CURSO DE SISTEMAS DE INFORMAÇÃO

OTIMIZAÇÃO DE ESCALAS DE PLANTÃO ORIENTADA POR ALGORITMO GENÉTICO: ESTUDO DE CASO NA SECRETARIA DE SEGURANÇA PÚBLICA DO TOCANTINS

PAULO DE SOUZA LIMA

Palmas - TO

2023

CURSO DE SISTEMAS DE INFORMAÇÃO

OTIMIZAÇÃO DE ESCALAS DE PLANTÃO ORIENTADA POR ALGORITMO GENÉTICO: ESTUDO DE CASO NA SECRETARIA DE SEGURANÇA PÚBLICA DO TOCANTINS

PAULO DE SOUZA LIMA

Trabalho apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS, como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação, sob a orientação do professor Me. Marco Antônio Firmino de Sousa.

Palmas - TO

2023

UNITINS TOCANTINS

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO DO CURSO DE SISTEMAS DE INFORMAÇÃO DA UNIVERSIDADE ESTADUAL DO TOCANTINS - UNITINS

Aos **24** dias do mês de **Junho** de **2023**, reuniu-se em sessão não presencial, às **08:30 horas**, por meio do Google Meet sob a Coordenação do Professor **Marco Antonio Firmino de Sousa** a banca examinadora de Trabalho de Conclusão de Curso em Sistemas de Informação, composta pelos examinadores Professor **Marco Antonio Firmino de Sousa** (Orientador), Professor **Douglas Chagas da Silva** e Professora **Tamirys Virgulino Ribeiro Prado**, para avaliação da defesa do trabalho intitulado **"Otimização de Escalas de Plantão Orientada por Algoritmo Genético: Estudo de Caso na Secretaria de Segurança Pública do Tocantins"** do acadêmico **Paulo de Souza Lima** como requisito para aprovação na disciplina Trabalho de Conclusão de Curso II (TCC II). Após exposição do trabalho realizado pelo acadêmico e arguição pelos Examinadores da banca, em conformidade com o disposto no Regulamento de Trabalho de Conclusão de Curso em Sistemas de Informação, a banca atribuiu a pontuação: 9,0.

Sendo, portanto, o Acadêmico: (X) Aprovado ( ) Reprovado

Assinam esta Ata:

Professor Orientador: _____

Examinador: _____

Examinador: _____

Marco Antonio Firmino de Sousa

Presidente da Banca Examinadora

Coordenação do Curso de Sistemas de Informação

Dados Internacionais de Catalogação na Publicação (CIP) Sistema de Bibliotecas da Universidade Estadual do Tocantins

S729o

SOUZA LIMA, Paulo de

Otimização de Escalas de Plantão Orientada por Algoritmo Genético: Estudo de Caso na Secretaria de Segurança Pública do Tocantins. Paulo de Souza Lima. - Palmas, TO, 2023

Monografia Graduação - Universidade Estadual do Tocantins – Câmpus Universitário de Palmas - Curso de Sistemas de Informação, 2023.

Orientador: Marco Antônio Firmino de Sousa

1. Gestão de Escalas. 2. Algoritmos Genéticos. 3. Aprendizado de Máquina. 4. Interface de Programação de Aplicativos (API).

CDD 610.7

Elaborado pelo sistema de geração automatica de ficha catalográfica da UNITINS com os dados fornecidos pelo(a) autor(a).

Este trabalho é dedicado a Deus e à minha família, pessoas que foram essenciais para que eu o conseguisse concluir com êxito.

Agradecimentos

Agradeço primeiramente a Deus pelo dom da vida e por ter me proporcionado chegar até aqui.

A minha família por toda a dedicação e paciência contribuindo diretamente para que eu pudesse ter um caminho mais fácil e prazeroso durante esses anos.

Ao professor Me. Marco Antônio Firmino de Sousa, por ter sido meu orientador e ter desempenhado tal função com dedicação e amizade.

Aos professores, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional ao longo do curso.

Aos meus colegas de curso, pelo companheirismo e trabalho em equipe nos momentos mais difíceis.

A Universidade Estadual do Tocantins, pela oportunidade de estar fazendo o curso.

Combati o bom combate, terminei a corrida, guardei a fé.
Agora me está reservada a coroa da justiça, que o
Senhor, justo Juiz, me dará naquele dia; e não somente
a mim, mas também a todos os que amam a sua vinda.
(Bíblia Sagrada, 2 Timóteo 4, 7-8)

Resumo

Este estudo aborda o desafio da gestão de escalas nos núcleos do Instituto de Criminalística, Instituto Médico Legal e Centro Integrado 18 de Maio, no Tocantins. Identificou-se uma deficiência no método atual de gestão manual das escalas, que, devido à sua inflexibilidade nos requisitos, sinaliza uma oportunidade valiosa para otimização, visando aumentar a eficiência e diminuir a incidência de erros. Com o objetivo de aprimorar a alocação da equipe e reduzir os custos administrativos, o estudo propõe uma abordagem híbrida que combina algoritmos genéticos e técnicas de aprendizado de máquina para criar uma solução otimizada. A solução proposta, desenvolvida em Python, tem como objetivo otimizar significativamente a gestão das escalas, buscando reduzir o tempo necessário para a sua montagem e minimizar a ocorrência de erros. A aplicação dessa solução será facilitada por meio de uma Interface de Programação de Aplicativos (API), que permitirá uma implementação suave e uma interação eficiente com outros sistemas existentes nos respectivos institutos. No entanto, a conclusão do estudo revelou que o algoritmo proposto não atingiu seus objetivos como esperado, principalmente devido às restrições estabelecidas pela instituição que não estavam totalmente em conformidade. Apesar de não produzir resultados tão promissores quanto esperado, a resiliência do algoritmo genético sob condições adversas reforça sua relevância na resolução de problemas complexos e a importância de sua futura exploração e otimização.

Palavras-chaves: Gestão de Escalas. Algoritmos Genéticos. Aprendizado de Máquina. Otimização. API. Python.

Abstract

This study addresses the challenge of shift management at the core of the Forensic Institute, Medical Legal Institute and Integrated Center 18 de Maio, in Tocantins. A deficiency was identified in the current manual shift management method, which, due to its inflexibility in requirements, signals a valuable opportunity for optimization, aiming to increase efficiency and decrease the incidence of errors. With the aim of improving team allocation and reducing administrative costs, the study proposes a hybrid approach that combines genetic algorithms and machine learning techniques to create an optimized solution. The proposed solution, developed in Python, aims to significantly optimize shift management, seeking to reduce the time required for its assembly and minimize the occurrence of errors. The application of this solution will be facilitated through an Application Programming Interface (API), which will allow a smooth implementation and an efficient interaction with other existing systems in the respective institutes. However, the conclusion of the study revealed that the proposed algorithm did not achieve its objectives as expected, mainly due to the restrictions established by the institution that were not fully complied with. Despite not producing results as promising as expected, the resilience of the genetic algorithm under adverse conditions reinforces its relevance in solving complex problems and the importance of its future exploration and optimization.

Key-words: Shift Management. Genetic Algorithms. Machine Learning. Optimization. API. Python.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API - *Application Programming Interface.*

ASGI - *Asynchronous Server Gateway Interface.*

CWI - *Centrum Wiskunde & Informatica.*

GPL - *General Public License.*

HTTP - *Hypertext Transfer Protocol.*

IML - Instituto Médico Legal.

OIT - Organização Internacional do Trabalho.

PSF - *Python Software Foundation.*

REST - *Representational State Transfer*

SI - Sistema de Informação.

SIG - Sistema de Informação Gerencial.

SOA - *Service Oriented Architecture.*

URI - *Uniform Resource Identifier.*

AG - Algoritmo Genético.

GRASP - *Greedy Randomized Adaptative Search Procedure.*

TS - *Tabu Search.*

ACO - *Ant Colony Optimization.*

SA - *Simulated Annealing.*

ILS - *Iterated Local Search.*

ML - *Machine Learning.*

IA - *Artificial Intelligence.*

Sumário

1 Introdução

Segundo Paz (PAZ; ODELIUS, 2021) a função gerencial desempenha, nas organizações, papel relevante para seu funcionamento e desenvolvimento, permitindo o alinhamento entre os setores dessas instituições. Tanto os níveis estratégicos e operacionais beneficiam-se com a adoção de medidas gerenciais no contexto das organizações, oportunizando assim o aumento no nível do engajamento individual dos colaboradores, bem como o desempenho das equipes. No contexto da administração pública brasileira, por mais que hajam esforços em se adotar políticas que fomentem modelos de competências gerenciais, poucas organizações mensuram suas competências, operações e lacunas por meio de instrumentos de coleta de dados quantitativos.

De acordo com Cerqueira (CERQUEIRA, 2021) a evolução do mercado de trabalho acarretou na transformação dos horários laborais. Algumas atividades humanas exigem atendimento e oferta contínuos, o que implica na necessidade de disponibilizar serviços em período integral, levando à adoção do sistema de trabalho por turnos. A adoção do trabalho por turnos possibilita a continuidade dessas atividades, garantindo que os serviços estejam disponíveis em diferentes horários, de acordo com a demanda e as necessidades dos usuários.

De acordo com Turhan et al (TURHAN; BILGEN, 2020), na gestão de recursos humanos, particularmente na alocação de funcionários, surge a necessidade de uma distribuição eficaz destes profissionais. No entanto, diversos fatores preponderantes podem complicar essa tarefa. Uma solução prática para essa problemática é a criação de um cronograma de trabalho que leve em consideração variáveis críticas como a legislação trabalhista, a disponibilidade dos funcionários, as solicitações pessoais de cada colaborador, além de organizar o serviço durante o período estipulado.

Diante dessa circunstância, este estudo propõe uma solução para um desafio encontrado nos Institutos de Criminalística, Médico Legal e Centro Integrado 18 de Maio em Tocantins, todos pertencentes à Secretaria de Segurança Pública do Estado. Atualmente, essas instituições enfrentam a ausência de um sistema automatizado para a gestão das escalas de trabalho dos servidores de plantão, levando a processos manuais que são propensos a erros e imprecisões.

Nessa perspectiva, a proposta busca otimizar as escalas de plantão, visando melhorar o dimensionamento das equipes de trabalho dentro das restrições de jornada impostas. O problema identificado foi a falta de eficiência na escala de plantões, que levou ao desenvolvimento. Esta aplicação, que tem como principal função redistribuir as escalas de plantão de maneira mais eficiente, foi testada no ambiente prático onde o problema

inicialmente surgiu. Com a finalidade de avaliar os resultados proporcionados por esta aplicação, conduzimos testes no ambiente prático.

A estruturação deste trabalho está organizado na seguinte configuração: no seguimento deste Capítulo 1 está presente os objetivo geral e os específicos. Na sequência, o Capítulo 2 evidencia a revisão de literatura retratando os assuntos a respeito de: **jornada de trabalho, aprendizado de máquina, sistemas de informação, arquitetura orientada a serviços e algoritmo genético,**. Em continuidade, o Capítulo 3 é manifestado a metodologia que elaborou o algoritmo, assim como as etapas do desenvolvimento efetuado. O Capítulo 4 aborda os resultados obtidos e, por último, o Capítulo 5 exprime as ponderações finais.

1.1 Justificativa

Aprimorar a eficiência operacional, otimizar a utilização dos recursos humanos e garantir a conformidade com os regulamentos trabalhistas são objetivos primordiais que podem ser alcançados por meio da automação da gestão de escalas com a utilização de uma API. Esta API permitiria a implementação eficiente de algoritmos genéticos, atendendo simultaneamente às necessidades operacionais das instituições públicas, especialmente nos Institutos de Criminalística, Médico Legal e no Centro Integrado 18 de Maio, localizados na Superintendência da Polícia Técnica Científica, no estado do Tocantins. A relevância do uso de algoritmos genéticos na gestão de escalas nesses institutos se dá por sua capacidade de otimizar soluções em ambientes complexos. Eles equilibram eficiência operacional, conformidade regulatória e demandas institucionais específicas de maneira adaptável e escalável.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo implementar uma Interface de Programação de Aplicativos (API) para um sistema de alocação de pessoal para composição de escalas de serviços baseados na formulação de um problema multiobjetivo restrito.

1.2.2 Objetivos Específicos

- Identificar a fundamentação teoria associada à problemas estocásticos;
- Identificar pelo menos seis algoritmos apropriados para composição de escalas;
- Selecionar um algoritmo viável para a implementação da solução;

- Caracterizar a escala de serviços através de uma representação de conhecimento apropriado para o algoritmo selecionado;
- Analisar os resultados dos experimentos.

2 Referencial Teórico

Este capítulo é constituído pelo embasamento teórico para a elaboração deste trabalho, composto pelas seguintes temáticas: jornada de trabalho, aprendizado de máquina, algoritmo genético, sistemas de informação, arquitetura orientada a serviços e as ferramentas utilizadas.

2.1 Jornada de Trabalho

2.1.1 Origem

A partir dos primórdios do desenvolvimento da humanidade, o indivíduo executava certas responsabilidades tendo a finalidade de assegurar a permanência da sua vida. De acordo com Cassar (2011), o significado da palavra trabalho advém do termo em latim *tripalium* no qual consistia ser um instrumento de tortura ou para dominação dos animais. Desta forma, os nobres não laboravam por julgarem ser uma penalização.

Segundo Martins (2010), a primeira constituição de trabalho se deu pela escravidão, uma vez em que os vassalos, classificados como objeto de posse, dedicavam-se em jornadas de trabalho duradouras desprovidos de algum benefício, logo não desfrutavam de nenhum direito e com possibilidades de serem punidos, cambiados, vendidos e inclusive mortos.

Nascimento (2017) complementa que mais adiante, a sociedade se organizou em feudos, iniciando um método de trabalho denominado de servidão onde os senhores feudais garantiam segurança aos servos porém os mesmos não possuíam liberdade e eram obrigados a pagarem imensa parcela de sua produção agrária aos seus senhores como forma de cobrança pelo serviço prestado.

Conforme Neto e Cavalcante (2007) na época da Idade Média manifestaram-se as corporações de ofício, ou guildas, compostas por mestres, oficiais e aprendizes, que tinham por objetivo agrupar os subordinados de áreas similares com a finalidade de elaborar condutas que doutrinariam os vínculos laborais e permitiam uma maior autonomia dos membros, entretanto existiam para assegurar os interesses das corporações e não favorecer os trabalhadores.

Porém, ao término do Século 18, as ideias surgidas da Revolução Francesa erradicaram quaisquer corporações de ofício e impedidas de serem restauradas, tais ações fundamentadas por serem discordantes com a independência individual. Portanto, os princípios de liberdade, igualdade e fraternidade deste marco autorizavam a pessoa praticar seja qual for atividade laboral, contanto que exercesse o pagamento dos tributos e acatasse

as normas definidas (MARTINEZ, 2020).

2.1.2 Revolução Industrial

Segundo Beltran (2002), o aumento da oferta de mão de obra associada a carência de amparo por parte do Estado acarretaram para que as empresas sujeitassem os seus funcionários com péssimas condições de trabalho, análogas ao período de escravidão, nos quais não possuíam preservação à higiene, saúde e assistência no serviço, além de causar enfermidades como asma, tuberculose e pneumonia. Caso estes operários fossem constatados com alguma doença que inviabilizasse a execução de sua tarefa, corresponderia a substituição por outro indivíduo que aceitasse desprovido de nenhuma garantia.

Hobsbawn (2015) explica que pela necessidade iminente de alguma resolução para o problema, manifestaram-se grupos, com ideais oriundos da Revolução Francesa, implorando que tivesse uma intervenção estatal nos vínculos empregatícios em benefício ao conforto da sociedade, principalmente dos menos favorecidos.

Conforme Delgado (2019), com o crescimento destes movimentos, se deu a origem das primeiras leis relacionadas ao direito do trabalho. Enfatiza-se a Lei Moral e Saúde promulgada na Inglaterra que lidava com o impedimento do expediente superior a doze horas e trabalho infantil no período noturno.

Posteriormente a conclusão da Primeira Guerra Mundial, os privilégios trabalhistas ganharam relevância através de mobilizações constitucionalistas a fim de que estes direitos pudessem estar presentes na carta magna do seu país até que no ano de 1919, por intermédio do Tratado de Versalhes, culminou na criação da Organização Internacional do Trabalho (OIT) incumbido de ser responsável pela estabilidade e proteção das relações de trabalho em âmbito universal (MARTINS, 2015).

2.2 Regime de Trabalho

De acordo com as garantias dadas pela Constituição Federal[1], com base na Lei nº 8.112/90 no Art. 1º. Esta Lei institui o Regime Jurídico dos Servidores Públicos Civis da União, das autarquias, inclusive as em regime especial, e das fundações públicas federais.

O Regime estatutário [2]. LEI Nº 1.818, DE 23 DE AGOSTO DE 2007 que "Dispõe sobre o Estatuto dos Servidores Públicos Civis do Estado do Tocantins." Art. 19. A mesma estabelece que, os servidores devem cumprir sua jornada de trabalho de acordo com as necessidades do exercício de suas atribuições, pertinentes a seus respectivos cargos,

[1] http://www.planalto.gov.br/ccivil$_0$3/*leis*/*l*8112*cons.htm*

[2] https://www.tjto.jus.br/index.php/documentos-licitacoes/173-lei-1-818-estatuto-do-servidor-publico-2/file

respeitada a duração máxima do trabalho semanal de 40 horas e observados os limites mínimo e máximo de 6 horas e 8 horas diárias, respectivamente.

§ 2º Regulamento disciplina a jornada de trabalho dos titulares de cargos de provimento efetivo cujo exercício exija regime de turno ou plantão.

2.3 Sistemas de Informação

2.3.1 Conceito

Sistema de informação é caracterizado pela composição de elementos associados com a finalidade de colher, processar, armazenar e compartilhar a informação para favorecer o manuseio, organização, estruturação, investigação e o método decisório nas instituições (LAUDON; LAUDON; TEIXEIRA, 2010).

Dornelas (2000) esclarece que o princípio da utilização de computadores no contexto de procedimentos empresariais se deu na década de 1950, com ascensão a partir no decênio seguinte, através do emprego de $mainframe$. Ergueram-se, neste período, sistemas gerenciais orientados a fornecer informações para logística decorrendo concessão gradual e cobertura.

Este progresso tecnológico viabilizou a ampliação e redução do curso de processamento e armazenamento, assim como o aprimoramento nas linguagens de programação próprias ao desenvolvimento. Também permitiu que os parâmetros destes sistemas estratégicos fossem baseados nas preferências do usuário (REZENDE; ABREU, 2006).

É possível catalogar os sistemas de informação (SI) em diferentes modelos. Uma destas categorizações tem sua origem pela necessidade das organizações na utilização destas aplicações em todos os aspectos relacionados, logo podem ser tipificados conforme o gênero da atividade envolvida. Bigdoli (1989) expõe três perspectivas a respeito dessa percepção para os sistemas de informação assim como Laudon *et al* (2010) caracteriza-os da seguinte forma:

- SPT (Sistema de Processamento de Transação): representa, detalhadamente, o progresso e as consequências das transações, procedimentos e técnicas diárias essenciais para construção dos negócios empresariais e indica as normas prefixadas a fim de domínio e decisão.

- SIG (Sistema de Informação Gerencial): opera a nível administrativo da instituição executando atribuições de organização, coordenação e determinações sobre práticas diárias e eventuais desvios.

- SAD (Sistema de Apoio à Decisão): encaminhado aos líderes da empresa tendo objetivo de conciliar as informações e modelos analíticos para auxiliar na tomada de decisão por meio de metodologias gráficas.

2.3.2 Sistemas de Informação Gerencial (SIG)

Segundo Stair e Reynolds (2020), o ponto focal dos sistemas é o resultado alcançado, por intermédio das informações, ser entregue para área responsável pela gestão. O sistema de informação gerencial é descrito como uma agregação dos conceitos da doutrina relacionada a administração com a construção de soluções através de recursos tecnológicos instruídos pela pesquisa operacional, convertendo dados em conhecimento.

Oliveira (2018) declara que o SIG é expressado tal qual uma maneira de apoiar e fundamentar as providências dentro da organização, tornando-as mais ágeis. Desta forma, é indispensável que seja competente com a finalidade de ser satisfatório nas metodologias de determinações, prevenindo incoerências e desperdícios.

Em conformidade com Lapolli (2003), o sistema de informação gerencial retrata tópicos comportamentais e o vínculo existente na utilização e repercussão dos SI na coordenação e colaboradores de uma instituição. Dentre os diversos benefícios, pode-se destacar o crescimento na produtividade, na tomada de decisão e nas atividades efetuadas assim como a diminuição dos custos.

2.4 Arquitetura Orientada a Serviços

2.4.1 Definição

Para Sommerville (2011), se faz imprescindível instituir uma arquitetura estruturada na aplicação por apresentar os principais componentes e como os relacionam mutuamente, tendo o objetivo de contribuir com as etapas de desenvolvimento e manutenção, além de reduzir os custos. Arquitetura em camadas, orientada a objetos, cliente servidor, repositórios e orientada a serviços são alguns dos estilos de arquitetura de $software$.

É possível caracterizar a arquitetura orientada a serviços (SOA) como um paradigma nos processos envolvendo sistemas distribuídos. Progressivamente sendo empregada nas organizações por se tratar de uma tática eficiente com cerne no desenvolvimento e produtividade em prol de elaborar uma aplicação modularizada (STAL, 2006).

A proposta do SOA é oportunizar as corporações desfrutarem de vantagens tecnológicas através da agregação de governança, estratégia e processos, resultando na integração entre os sistemas, baseada em três fundamentos: serviços, baixo acoplamento e interoperabilidade. Com este paradigma, torna-se capaz de incorporar recursos e/ou serviços de

terceiros, independente de idade ou linguagem de programação da aplicação (ENDREI et al., 2004).

O conceito no ambiente computacional de serviço refere-se a funcionalidade que dispõe de uma interface e requisitado por intermédio de mensagens, não apresentando conexão direta ao serviço. Esta interface é chamada de API *(Application Program Interface)* em que relaciona entre sistemas, favorecendo a troca de informações, sem deter o conhecimento da forma em que foi implementado (SCHELLER; KÜHN, 2015).

Em virtude que ocorra essa agregação entre aplicações, podendo serem desenvolvidas em linguagens de programação distintas, é fundamental que a documentação da API, contendo a entrada e saída dos dados bem como suas funcionalidades esteja explícita, concisa e de fácil entendimento a fim de prevenir inconsistências. Logo, existem dois modelos para a construção de APIs: SOAP e REST (RICHARDSON; AMUNDSEN; RUBY, 2013).

2.4.2 API REST

O termo *REST* (*Representational State Transfer*), originado por Roy Fielding (2000), representa um estilo arquitetural no qual compõe-se de uma coleção de princípios e restrições. No caso de uma API implementada seguindo as normas do modelo *REST*, a interface é denominada de *API RESTful*. Em outros termos, este padrão tem por objetivo de uniformizar as melhores convenções para conduzir os princípios do HTTP e URI com a intenção de explorar os recursos disponíveis.

Surgido em 1996, o *Hypertext Transfer Protocol* (HTTP) aborda ao protocolo na camada de aplicação, baseado no *Open System Interconnection* (OSI), no qual, a fim de auxiliar o controle da aplicação, as requisições são executadas no lado do cliente e os retornos por parte do servidor. O HTTP conta com métodos baseados nas funcionalidades dos recursos, dentre os quais os principais são: GET, POST, PUT e DELETE. Além do mais, por esse protocolo, o servidor consegue responder a requisição por um código de status, divididos em cinco categorias, com o propósito de favorecer o entendimento do cliente (KUROSE; ROSS, 2013).

Segundo Matos (2013), o *Uniform Resource Identifier* (URI) é uma série de caracteres que apontam os recursos encaminhados para os outros elementos da aplicação e os identificam. É composto pelo nome do protocolo, a autoridade (domínio e porta), caminho, recurso, *query string* e fragmento. Logo a URI é modelada na seguinte forma: ***protocolo://domínio:porta/caminho/recurso?querystring#fragmento***.

Há seis restrições que foram elaboradas para estabelecer esse estilo de arquitetura REST, possibilitando as melhores práticas de desenvolvimento, elaboradas por Fielding. As condições são: cliente-servidor, interface uniforme, cache, sem estado, sistema em camadas

e código sob demanda (FERREIRA; KNOP, 2017).

Em relação ao formato dos dados, no qual viabiliza a propagação das informações pela API, um modelo é usado em maior quantidade, o *JSON*[3]. Criado por Douglas Crockford, baseado na linguagem *JavaScript*[4], possui uma sintaxe compreensível de fácil manipulação e análise, além de ser estruturado através de pares compostos por chave e valor (LANTHALER; GÜTL, 2012).

2.5 Métodos de Otimização Multiobjetivo

Segundo Zhang (ZHANG; LI, 2007) a otimização multiobjetivo é um campo de pesquisa em constante evolução, que visa encontrar soluções que atendam a vários objetivos simultaneamente. Isso é especialmente relevante em problemas complexos, onde há uma grande variedade de objetivos conflitantes a serem considerados.

De acordo com (ZHANG; LI, 2007) existem vários métodos de otimização multiobjetivo, entre os quais os principais são:

- **Método de ponderação:** utiliza uma técnica estatística que atribui pesos diferentes a dados numa pesquisa para que eles representem adequadamente uma população;
- **Método de otimização baseados em dominância:** busca soluções que não sejam dominadas por outras soluções no espaço de busca. Uma solução é dominada por outra se ela for pior em todos os objetivos;
- **Método de otimização baseados em fronteiras de Pareto:** busca soluções que pertencem à fronteira de Pareto, que é a fronteira do conjunto de soluções não-dominadas;
- **Método de busca aleatória:** esses métodos utilizam técnicas de busca aleatória para explorar o espaço de busca e encontrar soluções ótimas;
- **Método híbrido:** combinam diferentes abordagens para a otimização multiobjetivo, a fim de obter um melhor desempenho.

Considerando resolver problemas que envolvem múltiplos objetivos, onde a otimização de um único objetivo pode levar a soluções sub-ótimas. Com os métodos, é possível encontrar um conjunto de soluções ótimas que representam um equilíbrio entre os diferentes objetivos e permitem ao tomador de decisão escolher a melhor opção.

[3] https://www.json.org
[4] https://www.javascript.com

2.5.1 Metaheurísticas

Para Osman(OSMAN; LAPORTE, 1996) são técnicas avançadas de otimização que buscam encontrar soluções ideais ou perto do ideal para problemas complexos, muitas vezes envolvendo grande espaço de busca ou número elevado de variáveis.

Existem vários métodos de otimização multiobjetivo, incluindo técnicas baseadas em heurística, algoritmos genéticos, algoritmos evolutivos, otimização por enxame de partículas, algoritmos de colônia de formigas e outros. Cada método tem suas próprias vantagens e desvantagens, dependendo do problema em questão e do ambiente de computação disponível.

Segundo Blum (BLUM; ROLI, 2003) as metaheurísticas têm a capacidade de encontrar soluções de qualidade em um tempo razoável, mesmo para problemas muito complexos, e são muito utilizadas em casos em que as técnicas de otimização clássicas não são eficazes.

2.5.2 Métodos de Otimização Combinatória

Conforme Blum (BLUM; ROLI, 2003) os principais algoritmos são:

- **Procedimento Aleatório Adaptativo Guloso** *Greedy Randomized Adaptative Search Procedure* (GRASP) se trata de um algoritmo utilizado em problemas de otimização combinatória. A aplicação do GRASP se baseia na criação de uma solução inicial e em seguida a utilização dessa solução para realizar uma busca local e aprimorar a qualidade do resultado. A diferenciação da técnica para as demais se dá no momento da concepção da solução inicial;

- **Busca Tabu** *Tabu Search* (TS) é um método de resolução de problemas de otimização de um processo adaptativo auxiliar que direciona um algoritmo de busca particular na exploração dentro de um delimitado espaço de pesquisa. A partir de um resultado inicial o método tenta avançar para uma solução melhor (dentro de um determinado intervalo ou espaço), até que satisfaça um um certo critério para sua parada;

- **Colônia de Formigas** *Ant Colony Optimization* (ACO) é um algoritmo utilizado para a solução de problemas combinatórios inspirado no comportamento de formigas na busca de alimentos. Quando uma formiga precisa decidir para onde ir, ela usa informação proveniente de feromônio previamente depositado por outras formigas que passaram por aquele local;

- **Recozimento Simulado** *Simulated Annealing* (SA) é um método de busca randômica sendo dessa forma classificado como método estocástico, que apresenta como critério de convergência princípios termodinâmicos;

- **Busca Local Iterativa** *Iterated Local Search* (ILS) é um procedimento de busca local que baseia-se na ideia que o procedimento de busca local pode ser melhorado gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações na solução correspondente ao ótimo local;
- **Algoritmos Genéticos** *Genetic Algorithms* (AGs) são algoritmos de otimização global, baseados nos mecanismos de seleção natural e da genética. A técnica de Algoritmos Genéticos fornece um mecanismo de busca adaptativa que se baseia no princípio Darwiniano de reprodução e sobrevivência dos mais aptos.

2.6 Características dos Métodos de Otimização Combinatória

2.6.1 Procedimento Aleatório Adaptativo Guloso

Segundo (FEO; RESENDE, 1995) o método GRASP é um processo iterativo que envolve a construção de soluções parciais e a melhoria dessas soluções por meio de uma busca local.

- Construção gulosa: Nessa etapa, uma solução parcial é construída de forma gulosa, ou seja, de maneira iterativa, adicionando elementos à solução de acordo com algum critério de seleção. A seleção é baseada em informações locais disponíveis no momento da escolha;
- Aleatoriedade: Após a construção gulosa, o método introduz aleatoriedade no processo de busca. Isso é feito por meio de uma fase de melhoria local, onde é permitido realizar movimentos aleatórios para explorar soluções vizinhas à solução parcial atual;
- Critério de parada: O processo de construção gulosa e melhoria local é repetido até que um critério de parada seja atingido. Esse critério pode ser um número fixo de iterações, um tempo máximo de execução ou a obtenção de uma solução considerada suficientemente boa.

2.6.2 Busca Tabu

Segundo (GLOVER; LAGUNA, 1997) A busca tabu é um método de otimização que visa superar as limitações dos algoritmos de busca local tradicionais, como o algoritmo de busca em profundidade ou busca em vizinhança.

O método consiste em repetir os seguintes passos:

- Geração de uma solução inicial;

- Seleção da melhor solução na vizinhança, levando em consideração as restrições da lista tabu e o critério de aspiração;
- Atualização da lista tabu com informações relevantes sobre a solução selecionada;
- Verificação do critério de parada.

2.6.3 Colônia de Formigas

De acordo com (DORIGO; STÜTZLE, 2004) é um método de otimização inspirado pelo comportamento de formigas reais em busca de alimentos.

O seu funcionamento básico do algoritmo:

- Inicialização: O algoritmo começa definindo um conjunto de formigas artificiais e inicializando as trilhas de feromônio. Cada formiga é colocada em uma posição inicial no espaço de busca;
- Construção de soluções: As formigas começam a construir soluções parciais para o problema. Elas se movem pelo espaço de busca seguindo regras heurísticas e estocásticas. A regra heurística geralmente é baseada em informações sobre a qualidade da solução, enquanto a regra estocástica considera os níveis de feromônio nas trilhas;
- Atualização das trilhas de feromônio: À medida que as formigas se movem, elas deixam rastros de feromônio nas trilhas percorridas. A quantidade de feromônio depositada é geralmente proporcional à qualidade da solução encontrada pela formiga. Trilhas mais curtas ou de melhor qualidade recebem uma quantidade maior de feromônio;
- Evaporação do feromônio: Para evitar a convergência prematura para soluções subótimas, o feromônio evaporará ao longo do tempo. Isso permite que o algoritmo explore diferentes áreas do espaço de busca. A taxa de evaporação é um parâmetro que pode ser ajustado para equilibrar a exploração e a intensificação da busca;
- Reforço de trilhas: Conforme as formigas seguem as trilhas de feromônio mais fortes, as trilhas mais curtas e de melhor qualidade são reforçadas. Esse reforço ocorre porque as formigas depositam mais feromônio nessas trilhas, tornando-as mais atraentes para as formigas subsequentes;
- Critério de parada: O algoritmo continua iterando, construindo soluções e atualizando as trilhas de feromônio até atingir um critério de parada. Isso pode ser um número máximo de iterações, uma condição de convergência ou qualquer outra métrica definida pelo problema em questão.

2.6.4 Recozimento Simulado

Conforme (KIRKPATRICK; JR; VECCHI, 1983) O método de Recozimento Simulado funciona através de um processo de busca probabilística, onde são geradas soluções candidatas e avaliadas em relação a uma função objetivo.

Os passos principais do algoritmo são:

- Inicialização: O algoritmo começa selecionando uma solução inicial de forma aleatória ou através de alguma heurística. Essa solução inicial serve como ponto de partida para a busca
- Geração de soluções vizinhas: A partir da solução atual, são geradas soluções vizinhas através de alguma estratégia de perturbação. Isso envolve modificar a solução atual de alguma forma para explorar regiões diferentes do espaço de solução. Essa perturbação pode ser feita de forma aleatória ou seguindo alguma heurística específica do problema;
- Cálculo da função de energia: A função de energia, também conhecida como função objetivo, é usada para avaliar a qualidade das soluções. Ela mede o desempenho ou a adequação da solução de acordo com o problema específico. O objetivo é maximizar essa função, embora em alguns casos possa ser necessário minimizá-la;
- Aceitação ou rejeição de soluções vizinhas: A aceitação de uma nova solução depende da função de energia e do estado atual do algoritmo. Uma solução melhor sempre é aceita. No entanto, soluções piores podem ser aceitas temporariamente com uma certa probabilidade. Essa probabilidade é controlada pela temperatura do sistema, que é gradualmente reduzida ao longo do tempo;
- Esquema de resfriamento: A temperatura é reduzida de acordo com um esquema de resfriamento específico. Esse esquema determina como a temperatura é atualizada a cada iteração do algoritmo. Geralmente, a temperatura é reduzida lentamente para permitir uma exploração mais global do espaço de solução no início e uma exploração mais localizada à medida que a temperatura diminui;
- Critério de parada: O algoritmo continua iterando pelos passos 2 a 5 até que seja atendido algum critério de parada. Isso pode ser um número máximo de iterações, um limite de temperatura ou a convergência para uma solução satisfatória;

2.6.5 Busca Local Iterativa

Conforme (LOURENÇO; MARTIN; STÜTZLE, 2003) trata-se de um procedimento que também utiliza técnicas meta-heurísticas de busca local. Este, baseia-se em fazer

uma busca local por melhores soluções a partir de soluções ótimas locais por meio de perturbações.

- Inicialização: O processo começa com a seleção de uma solução inicial para o problema. Isso pode ser feito de várias maneiras, dependendo do problema em questão. A solução inicial pode ser escolhida aleatoriamente, com base em algum conhecimento prévio ou por meio de heurísticas específicas;
- Geração de vizinhos: Uma vez que a solução inicial é selecionada, a busca local iterativa gera soluções vizinhas a partir da solução atual. A definição da vizinhança depende do problema em consideração. Pode envolver pequenas modificações na solução atual, como alterar um único elemento ou fazer uma pequena perturbação nos valores da solução;
- Avaliação da vizinhança: Cada solução vizinha gerada é avaliada para determinar se ela é melhor ou pior do que a solução atual. A função de avaliação é definida de acordo com o objetivo do problema de otimização. Pode ser uma função de custo a ser minimizada ou uma função de utilidade a ser maximizada;
- Atualização da solução atual: Se uma solução vizinha é melhor do que a solução atual, ela se torna a nova solução atual. Isso significa que a busca local está progredindo em direção a uma solução de melhor qualidade. Caso contrário, a solução atual permanece a mesma;
- Condição de parada: O processo de geração de vizinhos, avaliação e atualização da solução atual continua até que uma condição de parada seja satisfeita. Isso pode ser um número máximo de iterações, um limite de tempo, uma melhoria mínima na solução ou qualquer outra condição definida;
- Retorno da solução final: Quando a condição de parada é alcançada, o algoritmo retorna a solução atual como a solução final encontrada pela busca local iterativa.

2.6.6 Algoritmos Genéticos

Os algoritmos genéticos são uma técnica poderosa e versátil de otimização que pode ser usada em uma ampla variedade de problemas. Eles são baseados no conceito da seleção natural, têm a capacidade de lidar com espaços de busca complexos e são adaptativos a mudanças no ambiente de busca. No entanto, eles são limitados, mas permitem ajustes cuidadosos dos parâmetros para obter resultados eficazes.

De acordo com (BÄCK; FOGEL; MICHALEWICZ, 2018) os algoritmos genéticos funcionam com base em regras simples: a primeira geração é criada aleatoriamente, no espaço de solução do problema. A partir dela, as próximas gerações são criadas através de

regras probabilísticas envolvendo a geração anterior. Este processo nada mais é do que a cópia de strings completas ou de frações delas. As versões mais simples se resumem a três operadores: Seleção, *Crossover* e Mutação.

Segundo (KANIT et al., 2021) as condições de restrição de recursos é otimizado por meio de algoritmos genéticos, permitindo uma alocação eficiente de recursos e melhorando o desempenho geral do projeto.

O *Crossover* é o operador que combina os genes dos indivíduos selecionados para compor a próxima geração. Esse processo é inspirado na reprodução biológica, e pode gerar novas soluções que combinem características vantajosas de dois ou mais indivíduos.

A mutação é um operador que introduz uma pequena perturbação nos genes dos indivíduos selecionados. Esse processo é importante para evitar que o algoritmo fique preso em soluções locais ótimas e explore outras regiões do espaço de solução (BÄCK; FOGEL; MICHALEWICZ, 2018).

Para John Holland (JOHN, 1992) os Algoritmos Genéticos, são métodos de otimização e busca inspirada nos mecanismos de evolução de populações de seres vivos.

Foram fundamentados como:

- Utilização de codificação de soluções em forma de cromossomos, que representam as soluções candidatas do problema a ser resolvido;
- Aplicação de operadores genéticos, como a seleção, *crossover* e mutação, para criar novas soluções a partir das soluções existentes;
- Avaliação da qualidade das soluções criadas por meio de uma função objetivo, que atribui um valor numérico representando o desempenho da solução em relação ao objetivo a ser alcançado;
- Utilização de uma estratégia de seleção, que permite selecionar as soluções mais aptas para sobreviverem e se reproduzirem na próxima geração;
- Execução iterativa do algoritmo, com a geração de novas soluções a cada iteração, até que uma solução satisfatória seja encontrada ou um critério de parada seja atendido.

2.6.6.1 Análise Comparativa

O Algoritmo Genético se apresenta como uma escolha apropriada para dilemas de otimização e busca, particularmente quando a solução ideal não se revela prontamente acessível através de métodos tradicionais.

Aqui estão algumas justificativas potenciais para selecionar um algoritmo genético:

Tabela 1 – Comparação das características dos Algoritmos Genéticos

	Algoritmos Genéticos
Otimização de Problemas Complexos	Excelentes para resolver problemas de otimização complexos e não-lineares.
Busca Adaptativa	Capacidade de se adaptar a mudanças no ambiente de busca, útil para problemas dinâmicos.
Exploração de Novas Soluções	O recurso de mutação ajuda a evitar ficar preso em mínimos locais, explorando novas possibilidades.
Combinação de Soluções	O recurso de crossover permite combinar características de múltiplas soluções, levando a melhores resultados.
Otimização Multiobjetivo	Adequados para resolver problemas de otimização multiobjetivo, onde há várias funções objetivo que precisam ser otimizadas simultaneamente.

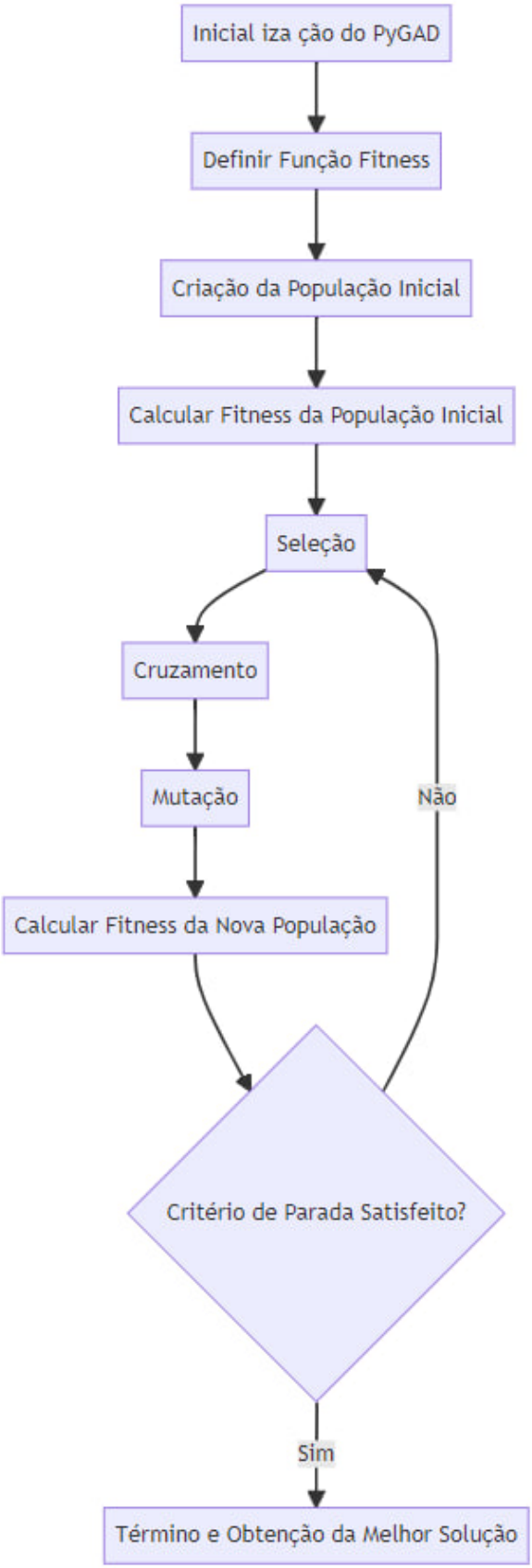
Fonte: Próprio Autor (2023)

2.6.6.2 Escolha e Estudo da Biblioteca PyGAD

O estudo se propõe a utilizar a biblioteca *PyGAD*, uma poderosa ferramenta de código aberto escrita em *Python. PyGAD*, especializada na implementação e manipulação de algoritmos genéticos. O propósito do uso do *PyGAD* neste estudo é explorar a capacidade desses algoritmos de encontrar soluções ótimas ou quase ótimas para complexos problemas de gerenciamento, conforme Figura (1).

Uma característica distintiva do *PyGAD* é sua flexibilidade e capacidade de personalização. Permite aos usuários definir suas próprias funções de adequação, operadores de cruzamento e mutação, estratégias de seleção e outros parâmetros. Isso torna a biblioteca adequada para uma ampla gama de problemas e aplicações, incluindo, mas não se limitando a, tarefas de aprendizado de máquina, otimização de estruturas de rede, agendamento de tarefas, otimização de rotas, entre outras.

Figura 1 – Fluxograma do Algoritmo Genético com PyGAD: utiliza técnica de otimização inspirada nos princípios da evolução biológica.



Fonte: Próprio Autor (2023)

2.6.7 Pseudocódigo

2.6.7.1 Funcionamento do Algoritmo Genético

Algoritmo 1: Algoritmo Genético

Entrada : Parâmetros do algoritmo genético

Saída: Melhor solução encontrada

```
1  Inicializar a população inicial;
2  Avaliar a aptidão de cada indivíduo;
3  enquanto critério de parada não for atingido faça
4      Selecionar indivíduos para reprodução;
5      Realizar cruzamento entre os indivíduos selecionados;
6      Aplicar mutação aos indivíduos resultantes do cruzamento;
7      Avaliar a aptidão dos novos indivíduos;
8      Substituir a população antiga pela nova população;
9  fim
10 Retornar o melhor indivíduo encontrado;
```

Um algoritmo genético pode ser descrito, superficialmente, através de um processo composto por etapas. Inicialmente, uma população inicial é criada e sua quantidade é determinada. Posteriormente, em cada iteração do algoritmo, os pais são selecionados dessa população para gerar filhos que são adicionados a ela. Além disso, cada indivíduo da população resultante pode sofrer alteração, o que consiste em alterações aleatórias nos cromossomos.

2.7 Aprendizado de Máquina

2.7.1 Conceito

Sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas através da solução bem sucedida de problemas anteriores. Os diversos sistemas de aprendizado de máquina possuem características particulares e comuns que possibilitam sua classificação quanto à linguagem de descrição, modo, paradigma e forma de aprendizado utilizado (MONARD; BARANAUSKAS, 2003).

Segundo (MONARD; BARANAUSKAS, 2003) o objetivo é o desenvolvimento de técnicas computacionais sobre o aprendizado bem como a construção de sistemas capazes de adquirir conhecimento de forma automática.

De acordo com (FERNANDES; FILHO, 2019) os algoritmos de aprendizado de máquina podem ser classificados, de maneira geral, em três categorias:

- **Aprendizagem Supervisionada** se baseia em dados preparados para treinamento

quando se sabe o desfecho de cada registro do conjunto de dados, em geral previamente rotulados por um especialista. Para a construção de um modelo de aprendizagem supervisionada, após a definição de um subconjunto da base de dados inicial para a análise, separa-se uma parte da amostra para realizar o treinamento e o ajuste do modelo, e outra parte para testar o desempenho do modelo;

- **Aprendizagem não Supervisionada** os algoritmos buscam padrões em registros com características similares, comparando os valores de seus atributos;
- **Aprendizagem por Reforço** o algoritmo interage repetidamente com um ambiente dinâmico com um objetivo específico como, por exemplo, ganhar um jogo ou dirigir um carro. O algoritmo chega à solução mais otimizada para o problema por meio de repetidos erros e tentativas.

A aprendizagem supervisionada utiliza dados com rótulos conhecidos, a não supervisionada busca por padrões em dados não rotulados, e a aprendizagem por reforço envolve interação com o ambiente para otimizar ações com base em recompensas e penalidades.

3 Metodologia

Este capítulo destaca uma abordagem de estudo de caso, tanto descritiva quanto exploratória, cujo objetivo principal é solucionar problemas inerentes à otimização e busca eficiente na gestão de horários de trabalho. Este estudo é direcionado especificamente para os núcleos do Instituto de Criminalística, do Instituto Médico Legal e Centro Integrado 18 de Maio.

3.1 Métricas de Avaliação

- A função precisa analisar o total de dias trabalhados, aplicando penalidades e recompensas conforme critérios definidos;
- A função tem como objetivo calcular a aptidão com base no custo;
- A função tem a finalidade de contar a quantidade de vezes que cada pessoa trabalhou em um determinado mês.

O objetivo é assegurar a distribuição otimizada de dias de trabalho e intervalos adequados de descanso.

3.2 Formulação da Equação da Função Objetivo

A função objetivo pode ser formulada como a diferença entre a soma dos produtos e a soma dos recursos, conforme expressa na Equação (3.2) abaixo:

$$C = \sum_{i=1}^{n} P(x_i) + \sum_{i=1}^{n} P(t_i) - \sum_{i=1}^{n} R(x_i) + \sum_{j=1}^{m} r_j$$

Onde:

- $P(x_i)$ é a penalidade para trabalhar menos que o mínimo de vezes no mês;
- $P(t_i)$ é a penalidade por ter um intervalo de dias de trabalho muito curto ou muito longo;
- $R(x_i)$ é a recompensa por trabalhar um número ideal de vezes no mês;
- r_j representa quaisquer outros custos.

3.3 Caracterização da Metodologia

Segundo Fonseca (FONSECA, 2002) *methodos* significa organização, e *logos*, estudo sistemático, pesquisa, investigação; ou seja, metodologia é o estudo da organização, dos caminhos a serem percorridos, para se realizar uma pesquisa ou um estudo, ou para se fazer ciência. Etimologicamente, significa o estudo dos caminhos, dos instrumentos utilizados para fazer uma pesquisa científica.

3.3.1 Descritiva

De acordo com (GIL et al., 2002) as pesquisas descritivas têm como objetivo primordial a descrição das características de determinada população ou fenômeno ou, então, o estabelecimento de relações entre variáveis. São inúmeros os estudos que podem ser classificados sob este título e uma de suas características mais significativas está na utilização de técnicas padronizadas de coleta de dados, tais como o questionário e a observação sistemática.

3.3.2 Exploratória

Para (GIL et al., 2002) a pesquisa exploratória normalmente ocorre quando há pouco conhecimento sobre a temática a ser abordada. Por meio de estudo exploratório, buscando conhecer com maior profundidade o assunto, de modo a torná-lo mais claro ou construir questões importantes para a condução da pesquisa.

3.3.3 Estudo de Caso

Segundo (GIL et al., 2002) é caracterizado pelo estudo profundo e exaustivo de uma ou de poucos objetos, de maneira a permitir conhecimento amplos e detalhados do mesmo, tarefa praticamente impossível mediante os outros tipos de delineamentos considerados.

3.3.3.1 Superintendência da Polícia Técnica Científica

A Polícia Científica é departamento ligado à polícia judiciária e ao sistema judiciário, especializada em produzir a prova técnica alicerçada em ciência, por meio da análise de vestígios produzidos e deixados durante a prática de delitos. A sua estrutura esta dividida em diretorias e seus núcleos.

A Superintendência da Polícia Técnica Científica do Estado do Tocantins atua em média aproximadamente com 40 (quarenta) escalas de plantão mensais de acordo com categoria do plantonista. Perícia de Trânsito, Crimes Contra Pessoal, Assistente Social, Psicologa, Agente de Necrotomia e Médico Legista.

3.3.4 Descrição do Problema

De acordo com a escala estabelecida, o colaborador trabalha por um dia inteiro e, em seguida, desfruta de três dias de descanso, totalizando um ciclo de quatro dias, ou 96 horas. Contudo, identificamos um desafio com o calendário, onde, em alguns meses, o início do mês não se alinha com a semana inicial. Isso se torna uma questão, considerando que é condição que o funcionário atue no mínimo duas vezes por semana.

Além disso, a escala de trabalho também considera a possibilidade de férias e licença para os funcionários. Durante o período de férias, o funcionário é liberado das suas responsabilidades laborais por um determinado tempo, permitindo que descanse e se recupere. As férias são um direito do trabalhador, assegurado por lei, e geralmente são concedidas após um período de trabalho contínuo.

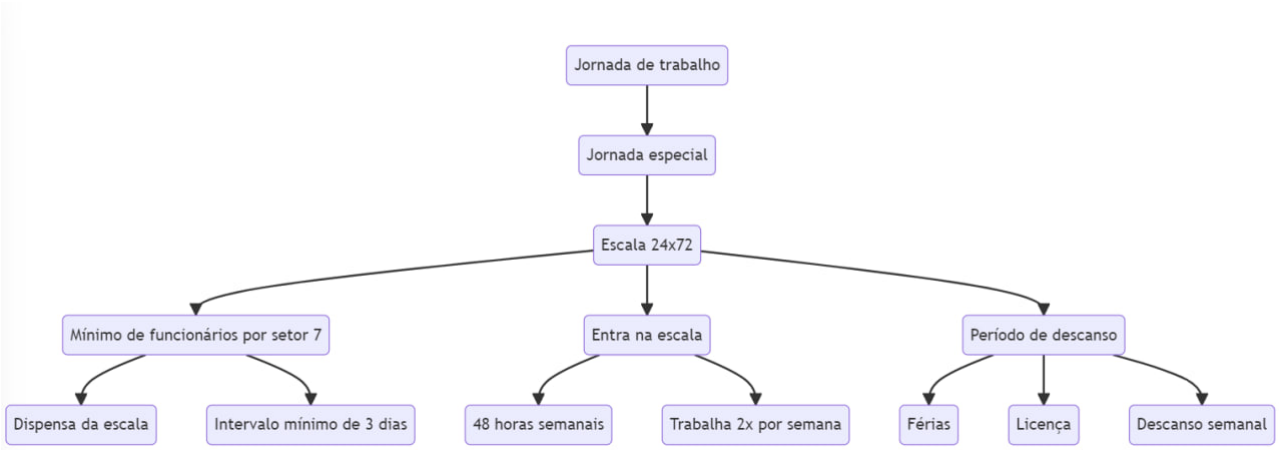
No entanto, se o início do mês não coincidir com o início da semana, o funcionário pode acabar trabalhando apenas uma vez em uma semana e três vezes em outra semana, o que não é permitido. Portanto, para garantir que o funcionário trabalhe pelo menos duas vezes por semana, ele deve começar a trabalhar sempre no início da semana, de modo que possa completar um ciclo de trabalho de 96 horas dentro de uma única semana. As restrições do problema são:

- Cada pessoa deve trabalhar pelo menos 8 vezes no mês (penalidade de 10 para cada vez que trabalhou menos que isso);
- Cada pessoa deve trabalhar no máximo 10 vezes no mês (recompensa de 100 para cada vez que trabalhou entre 9 e 10 vezes);
- Deve haver um intervalo mínimo de 3 dias entre os dias em que uma pessoa trabalha (penalidade de 50 para cada dia a menos do intervalo mínimo);
- Deve haver um intervalo máximo de 4 dias entre os dias em que uma pessoa trabalha (penalidade de 50 para cada dia a mais do intervalo máximo).

3.3.5 Mapa Mental

A partir das coletas obtidas na instituição, foi elaborado um mapa mental para visualizar e organizar as informações relevantes sobre a escala especial da instituição. Um mapa mental é uma representação gráfica que ajuda a mostrar as conexões entre ideias, conceitos, processos, departamentos e outras informações importantes.

Figura 2 – Mapa mental detalhado da escala especial da instituição



Fonte: Próprio Autor (2023)

3.4 Ferramentas Utilizadas

3.4.1 Visual Studio Code

O *Visual Studio Code* [1] é um editor de código-fonte desenvolvido pela Microsoft. Consiste em ambiente de desenvolvimento leve e altamente personalizável, projetado para atender às necessidades de programadores e desenvolvedores de software.

3.4.2 Python

Python[2] é uma linguagem de programação construída na conclusão do ano de 1989, entretanto somente ofertada para a comunidade no começo de 1991, por Guido van Rossum no Instituto de Matemática e Ciências da Computação (CWI), sediado em Amesterdã (Holanda), atualmente sendo sustentada pela *Python Software Fundation* (PSF). A origem do nome "*Pytho*" se dá através de um grupo humorístico denominado *"Monty Python's Flying Circus"*, reconhecido por suas obras cinematográficas britânicas (MANZANO, 2018).

De acordo (MANZANO, 2018) *Python* é uma linguagem de programação categorizada como alto nível, interativa, interpretada e multiparadigma, a linguagem engloba vários tipos de dados, com tipagem dinâmica, exceções e bibliotecas. Contendo uma sintaxe acessível e sucinta, proporciona melhor entendimento pela curva de aprendizagem menor, contribuindo com legibilidade e produtividade, aos novos adeptos da programação.

Para Borges (2014), o *Python* é considerado uma linguagem expressiva e dinâmica, no qual é empregado numa vasta esfera de aplicações. Inclui funcionalidades avançadas

[1] https://code.visualstudio.com
[2] https://www.python.org

como polimorfismo, herança múltipla, threads, sockets, expressões regulares e unidades de testes, auxiliando na estruturação e reutilização do código elaborado.

Conforme (CRUZ, 2015) *Python* é uma linguagem de código aberto *open-source*, compatível com os principais sistemas operacionais, como Linux, OSX, Windows, BSDs etc. Ela conta com uma vasta biblioteca padrão e documentação que possibilitam que muitas coisas sejam feitas sem dependências adicionais.

3.4.3 FastAPI

Conforme Palacio (2022), o FastAPI[3] é um *framework web Python*, isto é, conjunto de códigos (bibliotecas) responsáveis por funcionalidades no sistema, como requisições e servidores, desenvolvido por Sebastián Ramírez e lançado em 2018. Moderno, extremamente rápido e performático podendo ser objeto de comparação com outros *frameworks* tal qual *NodeJS* e *Go*, em virtude de utilizar as bibliotecas *Starlette* e *Pydantic*.

Shin e Han (2021) apresentam uma vantagem do *FastAPI* em relação aos demais *frameworks* desta linguagem, o uso do *Uvicorn* como interface de servidor assíncrona (ASGI). Isso significa que, por intermédio desse categoria de servidor, é capaz de processar altas taxas de transferências no contexto de entrada e saída dos dados, permitindo a criação de microsserviços assíncronos sem a necessidade adicional de controlar filas e rotinas.

Song e Kook (2022) também complementam outros benefícios no emprego do *FastAPI* na aplicação, bem como a redução de erros induzidos por humanos, velocidade no desenvolvimento, facilidade de aprendizado, produção de documentação instantânea, diminuição na duplicação do código, habilidade de validar entrada de dados automaticamente e compatibilidade nativa no formato *JSON*.

3.4.4 MongoDB

Por meio do surgimento da Web2.0[4], no qual a quantidade de dados elevou exponencialmente, as organizações especializadas em tecnologia careceram de inventar soluções que suportassem esse processamento, uma vez em que os bancos de dados relacionais já não estavam conseguindo satisfazer o consumo. Baseado nesse contexto, ocorreu a criação do banco de dados não relacional denominado comumente como *NoSQL* (DIANA; GEROSA, 2010).

Segundo Moniruzzaman e Hossain (2013), estes bancos, geralmente, evidenciam propriedades como: código aberto, flexibilidade no esquema, armazenamento de dados estruturados e não estruturados, replicação nativa, acesso via API e alta performance, escalabilidade, disponibilidade e confiabilidade. Existem quatro classificações para os

[3] https://fastapi.tiangolo.com

[4] Termo criado pela empresa O'Reilly em 2004 para caracterizar o conceito da internet como plataforma

banco de dados não relacionais, nos quais os modelos são orientados em: grafos, colunas, chave-valor e documentos.

De acordo com Kumar *et al* (2017), um dos bancos não relacionais orientado a documento mais notável refere-se ao *MongoDB*[5]. Criado no ano de 2007, a solução *open − source* conta com uma arquitetura composta de coleções (*collections*), em que expressa ser proporcional as tabelas no banco relacional, e os documentos são correspondentes aos registros.

Os documentos são armazenados no *MongoDB* no padrão *JSON* binário e aceitam variados tipos de dados como: *date*, *list*, *boolean*, *map*, *objects* e números em várias precisões. Além das características citadas acima relacionadas aos bancos não relacionais, também suporta índices e tipagem dinâmica. Por comportar esquema mutável, os documentos não necessitam ser idênticos na sua estrutura, logo podem conter atributos diferentes (BANKER et al., 2016).

3.4.5 Docker

Com o progresso tecnológico nos componentes que integram a parte física de um computador, estes recursos transformaram-se significativos em que sucedeu na construção de procedimentos de virtualização, técnica que oportuniza o desenvolvimento de variados ambientes computacionais, com o objetivo de rendimento superior relacionado a capacidade de memória, potência no processamento e armazenamento (MORAIS, 2015).

Docker[6], elaborado e mantido pela empresa Docker Inc, é uma plataforma de código fonte aberto focada na construção, execução e entrega de sistemas através de *containers*. Mouat (2015) revela que *containers* são elementos tecnológicos autônomos aptos a execução de aplicações no ambiente de computação. Desse modo, este componente concentra o sistema e suas respectivas dependências, proporcionando executar o programa isolado de outros aplicativos instalados na mesma máquina.

O *Docker* ganhou notoriedade e, portanto, difundido entre os desenvolvedores por se tratar de uma tecnologia em que viabiliza o processo de codificação e implantação de forma rápida e descomplicada. Seu ecossistema contempla um dispositivo para formação de imagens, em outros termos são uma sequência de camadas em modo de leitura nos quais os *containers* instanciados se baseiam, e o compartilhamento destas imagens por meio de um serviço em nuvem chamado *Docker Hub* (MERKEL, 2014).

[5] https://www.mongodb.com
[6] https://www.docker.com

3.4.6 Escolha do Software para Desenvolvimento

No que diz respeito ao software utilizado para desenvolver a aplicação, foi selecionado o *Visual Studio Code*. A escolha desse editor de texto se baseou na conveniência de ter todo o código da aplicação em um único ambiente e na vantagem de ser uma opção de licença gratuita. Para atingir os objetivos deste projeto, são utilizados os seguintes componentes de hardware.

Tabela 2 – Hardware

Descrição	**Fabricante**	**Versão / Modelo**
Sistema Operacional	Microsoft	Windows 10 Pro 64bits
Processador	Intel(R)	Intel(R) Core(TM) i7-2630QM
Memória RAM	Desconhecido	6GB
Armazenamento RAM	Kingston	SA400S37240GB
Placa de Video	Intel(R)	HD Graphics 3000
Placa-mãe	PHILCO	14A5

Fonte: Próprio Autor (2023)

3.5 Análise Detalhada do Critério de Avaliação

Flexibilidade é explorada pelo algoritmo por meio de variações nas atribuições de trabalho ao longo das gerações. O algoritmo permite a mutação dos genes, que representam as atribuições de trabalho para cada dia do mês. Essa mutação introduz uma certa aleatoriedade no processo, permitindo que novas combinações de atribuições sejam exploradas. Essa flexibilidade é essencial para evitar que o algoritmo fique preso em soluções sub-ótimas e para permitir a descoberta de soluções melhores ao longo do tempo.

Escalabilidade se relaciona com a capacidade de lidar com um número crescente de genes (atribuições de trabalho) e uma população maior de soluções.

Em relação ao número de genes, o algoritmo é flexível e pode ser facilmente adaptado para lidar com um número diferente de dias no mês. O código está configurado para receber a variável **dias-do-mes**, que determina o número de genes (dias) em uma solução. Portanto, é possível aumentar ou diminuir o tamanho do mês sem alterar significativamente a estrutura do algoritmo. Isso torna o algoritmo escalável em termos de lidar com diferentes tamanhos de problema.

Quanto à população de soluções, o algoritmo permite definir o número de soluções iniciais **sol-per-pop** e o número de pais selecionados para reprodução **num-parents-mating**. Esses parâmetros podem ser ajustados para lidar com uma população maior de soluções, permitindo explorar um espaço de busca mais amplo e potencialmente encontrar soluções melhores. No entanto, é importante notar que o aumento da população também

pode aumentar o tempo de execução do algoritmo, já que o número de avaliações *fitness* aumenta proporcionalmente.

Além disso, o algoritmo utiliza estratégias de seleção, *crossover* e mutação que são eficientes e escaláveis. A seleção dos pais é baseada no desempenho *fitness* e pode ser adaptada para lidar com um número maior de soluções. O *crossover* e a mutação são aplicados apenas nos pais selecionados, reduzindo o custo computacional dessas operações. Essas estratégias garantem que a evolução ocorra de forma eficiente e escalável.

Capacidade de Lidar com Múltiplas Variáveis se refere à habilidade do algoritmo em considerar e otimizar diversas características relacionadas à distribuição de trabalho entre as pessoas.

O algoritmo realiza a otimização considerando essas múltiplas variáveis através da função de avaliação *fitness*. A função de *fitness* calcula um valor que representa a qualidade da solução com base nas variáveis consideradas. Neste caso, a função de *fitness* é projetada para penalizar soluções que não atendem aos requisitos mínimos e recompensar aquelas que se enquadram nos critérios desejados.

O algoritmo utiliza operadores genéticos, como seleção, *crossover* e mutação, para explorar o espaço de busca de soluções e encontrar combinações ótimas das múltiplas variáveis. Através desses operadores, o algoritmo é capaz de gerar e evoluir soluções que atendam aos requisitos estabelecidos para cada variável.

Possibilidade de Gerar Soluções Ótimas ou Quase Ótimas embora não seja possível garantir a geração de soluções ótimas em todos os casos, o algoritmo busca encontrar soluções que sejam próximas do ótimo ou que atendam aos critérios estabelecidos de forma satisfatória.

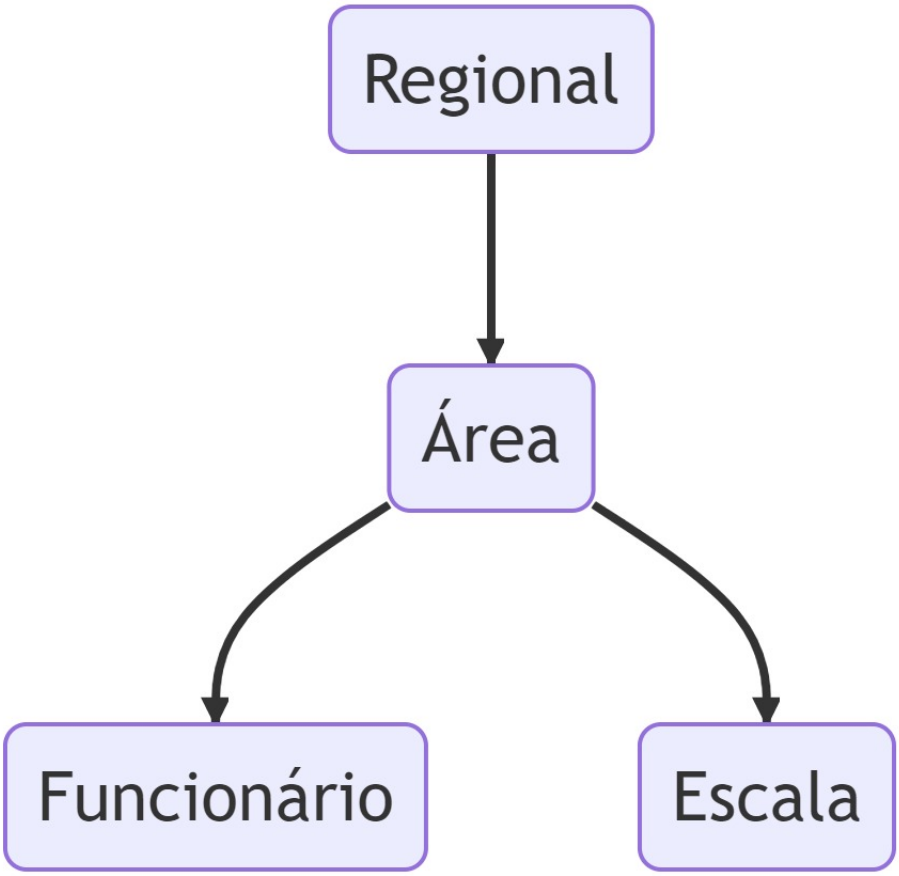
A população inicial de soluções é gerada aleatoriamente e, em seguida, o algoritmo evolui essa população por meio de seleção, *crossover* e mutação. A seleção favorece as soluções com melhor desempenho *fitness*, permitindo que elas sejam preservadas e combinadas para gerar soluções potencialmente melhores. O *crossover* combina informações genéticas das soluções parentais para gerar novas soluções, enquanto a mutação introduz pequenas alterações aleatórias para explorar novas regiões do espaço de busca.

Adaptabilidade o Algoritmo Genético possui certa adaptabilidade incorporada, principalmente por meio dos operadores genéticos, como seleção, *crossover* e mutação. Esses operadores permitem que o algoritmo explore e adapte as soluções à medida que evolui ao longo das gerações. A seleção dos pais com base no desempenho *fitness* permite que as soluções mais promissoras sejam preservadas, enquanto o *crossover* e a mutação introduzem variações nas soluções para explorar novas possibilidades e evitar a estagnação em mínimos locais.

É fundamental considerar a documentação da API em questão, que fornece infor-

mações supervisionadas pela instituição. É crucial compreender os requisitos e limitações específicas da API, levando em conta essas informações. Além disso, é importante projetar as classes de acordo com as necessidades, como ilustrado no diagrama mostrado na Figura 3.

Figura 3 – Representação das classes feita de acordo com as necessidades específicas, como exemplificado no diagrama.



Fonte: Próprio Autor (2023)

3.5.1 Framework FastAPI

Essa estrutura proporciona uma organização clara e eficiente das rotas e funções do sistema. Cada roteador desempenha um papel específico, contribuindo para uma distribuição lógica e bem definida. Como resultado, o sistema se torna mais gerenciável e eficiente.

- **Regionals-router**: Ele lida com as rotas relacionadas às regionais. Essas rotas têm o prefixo /api/v1/regionals e são agrupadas sob a *tag* **Regionals**;
- **Areas-router**: Esse roteador lida com as rotas relacionadas às áreas. Elas têm o prefixo /api/v1/areas e são agrupadas sob a *tag* **Areas**;

- **Employees-router**: Esse roteador lida com as rotas relacionadas aos funcionários. Elas têm o prefixo /api/v1/employees e são agrupadas sob a *tag* **Employees**;
- **Shifts-router**: Esse roteador lida com as rotas relacionadas aos turnos. Elas têm o prefixo /api/v1/shifts e são agrupadas sob a *tag* **Shifts**.

A abordagem de separar os *routers* por entidades é amplamente adotada e oferece benefícios significativos para a organização e compreensão do código.

O uso de uma API (Interface de Programação de Aplicações) proporciona benefícios tais como:

- Eficiência no desenvolvimento: reutilização de funcionalidades já prontas;
- Integração entre sistemas: comunicação eficiente entre softwares;
- Padronização: acesso uniforme a funcionalidades ou dados;
- Segurança: exposição controlada de funcionalidades ou dados;
- Escalabilidade: crescimento facilitado do sistema;
- Inovação: criação de novos serviços a partir de funcionalidades combinadas;
- Gerenciamento de dados: controle do fluxo de dados entre sistemas.

3.5.2 Aplicando o Algoritmo Genético para Otimização da Alocação de Tarefas

Para resolver o problema em questão, optamos por utilizar o algoritmo genético. Essa abordagem de otimização baseia-se na teoria da evolução natural e é implementada por meio da biblioteca PyGAD.

Os parâmetros foram definidos da seguinte maneira:

- num-parents-mating = 10: Este parâmetro define o número de pais que serão selecionados para a geração da próxima população por meio de cruzamento *(crossover)* e mutação;
- sol-per-pop = 50: Este parâmetro especifica o número de soluções (indivíduos) em cada população;
- num-genes = dias-do-mes: Define o número de genes em cada solução, que neste caso é igual ao número de dias do mês;
- gene-type = int: Especifica o tipo de dados que os genes podem assumir. Neste caso, são inteiros;

- init-range-low = 0 e init-range-high = len(pessoas) - 1: Definem o intervalo de valores que cada gene pode assumir inicialmente. Neste caso, qualquer valor inteiro entre 0 e o número de pessoas menos um;
- parent-selection-type = "sss": Define o método usado para selecionar os pais que gerarão a próxima geração;
- keep-parents = 1: Este parâmetro indica quantos dos indivíduos mais aptos (pais) da geração atual serão mantidos na próxima geração;
- crossover-type = "two-points": Define o método de *crossover* (cruzamento) que será usado para gerar novas soluções a partir dos pais. Neste caso, um cruzamento de dois pontos;
- mutation-type = "random": Define o método de mutação a ser usado. Neste caso, uma mutação aleatória;
- mutation-percent-genes = 10: Especifica a porcentagem de genes que sofrerão mutação;
- num-generations = 10000: Define o número total de gerações que o algoritmo percorrerá;
- on-generation = on-generation: Função de *callback* que será chamada em cada nova geração, passando a população atual e a geração atual como parâmetros.

Algoritmo 2: Conjunto de Parâmetros do Algoritmo Genético PyGAD

Result: População otimizada

Inicialize:

num-parents-mating = 10

sol-per-pop = 50

num-genes = dias-do-mes

gene-type = int

init-range-low = 0

init-range-high = len(pessoas) - 1

parent-selection-type = "sss"

keep-parents = 1

crossover-type = "two-points"

mutation-type = "random"

mutation-percent-genes = 10

num-generations = 10000

on-generation = on-generation

População = Inicialize uma população com sol-per-pop soluções, onde cada solução tem num-genes genes, cada gene é do tipo gene-type e tem um valor aleatório no intervalo de init-range-low a init-range-high

para *i = 0 até num-generations* **faça**

    Avalie a aptidão de todas as soluções na população;

    Selecione num-parents-mating pais da população usando o método parent-selection-type;

    Gere a nova população realizando crossover e mutação. Use crossover-type para crossover e mutation-type com mutation-percent-genes para mutação;

    se *keep-parents > 0* **então**

        Mantenha os keep-parents mais aptos da geração anterior na nova população;

    fim

    Chame a função on-generation com a população atual e a geração atual;

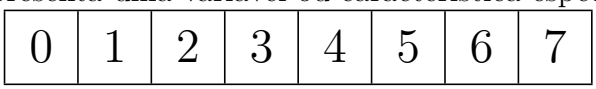
fim

retorna *População*

3.5.3 Representação do Cromossomo

No algoritmo genético, cada indivíduo da população é representado por um cromossomo único, que é uma lista com 30 elementos que correspondem aos **dias-do-mes**, uma lista composta por 30 inteiros, cada um correspondendo a um dia específico do mês. O valor do inteiro é o índice do perito na lista de peritos. Portanto, um cromossomo está definido como:

Cada gene representa uma variável ou característica específica da solução



Um cromossomo é uma lista de 30 números inteiros

A função de *fitness* avalia o custo de uma determinada solução, ou seja, a atribuição de peritos a dias, levando em consideração várias regras e restrições, como o número mínimo e máximo de dias que um perito deve trabalhar, intervalos de tempo entre dias de trabalho, e penalidades para trabalhar durante férias ou licenças. Essas regras determinam o quão boa ou ruim é uma solução.

As operações no algoritmo genético, como a seleção de pais, *crossover* e mutação, são aplicadas a esses cromossomos para gerar novas soluções ou seja, novas atribuições de peritos a dias na esperança de encontrar uma solução que minimize o custo.

3.6 Função Objetivo

A função objetivo do algoritmo é a função *fitness*, cujo objetivo é minimizar o custo e maximizar a recompensa ao atribuir dias de trabalho a um grupo de peritos. No contexto do algoritmo genético, a função objetivo é usada para avaliar a aptidão de cada indivíduo na população. Nesse caso, a função *fitness* é definida como o inverso do custo total mais um, para evitar divisões por zero.

Descrições detalhadas das funções:

- A função 3 **verificar-ferias-e-licencas** usada para calcular o custo associado a trabalhar durante as férias ou licenças de determinadas pessoas. Ela recebe uma lista de dias e uma lista de pessoas, onde cada pessoa possui uma lista de dias de férias ou licenças.

- A função 5 **calcular-custo** é usada para calcular o custo total das atribuições de pessoas aos trabalhos. Essa função verifica várias restrições, como o número mínimo de trabalhos por mês, recompensas para um número específico de trabalhos e penalidades por intervalos curtos ou longos entre os dias de trabalho. Ela calcula o custo baseado em várias condições:

  - As penalidades são aplicadas se um perito trabalhar menos que o mínimo de 8 vezes no mês;
  - As recompensas são dadas se um perito trabalha entre 9 e 10 vezes no mês;

  - As penalidades são aplicadas se o intervalo de dias entre dois dias de trabalho de um perito for muito curto (menos de 3 dias) ou muito longo (mais de 4 dias).

  Depois de calcular o custo, a função **fitness-func** calcula a aptidão como o inverso do custo mais um.

- A função 4 **vezes-que-trabalhou** conta o número de vezes que cada pessoa trabalhou no mês com base nas atribuições de pessoas em cada dia. Essa função retorna um dicionário em que cada chave é uma pessoa e o valor é o número de vezes que essa pessoa trabalhou.

A função objetivo, essencial para nossa análise, é estruturada com base na distinção entre a agregação dos produtos e o acumulado dos recursos. Essa formulação matemática, demonstrada na Equação (3.2).

O algoritmo genético busca otimizar a distribuição de dias de trabalho entre os peritos, garantindo que cada perito trabalhe um número adequado de vezes e que tenha um intervalo adequado de descanso entre os dias de trabalho.

3.6.1 Detalhamento das Funções

3.6.1.1 Verificar férias ou licenças

Algoritmo 3: Função verificar_ferias_e_licencas

```
Data: dias, pessoas
Result: custo
1  Função verificar_ferias_e_licencas(dias, pessoas);
2  custo = 0;
3  penalidade_ferias_licenca = 500;
4  para cada dia em dias faça
5  |   pessoa = pessoas[dia];
6  |   se pessoa em ferias_e_licencas e dia+1 em ferias_e_licencas[pessoa] então
7  |   |   custo += penalidade_ferias_licenca;
8  |   fim
9  fim
10 retorna custo;
```

A função **verificar-ferias-e-licencas** calcula o custo de penalidade por férias e licenças consecutivas em um conjunto de dias e pessoas.

3.6.1.2 Verificar dias trabalhados

Algoritmo 4: Função vezes_que_trabalhou

```
Data: dias, pessoas
Result: vezes_trabalhou_mes
Função vezes_que_trabalhou(dias, pessoas);
Inicialize vezes_trabalhou_mes como um dicionário vazio;
para cada pessoa em pessoas faça
    pessoa_idx = índice de pessoa em pessoas;
    count = 0;
    para cada dia em dias faça
        se dia é igual a pessoa_idx então
            count += 1;
        fim
    fim
    vezes_trabalhou_mes[pessoa] = count;
fim
retorna vezes_trabalhou_mes;
```

A função **vezes-que-trabalhou** conta quantas vezes cada pessoa trabalhou em um mês, analisando um conjunto de dias e pessoas.

3.6.1.3 Verificar o custo

Algoritmo 5: Função calcular_custo

```
Data: pessoas
Result: custo
Função calcular_custo(pessoas);
custo, recompensa, penalidade = 0;
intervalo_mínimo = 3;
intervalo_máximo = 4;
min_trabalhos_mes = 8;
para cada pessoa em min_trabalhos_mes faça
    se pessoa não estiver no dicionário então
        adicione-a com o valor 1;
    fim
    senão
        incremente o valor do contador para essa pessoa em 1;
    fim
fim
para cada pessoa em pessoas faça
    trabalhos = contagem de trabalhos da pessoa;
    se trabalhos < intervalo_mínimo então
        custo += penalidade;
    fim
    se 9 <= trabalhos <= 10 então
        recompensa += recompensa;
    fim
    dias_trabalhados = lista vazia;
    para cada índice, trabalho em min_trabalhos_mes faça
        se trabalho é igual à pessoa então
            Adicione o índice à lista dias_trabalhados;
        fim
    fim
    para cada índice em dias_trabalhados faça
        se índice < comprimento(dias_trabalhados) - 1 então
            intervalo = dias_trabalhados[índice + 1] - dias_trabalhados[índice];
            se intervalo < intervalo_mínimo ou intervalo > intervalo_máximo
             então
                custo += penalidade;
            fim
        fim
    fim
fim
retorna custo - recompensa;
```

4 Resultados

Neste capítulo, apresentam-se e discutem-se os resultados obtidos no desenvolvimento das funções do algoritmo, bem como as adversidades enfrentadas ao processar as restrições e garantir seu funcionamento de acordo com as necessidades da instituição.

4.0.1 Análise dos Resultados

Após a execução do Algoritmo Genético, as soluções mais promissoras são ordenadas com base em seu custo, e as quatro melhores são armazenadas na variável **top-1-solucoes**. Em seguida, um resumo das quatro melhores soluções é exibido, mostrando o custo total e a alocação de pessoas para cada dia de trabalho.

Com base nos resultados apresentados nas Tabelas (3 e 5), o número de gerações executadas pelo Algoritmo Genético foi definido como critério, onde cada geração representa uma nova população gerada pelo AG. Em cada geração, todas as soluções passam por etapas de seleção, reprodução e mutação.

O resultado do código apresenta a melhor solução encontrada pelo algoritmo genético para o problema de escalonamento de pessoas ao longo dos dias do mês. Cada solução é representada pelo custo total e pela distribuição de pessoas para cada dia.

Ao proceder com a avaliação das soluções propostas, adotamos uma métrica para mensurar seu desempenho efetivo. O custo total emergiu como um indicador crucial para a determinação da qualidade inerente a cada solução. Essa métrica foi concebida com base em múltiplos fatores, englobando penalizações para jornadas de trabalho inferiores ao mínimo estabelecido, recompensas para períodos de trabalho que se enquadram no intervalo desejado, além de penalidades para intervalos excessivamente curtos ou longos entre os dias trabalhados.

A distribuição de pessoas para cada dia é mostrada através do nome da pessoa que foi escalonada para trabalhar naquele dia. Cada dia é representado pelo número do dia no mês, seguido pelo nome da pessoa que irá trabalhar.

Ao analisar as soluções, é possível identificar as melhores combinações encontradas pelo algoritmo genético, onde as restrições e objetivos foram atendidos de forma mais satisfatória. Essas soluções representam uma alocação otimizada das pessoas ao longo dos dias do mês, considerando as restrições e critérios estabelecidos pela instituição.

É importante ressaltar que os resultados podem variar a cada execução do algoritmo genético, devido à natureza estocástica do processo. Portanto, é recomendado executar o algoritmo várias vezes e analisar a média ou a tendência dos resultados para obter uma

avaliação mais precisa.

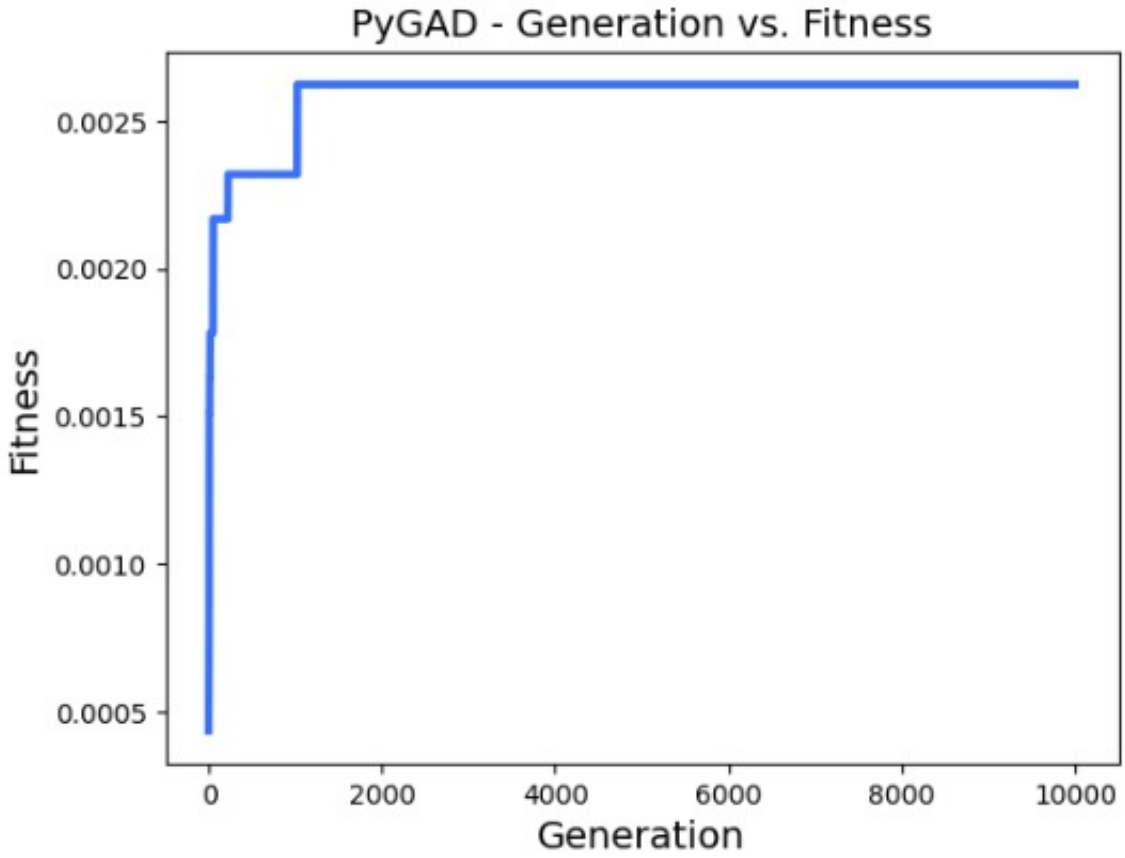
A fim de garantir a estabilidade da solução encontrada, foi definido o número de 10.000 gerações. Isso proporcionará a segurança de que a solução é estável.

Tabela 3 – Top: 1 - 10.000 Gerações

Melhor Solução: 1	
Custo: 430	
Dia	Funcionário
1	Perito01
2	Perito04
3	Perito07
4	Perito05
5	Perito04
6	Perito01
7	Perito05
8	Perito04
9	Perito07
10	Perito01
11	Perito02
12	Perito04
13	Perito01
14	Perito03
15	Perito07
16	Perito02
17	Perito04
18	Perito01
19	Perito07
20	Perito04
21	Perito02
22	Perito01
23	Perito07
24	Perito04
25	Perito02
26	Perito07
27	Perito01
28	Perito02
29	Perito07
30	Perito01

Fonte: Próprio Autor (2023)

Figura 4 – PyGAD - Generations vs Fitness não Satisfatória



Fonte: Próprio Autor (2023)

De acordo com os resultados obtidos o eixo x do gráfico representa as gerações e o eixo y representa o valor da função de *fitness*. O objetivo do algoritmo genético é maximizar a função de *fitness*, portanto, o gráfico deve mostrar uma tendência crescente ao longo das gerações.

De acordo com os padrões determinados, cada perito é obrigado a cumprir um mínimo de 8 dias de trabalho por mês. Logo, a fim de assegurar uma cobertura completa para todos os dias do mês, que compreendem 30 dias. No entanto, embora uma escala com 6 peritos tenha sido fornecida, o algoritmo gerou uma lista que também inclui estes 6 peritos, todos prontamente disponíveis para a execução da tarefa. Isso contrasta com um resultado ideal, que exigiria uma escala composta por apenas 4 peritos.

Tabela 4 – Tabela dias trabalhados por perito não satisfatório

Perito	Dias Trabalhados
Perito01	8
Perito02	5
Perito03	1
Perito04	7
Perito05	2
Perito07	7

Fonte: Próprio Autor (2023)

O algoritmo foi processado em um tempo estimado de 2 minutos e 48 segundos, demonstrando uma eficiência temporal razoável. Entretanto, o resultado gerado na Tabela (4) não cumpre integralmente os requisitos estabelecidos previamente.

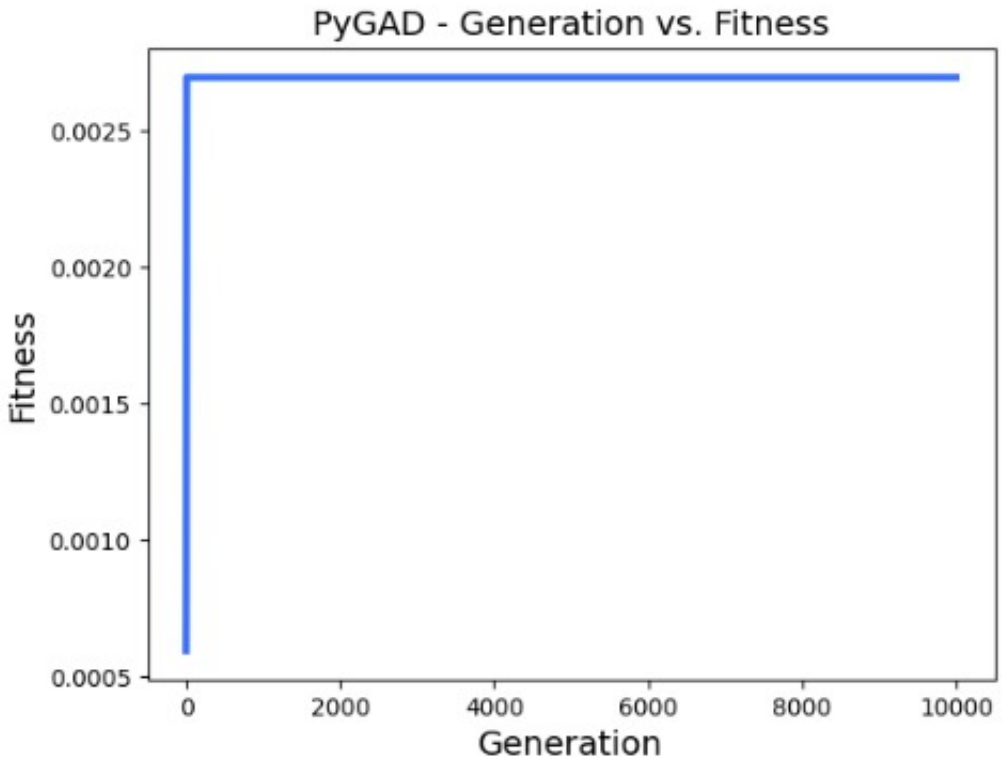
Dessa forma, diante desses contratempos, o estudo ressaltou a força e a adaptabilidade do algoritmo genético em circunstâncias desafiadoras. Apesar dos resultados não terem sido os mais encorajadores, isso reforça a importância de mantermos o esforço contínuo para explorar e aperfeiçoar a sua aplicação em uma diversidade de cenários e restrições.

Tabela 5 – Top: 1 - 10.000 Gerações

Melhor Solução: 1 Custo: 370 Dia	Funcionário
1	Perito01
2	Perito03
3	Perito01
4	Perito04
5	Perito03
6	Perito02
7	Perito01
8	Perito04
9	Perito02
10	Perito03
11	Perito02
12	Perito01
13	Perito03
14	Perito04
15	Perito02
16	Perito01
17	Perito03
18	Perito04
19	Perito02
20	Perito01
21	Perito02
22	Perito03
23	Perito03
24	Perito01
25	Perito04
26	Perito02
27	Perito03
28	Perito01
29	Perito02
30	Perito01

Fonte: Próprio Autor (2023)

Figura 5 – PyGAD - Generations vs Fitness Satisfatório



Fonte: Próprio Autor (2023)

De acordo com os critérios previamente definidos, é exigido que cada perito dedique no mínimo 8 dias de trabalho por mês. Assim, para assegurar uma cobertura ininterrupta durante todo o mês, que totaliza 30 dias, precisaríamos de, ao menos, 4 peritos à disposição.

Tabela 6 – Tabela dias trabalhados por perito satisfatório

Perito	Dias Trabalhados
Perito01	9
Perito02	9
Perito03	8
Perito04	9

Fonte: Próprio Autor (2023)

O algoritmo foi executado de maneira eficaz, consumindo apenas um tempo estimado de 1 minuto e 23 segundos, e seus resultados cumpriram satisfatoriamente com as exigências estabelecidas. Este desempenho atesta a eficiência do algoritmo e ilustra a adequação do mesmo em relação ao propósito para o qual foi criado.

Apesar de enfrentar certos obstáculos, o estudo ressaltou a solidez e a adaptabilidade do algoritmo genético, mesmo diante de circunstâncias desfavoráveis. Mesmo que os resultados obtidos não tenham sido os mais auspiciosos, isso reforça a importância de

prosseguir com a exploração e a otimização de sua aplicação em uma variedade de cenários e contextos distintos.

5 Conclusão

A conclusão do estudo apresentado revelou que o algoritmo proposto não conseguiu atingir seus objetivos como esperado. Isso se deve, principalmente, às restrições estabelecidas pela instituição, que não estava totalmente em conformidade. As regras em vigor geraram conflitos internos no algoritmo, impedindo a obtenção de uma solução ideal. Esta conclusão pode ser observada na Tabela (4), que evidenciou resultados não satisfatórios. Esses resultados foram contrastados com os resultados mais promissores encontrados na Tabela (6).

Este estudo evidenciou que a escala atual precisa de adaptações significativas para ser aplicada com sucesso em um algoritmo genético. Ainda que a solução inicialmente parecesse promissora, ela acabou não se encaixando na escala genética conforme o planejado. Este resultado levou a refletir sobre a necessidade de uma escala que atenda tanto às exigências legais quanto às demandas práticas para a aplicação eficaz de um algoritmo genético.

Contudo, a despeito dessas dificuldades, este estudo destacou a robustez e a resiliência do algoritmo genético, mesmo em condições adversas. Ainda que os resultados não tenham sido os mais promissores, o algoritmo mostrou-se um valioso aliado na busca por soluções complexas, reforçando a necessidade de se continuar explorando e otimizando seu uso em diferentes cenários e restrições.

5.1 Trabalhos Futuros

Como trabalhos futuros pretende-se:

- **Adicionar mais restrições e regras:** Considerar aspectos adicionais, como preferências individuais dos peritos e demandas específicas de trabalho, tornando o modelo mais adaptável à realidade e às necessidades da organização;
- **Implementar um sistema de interface gráfica:** Desenvolver uma interface amigável que permita aos gestores e peritos interagir com o algoritmo e ajustar parâmetros conforme necessário, facilitando o processo de construção e análise das escalas de trabalho.

Referências

BÄCK, T. et al. *Evolutionary computation 1: Basic algorithms and operators*. [S.l.]: CRC press, 2018.

BANKER, K. et al. *MongoDB in action: covers MongoDB version 3.0*. [S.l.]: Simon and Schuster, 2016.

BELTRAN, A. P. *Direito do trabalho e direitos fundamentais*. São Paulo, SP: LTr, 2002.

BIGDOLI, H. *Decision Support System-Principles and Practice*. New York, NY: West Publisbing Company, 1989.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308, 2003.

BORGES, L. E. *Python para desenvolvedores: aborda Python 3.3*. [S.l.]: Novatec Editora, 2014.

CASSAR, V. B. *Direito do trabalho*. [S.l.]: Impetus Niterói, 2011.

CERQUEIRA, I. d. C. C. *Trabalho por turnos: Implicações para os Colaboradores e Estratégias de Intervenção*. 2021.

CRUZ, F. *Python: Escreva seus primeiros programas*. [S.l.]: Editora Casa do Código, 2015.

DELGADO, M. G. *Curso de direito do trabalho*. [S.l.]: LTr, 2019.

DIANA, M. d.; GEROSA, M. A. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. *Workshop de Teses e Dissertações de Bancos de Dados do Simpósio Brasileiro de Bancos de Dados*, p. 8, 2010.

DORIGO, M.; STÜTZLE, T. Ant colony optimization algorithms for the traveling salesman problem. MIT Press, 2004.

DORNELAS, J. S. *Impactos da adoção de sistemas de apoio à decisão para grupos em um processo decisório público participativo: o caso do orçamento de Porto Alegre*. Tese (Doutorado) — UFRGS - Universidade Federal do Rio Grande do Sul, 2000.

ENDREI, M. et al. *Patterns: service-oriented architecture and web services*. New York, NY: IBM Corporation, International Technical Support Organization, 2004.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, p. 109–133, 1995.

FERNANDES, F. T.; FILHO, A. D. P. C. Perspectivas do uso de mineração de dados e aprendizado de máquina em saúde e segurança no trabalho. *Revista Brasileira de Saúde Ocupacional*, SciELO Brasil, v. 44, 2019.

FERREIRA, W. O.; KNOP, I. de O. Estruturação de aplicações distribuídas com a arquitetura rest. *Caderno de Estudos em Sistemas de Informação*, v. 3, n. 1, 2017.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, 2000.

FONSECA, J. J. S. da. *Apostila de metodologia da pesquisa científica*. [S.l.]: João José Saraiva da Fonseca, 2002.

GIL, A. C. et al. *Como elaborar projetos de pesquisa*. [S.l.]: Atlas São Paulo, 2002. v. 4. 42 p.

GLOVER, F.; LAGUNA, M. Tabu search principles. In: *Tabu Search*. [S.l.]: Springer, 1997. p. 125–151.

HOBSBAWM, E. *A era das revoluções: 1789-1848*. [S.l.]: Editora Paz e Terra, 2015.

JOHN, H. Holland. genetic algorithms. *Scientific american*, v. 267, n. 1, p. 44–50, 1992.

KANIT, R. et al. Scheduling of construction projects under resource-constrained conditions with a specifically developed software using genetic algorithms. *Tehnički vjesnik*, Strojarski fakultet u Slavonskom Brodu; Fakultet elektrotehnike, računarstva ..., v. 28, n. 4, p. 1362–1370, 2021.

KIRKPATRICK, S. et al. Optimization by simulated annealing. *science*, American association for the advancement of science, v. 220, n. 4598, p. 671–680, 1983.

KUMAR, K. et al. A performance comparison of document oriented nosql databases. In: IEEE. *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*. [S.l.], 2017. p. 1–6.

KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a internet: uma nova abordagem top-down*. [S.l.]: Pearson, 2013.

LANTHALER, M.; GÜTL, C. On using json-ld to create evolvable restful services. In: *Proceedings of the third international workshop on RESTful design*. [S.l.: s.n.], 2012. p. 25–32.

LAPOLLI, P. C. et al. *Implantação de sistemas de informações gerenciais em ambientes educacionais*. Dissertação (Mestrado) — UFSC - Universidade Federal de Santa Catarina, 2003.

LAUDON, K. et al. *Sistemas de informação gerenciais*. [S.l.]: Pearson Prentice Hall, 2010.

LOURENÇO, H. R. et al. Iterated local search. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 320–353.

MANZANO, J. A. N. *Introdução à linguagem Python*. [S.l.]: Novatec Editora, 2018.

MARTINEZ, L. *Curso de direito do trabalho*. [S.l.]: Saraiva Educação SA, 2020.

MARTINS, S. P. *Direito do trabalho*. [S.l.]: Atlas, 2010.

MARTINS, S. P. *Instituições de direito público e privado*. São Paulo, SP: Atlas, 2015.

MATOS, M. I. C. d. *Arquitetura de serviços web rest para domótica habitacional*. Dissertação (Mestrado) — Universidade de Aveiro, 2013.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, v. 239, p. 2, 2014.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, v. 1, n. 1, p. 32, 2003.

MONIRUZZAMAN, A.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *International Journal of Database Theory and Application*, v. 6, n. 4, 2013.

MORAIS, N. S. d. *Proposta de modelo de migração de sistemas de ambiente tradicional para nuvem privada para o Polo de Tecnologia da Informação do Exército brasileiro*. Dissertação (Mestrado) — Universidade de Brasília, 2015.

MOUAT, A. *Using docker: developing and deploying software with containers*. [S.l.]: O'Reilly Media, Inc., 2015.

NASCIMENTO, A. M. *Curso de direito do trabalho*. [S.l.]: Saraiva Educação SA, 2017.

NETO, F. F. J.; CAVALCANTE, J. d. Q. P. *Direito processual do trabalho*. [S.l.]: Brasiliense Coleções, 2007.

OLIVEIRA, D. d. P. R. d. *Sistemas de informações gerenciais*. [S.l.]: Atlas, 2018.

OSMAN, I. H.; LAPORTE, G. *Metaheuristics: A bibliography*. [S.l.]: Springer, 1996.

PALACIO, V. F. *Desarrollo de gateway de seguridad en Python con FastAPI*. Monografia (Graduação) — Universidad de Cantabria, 2022.

PAZ, L. M. C. ; ODELIUS, C. C. Escala de competências gerenciais em um contexto de gestão pública: desenvolvimento e evidências de validação. *Organizações Sociedade*, v. 28, p. 370–397, 2021.

REZENDE, D. A.; ABREU, A. F. d. *Tecnologia da informação aplicada a sistemas de informação empresariais*. São Paulo, SP: Atlas, 2006.

RICHARDSON, L. et al. *RESTful Web APIs: Services for a Changing World*. [S.l.]: O'Reilly Media, Inc., 2013.

SCHELLER, T.; KÜHN, E. Automated measurement of api usability: The api concepts framework. *Information and Software Technology*, Elsevier, v. 61, p. 145–162, 2015.

SHIN, Y.-E.; HAN, W.-J. Online finger circumference measurement system using semantic segmentation with transfer learning. *Journal of the Korea Information Technology Association*, v. 19, n. 12, p. 105–113, 2021.

SOMMERVILLE, I. *Software Engineering*. [S.l.]: Pearson Education, 2011.

SONG, J.; KOOK, J. Mapping server collaboration architecture design with openvslam for mobile devices. *Applied Sciences*, MDPI, v. 12, n. 7, p. 3653, 2022.

STAIR, R.; REYNOLDS, G. *Principles of information systems*. [S.l.]: Cengage Learning, 2020.

STAL, M. Using architectural patterns and blueprints for service-oriented architecture. *IEEE software*, IEEE, v. 23, n. 2, p. 54–61, 2006.

TURHAN, A. M.; BILGEN, B. A hybrid fix-and-optimize and simulated annealing approaches for nurse rostering problem. *Computers & Industrial Engineering*, Elsevier, v. 145, p. 106531, 2020.

ZHANG, Q.; LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, v. 11, n. 6, p. 712–731, 2007.