Algoritmo genético distribuído para problema de Timetabling

Distributed genetic algorithm for the Timetabling problem

DOI:10.34117/bjdv8n5-245

Recebimento dos originais: 21/03/2022
Aceitação para publicação: 29/04/2022

Iago Rosa Bianquini
Bacharel
Instituição: Instituto Federal de Santa Catarina
Endereço: Rua Heitor Vila Lobos, 225, Lages, SC, Brasil
E-mail: iagobianquini@gmail.com

Osmar José Hofman da Silva
Bacharel
Instituição: Instituto Federal de Santa Catarina
Endereço: Rua Heitor Vila Lobos, 225, Lages, SC - Brasil
E-mail: hofman.osmar@gmail.com

Wilson Castello Branco Neto
Doutor
Instituição: Instituto Federal de Santa Catarina
Endereço: Rua Heitor Vila Lobos, 225, Lages, SC - Brasil
E-mail: wilson.castello@ifsc.edu.br

RESUMO
Este artigo apresenta um Algoritmo Genético Distribuído para resolução do problema de Timetabling do Instituto Federal de Santa Catarina Câmpus Lages, que visa gerar um quadro de horários sem violar as restrições impostas pela instituição. Para tal, foi elaborado um algoritmo que pode ser executado em ambiente centralizado ou distribuído, que conta com um pré-processamento que é responsável por diminuir o espaço de busca, além de um algoritmo de árvore de busca em profundidade limitada para o ambiente distribuído, com o objetivo de resolver os conflitos restantes. Foi constatado que o algoritmo alcançou, em ambos os ambientes, a solução perfeita e, além disso, a solução em ambiente distribuído chegou nos resultados, em média, 26 segundos, enquanto a centralizada levou 70 segundos.

Palavras-chave: timetabling, algoritmo genético, inteligência artificial, sistema distribuído, busca em profundidade.

ABSTRACT
This paper presents a Distributed Genetic Algorithm for solving the Timetabling problem of the Federal Institute of Santa Catarina Campus Lages, which aims to build a timetable without violating the restrictions imposed by the institution. To this end, an algorithm that can be run in a centralized or distributed environment was developed. Such algorithm has a pre-processing step which is responsible for reducing the search space, in addition to a depth-first search for the distributed environment, aiming to solve the remaining conflicts. It was verified that the algorithm reached, in both environments, the perfect solution and, in addition, the solution in the distributed environment reached the results in 26 seconds

on average, while the centralized solution took 70 seconds.

Keywords: timetabling, genetic algorithm, artificial intelligence, distributed system, depth-first search.

1 INTRODUÇÃO

A cada início de semestre, em uma instituição de ensino, surge um problema recorrente: a elaboração dos quadros de horários de professores e estudantes. Esse problema é conhecido como *timetabling*, que segundo [1], consiste no agendamento de uma sequência de disciplinas e espaços físicos, como salas de aula e laboratórios, entre professores e estudantes num período de tempo previamente estabelecido, satisfazendo um conjunto de restrições. De acordo com [2], a elaboração desses horários causa um grande desgaste nos profissionais responsáveis por esta atividade, o que pode levar a erros causados por uma grande quantidade de variáveis, restrições e necessidades relacionadas aos recursos físicos e humanos. Essa complexidade é causada pelo fato de que não existe um algoritmo capaz de encontrar uma solução ótima em tempo polinomial para o problema de *timetabling*, o qual é categorizado como um problema NP-completo [3].

Dentro desse contexto, as técnicas mais utilizadas em problemas de *timetabling*, são metaheurísticas, tais como: *Greedy Randomized Adaptive Search Procedure* (GRASP), *Artificial Bee Colony* (ABC), busca tabu e Algoritmos Genéticos (AG). Esse último possui resultados expressivos em problemas de *timetabling*, sendo abordado neste trabalho.

Algoritmos genéticos é uma subárea da computação evolucionária, que segundo [4] é classificada como uma estratégia para resolver problemas genéricos, capaz de atuar em espaços não-lineares e não-estacionários. Apesar de não garantir o melhor resultado, geralmente chega-se a uma boa aproximação para a solução ótima em um tempo não exponencial. A partir dessa afirmação, é possível identificar que para problemas NP-Completo, como o *timetabling*, AG são capazes de encontrar boas soluções em um tempo computacional viável.

De acordo com [5]: “Nos algoritmos genéticos, populações de indivíduos são criados e submetidos aos operadores genéticos de seleção, *crossover* e mutação. Estes operadores utilizam uma caracterização da qualidade de cada indivíduo como solução do problema em questão chamada de avaliação deste indivíduo e vão gerar um processo de evolução natural destes indivíduos, que eventualmente gerará um indivíduo que

caracterizará uma boa solução (talvez até a melhor possível) para o nosso problema."

No Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina (IFSC), Câmpus Lages, o problema de *timetabling* também é uma realidade, pois a instituição utiliza um sistema terceirizado que não é capaz de gerar um horário completo, sem violar muitas restrições. Por consequência, é necessário a intervenção de servidores técnico-administrativos e coordenadores de curso para realizar ajustes nos horários gerados.

Este artigo apresenta um AG que resolve o problema *timetabling* no contexto do IFSC Câmpus Lages. Ele apresenta a modelagem do AG desenvolvido, sua implementação de diferentes maneiras, como *threads* e objetos distribuídos e o resultado da avaliação do desempenho das diferentes implementações do AG.

Visando melhorar o desempenho do processamento do AG foi implementado um módulo que tem o papel de processar os dados provenientes do IFSC, dividindo os cursos em conjuntos menores para serem posteriormente processados no AG. Após a conclusão desta etapa, inicia-se de fato a execução do AG, que pode acontecer de duas formas, a primeira é realizando o processamento de maneira centralizada, assim o AG é executado em um único computador, gerando a solução final. A segunda forma de execução é o processamento distribuído, que se utiliza dos conjuntos gerados no pré-processamento para enviá-los a diferentes computadores e assim tornar o processo mais rápido. Entretanto, ao final da execução de cada computador obtém-se uma solução para cada conjunto processado, o que pode gerar conflitos quando estas soluções são agrupadas em uma só. Devido a isso, foi implementada uma etapa posterior a execução do AG que utiliza um algoritmo de busca em profundidade para solucionar os possíveis conflitos gerados.

Este trabalho está dividido em cinco seções. A primeira seção contém esta introdução ao tema e aos objetivos do trabalho. Na seção 2 apresenta-se o referencial teórico sobre *timetabling* e alguns trabalhos similares. A seção 3 descreve a metodologia utilizada, incluindo a apresentação da modelagem e implementação do AG. A seção 4 apresenta os resultados encontrados. Por fim, na seção 5 são descritas as considerações finais deste trabalho.

2 TRABALHOS RELACIONADOS

Na visão de [6], *timetabling* pode ser descrito como a alocação de recursos, sujeito a restrições, de certos objetos alocados no espaço-tempo, de maneira a satisfazer o

máximo possível um conjunto de objetivos. Problemas de *timetabling* estão presentes em diferentes contextos, incluindo escalas de plantão médico, eventos esportivos, transporte e instituições educacionais [7]. Ainda sobre a abordagem em escolas e universidades, este problema pode ser dividido em três classes principais [1]:

• *Timetabling* de Escola: também conhecido como modelo turma/professor, no qual o agendamento é feito para todas as turmas de uma escola, evitando que o professor tenha duas turmas ao mesmo tempo. Este modelo é o mais utilizado em escolas de ensino fundamental e médio do Brasil;

• *Timetabling* de Curso: também conhecido como *timetabling* de cursos de universidade, que é o agendamento para todas as disciplinas de um conjunto de cursos da universidade, com o intuito de minimizar a sobreposição de diferentes matérias que tenham alunos em comum;

• *Timetabling* de Exame: agendamento de testes de um conjunto de cursos, com o objetivo de evitar que os alunos tenham duas provas em um mesmo horário, e minimizar o acúmulo de testes em um curto período de tempo.

De maneira geral, *timetabling* educacional possui dois tipos de restrições: *hard* e *soft*. Restrições *hard* devem necessariamente ser atendidas para que a solução seja viável. Algumas restrições *hard* comuns são: um professor não pode lecionar duas disciplinas em um mesmo horário, uma sala de aula não pode ser utilizada por duas turmas ao mesmo tempo e uma turma não pode ter duas disciplinas no mesmo horário. As restrições *soft* são definidas como desejáveis, porém não essenciais, como, por exemplo, um professor lecionar uma disciplina em um horário de sua preferência, ou não haver somente uma matéria durante um único turno. Os dois tipos de restrições são definidas pela instituição de ensino, levando em consideração suas necessidades.

Segundo [8], existe na literatura um número significativo de técnicas e abordagens utilizadas para a resolução de *timetabling*. Estes autores pontuam, ainda, que problemas pequenos podem ser resolvidos através de algoritmos exatos. Porém, conforme seu tamanho aumenta, a solução ótima fica mais distante de ser encontrada, devido à sua complexidade computacional. A dificuldade de resolução deste tipo de problema também está associada à discrepância de restrições, recursos e necessidades entre as instituições. Em função disto, meta-heurísticas vêm sendo aplicadas com o objetivo de encontrar uma solução satisfatória em um tempo aceitável.

O trabalho de [9] apresenta um algoritmo híbrido baseado em GRASP, em conjunto com as heurísticas de baixo nível de grafos, *Saturation Degree* (SD) e *Largest*

Weighted Degree (LDW), para um problema de *timetabling* de exame adaptativo. A heurística SD ordena os exames de acordo com os espaços de tempo, priorizando os mais restritivos, e a heurística LWD ordena os exames que possuem mais alunos com possíveis choques com outros exames. De maneira geral, o SD gera uma solução viável, porém não apresenta um bom desempenho nos estágios iniciais de construção, o que levou a utilização de LWD até um certo ponto dessa etapa. O algoritmo híbrido apresentou os melhores resultados, alcançando ao menos uma solução factível em 100% dos conjuntos de testes. Além disso, obteve um aproveitamento superior em relação aos algoritmos individuais, referente ao percentual de testes em que foi capaz de encontrar a solução em cada um dos conjuntos.

Outro tipo de técnica utilizada é a ABC. Essa técnica é baseada na estrutura de uma colônia de abelhas e tem como objetivo encontrar a fonte de alimento com maior qualidade, sendo essa fonte a provável solução do problema. O trabalho de [10] apresenta um algoritmo híbrido de ABC e um otimizador de subida de encosta (HCO), para um problema de *timetabling* de curso. O algoritmo é baseado na estrutura do ABC, a inicialização das fontes de alimento é feita através de LWD e *backtracking algorithm* e o processo de busca local feito pelas empregadas foi otimizado através do HCO. Os testes foram realizados no conjunto de instâncias de [11], com problemas de tamanho pequeno, médio e grande. Em comparação com vários outros algoritmos, apresentou os melhores resultados gerais, principalmente em problemas de média e grande escala.

A técnica de busca tabu também é uma alternativa interessante. Segundo [12], este é um algoritmo de busca local que restringe a vizinhança factível através dos vizinhos que são excluídos. Para isto, existe uma estrutura de dados chamada lista tabu, que armazena os estados inalcançáveis, evitando que a busca fique presa em máximos locais. Caso todos os vizinhos estejam na lista tabu, é permitido um movimento que piore o valor da função objetivo.

Além dessas técnicas, AG podem ser considerados para a busca de uma solução viável. De acordo com [13], a razão de AG ser um sucesso e ter uma ampla utilização para problema de escalonamento deve-se à combinação de poder e flexibilidade. O poder deriva da prova empírica que algoritmos evolucionários encontram de maneira eficiente a solução ótima em espaços de pesquisa grandes e complexos, característica típica dos problemas do mundo real. A flexibilidade de AG tem múltiplas facetas, já que podem efetivamente tratar problemas que muitos algoritmos de otimização tradicionais não conseguem, como espaços discretos e não-lineares.

O trabalho de [14] é direcionado ao *timetabling* escolar. Ele utiliza um algoritmo híbrido baseado em AG, combinado com busca tabu, como uma solução para o hdtt4, que é um conjunto de dados para *timetabling* presente em uma coleção de conjunto de dados de testes para pesquisa operacional, chamada *OR Library*. O funcionamento do algoritmo é baseado na estrutura fundamental do AG, a inicialização da população é feita através de busca tabu e a mutação e *crossover* foram modificados a fim de satisfazer os requisitos de carga de trabalho. Os autores concluíram que a solução produzida satisfaz todas as restrições para o problema, gerando o melhor resultado em 7 segundos, enquanto um AG simples alcançou esse resultado em 12 segundos. Além disso, técnicas baseadas em redes neurais, têmpera simulada, busca tabu e gulosa, obtiveram seus resultados em tempos ainda maiores.

O trabalho de [15] apresenta um AG recursivo para resolver o problema de *timetabling* de curso da Universidade Federal do Ceará. Além da recursividade, o AG tem como objetivo ser escalável e parametrizado a fim de encontrar uma solução factível para uma maior gama de cursos. O AG recebe como parâmetros a lista de cursos, a indisponibilidade dos agentes - que incluem professores e disciplinas, e os parâmetros do AG, como as taxas de mutação e cruzamento. Em cada execução do AG, busca-se por uma solução factível para um curso na lista, e uma vez que a solução é encontrada, essa lista é atualizada criando uma nova designação para o agente. Essa designação é utilizada para atualizar a indisponibilidade dos agentes para a próxima execução. Esse processo é repetido até ser encontrada a solução geral do problema. Os autores ressaltaram que houve dificuldade em comparar os resultados com outros presentes na literatura, devido à complexidade das divergências do ambiente. No entanto, os trabalhos de [16] e [17] apresentam diferenças mínimas em classes, agentes e restrições, o que possibilita a comparação. Os autores concluíram que na maioria das tentativas, o modelo encontrou a solução factível e também gerou mais de uma solução possível.

Os resultados indicaram que o algoritmo de [15] levou menos tempo que os outros, mesmo assim, vale ressaltar a divergência da modelagem e parâmetros entre os trabalhos. Os autores também reforçam que quanto maior a população, mais custosa é a execução, assim como o número de gerações em relação ao tempo de execução.

Um trabalho que tem como objetivo implementar novas abordagens para o uso de AG na nuvem, utilizando *softwares* de conteinerização e também *web services*, é descrito por [18]. A aplicação adota a arquitetura mestre/escravo, sendo o nodo mestre responsável pela execução do AG, exceto a função de avaliação que é delegada aos nodos escravos.

A comunicação entre esses nodos é feita através do *RabbitMQ*, um software *open source* utilizado para intermediar mensagens. Cada escravo atua sob o *CoreOS*, que é um sistema operacional focado em ambientes distribuídos.

Quanto aos resultados, segundo os autores, foi possível acelerar a execução do AG, com um total de 128 escravos. Notou-se, também, uma grande dependência entre a carga computacional e o custo de comunicação, ou seja, quanto maior a quantidade de informação associada ao cromossomo, maior é o custo da troca de mensagens entre os nodos. O desempenho e o tempo de configuração posicionam a abordagem em nuvem positivamente entre outras tecnologias empregadas à paralelização de AG disponíveis na literatura.

Os trabalhos citados nesta seção possibilitaram a compreensão das definições gerais das técnicas, conceitos, diferentes formas de resolução e tipos de problemas de *timetabling*. Embora não seja possível uma comparação quantitativa dos seus resultados, em função das divergências dos ambientes citada por [15], eles contribuíram para o entendimento sobre como a otimização do processamento geral pode ser feita a partir de pontos específicos e como a divisão do processo em partes menores pode impactar em um ganho de *performance*.

É visível dentre os trabalhos apresentados que a combinação de diferentes técnicas para buscas locais apresentam resultados mais satisfatórios do que quando comparados a técnicas puras, como mostra [9], [10] e outros. A priorização de alocação dos elementos mais restritivos primeiro, como feita por [9], e a resolução do problema em etapas, como demonstrada por [15], foram elementos importantes para a concepção da solução proposta neste artigo. Além disso, os trabalhos citados validam a utilização de AG como uma técnica viável para o problema abordado nesse artigo, como demonstra [13] e [14], apresentando flexibilidade na escolha da arquitetura de implementação e a possibilidade de utilização de agentes distribuídos, como destaca [18].

3 MATERIAIS E MÉTODOS

Este trabalho é de natureza aplicada, pois objetiva o desenvolvimento de conhecimento para uma aplicação prática, que visa a resolução de um problema de *timetabling* escolar. Para a abordagem do problema, a análise das informações é quantitativa, a partir de recursos e técnicas estatísticas. Do ponto de vista dos objetivos, apresenta um viés exploratório, aprofundando-se no problema para apresentar hipóteses sobre o mesmo, através do estudo de outros exemplos. Livros, artigos de periódicos e

materiais disponíveis da Internet, fundamentaram os procedimentos técnicos, o que caracteriza uma pesquisa bibliográfica. Além disto, adotou-se uma abordagem experimental, a fim de definir a estratégia com desempenho mais significativo em relação ao AG, e um estudo de caso com os dados disponibilizados pelo IFSC foi realizado.

O levantamento sobre os temas AG, *timetabling* e trabalhos relacionados foi desenvolvido por meio da pesquisa em sites acadêmicos como *google scholar*, *scielo* e *sciencedirect*.

Após essa etapa, a modelagem do AG foi realizada tomando como base o estudo de caso dos trabalhos de [16] e [19], a qual foi adaptada para as restrições do IFSC. A principal mudança é a flexibilização do número de aulas que determinada disciplina pode ter, possibilitando que as disciplinas tenham duas ou quatro aulas em um dia.

Na sequência, para a realização dos testes de desempenho do sistema, foi criado um ambiente para simular alguns cenários, como programação centralizada, com e sem a utilização de *threads*, e programação distribuída. Posteriormente, foi implementado o AG, de acordo com o método que gerou os melhores resultados nos testes. Para esse desenvolvimento foi utilizada a linguagem de programação *Java*, aliada ao *framework Spring* para o *backend*. Uma aplicação *Web* foi desenvolvida em relação ao *frontend* e para melhor integração de ambos, foi utilizado o padrão de projeto *Model-View-Controller* (MVC).

Para testar os resultados obtidos com o algoritmo desenvolvido na etapa anterior, dados de semestres anteriores do IFSC foram utilizados. Um destes testes, avaliou a influência de parâmetros como taxa de mutação, cruzamento, elitismo e tamanho da população nos resultados do AG, a partir da análise de variância e de testes de comparação de médias. Já o outro teste, calculou a qualidade dos resultados e o tempo médio de execução, por meio do valor da média e desvio padrão, uma vez que AGs usam valores aleatórios em sua execução e podem gerar resultados muito diferentes para os mesmos dados de entrada, em diferentes execuções.

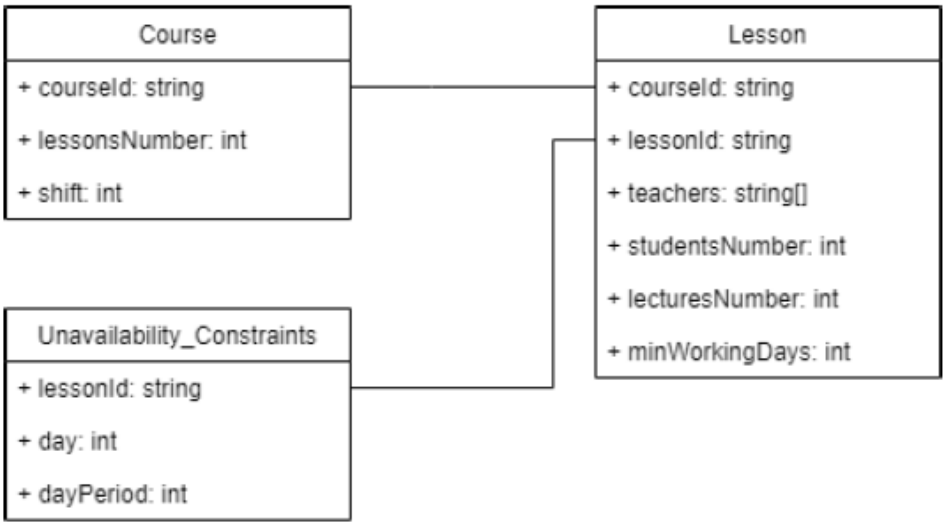
As subseções seguintes tratam da leitura dos dados, o pré-processamento dos dados, a modelagem e implementação do AG.

3.1 LEITURA DOS DADOS

O sistema utiliza um conjunto de dados gerado por um sistema do IFSC Câmpus Lages que está disponível através de um arquivo no padrão XML. A partir desse conjunto é gerada uma estrutura equivalente mais concisa, baseada na estrutura dos dados usados

na Competição Internacional de *Timetabling* (ITC) de 2007[1](Figura 1), com o intuito de possibilitar a comparação dos resultados em trabalhos futuros.

Figura 1: Modelo de dados adaptado



3.2 MODELAGEM DO ALGORITMO GENÉTICO

O objetivo do AG é gerar horários para os cursos que não violem as seguintes restrições:

- Turmas com duas aulas no mesmo período;
- Turmas com aulas de outras turmas;
- Disciplinas com aulas excedentes ou faltantes;
- Conflito de horários entre professores;
- Indisponibilidade dos professores.

As três primeiras restrições foram resolvidas durante a modelagem do AG e a implementação de seus operadores, como é explicado no decorrer desta seção. Desta forma, a função de avaliação não precisa verificar a existência das mesmas, o que representa um ganho de tempo na execução do AG.

As únicas duas restrições que podem ocorrer com a modelagem e operadores adotados e que precisam ser verificadas pela função de avaliação são:

- Conflito de horários: Essa restrição descreve os casos em que um professor

[1] A Competição Internacional de *Timetabling* (ITC) de 2007 foi um evento organizado por um grupo de pesquisa da Universidade do *Queen's* em conjunto de outras universidades, composta por três trilhas com diferentes problemas de *timetabling* educacional. O objetivo do ITC era reunir pessoas de diferentes áreas, a fim de encontrar novas abordagens e também melhorar a qualidade das pesquisas na área de *timetabling*. Os dados presentes nessa competição ainda são utilizados como referência para comparação de soluções de *timetabling* até hoje.

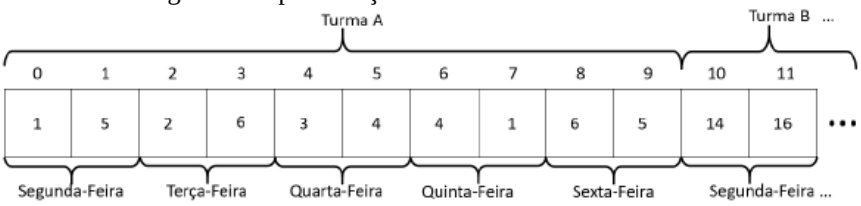
ministra duas disciplinas no mesmo período letivo e é classificada como uma restrição *hard*, ou seja, o horário torna-se inviável caso ocorra;

• Indisponibilidade dos professores: representa os cenários em que uma aula é atribuída a um professor num período que não pode lecionar. A classificação dessa restrição é *soft*, já que caso ocorra, o horário não será inviável, porém é desejável reduzir o seu número de ocorrências;

O cromossomo é a estrutura de dados que tem a capacidade de representar a solução final do problema. Ela é representada por um vetor, cujo tamanho é dez vezes o número de turmas porque o IFSC organiza seus cursos em 20 aulas semanais (4 aulas por turno a cada dia) e cada gene do cromossomo representa duas aulas consecutivas, pois esta é a situação mais comum na instituição. Desta forma, o AG precisa alocar duas aulas por turno para cada turma, o que reduz significativamente o espaço de busca, se comparado a uma solução modelada com quatro aulas individuais.

Como mostra a Figura 2, cada turma é representada por dez posições do cromossomo, e tais posições simbolizam o período e o dia da semana de cada uma das disciplinas. O valor armazenado em cada posição representa o identificador da aula alocada naquele dia e horário. Por exemplo, os valores das posições 0 e 1 representam, respectivamente, que as disciplinas 1 e 5 serão ministradas neste horário. Uma estrutura complementar armazena, além deste identificador, o nome e carga horária semanal das disciplinas.

Figura 2: Representação da estrutura do cromossomo



	Segunda-Feira	Terça-Feira	Quarta-Feira	Quinta-Feira	Sexta-Feira
Período 1	1	2	3	4	6
Período 2	1	2	3	4	6
Período 3	5	6	4	1	5
Período 4	5	6	4	1	5

O processamento do AG segue o modelo tradicional apresentado na literatura e

todos os parâmetros citados nas próximas etapas foram ajustados durante a fase de testes. As etapas do processamento do AG são: inicialização, avaliação, seleção (elitismo e roleta simples), cruzamento e mutação.

A etapa de inicialização tem como objetivo preencher as posições do cromossomo com valores aleatórios de acordo com as disciplinas de cada curso. No intervalo de posições pertencentes à turma, são inseridos os identificadores de suas disciplinas apenas e, após isso, são feitas trocas entre os identificadores em posições aleatórias desse intervalo. Tal processo é feito a fim de garantir que não existam disciplinas em excesso ou em falta em uma turma, assim como evitar que disciplinas sejam atribuídas a outras turmas.

A avaliação quantifica a qualidade da solução de cada cromossomo da população. Como já citado, há duas restrições presentes nesse processo, a primeira refere-se aos conflitos de horários de professores, ou seja, períodos em que um mesmo professor leciona para mais de uma turma (*hard constraint*). A segunda restrição refere-se a indisponibilidade dos professores, quando é verificado se os professores lecionam em dias e horários que não gostariam (*soft constraint*). Cada cromossomo é inicializado com uma pontuação pré-definida, e tal pontuação é decrementada a cada restrição violada. Conforme mencionado anteriormente, os problemas de turmas com duas disciplinas no mesmo horário ou com matérias faltantes não precisam ser verificados, pois, a modelagem do cromossomo e inicialização evitam essas situações. Da mesma forma, o problema de turmas com matérias de outras turmas não precisa ser verificado pela função de avaliação, devido à forma como os operadores de inicialização, cruzamento e mutação foram modelados, reduzindo o tempo de processamento da mesma.

Na seleção é feita a escolha dos cromossomos que são enviados para a próxima geração do AG. Essa escolha é feita por meio de dois sub-processos, levando em conta um parâmetro que indica a proporção da população que deve ser selecionada em cada um deles. Os sub-processos citados são:

• Elitismo: No elitismo os cromossomos melhores avaliados são selecionados de forma determinística e enviados diretamente para a etapa de mutação e, posteriormente, para a nova geração, sem passar pela etapa de cruzamento.

• Roleta: A roleta seleciona aleatoriamente pares de pais (cromossomos utilizados para gerar novos cromossomos), que são escolhidos a partir de valores gerados aleatoriamente, para a próxima etapa. A probabilidade de um cromossomo ser selecionado é diretamente proporcional a sua qualidade.

Na etapa de cruzamento, os pares de cromossomos gerados na roleta são usados para gerar novos cromossomos a fim de criar uma nova geração. Para o cruzamento foi utilizado o algoritmo descrito por [20], denominado *Recombinação Ordenada*. Neste algoritmo, a construção dos descendentes é feita a partir de uma subsequência de valores de um dos genitores, preservando também a ordem dos valores do outro genitor. A utilização desta técnica para realizar o cruzamento, garante que a quantidade de aulas de cada turma seja mantida ao longo das gerações, evitando que isto precise ser verificado pela função de avaliação, o que não seria possível com a utilização do cruzamento com pontos de quebra ou máscara binária.

Na mutação é realizada uma alteração de dois genes de forma aleatória a partir de um parâmetro, que representa a porcentagem de chance dessa etapa ocorrer. Isso é feito através da troca dos valores desses genes em posições aleatórias do cromossomo dentro de uma mesma turma, da mesma forma como na randomização dos elementos da etapa de inicialização, porém essa troca é feita somente uma vez.

Ao final desse processamento, cria-se uma nova geração a partir dos cromossomos que foram selecionados, cruzados e modificados e reinicia-se o processo até que se encontre uma solução satisfatória ou atinja um número limite de iterações.

3.3 IMPLEMENTAÇÃO

O Algoritmo Genético desenvolvido pode ser utilizado para a elaboração dos horários de um único curso ou de diversos cursos simultaneamente. A segunda opção é mais custosa em termos de tempo de processamento, pois o espaço de busca cresce de forma exponencial com o aumento do número de cursos. No entanto, uma vez que um resultado é encontrado, ele representa a solução final do problema. A elaboração dos horários de cursos individuais é mais simples e rápida devido ao espaço de busca reduzido e possibilita o processamento em ambiente distribuído. Porém, esse tipo de solução pode gerar conflitos nos horários de aula de professores que ministram aulas em mais de um curso. Neste artigo, uma solução intermediária foi elaborada visando aproveitar as melhores características dessas duas abordagens, através da criação de conjuntos que agrupam os cursos com muitos professores em comum. Esta abordagem apresenta um espaço de busca menor que o existente quando se tenta elaborar os horários de todos os cursos juntos, e minimiza o número de conflitos entre os professores que ministram aulas em diversos cursos.

A Figura 3 apresenta as etapas necessárias para elaborar os horários, independente

da abordagem escolhida e da forma de execução do AG (centralizada e distribuída). A execução deste algoritmo é iniciada pela leitura de cada um dos dados dos arquivos providos como entrada, como apresentado nas linhas 1, 2 e 3. Na linha 4, os dados do IFSC são convertidos para o modelo adaptado, que são efetivamente utilizados no decorrer do processamento do AG. A partir desses dados, na linha 5, é feito o pré-processamento como descrito na subseção *1) Pré-processamento*, e como resultado são obtidos os conjuntos que o AG processa de forma individual.

Logo após, os conjuntos são atribuídos aos computadores disponíveis de forma a distribuir a quantidade de conjuntos por máquina, que é dada pela divisão simples do número de conjuntos pela quantidade de computadores, como demonstrado na linha 8.

Figura 3: Algoritmo de processamento distribuído do AG

Algoritmo 1: Algorítmo de processamento distribuído do AG

Entrada: Arquivo de configuração do AG, XML com dados do IFSC, Arquivo com IPs dos computadores

início

```
1   configuracoesAG ← leConfigs(ArquivoConfigAg);
2   dadosIFSC ← leDadosIFSC(XmlIfsc);
3   listaIPsComputadores[] ← leArquivoIPs(IPsComp);
4   modeloAdaptado ←
      converteParaModeloAdaptado(dadosIFSC);
5   conjuntos[] ←
      preProcessamento(modeloAdaptado);
6   cromossomos[conjuntos.tamanho()];
7   para IPAtual ← listaIPsComputadores ate
      ultimoIP faça
8       conjuntosComputador ←
          obterConjuntos(conjuntos, IPAtual);
9       MelhoresCromossomos ←
          processamentoDoAG(conjuntosComputador,
          modeloAdaptado, configuracoesAG);
10      cromossomos.addCromo(MelhoresCromossomos);
    fim
11  retorna cromossomos;
fim
```

Saída: Lista com os melhores cromossomos gerados pelo AG

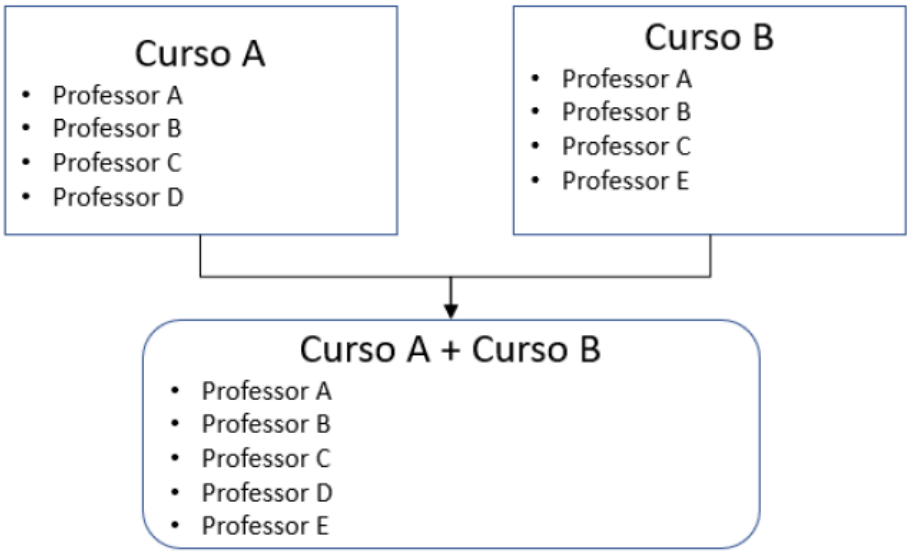
Na linha 9, é enviada a requisição de processamento do AG, que retorna como resposta os melhores cromossomos gerados a partir dos conjuntos enviados. A saída do algoritmo é a lista dos melhores cromossomos retornados por cada um dos processamentos ao final do laço de repetição. Caso o usuário opte por elaborar os horários de todos os cursos conjuntamente, basta definir o percentual de intersecção como 0. Neste caso, todos os cursos serão unidos em um único conjunto e apenas uma instância do AG será acionada para gerar os horários.

3.3.1 Pré-processamento

A etapa de pré-processamento, invocada na linha 5 da Figura 3, agrupa cursos que possuem uma porcentagem pré-determinada de professores em comum (intersecção). Ela tem por objetivo diminuir o tempo de processamento do algoritmo, dividindo seu espaço de busca em frações menores a partir dos conjuntos gerados e processando essas frações individualmente. Isso diminui a quantidade de choques entre os professores dos cursos, pois os cursos que possuem uma determinada proporção de professores em comum são processados como um único conjunto. Essa etapa também abre a possibilidade de utilizar agentes distribuídos para realizar o processamento paralelo dos conjuntos criados.

Como apresentado na Figura 4, o agrupamento é feito a partir da análise da quantidade de intersecções entre os cursos e assim é criado um novo conjunto formado pela união das informações de ambos os cursos.

Figura 4: Agrupamento de cursos referente ao pré-processamento



3.3.2 Algoritmo Genético

A Figura 5 refere-se ao processamento do método “processamentoDoAG”, na linha 9 da Figura 3. Ele inicia com uma estrutura de repetição para cada um dos conjuntos que o computador é responsável por processar.

Figura 5: Processamento do AG

```
Algoritmo 2: Processamento do AG
  Entrada: conjuntos[], modeloAdaptado,
           configuracoesAG
  início
1   para conjuntoAtual ← conjuntos ate
      ultimoConjunto faça
2     melhorAvaliacao ← defineMelhorAvalia-
        cao(conjuntoAtual.obtemNumeroCursos());
3     populacao ←
        inicializaPopulacao(modeloAdaptado);
4     avaliaPopulacao(populacao);
5     melhorCromossomo ←
        MelhorCromossomo(populacao);
6     enquanto não atender critérios de parada faça
7       cromossomosDeElite ←
          elitismo(porcentElitism, populacao);
8       faA ← calculaFaA(populacao);
9       cromRoleta ← roleta(porcentRoleta, faA,
          populacao);
10      cromossomosCruzados ←
          cruzamento(porcentCruzamento,
          cromossomosDaRoleta);
11      populacao ←
          criaNovaGeracao(cromossomosCruzados,
          cromossomosDeElite);
12      mutacao(porcentMutacao, populacao);
13      avaliaPopulacao(populacao);
14      melhorCromossomo ←
          MelhorCromossomo(populacao);
      fim
15    conjuntoAtual. atribuiMelhorCromos-
        somo(melhorCromossomo);
    fim
16  retorna conjuntos.obtemMelhoresCromossomos();
  fim
  Saída: Melhor cromossomo de cada um dos conjuntos
         processados
```

A seguir, na linha 2, é definido qual o valor máximo da função de avaliação. Esse valor é proporcional ao número de cursos do conjunto. Nas linhas 3 e 4 são realizadas as etapas de inicialização e avaliação da população, respectivamente. É importante destacar que são utilizadas *threads* durante a avaliação para paralelizar e otimizar esse processamento, visto que é um das etapas mais custosas em relação ao tempo de execução do AG. O melhor cromossomo da primeira geração é definido na linha 5. Na linha 6, é utilizado um laço de repetição que é executado até que um dos seguintes critérios de parada sejam atendidos:

- Caso o limite de iterações seja atingido;
- Caso encontre uma solução com a maior avaliação possível;

A seleção por elitismo encontra-se na linha 7, e na linha 8 o cálculo da função de avaliação acumulada é realizada, que é necessária para a seleção através do método da roleta, apresentada na linha seguinte. O cruzamento, na linha 10, é feito a partir dos cromossomos selecionados pela roleta. Para a criação de uma nova população, são

utilizados os cromossomos cruzados em conjunto com os selecionados pelo elitismo, como mostra a linha 11. Após a criação dessa nova população, a mesma passa pelo processo de mutação que consta na linha 12. Nas linhas seguintes, os processos de avaliação e obtenção do melhor cromossomo são refeitos para a nova população.

Após um dos critérios de parada serem aceitos, o melhor cromossomo é armazenado em seu respectivo conjunto. No final do processamento, é retornado o melhor cromossomo de cada conjunto.

3.3.3 Pós-Processamento

Quando o usuário opta por separar os cursos em mais de um conjunto, após a execução do algoritmo genético de maneira distribuída, o resultado obtido pode violar algumas restrições *hard*. Isto ocorre porque ao unir os horários gerados para cada conjunto de cursos pode-se gerar restrições de professores que lecionam disciplinas em um mesmo horário em cursos pertencentes aos diferentes conjuntos. Com o objetivo de resolver tais restrições, foi criado um algoritmo responsável por ajustar a solução e gerar um horário sem restrições *hard*.

O processamento do algoritmo é baseado em um algoritmo de busca em profundidade. Cada nodo da árvore gerada representa uma possível solução, ou seja, um cromossomo diferente. A raiz representa o cromossomo criado a partir da união das soluções dos AG distribuídos e a cada nível gerado na árvore significa que uma restrição foi resolvida, consequentemente, o último nível da árvore representa a solução com menos restrições possíveis.

Inicialmente, a lista com todos os conflitos de horário no cromossomo resultante, caso existam, é obtida. Em seguida, invoca-se o Algoritmo recursivo 3, que recebe o cromossomo gerado com os conflitos como raiz, executa o pós-processamento e retorna o cromossomo com o menor número de restrições possível.

Figura 6: Árvore de profundidade limitada

Algoritmo 3: Árvore de profundidade limitada

```
Entrada: pilha,listaConflitos
início
1   cromossomoAtual ← pilha.topo();
2   se cromossomoAtual.melhorAvaliacaoPossivel()
     então
3       retorna cromossomoAtual;
    fim
4   restricoesOrdenadas ←
     listaConflitos.obtemRestricoesOrdenadas();
5   para restricaoAtual ← restricoesOrdenadas ate
     ultimaRestricaoOrdenada faça
6       turmaConflito ←
         cromossomoAtual.obtemTurmaConflito();
7       filho ← geraFilho(turmaConflito,
         cromossomoAtual, restricaoAtual);
8       se filhoExiste(filho) então
9           pilha.empilha(filho);
10          listaConflitos.removeConflito(restricaoAtual);
        fim
11      senão
12          pilha.desempilha();
        fim
13      retorna arvoreProfundidadeLimi-
         tada(pilha,listaConflitos);
    fim
fim
Saída: Melhor cromossomo possível
```

Na linha 1 da Figura 6, o cromossomo do topo da pilha é obtido, logo após, na linha 2 é verificado se o cromossomo que foi passado para o método é a melhor solução possível, caso a condição seja satisfeita o cromossomo é retornado como mostra a linha 3, dando fim à recursividade. Na linha 4, o método retorna a lista de conflitos ordenada pelo seu grau de dificuldade, ou seja, quanto menor o tempo livre para troca dos horários dos professores de uma turma com conflito, maior é a sua dificuldade. Após esta definição, a lista é iterada na linha 5. Na linha 6 obtém-se a turma em que o conflito está presente e, na linha 7, é feito o processo de geração de um nodo filho. Esse processo parte do cromossomo atual e tenta fazer trocas entre as disciplinas da turma com conflito, após uma troca ser feita verifica-se se o conflito foi resolvido e também se um novo conflito não foi gerado. Caso esses critérios sejam atendidos, um novo nodo é gerado a partir do cromossomo alterado, caso contrário, nenhum filho é gerado. Na linha 8, verifica-se se o método anterior gerou um filho:

• Caso tenha gerado, esse filho é inserido na pilha e o conflito atual é removido da

lista de conflitos, como mostram as linhas 9 e 10.

• Caso não tenha gerado nenhum filho, o cromossomo atual é removido da pilha, como apresentado na linha 12, o que possibilita a realização do *backtracking* para que outros ramos da árvore sejam explorados.

Por fim, na linha 13, é aplicada a recursividade a esse método a qual levará a solução do próximo conflito.

4 RESULTADOS

Nessa seção são apresentados os testes realizados e seus respectivos resultados, com o objetivo de determinar a melhor combinação de parâmetros para a execução do AG, tanto em ambiente centralizado, quanto no ambiente distribuído.

4.1 CONFIGURAÇÃO DOS COMPUTADORES

Para a execução dos testes centralizados e distribuídos, foram utilizados de maneira remota seis computadores alocados no Laboratório de Informática 115 do IFSC. Nos testes centralizados, cada máquina executou de maneira independente um algoritmo genético, já nos testes distribuídos, foi adotada a arquitetura cliente-servidor, no qual uma máquina atuou como cliente e as demais como servidor. Também é importante ressaltar que tanto no ambiente centralizado, quanto no ambiente distribuído, foram utilizadas *threads* de maneira dinâmica, ou seja, o sistema utiliza o número de núcleos disponíveis de cada computador para otimizar o desempenho. Todas as máquinas possuem processador Intel Core 3 6100T 3.2 GHz de 64 bits, além de 8 GB de memória RAM.

4.2 DEFINIÇÃO DOS PARÂMETROS DOS AG - CENTRALIZADO E DISTRIBUÍDO

Foi realizado um teste fatorial com os parâmetros de porcentagem de mutação, elitismo, cruzamento e tamanho da população, com o intuito de analisar cada parâmetro e suas interações. Neste primeiro teste, o percentual de intersecção entre os professores foi definido como 0 para que os horários de todos os cursos fossem montados de uma única vez, sem processamento distribuído. Os valores testados para cada parâmetro são:

• Porcentagem de Mutação: 5, 10, 20, 30 e 50;

• Porcentagem de Elitismo: 4, 8, 20, 30 e 40;

• Porcentagem de Cruzamento: 30, 50, 60 ,70 e 90;

• Tamanho da População: 100, 200, 300, 400 e 500.

Foram executadas 10 replicações com cada combinação possível entre os valores dos parâmetros. Após a execução dos testes foram realizadas as análises de variância e os testes de *Tukey* em relação a três variáveis: resultado, tempo e total de gerações.

A primeira variável analisada foi o resultado, a qual indica a qualidade da solução gerada. Neste teste, 1500 indica uma solução sem a violação de nenhuma restrição. Este é o principal critério para escolher os melhores parâmetros do AG. Além dela, analisou-se o tempo de processamento e, por último, o total de gerações executadas como critérios de desempate caso existam mais de uma configuração de parâmetros que leve aos melhores resultados.

Com base nos resultados da análise de variância, foi possível identificar que ao menos um valor dentre os parâmetros: População, Mutação e Elitismo se difere dos demais. Porém, através dela não é possível constatar quais valores possuem uma diferença significativa. Devido a isso, foram realizados testes de *Tukey*, que faz a comparação de todos os possíveis pares de médias através da Diferença Mínima Significativa (D.M.S). Então, a partir desses testes, é possível identificar qual o melhor valor para cada parâmetro. Existem casos em que devido à variação residual dos resultados, alguns valores podem assumir duas classes em simultâneo. A partir destes testes, identificou-se que os valores que levam aos melhores resultados para cada parâmetro são:

• Porcentagem de Mutação: 20, 30 e 50;

• Tamanho da População: 300, 400 e 500;

• Porcentagem de Elitismo: 20, 30 e 40;

Considerando que mais de um valor para os parâmetros testados levam aos melhores resultados, realizou-se a análise de variância em relação ao tempo de execução e o total de gerações. Nelas foi possível identificar, novamente, que os parâmetros: população, mutação e elitismo apresentam diferenças significativas. O teste de *Tukey* em relação ao tempo de processamento e ao total de gerações permitiu identificar os valores que levam aos melhores resultados em menor tempo. A partir dos resultados da análise de variância, também identificou-se que o parâmetro de cruzamento não apresentou diferença significativa, o que indica que as demais variáveis não são influenciadas de maneira relevante, independente do valor que assumirem.

Baseado nestes testes identificou-se que os melhores valores para cada um dos parâmetros são:

• Porcentagem de Mutação: 50;

• Tamanho da População: 400;

• Porcentagem de Elitismo: 40;

• Porcentagem de Cruzamento: 60;

Vale ressaltar que o parâmetro taxa de cruzamento não afetou os resultados de maneira significativa. Assim, a taxa de cruzamento foi definida como 60 por ser um valor intermediário entre os testados.

Assim como nos testes feitos para elaborar os horários de todos os cursos juntos, foi executado um teste fatorial com os parâmetros de porcentagem de mutação, elitismo, cruzamento e tamanho da população para identificar os melhores parâmetros para a execução distribuída do AG. Nesse teste o percentual de intersecção foi definido como 60%, o que gerou quatro conjuntos, distribuídos entre os computadores servidores e um computador cliente, disponíveis no momento do teste. Os valores testados para os parâmetros são os mesmos apresentados no teste do algoritmo centralizado. Também foram realizadas 10 replicações com cada combinação de parâmetros e, a partir dos resultados dos testes, também foram realizadas as análises de variância e os testes de *Tukey* em relação ao resultado, tempo e total de gerações. Seguindo o modelo dos testes anteriores, conclui-se que os melhores valores para os parâmetros do AG distribuídos são:

• Porcentagem de Mutação: 5;

• Tamanho da População: 200;

• Porcentagem de Elitismo: 20;

• Porcentagem de Cruzamento: 30;

Vale ressaltar que o parâmetro porcentagem de elitismo não afetou os resultados de maneira significativa. Assim, ele foi definido como 20 por ser um valor intermediário entre os testados.

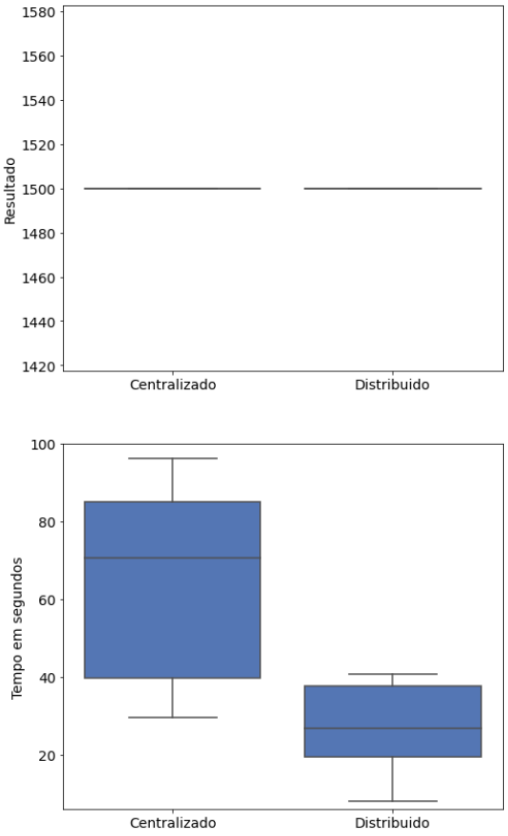
4.3 COMPARAÇÃO DA SOLUÇÃO CENTRALIZADA E DISTRIBUÍDA

Considerando os testes feitos em ambiente centralizado e distribuído, foram obtidas as melhores configurações de parâmetros de execução do AG para cada um dos casos. A partir disso, foi feito um teste, novamente com dez replicações, para comparar as duas formas de execução utilizando suas configurações ótimas, com o objetivo de identificar a forma de execução com melhor desempenho.

Na Figura 7 é possível visualizar a comparação dos resultados das duas execuções quando utilizadas as melhores configurações, identificadas nas duas seções anteriores. No primeiro gráfico pode-se visualizar que ambas as execuções obtiveram resultados

perfeitos em 100% dos testes realizados, sendo assim foi possível visualizar diferenças entre elas apenas quando se compara o tempo de execução (segundo gráfico). Nele pode-se observar que a execução distribuída obteve tempos melhores do que a execução centralizada, 26 segundos na execução distribuída e 70 segundos na execução centralizada, sendo esta diferença estatisticamente significativa, conforme análise de variância realizada que resultou em um valor de p = 0,000477.

Figura 7: Diagrama de caixas comparando a solução centralizada com a distribuída



5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo elaborar um AG para resolução de *timetabling* para o IFSC Campus Lages, de acordo com seus recursos e restrições. O algoritmo pode elaborar horários de vários cursos juntos, o que leva a um grande espaço de busca, mas gera uma solução final para o problema, ou de cada curso por vez, o que reduz drasticamente o espaço de busca, mas gera uma solução que pode ter restrições violadas quando as soluções de cada curso são unidas para formar a solução final.

Com o objetivo de melhorar o desempenho do processamento do AG foi implementado um módulo que tem o papel de processar os dados provenientes do IFSC, dividindo os cursos em conjuntos menores para serem posteriormente processados no

AG, o que representa um dos diferenciais do trabalho.

Outro aspecto importante é que, a fim de resolver algumas das restrições que podem ocorrer em um horário, a inicialização foi modificada, assim como os operadores de cruzamento, baseado na Recombinação Ordenada, e, para manter a estrutura das turmas com suas respectivas matérias, reduzindo o tempo de processamento da função de avaliação.

Em relação à comparação dos resultados das diferentes formas de implementação, através da análise de variância e testes de *Tukey* foi possível identificar os melhores valores para cada um dos parâmetros tanto no ambiente centralizado, quanto no distribuído. Após a obtenção destes valores, foi realizado um teste comparativo entre ambos os ambientes, no qual foi possível identificar que ambos obtiveram 100% de eficácia em relação aos resultados e a solução distribuída proposta chegou no resultado perfeito em um tempo menor, com o AG chegando a solução em uma média de 26 segundos, enquanto a solução centralizada levou, em média, 70 segundos. Assim, pode-se concluir que a solução distribuída proposta neste artigo apresenta um desempenho melhor.

Como possíveis trabalhos futuros, tem-se a possibilidade de realizar o balanceamento de carga no ambiente distribuído levando em consideração critérios de *benchmark* e configuração dos computadores. Além disso, é necessário realizar mais comparações utilizando conjuntos de dados de outros semestres, para assim garantir que o AG proposto pode substituir o *software* atual de geração de horários do IFSC. Por fim, cabe a melhoria na modelagem dos conjuntos de dados para possibilitar a utilização dos dados do ITC, para possibilitar a comparação com os resultados apresentados em outros trabalhos que usaram os dados deste conjunto em seus testes.

REFERENCES

[1] A. Schaerf, "A survey of automated timetabling," *Artificial Intelligence Review*, vol. 13, pp. 87–127, 01 1999.

[2] R. F. Cruz, G. P. dos Santos Júnior, L. B. Fontes, M. dos Anjos Santos, and B. L. C. da Silva, "Geração automática de horário escolar com algoritmo genético," *Eixo*, vol. 8, no. 2, pp. 230–241, 2019. [Online]. Available: http://revistaeixo.ifb.edu.br/index.php/RevistaEixo/article/view/565

[3] A. Colorni, M. Dorigo, and V. Maniezzo, "Metaheuristics for high-school timetabling," *Computational Optimization and Applications*, vol. 9, 04 1999.

[4] R. Concilio, "Contribuições à solução de problemas de escalonamento pela aplicação conjunta de computação evolutiva e otimização com restrições," Campinas, 2000.

[5] R. Linden, *Algoritmos Genéticos*, 3rd ed. Rio de Janeiro: Ciência Moderna, 2012.

[6] A. Wren, "Scheduling, timetabling and rostering—a special relationship?" in *International conference on the practice and theory of automated timetabling*. Springer, 1995, pp. 46–75.

[7] E. Burke, B. Mccollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, pp. 177–192, 01 2007.

[8] S. Daskalaki and T. Birbas, "Efficient solutions for a university timetabling problem through integer programming," *European Journal of Operational Research*, vol. 160, pp. 106–120, 01 2005.

[9] E. K. Burke, R. Qu, and A. Soghier, "Adaptive selection of heuristics within a grasp for exam timetabling problems," *Proceedings of the 4th Multidis ciplinary International Scheduling: Theory and Applications*, pp. 10–12, 2009.

[10] A. L. Bolaji, A. T. Khader, M. A. Al-Betar, and M. A. Awadallah, "University course timetabling using hybridized artificial bee colony with hill climbing optimizer," *Journal of Computational Science*, vol. 5, no. 5, pp. 809–818, 2014.

[11] K. Socha, J. Knowles, and M. Sampels, "A max-min ant system for the university course timetabling problem," in *International Workshop on Ant Algorithms*. Springer, 2002, pp. 1–13.

[12] S. Edelkamp and S. Schr¨odl, *Heuristic Search - Theory and Applications.*, 1st ed. Morgan Kaufmann, 01 2012.

[13] D. J. Montana, M. Brinn, S. Moore, and G. Bidwell, "Genetic algorithms for complex, real-time scheduling," *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, vol. 3, pp. 2213–2218 vol.3, 1998.

[14] S. R. Sutar and R. S. Bichkar, "High school timetabling using tabu search and partial feasibility preserving genetic algorithm," *International Journal of Advances in*

Engineering & Technology, vol. 10, no. 3, p. 421, 2017.

[15] S. S. A. Alves, S. A. F. Oliveira, and R. N. A. R., *A Recursive Genetic Algorithm-Based Approach for Educational Timetabling Problems*. Cham: Springer International Publishing, 2017.

[16] C. P. Borges, "Aperfeiçoamento de um algoritmo genético para elaboração de quadros de horários em instituições de ensino," Lages, 2007, trabalho de conclusão de curso em Sistemas de Informação. Universidade do Planalto Catarinense - SC.

[17] P. d. S. R. Ramos, "Sistema automático de geração de horários para a ufla utilizando algoritmos genéticos," 2002, trabalho de conclusão de curso em Ciência da Computação. Universidade Federal de Lavras - MG.

[18] P. Salza and F. Ferrucci, "Speed up genetic algorithms in the cloud using software containers," *Future Generation Computer Systems*, vol. 92, 10 2018.

[19] E. M. da Silva, "Algoritmos genéticos para geração de quadros de horários," Lages, 2010, trabalho de conclusão de curso em Sistemas de Informação. Universidade do Planalto Catarinense - SC.

[20] G. Luger, *Inteligência artificial*, 6th ed. São Paulo: Pearson Education do Brasil, 2013.