MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

ALOCAÇÃO DE HORÁRIO ESCOLARES COM ABORDAGEM HEURÍSTICA DE COLORAÇÃO DE GRAFOS

Wesley Henrique Batista Nunes

Orientador: Prof. Me. Everthon Valadão

Formiga - MG
2018

WESLEY HENRIQUE BATISTA NUNES

ALOCAÇÃO DE HORÁRIO ESCOLARES COM ABORDAGEM HEURÍSTICA DE COLORAÇÃO DE GRAFOS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais - *Campus* Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Everthon Valadão

Formiga - MG

2018

Nunes, Wesley Henrique Batista.
004 Alocação de horários escolares com abordagem heurística de Coloração de grafos / Wesley Henrique Batista Nunes. -- Formiga : IFMG, 2018.
81p. : il.

Orientador: Prof. MSc. Everthon Valadão dos Santos
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Alocação de Horários. 2. Colação de grafos. 3. Escola. 4. Cronograma de aula. 5 . Otimização. I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária MSc. Naliana Dias Leandro CRB6-1347

WESLEY HENRIQUE BATISTA NUNES

ALOCAÇÃO DE HORÁRIOS ESCOLARES COM ABORDAGEM HEURÍSTICA DE COLORAÇÃO DE GRAFOS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais — Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Aprovado em 24 de novembro de 2018.

BANCA EXAMINADORA

Me. Everthon Valadão
Professor no IFMG — Campus Formiga

Dr. Mayron César de Oliveira Moreira
Professor na UFLA (Universidade Federal de Lavras)

Me. Raí Caetano de Jesus
Professor no IFMG — Campus Formiga

Me. Walace de Almeida Rodrigues
Professor no IFMG — Campus Formiga

AGRADECIMENTOS

Agradeço primeiramente à Deus e a minha família, que sempre me apoiaram e me ajudaram nos momentos em que tive mais dificuldades.

Agradeço ao professor Everthon Valadão, por ter sido bem mais que um orientador, sempre tendo paciência em me ajudar nos momentos de dúvidas, não só durante a construção deste trabalho mas também durante toda a universidade.

Também gostaria de agradecer aos grandes amigos, Saulo Ricardo, Arthur Teodoro, Renato Borges, Samuel Gomes e Renata Pereira, que estiveram junto comigo durante toda essa longa caminhada e à minha namorada Ana Flávia por sempre estar comigo nos momentos difíceis.

"Com grande poderes vêm grandes responsabilidades"
(Parker, Ben)

RESUMO

Existe uma diferença essencial entre o problema da alocação de horários (*timetabling*) escolar e o de universidade. Nas escolas, a quantidade de dias e horários disponíveis é bem mais restrita, geralmente alocando-se as aulas de uma turma somente em um determinado período do dia e os professores possuem mais restrições de horário, por tipicamente trabalharem em mais de uma escola. Neste trabalho de conclusão de curso é abordado o *timetabling* escolar, que consiste em automatizar a elaboração dos horários de aula de uma escola, respeitando as restrições de horário impostas, bem como otimizando o atendimento às recomendações pedagógicas e preferências de horário. Ao invés da tradicional abordagem com busca e ajuste tabular, neste trabalho foi utilizada uma modelagem com a transformação do problema original para o de coloração de grafos. Entretanto, para viabilizar a utilização do software por pessoas leigas, o formato da entrada e saída de dados consiste em uma tradicional planilha eletrônica, facilitando as tarefas de impressão e publicação dos horários. A metaheurística GRASP+VND foi utilizada para encontrar soluções factíveis, geradas em tempo hábil, tendo sido foi comparada a outras abordagens e consistentemente propôs soluções próximas da solução ótima. Ou seja, soluções com a menor quantidade de horários utilizados, em dias úteis da semana, para encaixar a demanda de horários de aula e respeitar as restrições impostas. Para as quatro instâncias (reais) de escolas de ensino fundamental e médio utilizadas na validação, o algoritmo conseguiu atender a todas as restrições de horário utilizando a menor quantidade possível de horários para alocar as aulas, mostrando a viabilidade das soluções geradas. Ademais, a otimização da alocação de horários escolares com o GRASP+VND visou também maximização da qualidade da solução gerada, de maneira que apenas uma minoria das recomendações pedagógicas e parte das preferências de horários não puderam ser atendidas. Foram conduzidos experimentos fatoriais para calibrar a metaheurística de maneira a obter um bom custo-benefício, apresentando soluções de boa qualidade e consumindo um baixo tempo de execução para tal. Em um computador de configurações modestas, a ferramenta desenvolvida consumiu de 30 minutos a até 6 horas para gerar uma solução viável, de maneira automatizada, considerando o caso de teste mais trabalhoso (entrada com 15 turmas, 5 dias úteis e apenas 1 turno). Considerando o curto tempo investido na geração de cada solução de alocação de horários (1 solução por minuto) e a boa qualidade dos resultados obtidos, consideramos que o protótipo produzido tem potencial para crescer e amadurecer. Se for comparado ao processo manual de alocação de horários, considere a eficácia, eficiência e praticidade da ferramenta produzida, de maneira que seria de grande valia para as escolas de ensino básico mesmo enquanto protótipo.

Palavras-chaves: Alocação de horários. Coloração de grafos. Escola. Cronograma de aulas. Otimização.

ABSTRACT

There is an essential difference between the timetabling problem for a university and for some school. In schools, the number of available days and time slots is much narrower, usually by assigning a class at a particular period of day, and teachers have more schedule restrictions because they typically work in more than one school. In this work, the school timetabling variation of the problem is approached, which consists of automating the elaboration of class schedules for a school, respecting the imposed time restrictions, as well as optimizing the attendance to pedagogical recommendations and personal schedule preferences. Instead of the traditional approach with tabular search, in this work we modeled the timetabling as a graph colouring problem. However, in order to enable the use of the software by laypersons, the data input and output format consists of a traditional spreadsheet, facilitating the printing and publication of the generated class schedules. The GRASP+VND metaheuristic was used to find feasible solutions, generated in a timely manner, having been compared to other approaches and consistently proposed solutions close to the optimal solution. That is, solutions with the least amount of hours allocated on weekdays, to fit the demand of class scheduling and respect the imposed restrictions. Four (real) instances of elementary and secondary schools were used in the validation proccess. The algorithm was able to meet all the scheduling constraints using the minimum schedule to allocate all the classes. In addition, the optimization of school timetabling with GRASP+VND also aimed at maximizing the quality of the generated solution, so that only a minority of the pedagogical recommendations and part of the scheduling preferences could not be met. Factorial experiments were conducted to calibrate the metaheuristic in order to obtain a good tradeoff, presenting solutions of good quality and consuming a short execution time for such. In a computer of modest configurations, the developed tool consumed from 30 minutes to up to 6 hours to generate a viable solution, in an automated way, considering the most demanding test case (input with 15 classes, 5 working days and only 1 shift). Considering the short time invested in the generation of each timetabling solution (1 solution per minute) and the good quality of the obtained results, we consider that the prototype produced has the potential to grow and mature. If compared to the manual timetabling process, consider the effectiveness, efficiency and practicality of the tool produced, so that it would be of great value to elementary schools even as a prototype.

Key-words: Timetabling. Graph colouring. School. Class Schedules. Optimization.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

GPL	*General Public License*
FOSS	*Free Open Source Software*
GRASP	*Greedy Randomized Adaptive Search Procedure*
VND	*Variable Neighborhood Descent*
VNS	*Variable Neighborhood Search*
FO	Função Objetivo
FOs	Funções Objetivos
LCR	Lista de Candidatos Restrita
MEC	Ministério da Educação
CV	Coeficiente de Variação

LISTA DE SÍMBOLOS

Δ	Variação da Função Objetivo

SUMÁRIO

1 INTRODUÇÃO

O problema que será abordado no trabalho de conclusão de curso é o *timetabling* escolar, que consiste em alocar todos os horários de aula de uma escola de ensino fundamental e médio, respeitando as restrições impostas. Como diferencial deste trabalho de conclusão de curso, ao invés de modelar o problema tradicionalmente como uma busca e ajuste tabular, este trabalho utilizará uma modelagem com transformação no problema de coloração de grafos, conforme será explicado no Capítulo 2.

A implementação deste trabalho não deverá violar nenhuma restrição forte e buscar minimizar o não atendimento às restrições fracas. O problema considera as seguintes restrições fortes: não é permitido duas aulas com um mesmo professor no mesmo horário, não pode haver duas aulas com uma mesma turma no mesmo horário, todas as aulas devem ser alocadas ao longo dos dias de funcionamento da escola e turnos de aula predeterminados.

Já as restrições fracas variam de acordo com as recomendações pedagógicas e particularidades como, por exemplo: não deverá haver três ou mais aulas geminadas (com o mesmo professor em horários sequenciais), não deve haver horários de aula separados por lacunas muito grandes para uma mesma turma (janelas entre as aulas) e as aulas devem estar distribuídas de forma balanceada ao longo dos dias da semana, bem como buscar atender às particularidades de cada professor em relação a dias ou horários em que não possa lecionar.

1.1 Justificativa

O motivo da escolha deste trabalho é o meu grande interesse pela área de otimização, de heurísticas e meta-heurísticas. Também, por experiência pessoal tenho interesse no problema de alocação de horários pois observei que, na escola em que estudei durante meu ensino fundamental e médio, a supervisora tentava alocar todos os horários da escola manualmente. Sem o auxílio de qualquer ferramenta computacional, tal trabalho de alocar horários se mostra hercúleo. Então, tendo observado o imenso esforço e frustração que isso gera aos funcionários de uma escola, que devem periodicamente alocar os horários de aula, decidi propor esse projeto de TCC para poder ajudar pessoas e escolas.

Existe uma grande diferença entre o problema de alocação de horários escolar e de universidades. Nas escolas, a quantidade de horários é bem mais restrita, com aulas ocorrendo somente da parte da manhã, os professores podem ter mais restrições pois eles trabalham em várias escolas e cada aluno tem aulas com uma mesma turma ao longo do ano.

1.2 Objetivos

Criar um protótipo de software que, através de algoritmos heurísticos, gere a alocação de horários de aula para os dados fornecidos como entrada, software que será disponibilizado a uma escola de ensino fundamental e médio da região em que o *campus* do Instituto Federal de Minas Gerais de Formiga se encontra. São objetivos secundários e mais específicos os seguintes tópicos, apresentados na sequência.

- Elaborar um *software* que gere a alocação de horários de aula, modelando o problema como o de coloração de grafos e utilizando meta-heurística para encontrar as soluções;
- Padronizar o formato da entrada de dados (ex.: um modelo de planilha) para facilitar que pessoas leigas possam utilizar o *software*;
- Fazer com que a saída de dados do *software* seja uma página com as tabelas de horários de todas as turmas, facilitando para que possa ser impressa ou disponibilizada em um formato digital (ex.: PDF);
- Atender todas as restrições fortes e tentar minimizar o não atendimento às restrições fracas, para que os horários gerados agradem aos envolvidos.
- Apresentar algumas variações das tabelas de horários finais, para que as pessoas da escola possam inspecionar e escolher a solução que agrade à maioria dos envolvidos.

1.3 Estrutura do Trabalho

Esta monografia de conclusão de curso é organizada em 5 capítulos. No capítulo 2, após a contextualização feita na Introdução, são apresentados os conceitos teóricos necessários para a compreensão do trabalho. O capítulo 3 explicita as ferramentas utilizados no desenvolvimento do trabalho, como linguagens de programação, bibliotecas e plataformas de desenvolvimento. É abordada também a modelagem do problema proposto, juntamente com seus formas de entradas e saídas de dados. No capítulo 4, são apresentados como as heurísticas e metaheurísticas foram aplicadas, após é feito uma análise dinâmica da execução e o projeto fatorial 2^k. Finalizamos este documento com os resultado obtidos através dos experimentos realizados, também é realizado uma comparação entre as metaheurísticas abordadas, juntamente com as instâncias e seus cenários,

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados, de maneira objetiva, os principais conceitos necessários ao desenvolvimento e compreensão deste trabalho de conclusão de curso, tais como os problemas de alocação de horários, coloração de grafos e soluções metaheurísticas. Então, ao final deste capítulo serão apresentados alguns trabalhos relacionados.

2.1 *Timetabling*

O *timetabling*, ou problema de alocação de horários, consiste em alocar um conjunto de eventos entre professores, alunos e horários (que varia de acordo com a instituição), satisfazendo um conjunto de condições de vários tipos.

Obter uma solução manual poderá ser uma tarefa muito difícil ou até impossível, necessitando do trabalho de várias pessoas. Levando em conta esta dificuldade, o problema pode ser automatizado. Segundo Schaerf (1999) durante os últimos cinquenta anos, começando com Gotlieb em 1963, muitos artigos foram publicados em conferências e jornais, bem como Várias soluções foram desenvolvidas com algumas obtendo bons resultados (SCHAERF, 1999).

Diferentes tipos de *timetabling* foram propostos durante os anos nas literaturas, cada um com suas peculiaridades, baseado em sua instituição (escola ou universidade) e tipos condições. O *timetabling* escolar visa alocar as aulas de todas as turmas sem que os professores tenham que dar duas aulas ao mesmo tempo ou as turmas tenham duas aulas no mesmo horário. O *timetabling* universitário visa alocar aulas para um conjunto de discentes minimizando a sobreposição de aulas para um conjunto incomum de alunos, ou seja, esta variação do problema leva em consideração o currículo de cada aluno para que, caso ele tenha sido reprovado em alguma disciplina, o algoritmo tente alocá-lo nas disciplinas pendentes e nas correntes. O *timetabling* para alocação de exames visa alocar os exames de universidades para que as datas sejam o mais espalhadas possível e que não tenha o mesmo conjunto de alunos em dois exames ao mesmo tempo.

Em muitos casos o *timetabling* se limita a encontrar uma alocação viável, ou seja, que satisfaça todas as restrições propostas pelo problema, neste caso o *timetabling* se torna um problema de busca. Em outros casos o *timetabling* pode se tornar um problema de otimização em que todas as restrições fortes devem ser respeitadas obrigatoriamente para que a solução seja factível e as restrições fracas devem ser atendidas sempre que possível.

2.1.1 Restrições de horários de aula

Este trabalho se insere no contexto do *timetabling* escolar, visando sempre otimizar as restrições fracas e respeitar as restrições fortes.

O *timetabling* escolar consiste em associar as aulas em determinados períodos respeitando todas as restrições fortes, que são:

- Um professor não pode ministrar duas (ou mais) aulas ao mesmo tempo;
- Uma turma não pode ter duas (ou mais) aulas ao mesmo tempo;
- O professor não irá ministrar uma aula em horário com indisponibilidade do mesmo;
- A turma não irá assistir aula em horário com indisponibilidade da mesma.

Tais restrições fortes são representadas na Equação 2.1, na qual cada variável dentro do somatório representa uma restrição.

$$\sum_{i=1}^{horários} x_1 + x_2 + x_3 + x_4 = 0 \tag{2.1}$$

Observe que caso alguma restrição seja desobedecida em apenas um dos horários propostos, o valor do somatório será maior que zero e, portanto, tal solução deverá ser considerada como infactível.

2.1.2 Recomendações pedagógicas

O Ministério da Educação (MEC) é o órgão governamental brasileiro responsável pela política nacional de educação, educação infantil, avaliação e pesquisa institucional (MEC, 2018). Consultando alguns dos servidores técnico-administrativos da Secretaria de Controle e Registro Acadêmico do *campus* Formiga do IFMG, levantamos algumas recomendações para que os alunos tenham um melhor desempenho escolar: (i) evitar ter três (ou mais) aulas subsequentes de uma matéria, pois isso faz com que o rendimento dos alunos seja defasado no decorrer das aulas; (ii) conter poucos horários livres espaçados entre aulas ("janelas"), para que os alunos não percam o ritmo de estudo; (iii) duas aulas de uma determinada disciplina não sejam separadas por outra(s) aula(s) distinta(s), pois isso poderá fazer com que os alunos percam a linha de raciocínio.

Ademais, a ferramenta possibilita que professores informem os horários de sua preferência para ministrar as aulas. Tais preferências também foram modeladas como restrições fracas. As soluções de alocação de horários geradas pela heurística que atendam um maior número de recomendações e preferências serão melhor avaliadas pelo algoritmo, buscando assim atendê-las quando possível.

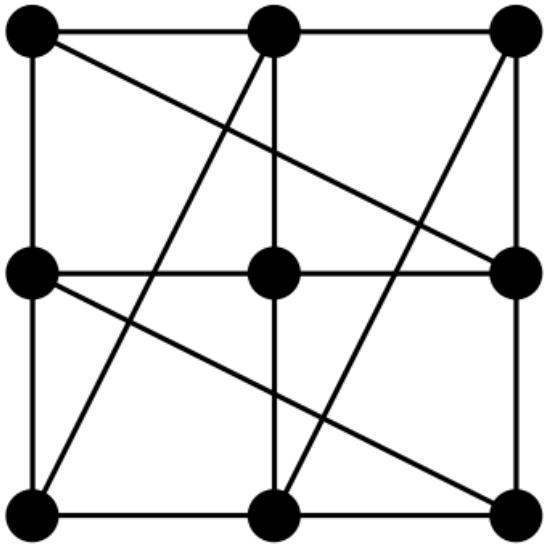
2.2 Coloração de Grafos

O problema de coloração de grafo é um dos mais velhos e conhecidos problemas da teoria de grafos (LEWIS, 2016). Este problema pode ser utilizado para resolver diversos problemas diferentes de alocação de horários, alocação de produtos dentre outros. O problema se encontra na classe dos NP-Completo (I. et al., 1988) e obter uma solução para ele com uma instância de grande escala pode fazer com que leve anos para ser calculada.

2.2.1 Conceito de Grafo

Um grafo é uma estrutura constituída pelo par G = (V,E) em que V é o conjunto dos vértices e o conjunto dos arcos ou arestas. Cada aresta $(u, v) \in E$ está associada estreitamente a dois vértices. Poderá haver também um laço, onde um vértice tem uma aresta que liga nele próprio. A Figura 1 apresenta um exemplo de grafo com 9 vértices.

Figura 1 – Representação de um grafo



Fonte: (PINA; FEOFILOFF, 2018)

Segundo Biggs, Lloyd e Wilson (2006) a origem da teoria de grafos foi traçado por Euler para a resolução do problema das pontes de Konigsberg, originado em 1735. A partir da origem da teoria surgiu o conceito de um grafo Euleriano. Os estudos de ciclos em poliedros de Thomas Penyngton Kirkman (1806 a 1895) e Sir William Rowan Hamilton (1805 a 1865) levou ao conceito de grafo Hamiltoniano.

O tipo de grafo utilizado neste trabalho é o grafo simples. Um grafo simples é não direcionado, ou seja, caso o vértice x esteja ligado em y, isso implica que y também está ligado em x. Neste grafo não existem laços e existe somente uma aresta entre dois vértices.

2.2.2 Origem do problema

Segundo Kubale (2004) a origem do problema de coloração de grafos foi em meados de 1852 quando Morgan escreveu uma carta para seu amigo Hamilton informando que um de seus estudantes teria observado que é necessário 4 cores para colorir o condados da inglaterra sem que nenhum condado adjacente contenha a mesma cor. Então uma pergunta apareceu: "*Qual a quantidade de cores mínimas para se colorir determinado mapa (real ou imaginário)?*" O problema foi publicado em um formato de desafio para o público por Cayley em 1878.

Existem vários modelos de coloração de grafos, mas em geral, o problema consiste em colorir vértices, arestas, faces de um grafo planar ou qualquer combinação de conjuntos. Além disso, para cada modelo de coloração de grafo existem regras para se poder otimizar a solução. Alguns modelos não clássicos permitem que condições adicionais como, colocar mais de uma cor em um mesmo elemento, permitindo a divisão de cores entre frações ou até mesmo admitindo a desenvoltura das cores.

A maioria destes modelos descritos acima são generalizações do modelo clássico de coloração de grafos. Neste trabalho foi utilizada a coloração de grafos clássica.

2.2.3 Coloração de vértices

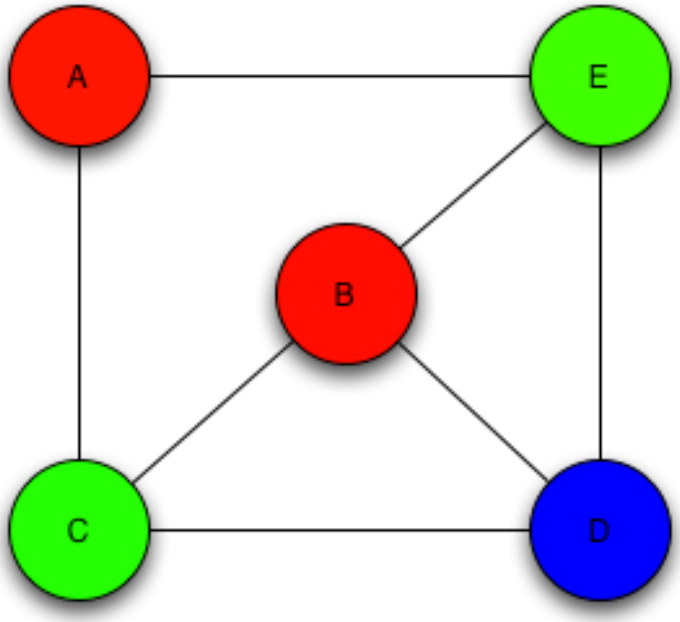
De todos os modelos descritos acima um dos mais estudados é o coloração de grafos clássico. A classe de problemas NP-Completo (I. et al., 1988) faz com que encontrar uma solução ótima para o problema seja uma tarefa bem trabalhosa, então os problemas que utilizam a coloração de grafos como base, caso o tamanho dos dados sejam muito grandes, é necessário trabalhar com valores sub-ótimos.

Para a definição do problema descrita por Kubale (2004), temos um grafo G que é ordenado pelo par G = (V,E) , onde V é um conjunto finito de elemento chamado vértices, enquanto E é também um conjunto finito de elementos chamados arestas. Os vértices u,v que pertencem a V são chamados de vértices adjacentes (ou vizinhos) se u,v pertencerem a E. Os vértices que estão no conjunto E não poderão conter a mesma coloração.

A Figura 2 demonstra um exemplo de um grafo colorido e, como pode-se observar, os vértices adjacentes, não possuem a mesma coloração. O número cromático é a quantidade mínima de cores necessárias para colorir um determinado grafo, ou seja, o número mínimo de cores necessárias para colorir todos os vértices de um grafo, sendo que os vértices adjacentes não poderão conter a mesma cor.

Baseado neste problema foi possível modelar o problema de alocação de horários (*timetabling*) descrito neste trabalho. Tal modelagem é explicada na Subseção 3.3.1.

Figura 2 – Grafo Colorido



Fonte: (LUMSDAINE, 2018)

2.2.4 Abordagens Algorítmicas

Vale ressaltar que o problema de coloração de grafos é classificado como um problema NP-completo (I. et al., 1988), ou seja, um problema de decisão que pode ser solucionado em tempo polinomial apenas em uma Máquina de Turing não determinística, podendo ser transformado em qualquer outro problema NP em tempo polinomial (KARP, 1972) e podendo ser transformado em qualquer outro problema NP em tempo polinomial. Entretanto, apesar da validade de uma solução proposta para o problema poder ser verificada rapidamente (em tempo polinomial), não é conhecida uma forma eficiente de se encontrar uma solução para o problema, de maneira que o tempo necessário para resolver deterministicamente o problema aumenta rapidamente à medida que cresce o tamanho da instância do problema.

Tentar resolver o problema de coloração de grafos de uma maneira em que se é testado todas as possibilidades (força bruta), dependendo do tamanho da instância em que se está trabalhando pode levar alguns anos para poder obter o melhor resultado. Então outras maneiras de abordagem do problema foram buscadas para tentar resolver o problema em um tempo viável e obter o melhor resultado.

Além do algoritmo de força bruta, existem algumas outras abordagens como o algoritmo de coloração gulosa. Esta abordagem do algoritmo consiste em selecionar vértices consecutivamente para serem coloridos até que o grafo esteja completamente colorido. A

maneira que estes vértices são escolhidos pode ser de diferentes formas, como, levar em consideração o grau de cada vértice e escolher o vértice com maior ou menor grau.

Com a utilização do algoritmo de coloração gulosa, não temos a garantia de que em todas as instâncias, será obtido o melhor resultado, então com isso, as Heurísticas e Metaheurísticas foram adotadas em alguns casos para a resolução dos problemas da classe NP-Completo (I. et al., 1988).

2.3 Metaheurísticas

Alguns problemas que encontramos no cotidiano são facilmente resolvidos por um método computacional determinístico mas, para alguns problemas complexos como o *timetabling* e a coloração de grafos, até o presente momento não se conhece uma maneira viável de resolvê-los com um algoritmo determinístico de tempo polinomial em uma máquina determinística.

Então, para abordar problemas com esse grau de dificuldade foram propostos vários tipos de metaheurísticas ao longo dos anos, para que uma boa solução fosse possível de ser encontrada em tempo hábil. Porém, abordagens metaheurísticas não garantem que encontrarão a melhor solução possível (ótimo global).

Segundo a definição original, metaheurísticas são métodos de solução que coordenam procedimentos de busca locais com estratégias de mais alto nível, de modo a criar um processo capaz de escapar de mínimos locais e realizar uma busca robusta no espaço de soluções de um problema (GLOVER; KOCHENBERGER, 2003). As metaheurísticas abordadas neste documento são: GRASP; VND; *Hill Climbing* e *Simulated Annealing*.

2.3.1 GRASP

O GRASP (Greedy Randomized Adaptive Search Procedure) é uma metaheurística que foi apresentada proposta por Feo e Resende (1995) para a resolução de problemas combinatórios. Esse algoritmo se baseia em um processo de busca adaptativa aleatória e gulosa, que consiste em duas etapas: (i) geração de uma solução inicial que é construída com elementos selecionados de forma aleatório-gulosa; e (ii) exploração da vizinhança do elemento gerado visando melhorar a função objetivo da solução proposta.

O pseudocódigo do algoritmo do GRASP é descrito no Quadro 1.

Na fase de construção do GRASP, uma solução é iniciada com um conjunto vazio de elementos e então, cada elemento é colocado na solução incrementalmente até que ela esteja completa, respeitando as restrições do problema. A escolha de cada elemento que será colocado na solução é determinada por uma lista ordenada, denominada lista de candidatos (LC). Esta lista é ordenada de acordo com uma função de avaliação dos candidatos, que

Quadro 1 – Pseudocódigo do GRASP

```
Funcao grasp()
1   InstanciaEntrada();
2   enquanto critério de parada GRASP não satisfeito ->
3       construcaoRandomicaGulosa(Solucao);
4       buscaLocal(Soluca);
5       atualizaSolucao (Solucao,MelhorSolucaoEncontrada);
6   fim enquanto;
7   retorno(MelhorSolucaoEncontrada);
```

Fonte: Adaptado de Feo e Resende (1995)

avalia qual seria o ganho em colocar determinado candidato na solução em construção. Assim, na primeira posição da Lista de Candidatos estará o candidato que apresentar o maior benefício a um menor custo, sendo portanto ordenada em função do ganho dos candidatos. O comportamento aleatório-guloso provêm do uso de um subconjunto desta lista, chamado de lista de candidatos restrita (LCR), na qual estarão apenas os melhores candidatos a entrar no conjunto da solução. O tamanho da LCR é determinado como segue: caso a LCR seja de tamanho 1, o método será puramente guloso e, caso o tamanho da LCR seja igual ao tamanho da LC, o método tende a uma maior aleatoriedade.

Da lista de candidatos restrita, o candidato que irá entrar na solução em construção é escolhido através do sorteio de um número para uso em uma "roleta estocástica", na qual os melhores candidatos tem uma "fatia" maior na roleta. Assim, garante-se que a construção da solução não seja puramente aleatória (qualquer candidato teria a mesma chance) nem puramente gulosa (o primeiro candidato tem 100% de chance), pelos candidatos terem maior ou menor chance de serem selecionados em função de seu ganho ao ser incorporado à solução.

O pseudocódigo da fase construtiva está descrito no Quadro 3.

Quadro 2 – Pseudocódigo da fase construtiva

```
Funcao construcaoRandomicaGulosa(Solucao)
1   Solucao = {};
2   enquanto construcao da solucao nao termina ->
3       FazRCL(RCL);
4       s = SelecionaElementoAleatorio(RCL);
5       Solucao = Solucao U {s};
6       AdaptaFuncaoGulosa(s);
7   Fim enquanto;
fim construcaoRandomicaGulosa
```

Fonte: Adaptado de Feo e Resende (1995)

Um passo muito importante no GRASP é a busca local, após uma solução factível

ser construída pela fase inicial. A busca local consiste em gerar soluções denominadas "vizinhos" bem próximas da solução corrente, sempre tentando explorar ao máximo o espaço de busca. Após técnicas serem aplicadas na solução inicial, vizinhos são gerados e analisados com o intuito de otimizar a função objetivo.

O pseudocódigo da fase de busca local é:

Quadro 3 – Pseudocódigo da busca local

```
Funcao local(P,N(P),s)
1   Enquanto não encontrar otimo local ->
2       Encontre uma melhor solução t em N(s);
3       Coloca t em s;
4   fim enquanto;
5   retorna(s e o ótimo local de P)
fim local;
```

Fonte: Adaptado de Feo e Resende (1995)

2.3.2 *Hill Climbing*

O *Hill Climbing* é uma metaheurística de busca local simples onde a vizinhança da solução é explorada até que nenhum vizinho melhor seja encontrado. O algoritmo é utilizado para se maximizar ou minimizar uma função objetivo de um determinado problema. O motivo da escolha desta heurística foi para servir como um filtro na geração de vizinhança. O *Hill Climbing* irá receber como entrada uma solução da fase construtiva do GRASP e trabalhará em cima das soluções (vizinhos) geradas pelo VND, que será explicado adiante na Subseção 2.3.3.

O *Hill Climbing* irá selecionar entre as soluções geradas pelo VND e verificar se algum vizinho tem a função objetivo inferior aos vizinhos gerados anteriormente, esse processo irá ocorrer até que não houver mais melhorias na geração de vizinhança

2.3.3 VNS e VND

O VNS (*Variable Neighborhood Search*) é uma metaheurística proposta por Hansen et al. (2006). Este método consiste em ter um conjunto finito de estruturas de vizinhança e, com isso, executar buscas locais em distintas partes da região factível buscando evitar que o algoritmo fique preso em mínimos locais.

Neste trabalho foi utilizada a variação VND (*Variable Neighborhood Descent*) pois esta meta-heurística faz com que estruturas de vizinhanças diferentes sejam utilizadas, fazendo com que o espaço de busca na geração de um novo vizinho seja bem explorado. O funcionamento deste método consiste em pré definir quais estruturas de vizinhança

poderão ser utilizadas e a quantidade delas. Para cada iteração que o algoritmo fizer, a estrutura de vizinhança que está sendo utilizada na solução corrente será alterada para poder assim aumentar o espaço de busca. Caso uma solução melhor seja encontrada, a solução corrente passa a ser a encontrada e é a mesma volta a ser explorada desde a primeira estrutura de vizinhança. Caso todas as estruturas sejam utilizadas e não há melhoria, o algoritmo retorna a melhor solução encontrada.

Para melhor entendimento do algoritmo o pseudocódigo está ilustrado no Quadro 4.

Quadro 4 – Pseudocódigo do algoritmo VND

```
1 Seja s0 uma solução inicial e r o número de estruturas de vizinhanças.
2 s = solucaoInicial;
3 k = 1;
4 enquanto(k <=  r):
5   viz = geraNovoVizinho(s);
6   se(fo(viz) < fo(s):
7       s = viz;
8       k = 1;
9   senao:
10       k = k +1;
11 fim enquanto
12 retorna s;
13 fim VND
```

Fonte: Adaptado de (MARTINS et al., 2009)

2.3.4 Critério de Boltzmann

O critério de Boltzmann é um critério de aceitação característico da metaheurística *Simulated Annealing* (LAARHOVEN, 1987), cujo funcionamento consiste em sempre aceitar uma solução em que a função objetivo teve alguma melhora, mas aceitar ou não uma solução em que a função objetivo piora. O critério de aceitação decide se a função objetivo pior irá ser aceita ou não de acordo com a Equação 2.2.

$$P(novo_estado) = e^{-\Delta/T} \tag{2.2}$$

P irá receber o resultado da equação onde o Δ é o resultado da subtração da função objetivo atual com função objetivo após os métodos aplicados e o T é a temperatura do recozimento. Quando o valor P é calculado, um número aleatório entre 0 e 1 é sorteado, caso o número aleatório for menor ou igual a P, então a solução é aceita, mesmo sendo pior que a melhor conhecida até então.

O critério de Boltzmann tem como característica, aceitar mais pioras na solução quando a temperatura está mais alta pois, quando o número de Euler estiver elevado a um

número muito baixo, o resultado será um número bem próximo de 1, então a probabilidade do número aleatório sorteado for menor que o resultado do critério de Boltzmann é bem grande.

2.4 Trabalhos Relacionados

No trabalho de Souza, Costa e Guimarães (2002) o problema de *timetabling* tem uma abordagem diferente. Uma meta-heurística híbrida é utilizada para a resolução do problema, o algoritmo genético, busca tabu e GRASP. A estrutura de dados utilizada neste trabalho é matricial. O procedimento de resolução do problema consiste em gerar soluções factíveis com o GRASP, gerar vizinhos com as mutações do algoritmo genético e refinar os melhores indivíduos com a busca tabu.

Pereira, Netto e Lacruz (2007) utilizam grafos como estrutura de dados para a resolução do problema de alocação de horários. Quando um professor não poder lecionar em certos horários, é criado um vértice para essa restrição, então quando o grafo for passar pelo processo de coloração, este vértice será considerado. É adicionado ao grafo vértices representando horários, que ligam os professores/turmas que não podem dar aula nos respectivos horários, então, resolvendo o problema de coloração, os professores/turmas não terão aulas nos horários que eles não podem.

A meta-heurística utilizada neste artigo foi o GRASP, o tamanho da lista de candidatos restrita com um tamanho fixo em 6. A lista de candidatos restrita é formada com os 6 professores que têm mais aulas pendentes e não por vértices de maior grau. Os vértices também são avaliados com as restrições antes mesmos de entrarem na lista de candidatos restrita, fazendo assim a possibilidade de gerar uma solução de qualidade que viola poucas restrições. Existe um número máximo de impedimentos para a não entrada na lista. Quando esse número é atingido o vértice (mesmo violando uma restrição) é inserido na lista.

Com o GRASP sem nenhuma modificação, foi possível alocar os horários da instância, mas com o GRASP modificado, as funções objetivos obtever uma pequena diferença, fazendo com que as soluções propostas pelo GRASP modificado se sobressaíssem.

No trabalho de Pandey e Mohan (2016) é ilustrado como o uso de um algoritmo genético pode resolver o problema de *timetabling*. É dito que o algoritmo não foi testado com uma instância muito grande pois levaria muito tempo de processamento. Depois também é explicado que o *timetabling* é um problema que se fosse resolvido por uma busca exaustiva, levaria muito tempo dependendo do tamanho da instância.

Também é mostrado como o algoritmo genético representou o problema. O autor do artigo simplificou o problema para quatro tipos de variáveis, que são os dias da semana,

a quantidade de horários vagos, o conjunto de cursos e os estudantes. Então é dito que uma lista de tuplas irá definir o problema, que é um dia e um slot de horário para cada aula.

A função objetivo de cada indivíduo da população é calculada de acordo com: o número de choques de horários vezes uma constante alta somada ao número de restrições não respeitadas que multiplica uma constante também com um valor muito alto. Então quando um indivíduo apresentar o número zero de choques de horários significa que esta solução poderá ser usada.

Neste capítulo foram apresentados os principais conceitos nos quais este trabalho de conclusão de curso foi embasado, tais como os problemas de alocação de horários, coloração de grafos e algumas soluções metaheurísticas. No próximo capítulo, será apresentada a metodologia, as ferramentas e técnicas utilizadas para abordar o problema.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentados os materiais e métodos utilizados para o desenvolvimento desse projeto, tais como as bibliotecas de Python, a manipulação dos arquivos de entrada e saída (tabelas XLSX), a transformação do problema de alocação de horários de aula em um grafo e sua modelagem como o problema de coloração de grafos, os formatos de entrada e saída dos dados e, por fim, como foi realizada a análise dinâmica da execução do programa.

3.1 Linguagens e Bibliotecas

A implementação dos algoritmos para a resolução do problema de coloração de grafos e consequentemente do *timetabling* foi realizada na linguagem de programação Python. A escolha desta linguagem é pela sua rápida prototipação, facilidades nativas para manipulação de dados e arquivos, grande disponibilidade de bibliotecas gratuitas e ferramentas para a depuração do código e análise dinâmica do programa.

3.1.1 Python

O Python foi criado em 1989 por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (PILGRIM, 2008). O projeto foi baseado na linguagem ABC, com algumas partes da sintaxe derivadas de C e algumas funcionalidades inspiradas em outras linguagens.

3.1.2 PyExcel

Para o desenvolvimento deste trabalho foi utilizado a biblioteca PyExcel, para coletar os ados presente no arquivo de entrada e carregá-los na respectiva estrutura de dados. Utilizando esta biblioteca temos a opção de editar um arquivo XLSX, CSV, ODS, XLS, XLSM ou somente ler os dados contidos nele.

Para a instalação desta biblioteca basta utilizar o seguinte comando no console do interpretador de comandos (BASH):

```
pip install pyexcel-xlsx
```

Para utilizar os recursos de leitura dos dados de uma tabela XLSX, é necessário importar no código uma função da biblioteca com a seguinte instrução em Python:

```
>>> from pyexcel_xlsx import get_data
```

A estrutura de dados do arquivo é obtida pelo seguinte comando:

```
>>> data = get_data("filename.xlsx")
```

Os recursos de escrita dos dados em tabela XLSX, exigem a importação no código de uma biblioteca em Python com a seguinte instrução:

```
>>> from pyexcel_xlsx import save_data
```

Para salvar a estrutura de dados no arquivo, basta utilizar a instrução:

```
>>> save_data("filename.xlsx",data)
```

Observe que a variável *data* é um dicionário do tipo *OrderedDict*, que pode ser obtido através da instrução (em Python):

```
>>> from collections import OrderedDict
```

E ser atualizado com a instrução:

```
>>> data.update({sheet1:matriz})
```

3.1.3 Pickle

A biblioteca *Pickle* implementa um algoritmo para serializar e desserializar objetos em Python. Esta biblioteca funciona da seguinte maneira, o objeto é convertido em bytes (processo de serialização) e depois o objeto pode sofrer a operação inversa (processo de "desserialização"). Existe também uma biblioteca com o nome de *cPickle*, que desempenha a mesma função mas é completamente escrita na linguagem C e com um desempenho mais rápido que a biblioteca nativa *Copy*.

A biblioteca *cPickle* foi utilizada na construção do trabalho com a finalidade de criar uma cópia da estrutura de dados na hora do coloração do grafo e também na geração de novos vizinhos. Para se utilizar esta biblioteca terá que primeiramente importar-lá com a seguinte instrução (em Python):

```
>>> import_pickle as cPickle
```

Para fazer a cópia do objeto basta utilizar a instrução:

```
>>> copia = cPickle.loads(cPickle.dumps(objeto))
```

Através de testes observamos que a utilização desta biblioteca é mais eficiente do que a utilização da biblioteca *Copy* e sua função *deepcopy* pois, com a utilização da biblioteca *cPickle* houve uma melhora de 90% em relação ao tempo gasto na cópia de objetos.

3.1.4 cProfile

Segundo Foundation (2018) o cProfile provê análise dinâmica da execução de um programa (*profiling*) na linguagem de programação Python. Um profile é um conjunto de de estatísticas que descrevem quantas vezes ou quanto tempo o seu programa demora para escutar as partes de seu código.

Para a realização do *profiling* em Python, é necessário importar a biblioteca cProfile com a instrução:

```
>>> import cProfile
```

Para analisar a execução do programa, é necessário que na função principal de seu código (*main*) seja utilizado a seguinte instrução (em Python):

```
>>> cProfile.run(statement='run()', filename='saida.cprof')
```

No comando acima, *statement* está relacionado ao bloco de código que você quer que o cProfile faça a análise, já *filename* é o arquivo de saída que terá o conjunto de estatísticas coletadas.

3.1.5 Snake Viz

Para a visualização do arquivo gerado pelo cProfile, é necessário que se instale o SnakeViz que é um Browser para a visualização gráfica do arquivo de saída do cProfile. Para a instalação no Linux, basta entrar no console do interpretador de comandos (shell) e digitar o seguinte comando:

```
pip3 install snakeviz
```

Para a visualização do arquivo de saída, é necessário definir o diretório de trabalho atual para aquele onde está localizado o arquivo gerado pelo cProfile e digitar o comando:

```
snakeviz filename.cprof
```

No comando acima, *filename* é o nome do arquivo previamente gerado com o cProfile. Logo após será aberto em um navegador Web uma página com a visualização das informações de *profiling*.

3.2 Pesquisa Bibliométrica

Referência bibliométrica, segundo Chueke e Amatucci (2015), consiste em aplicar métodos estatísticos e matemáticos para análise de obras literárias. Após obtenção de listagem de materiais nas bases de dados da CAPES, para selecionar os melhores textos foram criados scripts nas linguagens Python e Shell, conforme será explicado a seguir.

Utilizando o acesso à rede do campus Formiga do IFMG, é possível realizar pesquisas no Portal de Periódicos[1] da CAPES em bases de periódicos e, então, fazer o download de planilha com os dados bibliográficos de artigos publicados por organizações como IEEE, ACM, Springer e Elsevier. As planilhas obtidas continham milhares de entradas bibliográficas, correspondendo a materiais sobre o(s) tema(s) pesquisado(s) e suas respectivas informações: DOI (*Digital Object Identifier*[2]), título do artigo, autor(es), ano de publicação e, no caso de algumas bases de periódicos, a quantidade de citações.

O script Python desenvolvido lê as informações contidas na planilha em uma estrutura de dados, filtra os autores que mais publicaram sobre aquele assunto específico e os artigos com maior número de citações e, então, escreve como saída, em um arquivo texto, os respectivos dados dos artigos filtrados, com uma entrada por linha do arquivo texto. Logo após, o Shell script seleciona no arquivo texto os DOIs das entradas bibliográficas, procura no Google Acadêmico[3] quais deles contém mais citações (nem toda base de periódicos disponibiliza esta informação). Por fim, seleciona os artigos[4] com maior número de citações.

Esse método é bastante importante para que os textos utilizados para a construção do trabalho sejam de alta relevância, bem como de autores influentes no assunto pesquisado.

3.3 Modelagem do Problema

Considerando o problema da alocação de horários, é possível transformá-lo no referido problema da coloração de grafos. Para realizar tal transformação, a estrutura de dados utilizada para armazenar as aulas, professores e horários, deverá ser modelada como um grafo.

[1] http://www.periodicos.capes.gov.br
[2] https://www.doi.org/publications.html
[3] https://scholar.google.com.br
[4] por default, a quantidade de artigos a serem selecionados foi configurada como 10 (dez)

3.3.1 Transformação do Problema

O problema de coloração de grafos pode ser utilizado para a resolução do *timetabling*, já que se os vértices adjacentes não podem conter a mesma cor, isso implica que as aulas que contém o mesmo professor ou a mesma turma ou ambos, não poderão ser alocadas em um mesmo horário. Assim, pode ser interpretado de maneira que cada cor utilizada será um horário de aula, (por exemplo: cor 1 seria "segunda-feira às 7:00"). Com a solução para o problema da coloração de grafo, com o grafo devidamente colorido tem-se um resultado válido e mapeável para o problema original da alocação de horários.

Cada vértice que compõe o grafo deverá ter algumas informações, a saber: o nome do professor, a turma para qual ele estaria lecionando em um dado momento, a cor do vértice e o número que aquela aula representa na semana, de acordo com a quantidade de aulas semanais. Ou seja, se um professor deve ministrar aula 3 vezes por semana para uma mesma turma, então seriam a 1ª, 2ª e 3ª aula.

As arestas ligarão todos os vértices que carregam algum tipo de informação semelhante. Por exemplo, todos os vértices que são de uma determinada turma T, estarão ligados entre si, todos os vértices de um certo professor P também estarão ligados entre si.

A Tabela 1 abaixo, será utilizada para exemplificar como transformar os dados de entrada do problema em um grafo.

Tabela 1 – Tabela de horários fictícios.

Professor	**Turma A**	**Turma B**	**Turma C**
Josiane (J)	2	1	0
Leonardo (L)	0	1	1
Reginaldo (R)	1	1	0
Wagner (W)	1	0	1

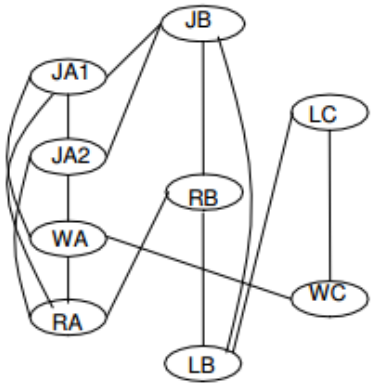
Fonte: (PEREIRA; NETTO; LACRUZ, 2007)

Observe que a professora Josiane (J) leciona duas aulas para a turma A, uma aula para a turma B e nenhuma aula para a turma C. As demais linhas devem ser interpretadas de maneira semelhante. Assim, de posse de tais dados é possível montar o grafo apresentado na Figura 1.

Quando a instância de dados for bem maior do que a representada acima, naturalmente o grafo gerado por ela também será bem maior também do que o representado na Figura 3. Entretanto, com esse tipo de estrutura de dados é possível visualizar os relacionamentos e melhor organizar os dados, bem como aplicar algoritmos específicos para tratamento de grafos e para sua coloração.

Como foi dito anteriormente, os vértices contém três tipos de informação básica, o professor, a turma e qual aula será lecionada e as arestas conectam todos os vértices que carregam o mesmo professor ou a mesma turma ou ambos.

Figura 3 – Grafo gerado a partir da instância de dados da tabela.



Fonte: (PEREIRA; NETTO; LACRUZ, 2007)

3.3.2 Representação da Solução

A estrutura utilizada para a representação da solução do problema consiste em um grafo não dirigido. O vértices são representados por um objeto que contém as seguintes informações:

- Id do Vértice;
- Nome do Professor;
- Matéria que será ministrada;
- Turma que irá assistir a aula;
- Cor do Vértice.

As arestas são representadas por um objeto que contém as informações:

- Id da Aresta;
- Vértice de Origem;
- Vértice de Destino.

A classe que representa o grafo contém duas listas, uma lista de vértices e outra lista de arestas. Essa classe contém algumas funções gerais para se trabalhar com esta estrutura de dados como, saber se existe um vértice ou aresta com um ID específico, pegar o grau (quantidade de arestas que saem ou entram no vértice) de um determinado vértice, retornar todos os vértices adjacentes, dentre outras. Também existem algumas funções específicas para o problema de coloração de grafos como, verificar se algum dos vértices

adjacentes tem uma coloração de um vértice específico, fazer alterações nas cores dos vértices.

As informações fornecidas pelo usuário no arquivo de entrada são armazenados na estrutura de dados através de uma função da classe do grafo. Esta função primeiramente vai pegando as informações de cada célula do arquivo de entrada e vai criando os vértices. O número de vértices criados é o mesmo da quantidade de aulas ou seja, se um professor ministra cinco aulas para a turma, será criado cinco vértices com esta informação, sendo eles diferenciados somente pelo ID.

Logo após todos os vértices terem sido criados, uma função irá criar as arestas do grafo. A condição existente para os vértices terem uma ligação entre si são: eles precisam ter alguma informação em comum, ou seja, vértices que contém o mesmo nome de professor ou a mesma turma, terão uma aresta que os ligam. Esta condição está presente pois com esta ligação, quando o grafo estiver colorido, os vértices que contém os mesmos professores e as mesmas turmas nunca irão estar ministrando ou tendo aula no mesmo horário, assim já respeitando algumas restrições fortes do problema.

3.3.3 Formato de entrada e saída

A entrada de dados do sistema, consiste em um arquivo ".xlsx" que contém um padrão único. Este arquivo está subdividido em cinco abas diferentes.

Na primeira aba com o nome de "Dados" será onde o usuário do sistema irá inserir os horários que irão ser alocados. Nesta aba existem quatro colunas, "Matéria", "Turma", "Professor" e "Quantidade de Aulas".

Na segunda aba com o nome de "Restricoes Professores" será onde o usuário do sistema irá inserir as restrições dos professores presentes na aba de "Dados". Está aba contém três colunas que são elas "Professor", "Restrição de Horário" e "Dia da Semana". Na terceira aba com o nome de "Configuracoes" será onde o usuário irá inserir qual é o horário de início das aulas, sendo assim pode-se obter também quantas aulas serão ministradas diariamente. Esta aba contém somente uma coluna que é "Horário de início de aulas a cada dia". Na quarta aba com o nome de "Preferências" será onde o usuário do sistema irá inserir os horários em que os professores preferem ministrar as suas aulas. Com essas informações o algoritmo irá tentar sempre gerar uma solução que atende a maioria das preferências.

A última aba da entrada de dados "Restricoes Turmas" é referente as restrições das turmas inseridas na aba "Dados". Esta aba segue o mesmo padrão da aba "Restricoes Professores" contendo três colunas com o nome de "Turma", "Horário da Restrição" e "Dia da Semana".

A saída de dados do sistema é um arquivo ".xlsx". Este arquivo está subdividido

Figura 4 – Exemplo de entrada de dados

	A	B	C	D
1	Matéria:	Turma:	Professor:	Quantidade de Aulas:
2	a	1	1	2
3	b	1	2	3
4	c	1	3	2
5	d	1	4	3
6	e	1	5	5
7	f	1	6	1
8	g	1	7	5
9	h	1	8	3
10	i	1	9	2
11	j	1	10	1
12	k	1	11	1
13	a	2	1	2
14	b	2	2	3
15	c	2	3	2
16	d	2	4	3
17	e	2	5	5
18	f	2	6	1
19	g	2	7	5
20	h	2	8	3
21	i	2	9	2
22	j	2	10	1
23	k	2	11	1
24	a	3	12	5
25	b	3	4	3
26	c	3	1	2
27	d	3	2	3

Fonte: Próprio autor

Figura 5 – Exemplo de configuração de horários

	A
1	Horários de Inicio de aulas a cada dia.
2	07:00
3	07:50
4	08:40
5	09:50
6	10:40
7	11:30

Fonte: Próprio autor

em abas, onde cada aba é o horário semanal de cada turma que foi especificada na entrada de dados.

Para gerar a saída de dados foi utilizado a função "*save_data("filename.xlsx",data);*" da biblioteca PyExcel, onde o os dados que esta função recebe por parâmetro é um dicionário onde cada uma de suas chaves contém uma matriz.

Cada chave do dicionário de saída está relacionado com uma turma diferente e cada matriz do dicionário está o horário semanal respectivamente. O formato do arquivo de saída é com cada turma contendo sua aba separada e para cada aba, a primeira coluna contém os horários das aulas e na primeira linha os dias da semana. Em cada célula do horário está o professor que irá ministrar aula no respectivo horário com o seguinte formato

"Professor(Matéria)".

A escolha do formato de arquivo ser ".xlsx" é que as pessoas que irão utilizar o sistema poderão fazer alterações manuais caso necessário, para que o resultado obtido pelo algoritmo seja mais agradável para a utilização do horário.

3.4 Análise dinâmica da execução do programa

O *profiling* é uma forma de análise dinâmica de programa que mensura a complexidade do programa, seja em espaço (memória) ou tempo (de execução). Por tanto levamos em consideração certas instruções de código ou a frequência e duração de chamadas de funções. Tipicamente, o *profiling* é utilizado com finalidade de identificar partes do programa que carecem de otimização. Durante todo o desenvolvimento deste trabalho, foram realizadas análises de *profiling* para identificar possíveis gargalos no programa. Os principais pontos identificados através da análise dinâmica da execução do programa e as melhorias a partir dela realizadas serão apresentados no Capítulo 5 (RESULTADOS E ANÁLISE).

3.5 Projeto fatorial 2^k

O projeto fatorial 2^k consiste em conduzir experimentos para que se possa identificar qual a melhor configuração entre duas opções para cada fator do algoritmo (2 níveis para k parâmetros de configuração), a fim de calibrá-lo para melhor atuar em determinado problema. Segundo Brandão, Moro e Almeida (2013), a análise realizada com o projeto fatorial mostra que diferentes fatores impactam no resultado das métricas de diferentes formas. Tal avaliação mostra o potencial de utilizar o projeto fatorial na avaliação de sistemas computacionais dependentes da definição de parâmetros de entrada. Foram conduzidas várias execuções do projeto fatorial 2^k, visando calibrar o algoritmo de maneira a gerar soluções com alta qualidade reduzindo o intervalo de tempo. A proposta de calibração dos parâmetros da metaheurística, fruto da realização do projeto fatorial 2^k, será apresentada no Capítulo 5 (RESULTADOS E ANÁLISE).

Neste capítulo foi apresenta a metodologia utilizada no desenvolvimento desse projeto, especialmente no que tange à transformação do problema de alocação de horários de aula em um grafo e sua modelagem como o problema de coloração de grafos.

4 PROJETO E DESENVOLVIMENTO

Este capítulo apresentará todo o processo de planejamento e implementação da ferramenta para alocação de horários escolares. Em função da proposta de adoção da modelagem do problema como coloração de grafos, foi definida a metaheurística a ser utilizada na resolução do problema de coloração de grafos, a estratégia de geração da estrutura de vizinhança válida e exploração do espaço de soluções, bem como a definição da função objetivo e pesos das suas penalidades. Por fim, com a definição da melhor solução que atenda às restrições impostas, é realizado seu mapeamento para o problema original de alocação de horários de aula. Para garantir um bom custo-benefício da solução desenvolvida, também será apresentado neste capítulo o projeto fatorial 2^k conduzido para calibração dos parâmetros da metaheurística, visando obter a melhor solução possível no menor tempo disponível.

4.1 Metaheurística para resolução do problema

Inicialmente foram implementados duas classes para a resolução do problema, uma das classes contava somente com a Metaheurística GRASP para resolver o problema da coloração de grafos e atender todas as restrições fortes do problema, já a outra continha o GRASP e a Metaheurística VND para auxiliar na construção de novos vizinhos. O GRASP e o VND necessitam de alguns parâmetros para a execução do algoritmo. Os parâmetros são, número de execuções, número de vizinhos que serão gerados em cada busca local, o tamanho da lista de candidatos restrita presente na geração da solução inicial e a quantidade de estrutura de vizinhança que será utilizada pelo VND.

Para a resolução do problema, o GRASP gera a solução inicial e logo após é feito uma busca local a fim de gerar uma solução vizinha melhor que a inicial. Após a geração do vizinho, uma função irá avaliar qual função objetivo é menor, já que o problema consiste em minimizar as cores da coloração do grafo.

A metaheurística VND está contida dentro da busca local para o auxílio da geração de vizinhos. Depois que o algoritmo executa a quantidade de vezes determinada por um parâmetro de entrada, é gerada uma saída com o mesmo formato do arquivo de entrada, contendo os dados da melhor solução gerada pelo GRASP.

Para os testes iniciais de desempenho, foram padronizados alguns parâmetros de entrada:

- Número de execuções: 10;

- Quantidade de vizinhos: 20;
- Tamanho da lista de candidatos restrita: 30%;
- Quantidade de Estruturas de vizinhanças: 2.

A instância para teste iniciais foi metade dos dados de uma instância completa presente nos resultados iniciais. A utilização de somente metade desta instância se deve a necessidade de respostas rápidas nos testes iniciais. Não foi considerada a utilização de uma instância artificial pois queríamos lidar com uma demanda real de dados.

4.2 Construção da solução inicial

Na construção da solução inicial, o algoritmo avalia quais vértices são melhores candidatos para serem inseridos na solução e inicialmente coloridos. A forma de avaliação desses vértices é de acordo com o seu grau, ou seja, quanto mais arestas incidentes em um vértice, ele terá uma maior chance de entrar na lista de candidatos restrita. Após a construção de tal lista com os vértices mais "interessantes" para compor a solução naquele momento, uma função estocástica determina qual dos vértices da lista será escolhido. Este processo é repetido até todos os vértices do grafo terem sido efetivamente coloridos.

Com o intuito de tentar amenizar a ocorrência do não atendimento das restrições fracas foi adicionado ao algoritmo o critério de Boltzmann. Sempre em que um vértice for escolhido para ser adicionado na solução, é feito um delta entre a função objetivo antes do vértice escolhido ser colocado na solução e após. Caso o delta for negativo, ou seja, caso a função objetivo fique pior com a adição deste vértice, o critério de Boltzmann é ativado para "decidir" se este vértice irá continuar na solução ou irá sair.

O critério de Boltzmann original utiliza como uma de suas variáveis a temperatura da solução, mas já que este algoritmo não aborda esse conceito, esta variável foi substituída pela quantidade de vértices que ainda faltam para serem colocados na solução (coloridos).Esta decisão foi tomada pois, como esta lista diminui a cada iteração, é possível ter uma mesma ideia da temperatura que decai a cada iteração. E com isso, o conceito básico do critério de Boltzmann se mantém diante dessa adaptação.

4.3 Busca local no espaço de soluções

A busca local é baseada na metaheurística *Hill Climbing*, em que o espaço de busca de uma solução é explorado até que não se encontre uma solução melhor que a atual. A quantidade de vizinhos que serão gerados para avaliar se sua função objetivo é melhor que a solução atual é determinada a partir de um parâmetro de entrada. Após ser gerada tal

quantidade de soluções vizinhas, verifica-se as mesmas para identificar se houve alguma melhoria em relação à solução atual.

O VND compõe a função de busca local e trabalha com diferentes estruturas de vizinhanças. Caso um vizinho que foi gerado não tenha uma função objetivo melhor do que a solução fornecida, a estrutura de vizinhança é trocada, ou seja, o VND altera a maneira de gerar um vizinho para que haja um nova chance de melhorar a solução fornecida.

A diferença entre as estruturas de vizinhanças utilizadas no algoritmo é somente a quantidade de vértices que serão alterados. Após o VND explorar todas as estruturas de vizinhanças, é retornado uma solução vizinha com grandes chances de ter uma função objetivo melhor que a solução fornecida.

4.4 Geração da estrutura de vizinhança

A geração de uma nova solução para otimizar o resultado do algoritmo é denominado "vizinho" pois as modificações feitas na solução corrente (para obter algum ganho) são operações "leves" e que não afetam a solução por completo, apenas alguns elementos presentes na mesma. Existem inúmeras estruturas de vizinhanças, mas quando são bem escolhidas, fazem com que a convergência da heurística seja mais rápida, evitando com que a solução fique presa em um ótimo local.

Uma estrutura de vizinhança bem conhecida no problema de coloração de grafos é, sortear dois vértices aleatoriamente e trocar as suas cores, mas, com esse tipo de estrutura de vizinhança, o grafo sempre permanecerá com a quantidade máxima de cores igual a solução corrente. Com essa observação feita, foi optado por desenvolver outra estrutura de vizinhança.

A estrutura de vizinhança desenvolvida consiste em sempre tentar minimizar a quantidade de cores presente no grafo, a maneira que é feita essa minimização é fazendo uma operação de aumentar uma cor de um vértice que inicialmente tinha uma cor baixa e depois pegar essa cor antiga e atribuir a outro vértice.

Primeiramente os vértices que compõem o grafo são ordenados pela sua cor para que o algoritmo dê preferência em escolher os vértices de grau maior. Um vértice é inicialmente sorteado da lista de vértices. Na sequência, uma função retorna uma lista com todos os vértices adjacentes àquele sorteado. Dela, é sorteado um dos vértices adjacentes, compondo assim um par de vértices cujas cores serão modificadas para gerar a solução vizinha.

Após os dois vértices escolhidos, é verificado qual destes vértices tem a maior cor para que ele, no final do processo, receba a cor do outro vértice. Sabendo qual dos dois vértices tem a menor cor, é iniciado um processo de aumentar a sua cor, uma unidade de cada vez, até que a sua coloração esteja correta em relação aos vértices adjacentes a ele,

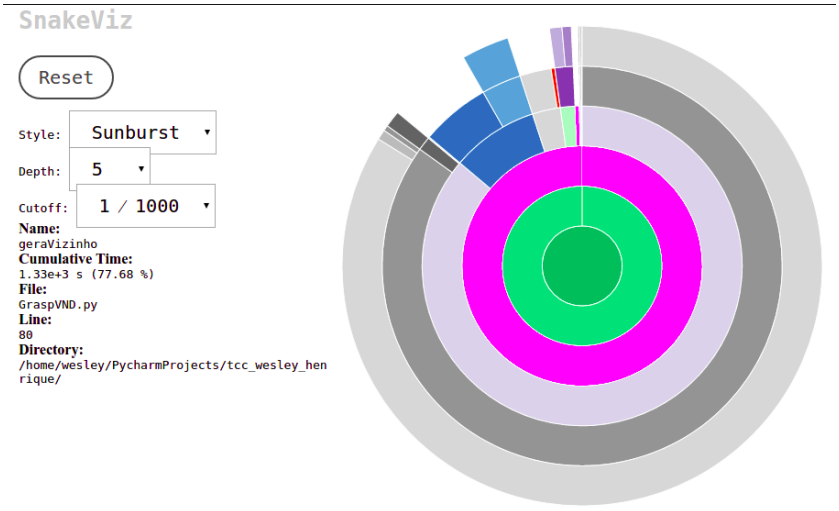
ou seja, que a nova cor proposta não exista em sua adjacência.

Após a solução tornar-se factível, a cor inicial do vértice que foi alterado, é atribuída ao outro vértice que tinha a maior cor e então é verificado novamente a factibilidade da solução. Caso a solução esteja infactível, o processo é reiniciado e um contador é incrementado para que se conte o número de tentativas de geração de um novo vizinho. Se o contador chegar em dez tentativas, o processo é finalizado e a solução que foi recebida por parâmetro é retornada.

4.4.1 Impacto da escolha do vértice inicial

Inicialmente o vértice escolhido era de maneira aleatória mas, depois, resolvemos alterar a forma de escolha do vértices baseado em sua coloração, ou seja, vértices com cores de numeração maior terão maiores chances de serem escolhidos pelo algoritmo. Uma explicação mais detalhado será oferecida na próxima subseção. Analisando o *profiling* do algoritmo antes e depois dessa alteração, pode ser observado que melhorias foram obtidas em relação ao tempo gasto pelo algoritmo para gerar a solução.

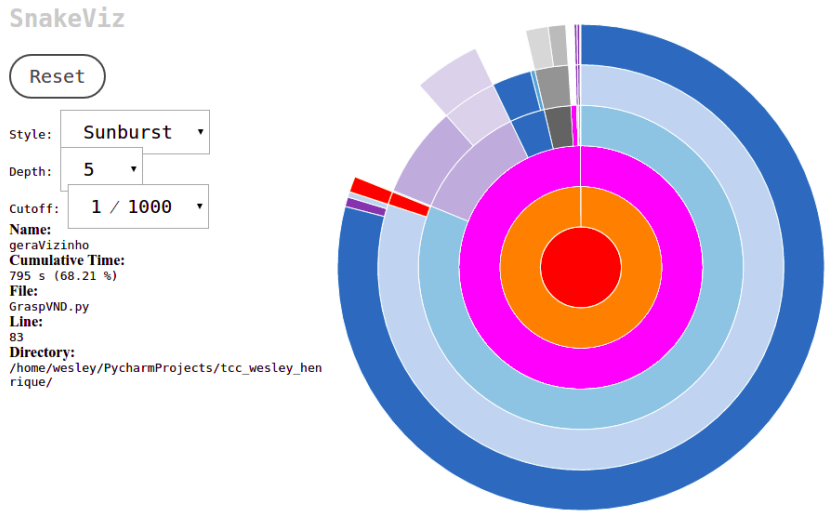
Figura 6 – Análise dinâmica da antiga estrutura de vizinhança.



Fonte: Próprio autor

Como pode ser observado na Figura 6, o tempo que é gasto para executar a função de geração de um novo vizinho equivale a 77.68% do tempo total do algoritmo (cerca de 17 minutos do tempo de execução total de 22 min). Baseado nesta observação, a nova estrutura de vizinhança explicada anteriormente foi implementada. Então, realizamos nova execução do algoritmo com os mesmos parâmetros de configuração e com a mesma instância de entrada de dados. O resultado obtido com tal modificação na estrutura de vizinhança pode ser observado na Figura 7, discutido a seguir.

Figura 7 – Análise dinâmica da nova estrutura de vizinhança.



Fonte: Próprio autor

Como pode ser observado na Figura 7, a parcela do tempo de execução gasto pela nova estrutura de vizinhança diminuiu de 77,68% para 68,21% (cerca de 9 minutos do tempo de execução total de 13 min). Assim, ao atentar para o custo computacional desta pequena mas pesada etapa do algoritmo, reduzimos o tempo de execução total de 22 min. para 13 min, ou seja, uma economia relativa de 40,9% de tempo em relação à estrutura de vizinhança anterior.

4.4.2 Roleta para escolha do vértice inicial

A nova forma de escolha do vértice inicial funciona da seguinte maneira: ordenar todos os vértices pelo número de sua cor, posicioná-los em uma fila circular, calcular a soma total dos valores correspondentes às cores de cada vértice. No exemplo apresentado na Figura 8, a soma dos números correspondentes a cada cor será 21.

Após o somatório ser calculado, um número aleatório será sorteado dentre valores de 1 até o valor do somatório. Tal valor indicará a posição da cor sorteada em uma "roleta estocástica", conforme ilustra a Figura 8. Um cálculo é feito para indicar em que ponto da soma cumulativa tal número sorteado incide, correspondendo à posição de determinado vértice que contribuiu com o último valor adicionado ao somatório que faça com que o valor daquela soma seja maior ou igual ao valor sorteado na roleta.

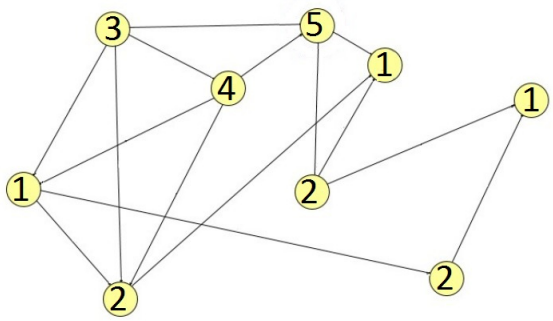
Suponha que o valor 13 tenha sido sorteado. Conforme já definido, os vértices são ordenados de acordo com o número correspondente à sua cor. Sendo assim, 13 é maior que 5, maior que 9 (5+4) e maior que 12 (5+4+3), porém é menor que 14 (5+4+3+2). Assim, o vértice que contribuiu com a primeira das cores de número 2 corresponde ao vértice escolhido. Por outro lado, se o valor sorteado fosse 4, neste mesmo exemplo o primeiro

Figura 8 – Roleta estocástica para o sorteio de uma cor.

Fonte: Próprio autor

vértice seria escolhido, uma vez que 4 é menor que 5, o valor da primeira soma.

Figura 9 – Solução que será base para criação de um vizinho.



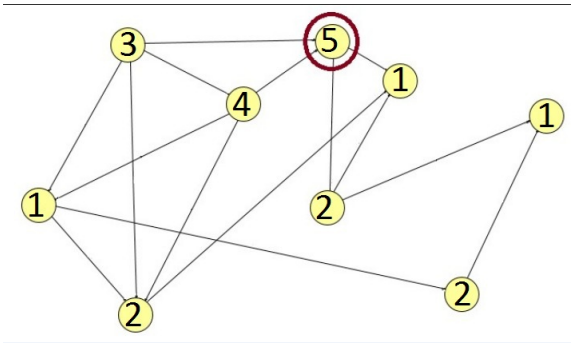
Fonte: Próprio autor

A Figura 9 oferece uma simulação do funcionamento da estrutura de vizinhança com esta "nova" maneira de escolha do vértice inicial. O número que está disposto dentro de cada vértice representa sua coloração. Inicialmente temos um grafo cujo número cromático é 5. Conforme visto, o somatório dos números correspondentes às cores tem valor 21. Vamos novamente supor que o valor sorteado seja 4 e, conforme anteriormente já explicado, o vértice escolhido corresponde ao vértice que possui a cor de número 5, conforme destacado na Figura 10.

Após a escolha do vértice inicial, é feita a escolha aleatória de um dos vértices adjacentes a ele, com igual probabilidade. Vamos supor que o vértice adjacente sorteado seja o vértice com coloração correspondente ao valor 1, conforme ilustrado na Figura 11.

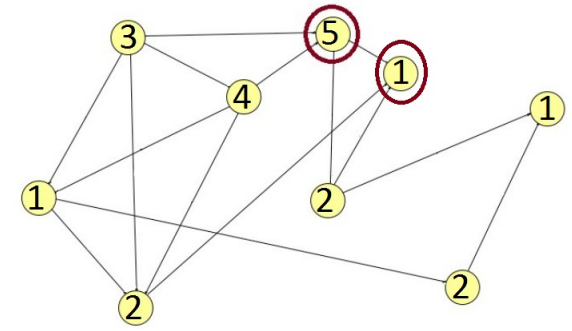
Com a escolha do vizinho, é então verificado qual destes dois vértices contém

Figura 10 – Vértice escolhido pela estrutura de vizinhança.



Fonte: Próprio autor

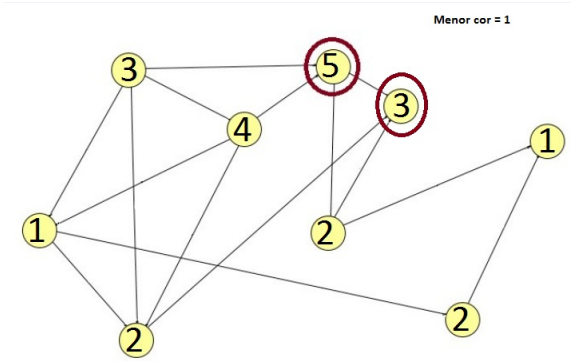
Figura 11 – Vizinho do vértice escolhido aleatoriamente.



Fonte: Próprio autor

uma menor coloração que, no caso, é o vértice com a coloração 1. Então um processo de incremento da coloração desse vértice é iniciado, sempre checando a viabilidade da nova coloração proposta. Conforme ilustra a Figura 12, o processo irá terminar quando o vértice chegar à coloração 3 pois nenhum dos vértices adjacentes tem esta coloração.

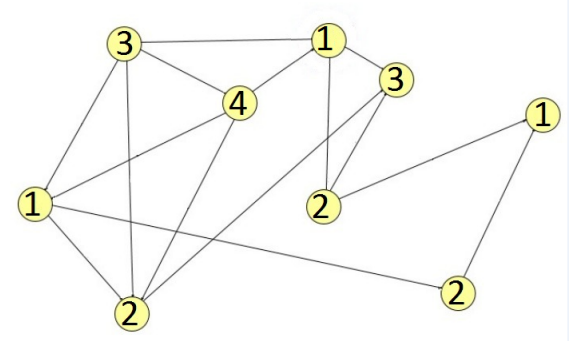
Figura 12 – Vértice com coloração incrementada.



Fonte: Próprio autor

Então com o vértice já colorido com a nova cor proposta (sem conflitos com nenhum dos vértices adjacentes), a cor antiga do vértice é atribuída àquele outro vértice, o que não sofreu nenhuma alteração. A nova estrutura de vizinhança gerada por este mecanismo é apresentada na Figura 13.

Figura 13 – Nova solução vizinha.



Fonte: Próprio autor

Como pode ser observado, comparando a solução inicial (Figura 9) com sua solução vizinha (Figura 13), foi possível diminuir a quantidade de cores necessárias na solução. A solução inicial tinha uma coloração de cinco cores e, depois do processo, a solução vizinha demandava somente quatro cores.

4.5 Função Objetivo

A função objetivo consiste em qual é a métrica que é utilizada para medir a qualidade de uma soluçao, além de identificar se ela atende ou não as restrições fracas (recomendação, preferência) que foram propostas no trabalho. Para cada restrição fraca não atendida, a função objetivo ganha como penalidade o incremento de um determinado valor, de acordo com o peso atribuído a cada restrição fraca.

Visando ilustrar a ordem de importância das restrições fracas, apresentamos abaixo os pesos utilizados na fase inicial deste trabalho:

- Peso 1: Não atendimento a **preferência** de horário de determinado professor;
- Peso 2: Existência de **lacuna** (horário vago) entre duas aulas;
- Peso 3: Existência de aula **interposta** entre duas aulas de determinada matéria (não geminação);
- Peso 4: Existência de três ou mais aulas **subsequentes** de uma mesma matéria (poligeminação).

A Equação 4.1 apresenta o somatório das penalidades realizado na função objetivo, onde cada infração ao não atendimento de uma restrição fraca (acima listadas) é multiplicada pelo seu respectivo peso. Existe também uma função que avalia o número cromático das soluções quando se é gerado uma nova solução vizinha. Tal função é chamada dentro do algoritmo VND.

$$\sum_{i=1}^{horários} 1*\textbf{Preferência}(i)+2*\textbf{Lacuna}(i)+3*\textbf{Interposta}(i)+4*\textbf{Poligeminada}(i) \quad (4.1)$$

Vale ressaltar que, conforme será apresentado na Subseção 5.2.1, uma configuração diferente de pesos será adotada *a posteriori*, na qual os pesos das restrições mais críticas foram inflacionados.

4.6 Mapeamento da solução para o problema original

O término de execução do algoritmo proposto gera um arquivo com a melhor alocação de horários encontrada. Na Figura 14, observe que esse arquivo possui a mesma extensão (planilha XLSX) que o arquivo de entrada, formato adotada para que, caso o usuário do sistema queira fazer alguma alteração manual, possa fazê-lo diretamente na interface gráfica do programa de planilhas de sua preferência.

Figura 14 – Exemplo de saída de dados.

	A	B	C	D	E	F	G	H
1		Segunda	Terça	Quarta	Quinta	Sexta	Sábado	Domingo
2	07:00:00	12 (a)	1 (c)	2 (d)	3 (g)	12 (a)		
3	07:50:00	8 (f)	1 (c)	7 (e)	12 (a)			
4	08:40:00	7 (e)	10 (h)	7 (e)	8 (f)	28 (i)		
5	09:50:00	4 (b)	7 (e)	8 (f)	12 (a)			
6	10:40:00	2 (d)	11 (k)	3 (g)	12 (a)	4 (b)		
7	11:30:00	2 (d)	6 (j)	7 (e)	4 (b)			

Fonte: Próprio autor

Para a criação desta saída de dados também foi utilizada a biblioteca *PyExcel*. Na planilha contendo a solução proposta, cada aba corresponde a uma turma que estava presente no arquivo de entrada. Vale ainda notar que os horários em que estão dispostas as aulas correspondem somente àquelas opções de horários fornecidas no arquivo de entrada.

Para geração da saída, inicialmente o algoritmo insere cada uma das turmas em uma lista, para saber quantas abas serão criadas. Então, para cada turma uma matriz de horários será criada com oito colunas (uma coluna para representar os horários e as outras sete para os dias da semana) e N linhas, onde N é a quantidade máxima de aulas em determinado dia mais o cabeçalho dos nomes dos dias da semana. Após isso, a matriz é preenchida com os dados presente na estrutura de dados.

4.7 Análise dinâmica da execução (*Profiling*)

A seguir, serão apresentados os principais gargalos do programa identificados através da análise dinâmica de sua execução, bem como as melhorias de desempenho obtidas ao modificá-los.

4.7.1 Economia de chamadas a funções

Todas as classes presentes no código do algoritmo, inicialmente programado de maneira orientado a objetos, estavam devidamente encapsuladas com funções Get e Set. Após testes iniciais de desempenho, foi percebido que o tempo de execução do algoritmo estava demasiado alto, de maneira que inviabilizaria o ciclo de modificações e testes, utilizado no desenvolvimento deste projeto. Com a percepção Dada a necessidade de otimizar o tempo de execução do algoritmo, uma das alternativas foi retirar o encapsulamento utilizado nas classes, economizando chamadas a funções. Assim, apenas com a retirada das funções *Get e Set* (permitindo acesso direto às variáveis de cada classe), o tempo de execução do algoritmo foi diminuído pela metade. Tal economia viabilizou que diferentes abordagens fosse exploradas, testadas e ajustadas durante o desenvolvimento deste projeto.

4.7.2 Eficiência na clonagem de objetos

A viabilização de melhorias e a subsequente validação do algoritmo, trouxe a necessidade de agilizar o tempo de execução do protótipo. Com a utilização da ferramenta cProfile, foi percebido que uma das funções que estava consumindo mais tempo de utilização era a função *Copy.deepcopy*[1] (da biblioteca *Copy*, no Python), que faz uma cópia recursiva de um determinado objeto.

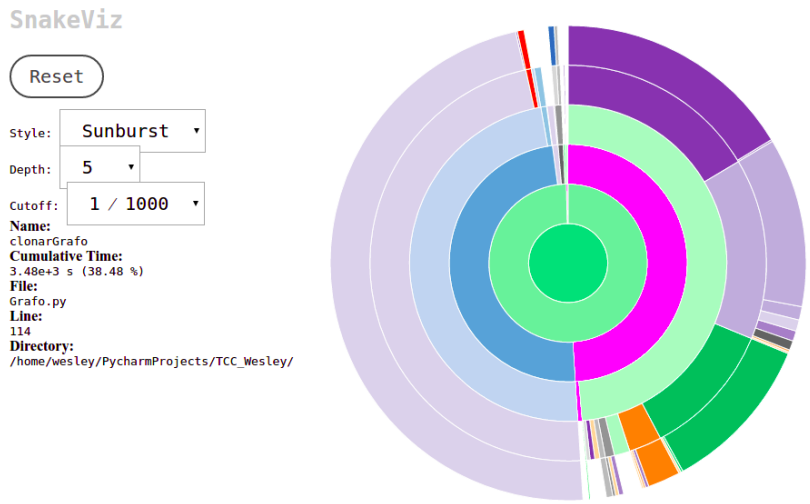
Esta função era utilizada em várias partes do código mas, principalmente, na geração de um novo vizinho. O procedimento de geração de vizinhança, explicado na Seção 4.4, recebe como parâmetro a solução corrente, melhor solução até então encontrada. Tal solução precisa ser clonada pois, com as modificações propostas na geração de um novo vizinho, não poderíamos alterar diretamente a solução corrente e perder seu estado, caso o novo vizinho gerado não apresente melhora em relação à ela.

Conforme pode ser observado na Figura 15, a função *clonarGrafo* (responsável por fazer a cópia completa de um objeto), era responsável por 38,48% do tempo de execução do algoritmo (58 minutos dos 150,7 minutos totais). Após essa análise dinâmica da execução do algoritmo, via *cProfile*, decidimos trocar a biblioteca *Copy* pela biblioteca *Pickle*.

De acordo com a documentação oficial[2] da função *Copy.deepcopy*, uma "cópia profunda" constrói um novo objeto composto e então, recursivamente, insere nele cópias

[1] https://docs.python.org/3/library/copy.html
[2] https://docs.python.org/3/library/copy.html

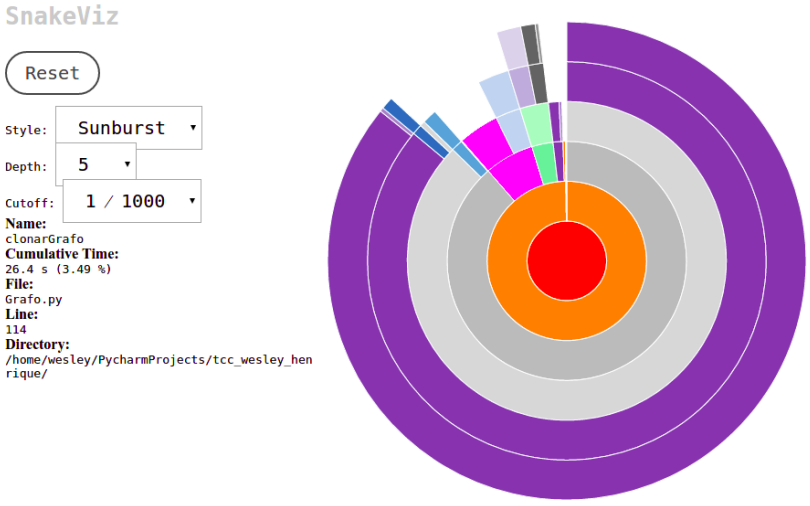
Figura 15 – Análise dinâmica da execução com *deepcopy*.



Fonte: Próprio autor

de todo e cada objeto referenciado no original. Por outro lado, a biblioteca *Pickle* realiza um processo mais simples, onde um objeto Python (hierárquico) é convertido em um fluxo de bytes, conforme documentação oficial[3]. Informalmente interpretado como um *dump* de memória, tal processo é formalmente conhecido como serialização ou achatamento.

Figura 16 – Análise dinâmica da execução sem *deepcopy*.



Fonte: Próprio autor

Conforme pode ser observado na Figura 16, após a substituição do *deepcopy* pelo *Pickle*, a função *clonarGrafo* (responsável por clonar o objeto contendo uma solução) ficou mais eficiente: passou a ser responsável por apenas 3,49% do tempo de execução do algoritmo (26 segundos dos 12,6 minutos totais).

[3] https://docs.python.org/3/library/pickle.html

A função *clonarGrafo* passou então a consumir de 58 minutos para menos de meio minuto (uma melhoria relativa de 99% no tempo gasto por esta função). Observe que tal função é invocada inúmeras vezes ao longo da exploração do espaço de soluções promovida pela heurística. Portanto, o tempo de execução do algoritmo caiu drasticamente: de 2,5 horas para 12,6 minutos (uma melhoria relativa de 92% no tempo de execução do algoritmo).

4.8 Projeto fatorial 2^k

Os parâmetros (calibração) que serão utilizados no projeto fatorial 2^k são:

- **Quantidade de Execuções** que a metaheurística irá executar;
- **Tamanho da LCR** (lista de candidatos restrita), em porcentagem;
- **Vizinhos** que serão explorados na busca local do GRASP;
- **Estruturas de vizinhanças** que o VND irá utilizar, em sua busca por uma solução com menor FO.

As métricas consideradas são:

- Média do **tempo gasto** na execução total do algoritmo;
- Média do **valor da FO** (de cada solução) gerada pelo algoritmo.

Ao todo, foram realizados três projetos fatoriais diferentes, incrementalmente adotando um par de valores limítrofes para cada parâmetro. A cada novo projeto fatorial, um dos valores propostos era mantido e o outro era modificado (ex.: dobrado ou dividido, de maneira similar a uma busca binária).

4.8.1 Primeiro projeto fatorial 2^4

Na primeira etapa do experimento fatorial, o valor previamente configurado para cada um dos quatro fatores foi dobrado ou reduzido pela metade, conforme o caso. A Tabela 2 apresenta os níveis (valores) propostos para os fatores (parâmetros) de calibragem da metaheurística.

Nesta primeira fase, os níveis (1 e 2) de cada fator (A, B, C e D) foram escolhidos de acordo com o desempenho atual algoritmo, na expectativa de melhorar a solução gerada e reduzir o tempo de execução para tal. Como há dois níveis e quatro fatores, foram testadas todas as 16 combinações (2^4) e analisados os resultados obtidos em relação à qualidade da solução e o tempo de execução para que fosse gerada. A Tabela 3 apresenta as métricas

Tabela 2 – Níveis dos fatores no 1º projeto fatorial

	Fatores				**Fatores**			
k=4	A (Execuções)		B (nº vizinhos)		C (Tam. Lista Cand. Restrito) (%)		D (Tam. estrut vizinhança)	
Níveis	**1**	**50**	**1**	**10**	1	30	**1**	**2**
	2	100	2	20	**2**	**50**	2	3

Fonte: Próprio autor.

coletadas neste 1º experimento fatorial, ressaltando os melhores e piores resultados das dezesseis variações.

Como pode ser observado na Tabela 3, a configuração que obteve o pior resultado foi 1-2-1-2 (50 execuções, 20 vizinhos, 30% da lista de candidatos restrita e 3 estruturas de vizinhança). A configuração com o melhor resultado foi 1-1-2-1 (50 execuções, 10 vizinhos, 50% da lista de candidatos restrita e 2 estruturas de vizinhança).

Tabela 3 – Métricas obtidas no 1º projeto fatorial

Melhores e Piores Configurações						
Configurações				Métricas		Custo Benefício
A	B	C	D	Qualidade do Resultado	Tempo Gasto (Segundos)	FO X (Tempo/#Soluções)
1	2	1	2	53,68	16035	17215
2	2	1	1	52,56	12539	6590
2	**1**	**2**	**1**	**38,23**	**8079**	**3088**
1	**1**	**2**	**1**	**38,9**	**4017**	**3125**

Fonte: Próprio autor.

É desejável ter uma noção do desempenho da heurística conforme sua configuração, por exemplo, comparando-se o tempo total gasto com a qualidade das soluções geradas. Assim, uma estimativa do custo para a geração de cada solução poderia ser calculada ao multiplicar-se o valor da função objetivo (FO) pelo tempo gasto na geração de uma solução (Tempo total / Quantidade de execuções). A última coluna da Tabela 6.2.1.2 apresenta tais custos, de maneira que as duas últimas configurações apresentaram um melhor custo-benefício. Ao comparar as duas últimas linhas entre si (os melhores resultados do experimento) pode-se observar que mantendo-se fixos os fatores B, C e D e alterando-se os níveis do fator A (quantidade de execuções), de 50 para 100 execuções não houve diferença significativa na qualidade da melhor solução obtida. Com 100 execuções, o algoritmo obteve uma solução 1,7% relativamente melhor que com 50 execuções porém, ao custo do dobro do tempo de execução. Assim, como uma boa solução conseguiu ser obtida dentre 50 soluções factíveis geradas, insistir em gerar mais soluções além disso provavelmente apenas desperdiçará tempo. Em ambos os fatores B (quantidade de vizinhos) e D (quantidade de estruturas de vizinhança), o valor escolhido para o nível 2 impactou negativamente no tempo de execução. O que mais afetou na minimização da função objetivo

foi o fator C (tamanho da LCR — lista de candidatos restrita), gerando uma melhoria de 27% na qualidade das soluções geradas. Na configuração de parâmetros mais promissora (1-1-2-1) foram necessários cerca de 80 segundos para gerar cada solução factível, com uma média de 38,9 para a função objetivo (penalizações). Já na configuração de parâmetros mais dispendiosa (1-2-1-2) foram necessários cerca de 320 segundos para gerar cada solução, com uma média de 53,68 para a função objetivo (penalizações).

4.8.2 Segundo projeto fatorial 2^3

A partir da análise da configuração de parâmetros mais bem sucedida no projeto fatorial, elaboramos uma nova proposta de valores para calibração dos parâmetros dos algoritmos. Para tal, alteramos os níveis de cada fator, reduzindo seu valor limítrofe ou aumentando-o, conforme o caso.

Pela análise apresentada no 1º projeto fatorial, observamos que o fator A (quantidade de execuções) fazia com que o tempo de execução do algoritmo crescesse linearmente mas sem garantir uma melhoria significativa na qualidade da solução encontrada. Assim, neste novo projeto fatorial pretende-se avaliar a possibilidade de reduzir ainda mais o valor de calibração do fator A (quantidade de execuções, i.e. soluções factíveis geradas). Já os o fatores B (com 10 vizinhos) e C (LCR de 50%) impactaram positivamente, tanto na qualidade da solução quanto no tempo gasto, devendo ser explorados novos níveis para sua calibração. A Tabela 4 apresenta os novos valores para os fatores a serem avaliados nesta segunda rodada do projeto fatorial.

Tabela 4 – Níveis dos fatores no 2º projeto fatorial.

	Fatores				**Fatores**			
k=4	A (Execuções)		B (nº vizinhos)		C (Tam. Lista Cand. Restrito) (%)		D (Tam. estrut vizinhança)	
Níveis	1	50	1	10	1	20	1	**2**
	2	**25**	2	**5**	2	**50**	2	**2**

Fonte: Próprio autor.

Tais configurações de parâmetros foram escolhidas conforme as justificativas abaixo apresentadas:

- **Fator A:** 50 e 25 execuções, visando reduzir o tempo gasto;
- **Fator B:** 10 e 5 vizinhos, visando reduzir o tempo gasto e reduzir a FO;
- **Fator C:** 20% e 50% para a LCR. Apesar da LCR maior aparentemente ter melhorado a eficiência da solução, não aumentamos-na para além de 50%. O motivo é que este parâmetro define o quão guloso ou aleatório o GRASP será;

- **Fator D:** fixamos a quantidade de estruturas de vizinhança em 2, por ser o melhor valor observado no último projeto fatorial.

O motivo de manter o fator D fixo com o valor 2 é que observamos, experimentalmente, que com valores maiores que 3 o tempo de execução da metaheurística crescia exponencialmente, porém sem apresentar melhoria significativa na qualidade dos resultados. Assim, uma nova rodada de experimentos foi conduzida, desta vez com com 8 combinações (2^3) de calibração do algoritmo.

Como pode ser observado na Tabela Tabela 5, a configuração que obteve o pior resultado foi 1-1-1-2 (50 execuções, 10 vizinhos, 20% da lista de candidatos restrita e 2 estruturas de vizinhança). E a configuração com o melhor resultado foi 2-2-2-2 (25 execuções, 5 vizinhos, 50% da lista de candidatos restrita e 2 estruturas de vizinhança).

Tabela 5 – Métricas obtidas no 2º projeto fatorial.

Melhores e Piores Configurações						
Configurações				Métricas		Custo Benefício
A	B	C	D	Qualidade do Resultado	Tempo Gasto (Segundos)	FO X (Tempo/#Soluções)
1	1	1	2	64.82	4096	5310
1	2	1	2	64.84	2760	3579
2	**1**	**2**	**2**	**40.68**	**1940**	**3156**
2	**2**	**2**	**2**	**39.88**	**1415**	**2257**

Fonte: Próprio autor.

A última coluna da Tabela 5 apresenta a estimativa do custo para a geração de cada solução, calculada ao multiplicar-se o valor da função objetivo (FO) pelo tempo gasto na geração de uma solução (Tempo total / Quantidade de execuções). As duas últimas configurações, destacadas em negrito, apresentaram o melhor custo-benefício. Ao comparar as duas últimas linhas entre si (os melhores resultados do experimento) pode-se observar que mantendo-se fixos os fatores A, C e D e alterando-se os níveis do fator B (número de vizinhos), de 10 para 5 vizinhos não houve diferença significativa na qualidade das soluções obtidas (melhoria relativa de 1,9%) porém, o tempo gasto na execução reduziu significativamente apresentando uma melhoria relativa de 27%. O mesmo pode ser observado ao comparar as duas primeiras linhas entre si: mantendo-se fixos os fatores A, C e D e alterando-se os níveis do fator B (número de vizinhos), observa-se uma melhoria relativa de 32% no tempo gasto mas não houve diferença significativa na qualidade das soluções obtidas ($< 0{,}03\%$). Comparando-se a segunda com a quarta linha (configurações 1-2-1-2 e 2-2-2-2), pode-se observar o impacto dos fatores A e C no desempenho da heurística. Reduzindo-se ainda mais a quantidade de execuções (fator A nível 2), menos soluções precisaram ser geradas melhorando o tempo gasto em 48%. Por outro lado, caso a heurística se comportasse de forma mais gulosa (fator B nível 1) haveria uma piora significativa na qualidade do resultado porém, mantendo-a no patamar

da melhor configuração (fator B nível 2) dessa rodada de resultados, pode-se observar uma melhoria relativa de 38% em relação à qualidade das soluções obtidas.

Na configuração de parâmetros mais promissora (2-2-2-2) foram necessários cerca de 57 segundos para gerar cada solução factível, com uma média de 39,88 para a função objetivo (penalizações). Já na configuração de parâmetros mais dispendiosa (1-1-1-2) foram necessários cerca de 82 segundos para gerar cada solução, com uma média de 64,82 para a função objetivo (penalizações).

4.8.3 Terceiro projeto fatorial 2^4

Com tais observações em mente, um terceiro e último projeto fatorial foi realizado, visando experimentar o impacto de um aumento na lista de candidatos restrita (fator C), bem como aproveitar o projeto fatorial para explorar o impacto de uma nova redução na quantidade de execuções e de vizinhos (fatores A e B).A Tabela 6 apresenta os valores propostos para os fatores a serem avaliados nesta terceira e última rodada do projeto fatorial.

Tabela 6 – Níveis dos fatores no 3º projeto fatorial.

	Fatores				**Fatores**			
k=4	A (Execuções)		B (nº vizinhos)		C (Tam. Lista Cand. Restrito) (%)		D (Tam. estrut vizinhança)	
Níveis	1	5	1	3	1	50	1	**2**
	2	**15**	2	**5**	2	**65**	2	**3**

Fonte: Próprio autor.

Considerando que o tempo total de execução da metaheurística foi drasticamente reduzido com a nova calibração dos parâmetros A e B, optamos por também verificar qual seria o impacto de um aumento na quantidade de estruturas de vizinhança (fator D). Assim, ao todo foram conduzidos 16 testes para verificar o comportamento da metaheurística com as 2^4 combinações de calibração propostas. Como pode ser observado na Tabela 6, as configurações que obtiveram o pior resultado foram 1-1-2-1 e 1-2-1-2 (5 execuções, ..., estrutura de vizinhança de tamanho 3), obtendo uma função objetivo com um valor de 999 (cumulativo de de penalidades da solução encontrada). Um possível causa que inviabilizou ao algoritmo de encontrar uma solução factível pode ter sido a baixa quantidade de execuções (fator A nível 1), ou seja, uma quantidade insuficiente de repetições em que o algoritmo foi executado.

A estimativa do custo para a geração de cada solução é novamente apresentada na última coluna da Tabela 7. Considerando as configurações com o melhor resultado (2-2-2-2 e 1-2-2-1), o aumento na quantidade de execuções (fator A) e na quantidade de

Tabela 7 – Métricas obtidas no 3º projeto fatorial.

Melhores e Piores Configurações						
Configurações				Métricas		Custo Benefício
A	B	C	D	Qualidade do Resultado	Tempo Gasto (Segundos)	FO X (Tempo/#Soluções)
1	1	2	1	999	371	74125
1	2	1	2	999	313	62537
2	**2**	**2**	**2**	**34,40**	**1517**	**3377**
1	**2**	**2**	**1**	**36**	**317**	**2282**

Fonte: Próprio autor.

estruturas de vizinhança (fator D), impactaram numa melhoria relativa de 7,2% na função objetivo. Uma estrutura de vizinhança maior faz com que as soluções que serão geradas passem por um tempo maior de perturbação, podendo gerar assim soluções melhores. Com a utilização de uma estrutura de vizinhança maior (fator D nível 2), obtivemos a **melhor qualidade para as soluções geradas (função objetivo de 33,40)** dentre todos os três projetos fatoriais conduzidos. Naturalmente, isto demandará um tempo de execução um pouco maior, mas pelo qual vale a pena esperar. Em contrapartida, pode-se creditar uma pequena redução no tempo de execução do algoritmo, com a utilização de uma maior quantidade de vizinhos (fator B nível 2). Levando em conta que tanto os piores quanto os melhores resultados tiveram em comum a baixa quantidade de 5 execuções (fator A nível 1), bem como recapitulando os melhores resultados observados no segundo projeto fatorial, decidimos por não arriscar a apresentação de uma boa solução factível ao reduzir demais a quantidade de execuções. Assim, recomendamos o uso de 15 execuções (fator A nível 2). Na configuração de parâmetros mais promissora (2-2-2-2) foram necessários cerca de **101 segundos para gerar cada solução factível** porém, com uma média de 33,40 para a função objetivo (penalizações). Já na segunda melhor configuração de parâmetros (1-2-2-1) foram necessários cerca de 63 segundos para gerar cada solução porém, com uma média de 36,00 para a função objetivo (penalizações).

4.8.4 Configuração da metaheurística para o protótipo

Considerando os resultados do terceiro e último projeto fatorial, elegemos como melhor a configuração aquela em que o algoritmo obteve o melhor custo-benefício. A calibração da metaheurística será fixada em:

- 15 execuções;
- 5 vizinhos;
- LCR de 65%;
- 3 estruturas de vizinhanças.

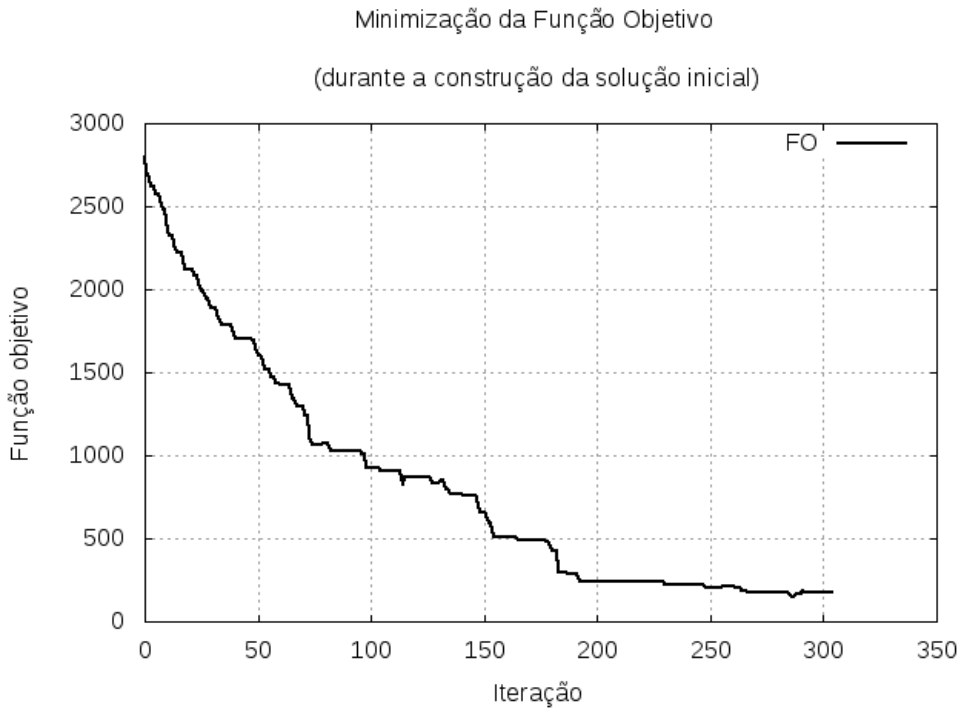
5 RESULTADOS E ANÁLISE

De posse do algoritmo projetado, implementado, otimizado e calibrado, neste capítulo passamos à apresentação do desempenho do algoritmo e análise dos resultados com ele obtidos. Posteriormente, com a averiguação da qualidade das soluções geradas pelo GRASP+VND, serão apresentados experimentos de validação da abordagem, nos quais instâncias reais de escolas da região foram utilizadas e seus resultados comparados.

5.1 Convergência da FO das metaheurísticas

A cada iteração do algoritmo na sua construção da solução inicial, a função objetivo poderá sofrer uma alteração visto que o objetivo é sempre tentar minimizar a mesma. Foram capturados os dados de todas as iterações na construção de uma solução inicial para prover uma visualização de como foi a convergência da função objetivo em relação a quantidade de iterações.

Figura 17 – Minimização da função objetivo do GRASP+VND.



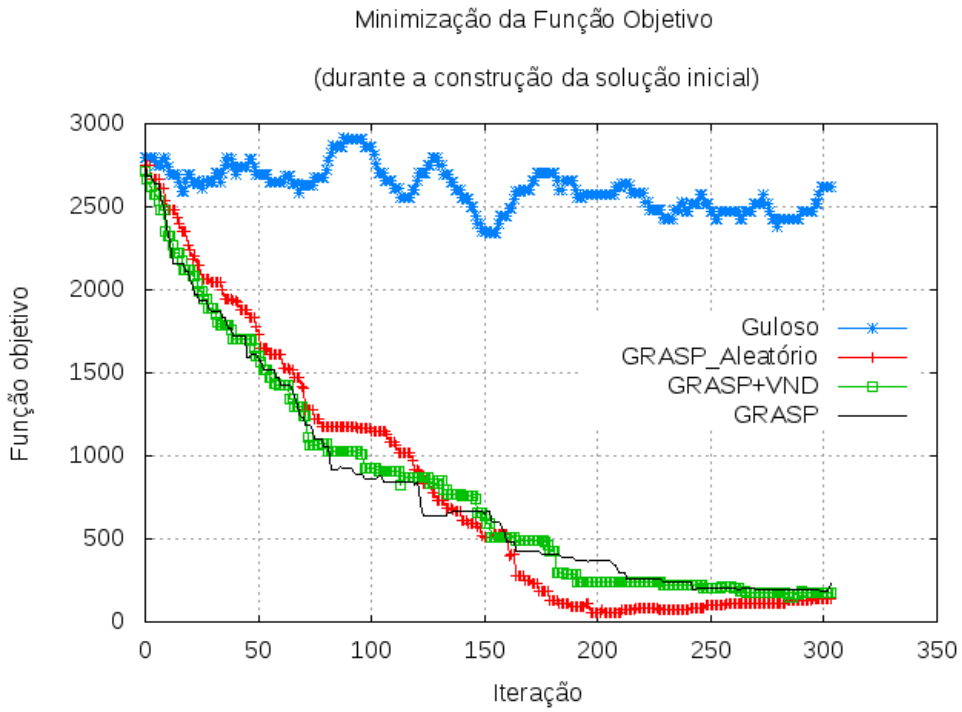
Fonte: Próprio autor

Como pode ser observado na Figura 17, em um pouco mais de 300 iterações a função objetivo caiu de 2800 unidades para cerca de 180. Observe que em alguns momentos

a função objetivo aumenta e depois volta a diminuir, isso pode ser justificado devido a inserção do critério de Boltzmann na metaheurística, visando aceitar pioras na função objetivo com o intuito de tentar melhorar a solução ao atender a mais restrições fracas.

Vale ressaltar que na metaheurística proposta para atacar o problema tratado neste trabalho (GRASP+VND), tivemos o cuidado de não degradar o GRASP para um comportamento puramente aleatório (caso definíssemos a LCR com tamanho 100% do total de candidatos) bem como não degradá-lo a um comportamento puramente guloso (LCR de tamanho 1 candidato). Entretanto, visando comparar o comportamento de diferentes abordagens para o problema, foram implementadas variações do GRASP, para que se comportasse de maneira totalmente gulosa ou totalmente aleatória. Durante a realização de testes com tais variações do GRASP, além do resultado final (função objetivo e tempo gasto), também armazenamos a cada iteração o valor da função objetivo. Isso nos possibilitou comparar a taxa de decaimento entre elas, visando identificar qual das abordagens metaheurísticas converge mais rapidamente para uma boa solução. Na Figura 18 pode-se visualizar o comportamento dos algoritmos implementados.

Figura 18 – Comparação do decaimento das funções objetivos.



Fonte: Próprio autor

Como pode ser observado na Figura 18, o algoritmo guloso conseguiu diminuir ligeiramente o valor a função objetivo, para em seguida aumentá-la, mas nunca conseguindo diminuí-la o suficiente para gerar uma solução viável (FO minimizada, com baixa penalização) em comparação às outras implementações de heurísticas. As três variações

do GRASP com as modelagens propostas obtiveram resultados relativamente parecidos, apresentando um decaimento que segue um comportamento parecido. Vale relembrar que os aumentos temporários na FO (pioras) é devido ao critério de Boltzmann inserido ao GRASP.

Curiosamente, o GRASP (totalmente) aleatório foi a abordagem que mais rapidamente conseguiu diminuir a função objetivo em relação às demais. Conforme os experimentos com projeto fatorial 2^k insinuaram (vide Seção 4.8), ao aumentarmos o tamanho da lista de candidatos restrita (LCR) o valor da função objetivo e o tempo gasto na execução caíram drasticamente. Ao final, com o término da quantidade máxima de iterações que permitia à heurística explorar o espaço de soluções, o valor da função objetivo da solução gerada por cada abordagem se aproximou. Vale ressaltar que a Figura 18 apresenta o processo *gradual* de construção da solução inicial, que ao final deve atender todas as demandas de horários de aula e respeitar todas as restrições fortes. Portanto, faz sentido que ao valor final da função objetivo da solução inicial seja pior do que o valor da função objetivo de alguma iteração prévia, uma vez que essa conteria uma solução parcial, apenas.

Nas próximas subseções, serão apresentadas estatísticas descritivas da função objetivo para uma população de de 50 soluções geradas por cada abordagem metaheurística: GRASP (totalmente) aleatório, GRASP sem VND (tradicional), e GRASP+VND que é a proposta deste estudo.

5.2 Desempenho das metaheurísticas

Existem diversas maneiras de se modelar um problema para se encaixar na solução proposta por uma determinada metaheurística ou, até mesmo, unir os esforços de duas ou mais metaheurísticas para que a segunda procure melhorar o resultado obtido pela primeira. Além da abordagem metaheurística proposta (GRASP+VND), neste trabalho foram também implementadas variações do GRASP (alterando a LCR) e, também, um algoritmo guloso. O objetivo foi comparar o desempenho da abordagem proposta (GRASP+VND) com as demais, para determinada instância de dados como entrada. Os algoritmos implementados foram:

- GRASP com busca local **utilizando o VND**;
- GRASP **tradicional**, i.e., sem a utilização do VND;
- GRASP com a LCR de tamanho máximo, i.e., **totalmente aleatório**;
- Algoritmo **totalmente guloso**, que escolhe sempre o vértice de maior grau como melhor candidato compor a solução.

A comparação dos algoritmos se dará entre a quantidade de cores das soluções geradas (qualidade da solução) e, também, pelos valores obtidos com uma mesma função objetivo (penalidades da solução). Observe que, pela transformação dos problemas, a quantidade de cores necessárias à coloração do grafo correspondente à quantidade de horários demandada para atender às restrições do problema de alocação. Dentre a quantidade de cores e a função objetivo, a primeira é tida como de maior importância (**restrições fortes**), uma vez que reflete o desempenho do algoritmo quanto à minimização da quantidade de dias da semana e turnos disponíveis para atender à demanda de alocação. Ao compararmos o desempenho das metaheurísticas, o valor da função objetivo é complementar e de importância secundária, uma vez que exprime a quantidade de penalidades da solução proposta por não atender algumas das **restrições fracas**.

5.2.1 Função objetivo e seus pesos

Como todas as restrições fortes devem ser atendidas para que a solução gerada seja factível, a função objetivo contém as penalidade aplicadas ao não serem atendidas algumas das restrições fracas do problema de alocação de horários (*e.g.*, recomendações pedagógicas, preferências de horários). Como uma restrição pedagógica e uma preferência pessoal de horário não tem a mesma importância no processo de decisão, utilizamos diferentes pesos para cada tipo de penalidade, ponderando-a conforme sua gravidade:

- Peso 4: Não atendimento a **preferência** de horário de determinado professor;
- Peso 16: Existência de **lacuna** (horário vago) entre duas aulas;
- Peso 16: Existência de aula **interposta** entre duas aulas de determinada matéria (**não geminação**);
- Peso 49: Existência de três ou mais aulas subsequentes de uma mesma matéria (**poligeminação**).

A Equação 5.1 apresenta o somatório das penalidades realizado na função objetivo, em que cada infração ao não atendimento de uma restrição fraca (acima listadas) é multiplicada pelo seu respectivo peso.

$$\sum_{i=1}^{horários} 4 * \textbf{Preferência}(i) + 16 * \textbf{Lacuna}(i) + 16 * \textbf{Interposta}(i) + 49 * \textbf{Poligeminada}(i) \tag{5.1}$$

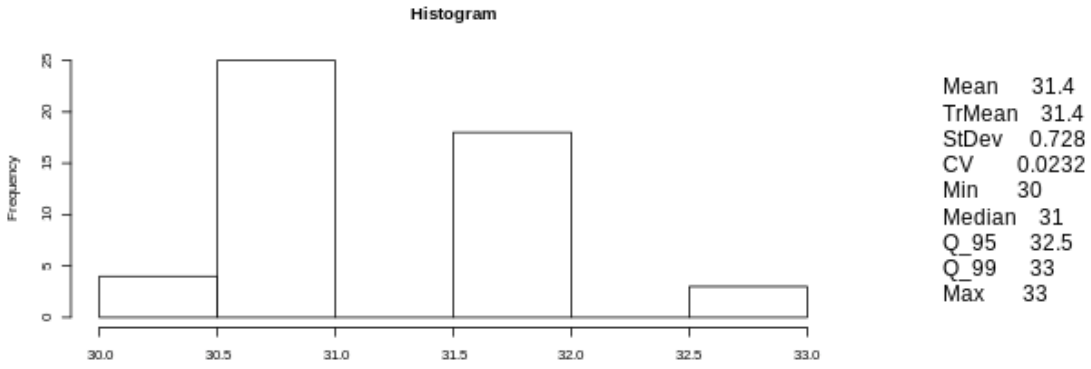
5.2.2 Minimização da quantidade de horários necessários (cores)

Para cada variação do GRASP (tradicional, com VND ou totalmente aleatório), foi gerada uma população de 50 soluções. Já no caso do algoritmo Guloso, foi necessário gerar apenas uma solução pois, sendo "guloso", sempre retornará o mesmo resultado.

Para melhor interpretar os resultados abaixo, vale ressaltar que a quantidade de cores de uma solução para o problema de coloração de grafos equivale à quantidade de horários ao longo da semana que deverão ser alocados para atender à demanda da instância de entrada. Como entrada foi utilizada a mesma instância do projeto fatorial (Seção 4.8). Nesta instância, as aulas devem ser alocadas de segunda a sexta e apenas no turno matutino: de 7:00 às 12:20, com aulas de 50 min. e um intervalo de 20 min. Ou seja, uma solução viável para esta instância deverá ter no máximo 30 cores (5 dias na semana * 6 horários matutinos), bem como atender a todas as restrições fortes impostas. As variações do GRASP conseguiram gerar resultados factíveis e viáveis na prática, conforme será apresentado a seguir. Já o algoritmo Guloso não propôs uma solução utilizável na vida real, exigindo no mínimo 33 cores, ou seja, mais que os 30 horários disponíveis para alocação no turno matutino.

Como pode ser observado na Figura 19, o GRASP **totalmente aleatório** conseguiu gerar soluções viáveis (mínimo de 30 cores) porém, a média de cores das 50 soluções propostas (31,4) foi maior que a do GRASP tradicional.

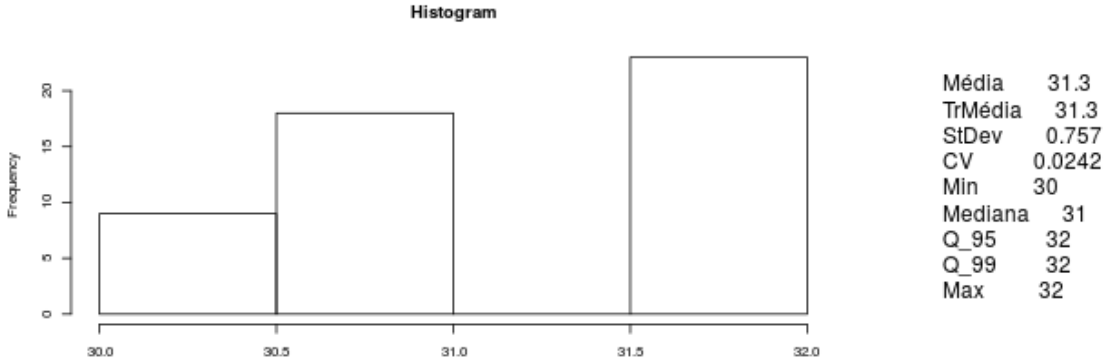
Figura 19 – Estatísticas do **nº de cores** obtido pelo GRASP **totalmente aleatório**



Fonte: Próprio autor

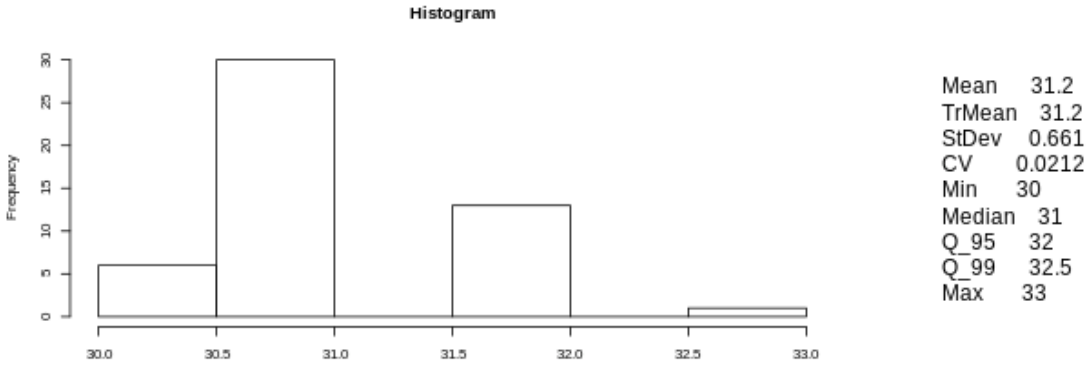
De forma equivalente, o GRASP **tradicional** (sem o VND) conseguiu gerar soluções viáveis e de utilização prática (mínimo de 30 cores) Figura 20 no entanto, suas soluções foram mais heterogêneas e propuseram em média uma maior quantidade de cores que o GRASP+VND. Estatisticamente, o GRASP **com o VND** foi o que obteve os resultados mais robustos: a quantidade média de cores (31,2), seu desvio padrão e coeficiente de variação foram os mais baixos em relação aos demais, como pode ser observado na Figura 21.

Figura 20 – Estatísticas do **nº de cores** obtido pelo **GRASP tradicional**



Fonte: Próprio autor

Figura 21 – Estatísticas do **nº de cores** obtido pelo GRASP **com VND**



Fonte: Próprio autor

Conforme pode ser observado, as três variações do GRASP alcançaram a menor quantidade de cores possível (30 cores), em **ao menos uma** das soluções geradas. Mas, para atingir tal solução ótima global, uma quantidade maior ou menor de execuções do algoritmo deveria ser utilizada conforme a robustez e eficiência da metaheurística. Deve-se ressaltar que cada uma das 50 soluções geradas (população da qual as estatísticas proveem) consiste na melhor solução encontrada pela abordagem metaheurística em uma repetição do experimento. Ou seja, para cada uma destas 50 amostras o espaço de soluções foi devidamente explorado conforme a estratégia heurística. Tendo isto em mente, pode-se afirmar que a abordagem metaheurística que apresente a menor quantidade média de cores e o menor desvio padrão comporta-se de maneira mais **robusta**.

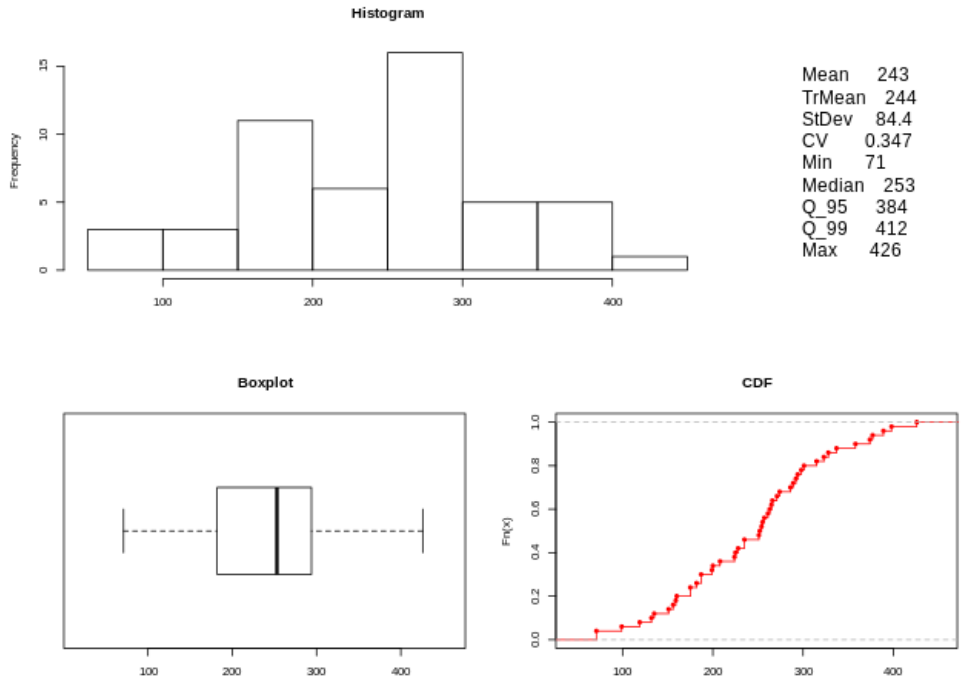
Dentre as três abordagens ressaltamos que o **GRASP+VND** apresentou a menor quantidade média de cores (31,2), o menor desvio padrão (0,661) e, consequentemente, o menor coeficiente de variação (0,0212). Isso significa que das três abordagens analisadas,

o GRASP+VND foi a que mais consistentemente propôs soluções próximas da solução ótima (30 cores). Ou seja, soluções com a menor quantidade de horários utilizados, em dias úteis da semana, para encaixar a demanda de horários de aula e respeitar as restrições impostas. Assim, devido a esta metaheurística ter apresentado robustez na minimização da quantidade de cores para as soluções geradas, dentre as abordagens analisadas recomenda-se a utilização do GRASP+VND para resolver o problema.

5.2.3 Maximização do atendimento às restrições (FO)

Foram também coletados os valores das funções objetivo (FO) de cada uma das 50 soluções gerada pelas variações do GRASP e pela solução única do algoritmo Guloso. Os resultados desta métrica são apresentados a seguir. Para melhor interpretar os resultados abaixo, vale ressaltar que a função objetivo penaliza cada restrição fraca que não foi atendida, de acordo com os pesos apresentados na Subseção 5.2.1.

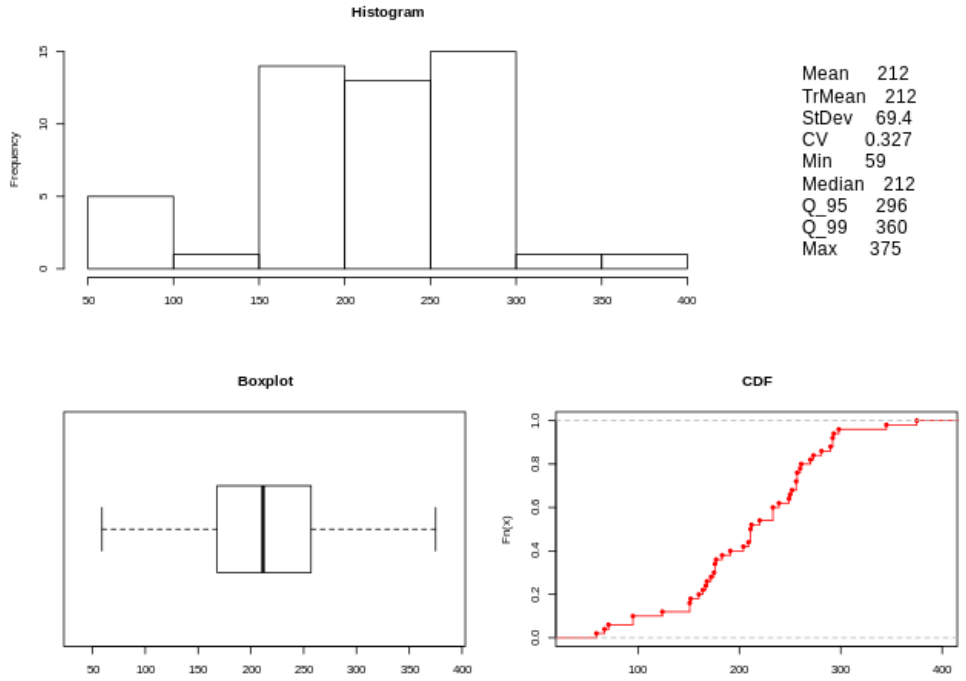
Figura 22 – Estatísticas da **FO** obtida pelo **GRASP tradicional**



Fonte: Próprio autor

Como pode ser observado na Figura 22, a maioria das soluções geradas pelo **GRASP tradicional** ficaram entre um valor de 180 e 295. Os resultados foram bons em relações as outras metaheurísticas pois com a utilização dos pesos das restrições fracas proporcionais à gravidade da infração, com o GRASP tradicional a função objetivo conseguiu ser minimizada até o valor de 71 unidades, no caso da melhor solução dentre as 50 soluções por ele geradas.

Figura 23 – Estatísticas da **FO** obtida pelo **GRASP aleatório**



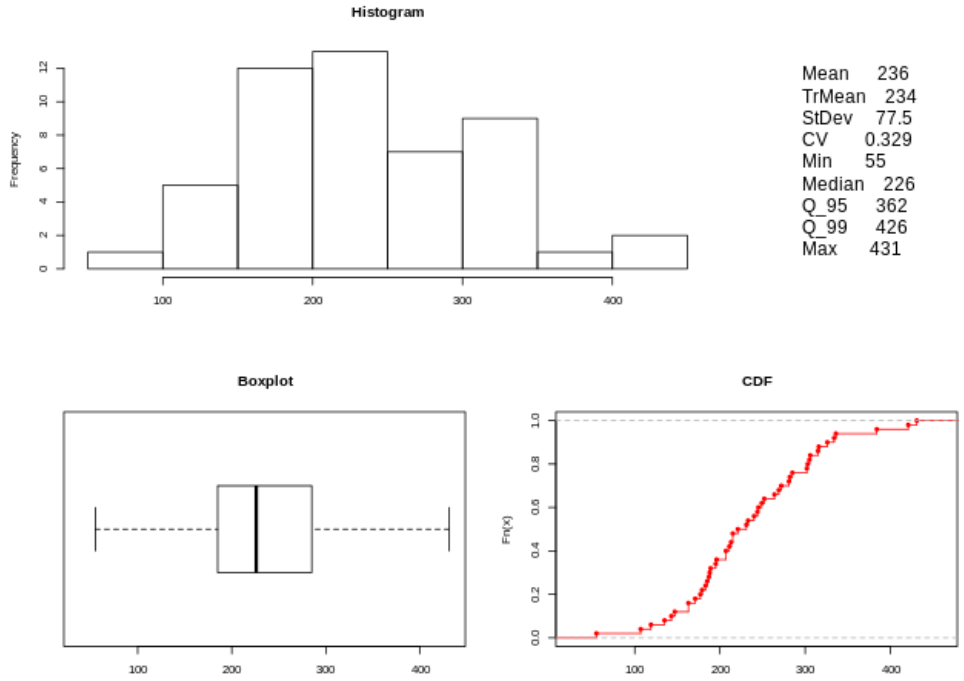
Fonte: Próprio autor

Tal qual no comparativo da seção anterior (minimização de cores), o **GRASP totalmente aleatório** forneceu melhores resultados em relação ao GRASP tradicional pois, com a aleatoriedade, ele consegue explorar mais livremente os vértices (vide Figura 23). A melhor solução dentre as 50 geradas pelo GRASP totalmente aleatório conseguiu minimizar a função objetivo para 59 unidades, ontendo um desempenho relativamente **16,9% melhor** que o GRASP tradicional. Também, a média de valores das funções objetivo desta abordagem foi a menor (212) dentre as três variações do GRASP analisadas, bem como seu coeficiente de variação foi menor (0,327).

Por fim, o **GRASP+VND** apresentou um desempenho entre o GRASP tradicional e o GRASP totalmente aleatório, conforme estatísticas apresentadas na Figura 24, tanto em relação à média de valores das funções objetivo (236) quanto à sua mediana (226). Porém, vale ressaltar que o GRASP com o VND consistiu na abordagem que obteve a menor função objetivo dentre todas os demais, chegando a penalidade de apenas 55 unidades, ou seja, apresentou um desempenho relativamente **22,5% melhor** que o GRASP tradicional.

Para o algoritmo Guloso não foi gerada análise estatística descritiva nem seu gráfico, uma vez que a solução única gerada foi penalizada pela função objetivo em 2620 unidades. Portanto, o algoritmo Guloso foi a abordagem que gerou o pior resultado dentre as demais. Enquanto os demais algoritmos apresentaram FO de 375 a 431 no caso das piores soluções por eles encontradas, a solução gerada pelo algoritmo Guloso foi penalizada em 2620

Figura 24 – Estatísticas da **FO** obtida pelo **GRASP com VND**



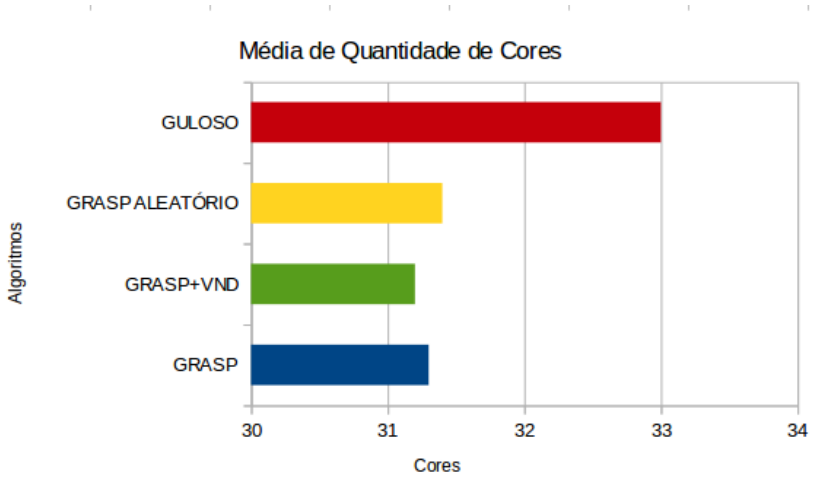
Fonte: Próprio autor

unidades (cinco vezes pior que os demais).

5.2.4 Comparação do desempenho das metaheurísticas

Após ter sido apresentada a caracterização das estatísticas descritivas das abordagens metaheurísticas, nesta seção serão apresentados gráficos de barras visando comparar de maneira mais tangível o desempenho apresentado pelas metaheurísticas. A seguir, será apresentada comparação da média da quantidade de cores para cada conjunto de 50 soluções gerado pelas abordagens.

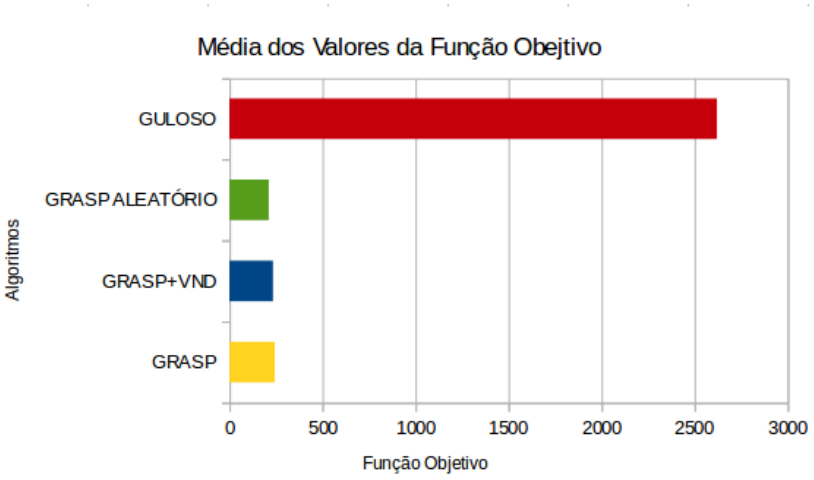
Figura 25 – Quantidade média de cores proposta pelos algoritmos



Fonte: Próprio autor

Conforme observa-se na Figura 25, o algoritmo Guloso obteve os piores resultados em relação à quantidade de cores da solução encontrada, ficando bem acima da quantidade média apresentada pelas outras três metaheurísticas. Já o GRASP+VND foi a abordagem que conseguiu gerar soluções em média com a menor quantidade de cores em relação aos outros algoritmos.

Figura 26 – Penalidade média da função objetivo sofrida pelos algoritmos

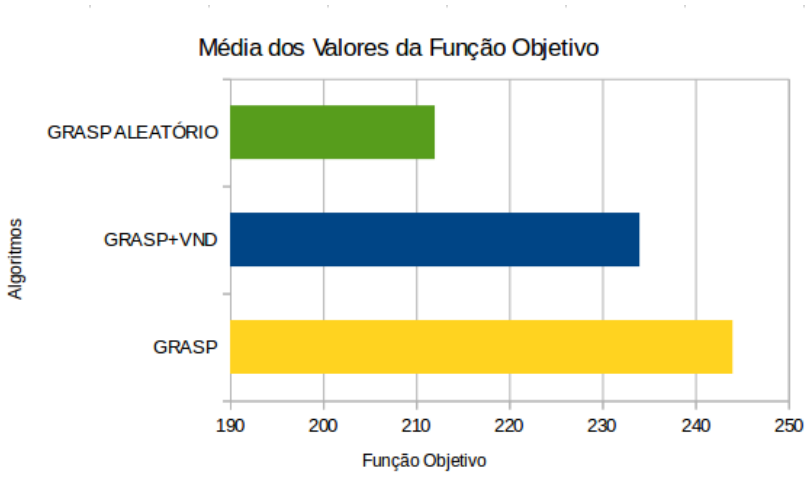


Fonte: Próprio autor

Em relação às penalidades aplicadas pela função objetivo, o algoritmo Guloso também obteve os piores resultados, gerando uma solução de cinco a dez vezes pior que os demais algoritmos implementados, como pode ser observado na Figura 26.

Na figura Figura 27 foi excluído o algoritmo Guloso por questões de escala. Observe por ela que o GRASP totalmente aleatório obteve o melhor desempenho médio, ao

Figura 27 – Comparação entre as funções objetivos geradas pelos GRASP



Fonte: Próprio autor

minimizar as penalidades em relação aos demais algoritmos. Entretanto, conforme visto anteriormente na Figura 24 o GRASP+VND apresentou uma solução com FO 6,8% menor em relação ao GRASP totalmente aleatório. Ademais, vale lembrar que no quesito quantidade média de cores para as soluções geradas, o GRASP totalmente aleatório comportou-se de maneira pior que os algoritmos GRASP e GRASP+VND.

Para as instâncias utilizadas nos experimentos, após comparar os gráficos e estatísticas de desempenho dos algoritmos avaliados observamos que o GRASP+VND foi a abordagem que obteve os resultados mais consistentes (bons resultados com uma maior frequência), tendo obtido o melhor desempenho em relação à minimização da função objetivo utilizada e atingido a quantidade mínima possível de cores.

5.3 Experimentos com outras instâncias do problema

O algoritmo utilizado para a resolução do problema foi submetido a uma bateria de testes utilizando quatro instâncias reais, de escolas situadas na cidade de Formiga. Os dados disponibilizados pelas escolas se limitaram às disciplinas e quantidade de aulas que cada professor deve ministrar para cada turma. Juntamente com quais dias da semana são letivos e quantos horários compõe o turno das aulas, tais dados consistem nos requisitos fortes aos quais uma solução proposta deve atender. Já as recomendações pedagógicas e preferências docentes por horários, consideradas restrições fracas, não nos foram disponibilizadas pois algumas escolas não puderam divulgar tais informações ou simplesmente não mantinham um registro permanente de tais preferências. A instância que foi utilizada para realizar o projeto fatorial, também consiste em uma instância real, mas em que a quantidade de turmas foi alterada.

Com a carência de dados sobre recomendações/preferências, tais restrições fracas foram geradas sinteticamente baseando-nos em recomendações pedagógicas do próprio MEC e em preferências pessoais de alguns professores aos quais tivemos acesso. Assim, a formamos uma pequena base de dados de preferências de horários de professores, a qual foi utilizada como demanda sintética de restrições fracas, para as outras instâncias de escolas que serão analisadas a seguir.

5.3.1 Configuração dos experimentos de validação

As restrições fortes (SR) consistem em: (SR-1) nenhum professor ou turma poderia estar em duas aulas diferentes simultaneamente, requisito que a solução inicial do GRASP atende para que a solução seja factível; e (SR-2) respeitar as restrições de horários dos professores, nas quais em hipótese alguma poderia lecionar. Já as restrições fracas consistem em: (WR-1) tentar atender às recomendações pedagógicas; e (WR-2) tentar atender às preferências de horários dos professores.

Para que pudesse ser observado a diferença do comportamento do algoritmo quando uma instância continha em maior ou menor grau de restrições, foram **mantidas fixas** a restrição forte SR-1 e a restrição fraca WR-1, para cada instância de escola analisada, foram geradas quatro variações combinando-se as restrições SR-2 e WR-2, conforme detalhado abaixo:

- Cenário 1: SR-1 & WR-1, mais as Restrições de horário (SR-2) e Preferências de horário (WR-2);
- Cenário 2: SR-1 & WR-1, mais as Restrições de horário (SR-2);
- Cenário 3: SR-1 & WR-1, mais as Preferências de horário (WR-2);
- Cenário 4: SR-1 & WR-1, apenas.

Para a geração dos resultados com estas outras instâncias do problema, baseadas em dados de escolas de Formiga/MG, foi utilizada a metaheurística GRASP juntamente com o VND (GRASP+VND), utilizando como parâmetros:

- 200 execuções (soluções a serem geradas);
- 5 vizinhos (a serem explorados pelo busca local do GRASP);
- 65% para o tamanho da LCR (lista de candidatos restrita);
- 3 estruturas de vizinhança (que o VND irá utilizar em sua busca).

Observe que tais parâmetros são provenientes da calibração recomendada a partir do projeto fatorial previamente realizado (Subseção 4.8.4), visando maximizar a qualidade da solução e minimizar o tempo gasto na execução do algoritmo. A exceção é a quantidade total de soluções a serem geradas no experimento, a qual aumentamos de 15 para 200 execuções com finalidade de prover uma maior amostra populacional para a análise estatística descritiva. Para fins de comparação da robustez das metaheurísticas discutiremos os valores mínimos e máximos obtidos segundo a função objetivo.

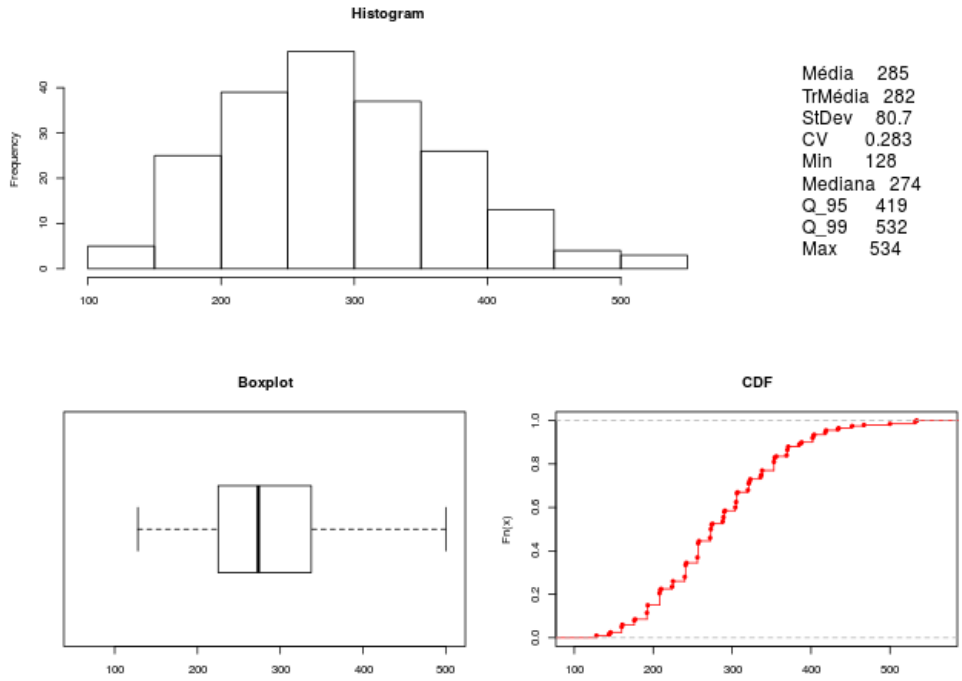
5.3.2 Instância "Escola A"

A instância fornecido pela escola A é composta por 11 turmas, onde cada turma pode ter de 26 a 29 aulas por semana, dependendo de qual seja o ano (grau) da turma. Nesta instância, as aulas devem ser alocadas de segunda a sexta e apenas no turno matutino: de 7:00 às 12:20, com aulas de 50 minutos e um intervalo de 20 min. Ou seja, uma solução viável para esta instância deverá ter **no máximo 30 cores** (5 dias na semana * 6 horários matutinos), bem como atender a todas as restrições fortes impostas. Como há 11 turmas, a demanda por alocação de horários é de um total de **330 aulas** (30 aulas * 11 turmas). Quando esta instância foi coletada *in loco*, foi obtido da direção da escola a informação de que as turmas poderiam ter **horários vagos somente no último horário do dia** e em dias específicos. Isso consistiu em mais algumas restrições fortes a serem consideradas pelas metaheurísticas. Foram sintetizadas no total 44 restrições de horários e outras 44 preferências de horários, de maneira que os professores possuíam de zero a três restrições e a mesma quantidade de preferências. Com os dados que a escola forneceu, foram codificadas 29 restrições de horários para as turmas, visto que elas podem ter aulas vagas somente nos últimos horários do dia.

O algoritmo GRASP+VND conseguiu alocar com sucesso os horários em todas as variações da instância "Escola A" (conforme Subseção 5.3.1), ou seja, em todos os quatro cenários da instância ao menos uma das soluções geradas **obtiveram um número cromático de 30**. Vale lembrar que a quantidade de cores da solução de coloração de grafos corresponde à quantidade de horários alocados ao longo da semana, assim, nenhuma aula ficou fora das configurações exigidas pela instância.

Na Figura 28 são apresentadas as estatísticas descritivas dos resultados obtidos para a instância "Escola A", em relação ao cenário no qual o algoritmo apresentou soluções com baixa penalidade na função objetivo (FO), dentre as variações propostas. A variação da instância "Escola A" que apresentou baixa penalidade na FO, intuitivamente, foi o **cenário 4** que não impunha nenhuma restrição e nenhuma preferência (SR-1 e WR-1 apenas), de maneira que o algoritmo pode explorar bastante o espaço de busca sofrendo menos penalidades. A melhor solução gerada pelas 200 execuções com o melhor cenário desta instância foi um quadro de horários com todas as aulas alocadas como previsto,

Figura 28 – Resultados das FOs com a melhor média da instância A.

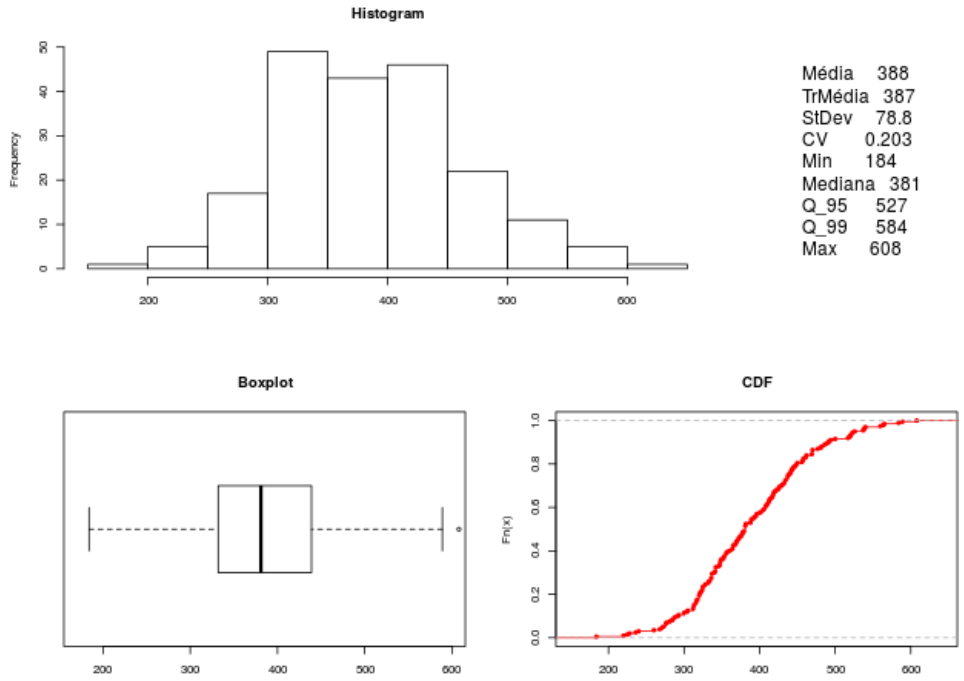


Fonte: Próprio autor

ou seja, sem nenhum aula fora do horário configurado na instância. A quantidade de restrições fracas violadas foi de 9 no total, sendo 9 aulas interpostas entre aulas que seriam geminâveis. Portanto, visto que a instância tem no total de 330 aulas, apenas **2,72% do total de aulas infringe** alguma restrição fraca.

Na Figura 29 são apresentadas as estatísticas descritivas dos resultados obtidos para a instância "Escola A", em relação ao cenário no qual o algoritmo obteve soluções com maior penalidade na função objetivo (FO). Esta variação da instância corresponde ao **cenário 1**, que impõe ambas as restrições e preferências (SR-1, WR-1, SR-2 e WR-2). Assim, quando a instância apresenta restrições em excesso, o algoritmo apresenta uma maior dificuldade em se deparar com soluções boas ao explorar o espaço de busca. A melhor solução dentre aquelas geradas para o pior cenário desta instância foi um quadro de horários com todas as aulas alocadas conforme demanda e restrições específicas da escola, ou seja, sem nenhum aula fora dos horários de aula válidos para esta instância. A quantidade de restrições fracas violadas foram 4 no total, sendo 4 aulas interpostas entre aulas que seriam geminâveis. Visto que a instância tem no total de 330 aulas, apenas **1,21% no total de aulas infringe** alguma restrição fraca. Também houve penalidades por não atender a outro tipo de restrições fracas. Em tempo, este cenário continha preferências de horários e a melhor solução gerada conseguiu atender 13 das 44 preferências ou seja apenas **29,5% das preferências foram atendidas**.

Figura 29 – Resultados das FOs com a pior média da instância A.



Fonte: Próprio autor

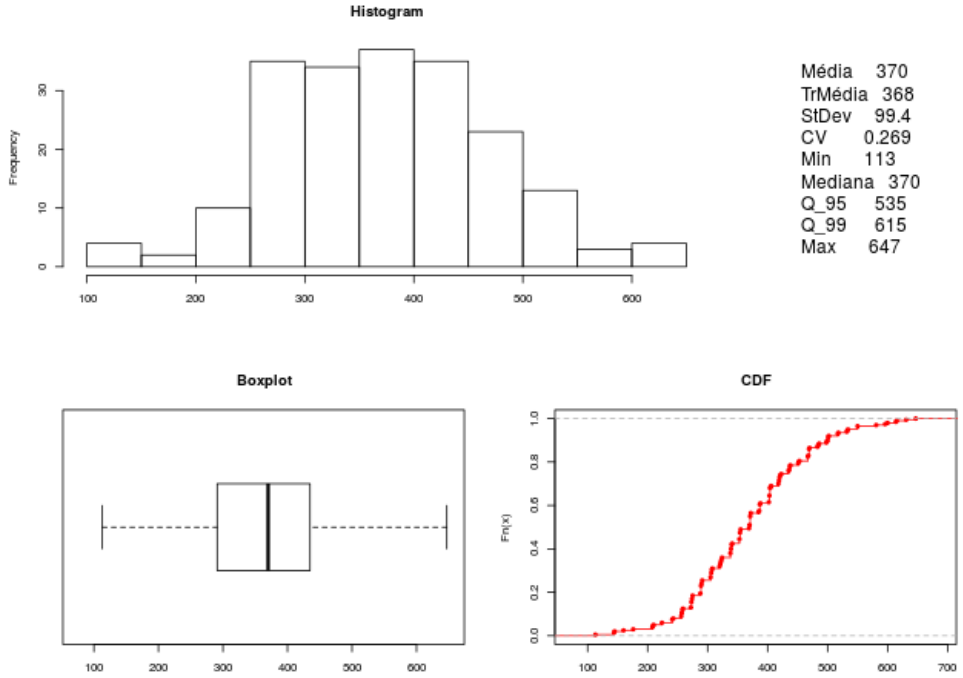
Apesar da melhor função objetivo (FO) obtida no cenário 4 ser de valor menor que a melhor FO do cenário 1, um aspecto interessante a ser observado é a restrição fraca onde deve-se evitar lacunas entre as aulas (horários vagos). Olhando apenas este aspecto, o cenário 1 (todas as restrições e preferências) obteve na solução final um melhor resultado que o cenário 4 (sem restrições ou preferências), ou seja, mesmo com estes estas restrições adicionais o algoritmo ainda conseguiu gerar um quadro de horários viável, apenas deixando de atender a parte das preferências de horários dos professores.

5.3.3 Instância "Escola B"

A instância coletada da "Escola B" é composta por 12 turmas, onde cada turma tem estritamente 25 aulas durante a semana, não podendo haver **nenhum horário vago** entre elas. Nesta instância, as aulas devem ser alocadas de segunda a sexta e apenas no turno matutino: de 7:00 às 11:30, com aulas de 50 minutos e um intervalo de 20 min. Ou seja, uma solução viável para esta instância deverá ter **no máximo 25 cores** (5 dias na semana * 5 horários matutinos), bem como atender a todas as restrições fortes impostas. Como há 12 turmas ao todo, a demanda por alocação de horários é de um total de **300 aulas** (25 aulas * 12 turmas). Foram sintetizadas 20 restrições e também 20 preferências, de maneira que foi gerada uma restrição de horário e uma preferência de horário aleatoriamente para cada um professor.

O algoritmo conseguiu alocar os horários em todas as variações da instância B, ou seja, em todos os quatro cenários da instância ao menos uma das soluções geradas obteve um **número cromático de 25**. Assim, nenhuma aula ficou fora das configurações demandadas pela instância.

Figura 30 – Resultados das FOs com a melhor média da instância B.

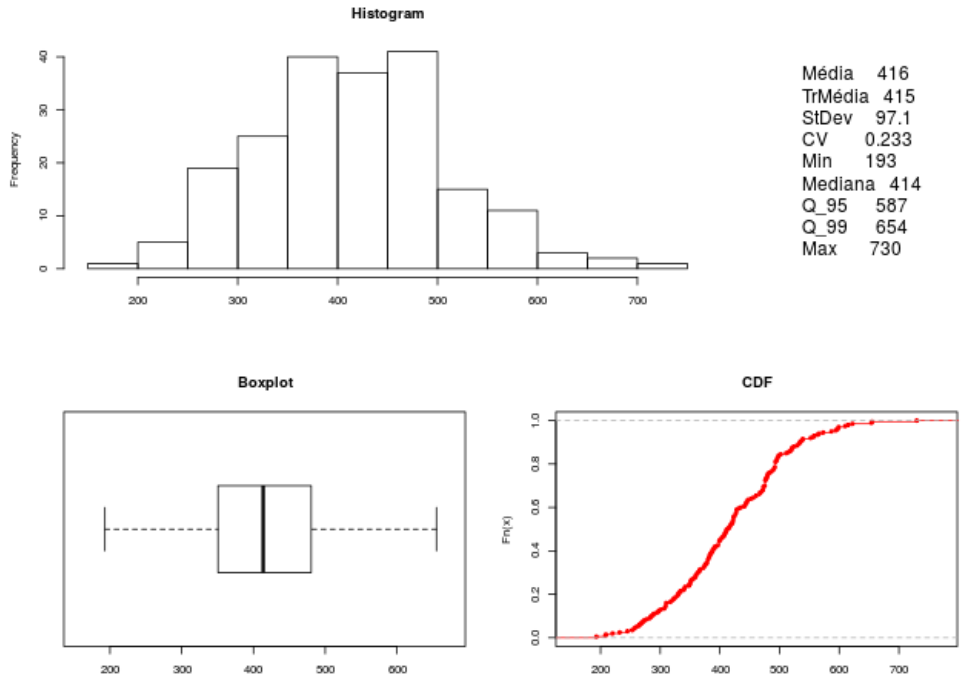


Fonte: Próprio autor

Na Figura 30 podem ser observadas as estatísticas descritivas dos resultados da instância "Escola B", em relação ao cenário no qual o algoritmo obteve soluções com menor penalidade na função objetivo (FO). Esta variação da instância corresponde ao **cenário 2**, que impõe todas as restrições mas nenhuma preferência (SR-1, WR-1 e SR-2), visto que esta instância é bastante complexa, já que não existe muita oferta de horários vagos para encaixar as aulas. A melhor solução gerada pelas 200 execuções com o melhor cenário desta instância foi um quadro de horários com todas as aulas alocadas conforme demanda e restrições específicas da escola, ou seja, sem nenhum aula fora dos horários válidos para esta instância. A quantidade de restrições fracas violadas foram 19 no total, sendo 3 ocorrências de três aulas subsequentes (poligeminação) e 16 aulas interpostas entre aulas que seriam gemináveis (não geminação). Visto que a instância tem no total de 300 aulas, apenas **6,33% no total de aulas infringe** alguma restrição fraca.

Na Figura 31 são apresentadas as estatísticas descritivas dos resultados obtidos para a instância "Escola B", em relação ao cenário no qual o algoritmo obteve soluções com maior penalidade na função objetivo (FO). Esta variação da instância corresponde ao

Figura 31 – Resultados das FOs com a pior média da instância B.



Fonte: Próprio autor

cenário 3, que inclui todas as preferências mas nenhuma restrição (SR-1, WR-1 e WR-2). O motivo dessa variação ter sido a pior em relação as outra é pela falta de mobilidade que o algoritmo tem de alocar os horários, então ele não conseguiu atender tantas preferências, deixando assim a função objetivo de uma maneira geral, com um valor numérico médio alto em relação às outras. A melhor solução gerada pelas 200 execuções com o pior cenário desta instância foi um quadro de horários com todas as aulas alocadas como previsto, ou seja, sem nenhum aula fora dos horários válidos para a instância "Escola B". A quantidade de restrições fracas violadas foi de 14 no total, sendo 13 aulas interpostas entre aulas que seriam gemináveis (não geminação) e 1 ocorrência de três aulas subsequentes (poligeminação). Visto que a instância tem no total de 300 aulas, **4,6% no total de aulas infringem** uma restrição fraca. Em tempo, este cenário continha preferências de horários e a melhor solução gerada conseguiu atender 15 das 20 preferências ou seja **75% das preferências foram atendidas**.

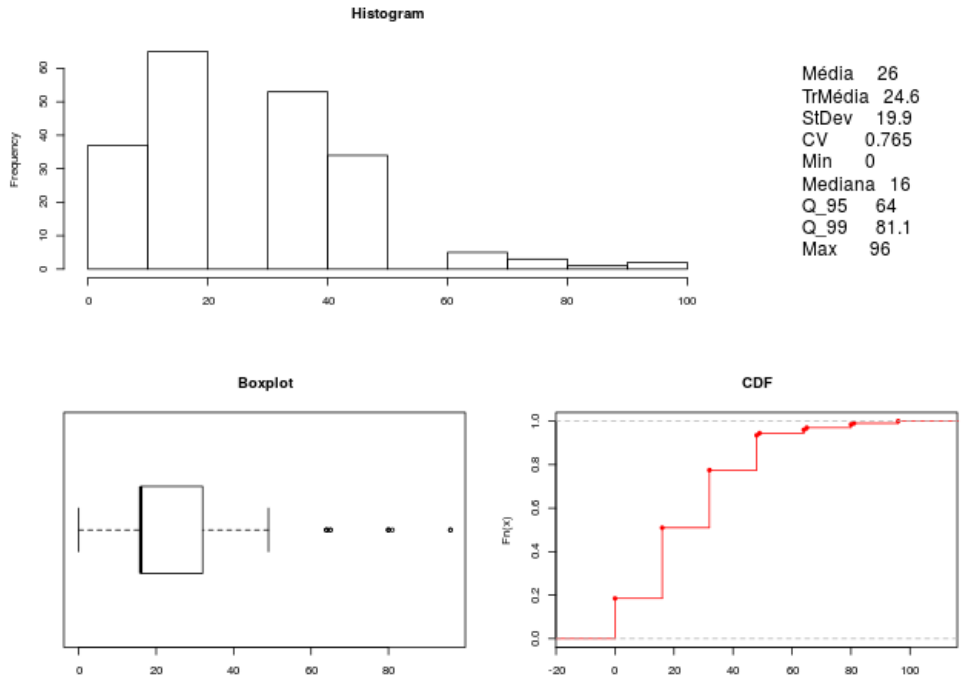
5.3.4 Instância "Escola C"

A instância coletada da escola C é **um pouco menor** em comparação às outras. Esse exemplo representa as aulas de uma escola noturna, a qual quantidade de aulas por turma é bem menor. Esta instância é composta por 4 turmas com 3 delas contendo 16 aulas semanais e somente uma com 17 aulas. Essa aula adicional que é vista em uma das

turmas da instância tem que ser ministrada em um horário extra onde nenhuma outra turma tem aula. Nesta instância, as aulas devem ser alocadas de segunda a sexta e apenas no turno noturno: de 19:00 às 22:15 (ou de 18:15 às 22:15 no caso da turma especial), com aulas de 45 minutos e um intervalo de 20 min. Ou seja, uma solução viável para esta instância deverá ter no **máximo 21 cores** (5 dias na semana * 4 horários noturnos + 1 horário extra em um dia), bem como atender a todas as restrições fortes impostas. Como há 4 turmas, a demanda por alocação de horários é de um total de **65 aulas** (3*16 + 1*17). Foram sintetizadas no total 27 restrições de horários e outras 27 preferências de horários, de maneira que os professores possuíam de zero a três restrições e a mesma quantidade de preferências.

O algoritmo conseguiu alocar os horários em todas as variações da instância C, ou seja, em todas as ocasiões a solução melhor obteve um **número cromático de 21**. Nenhuma aula ficou fora das configurações de horários demandada pela instância. A alocação de horários gerada conseguiu respeitar a peculiaridade da instância "Escola C" de só uma turma conter uma aula a mais que as demais, em um horário semanal extra. Na entrada de dados da instância C, todas as turmas foram modeladas com o 5º horário, mas foi imposto uma restrição para elas (menos a turma especial) para que não tenham mais de 4 aulas diárias.

Figura 32 – Resultados das FOs com a melhor média da instância C.

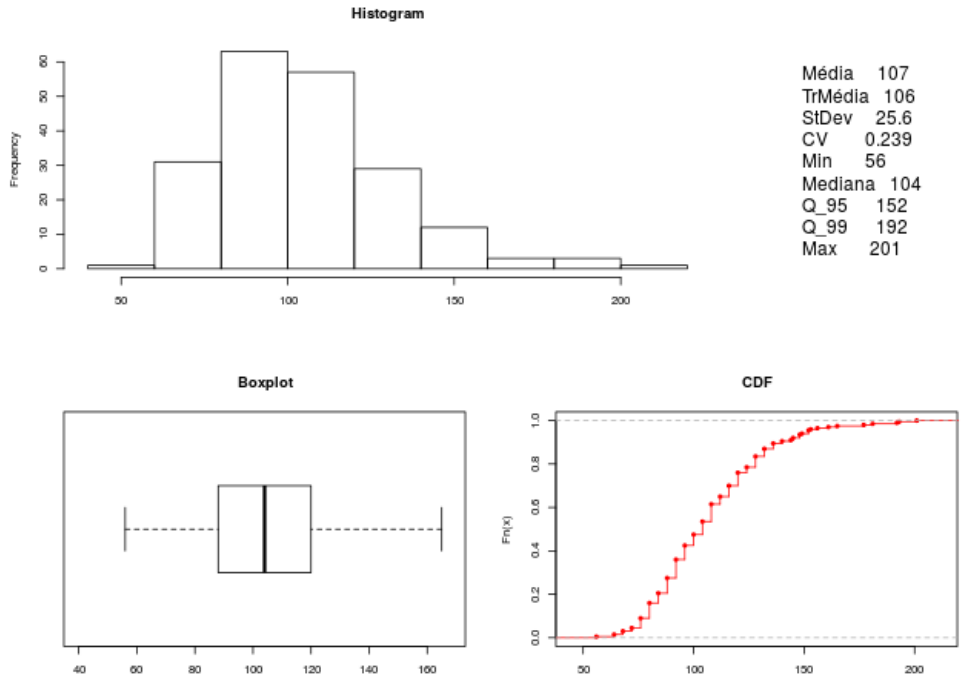


Fonte: Próprio autor

Na Figura 32 podem ser observadas as estatísticas dos resultados obtidos para esta

instância, em relação ao cenário no qual o algoritmo obteve soluções com menor penalidade na função objetivo (FO). Esta variação da instância corresponde ao **cenário 4**, sem nenhuma restrição nem preferência (SR-1 e WR-1, apenas). A melhor solução gerada pelas 200 execuções com o melhor cenário desta instância foi um quadro de horários com todas as aulas alocadas como previsto, ou seja, sem nenhum aula fora do horário configurado na instância. Não houve **alguma ocorrência de restrição fraca violada**, ou seja, não foi gerada lacuna entre aulas, não houve aula interposta entre aulas gemináveis, nem a ocorrência de três aulas subsequentes (poligeminação). Com esta instância o algoritmo pode explorar bastante o espaço de busca pois a quantidade de turmas é bem menor em relação às outras instâncias utilizadas.

Figura 33 – Resultados das FOs com a pior média da instância C.



Fonte: Próprio autor

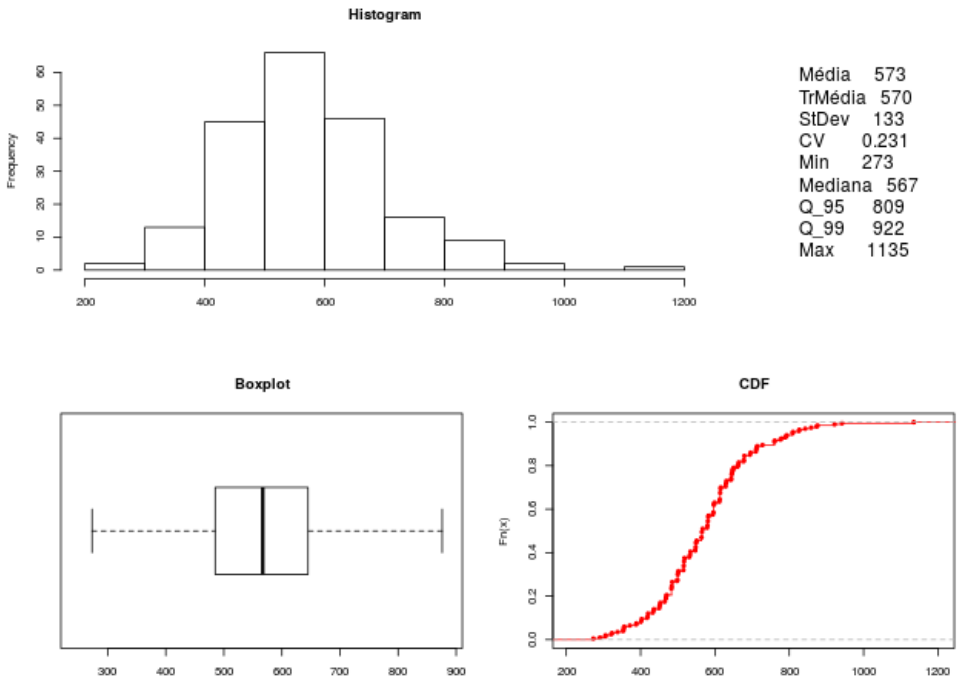
Na Figura 33 são apresentadas as estatísticas descritivas dos resultados obtidos para a instância "Escola C", em relação ao cenário no qual o algoritmo obteve soluções com maior penalidade na função objetivo (FO). Esta variação da instância corresponde ao **cenário 1**, que impõe todas as restrições e preferências (SR-1, WR-1, SR-2 e WR-2). Neste caso o algoritmo obteve a pior média dentre as variações propostas pois nem sempre ele conseguia alocar todos os professores com as preferências sugeridas e ainda assim atender todas as restrições. A melhor solução gerada pelas 200 execuções com o pior cenário desta instância foi um quadro de horários com todas as aulas alocadas como previsto, ou seja, sem nenhum aula fora do horário configurado na instância. Não houve **nenhuma**

ocorrência de restrição fraca violada. Em tempo, este cenário continha preferências de horários e a melhor solução gerada conseguiu atender 12 das 27 preferências ou seja **44,44% das preferências foram atendidas**.

5.3.5 Instância "Escola D"

A instância coletada na escola D é a instância com o **maior número de aulas**: é composta por 15 turmas, sendo que cada turma contém 25 aulas semanais. As 15 turmas da instância são exclusivamente do Ensino Médio, então existem professores que dão aula em quase todas as turmas. Nesta instância, as aulas devem ser alocadas de segunda a sexta e apenas no turno matutino: de 7:00 às 11:30, com aulas de 50 minutos e um intervalo de 20 min. Ou seja, uma solução viável para esta instância deverá ter **no máximo 25 cores** (5 dias na semana * 5 horários matutinos), bem como atender a todas as restrições fortes impostas. Como há 15 turmas, a demanda por alocação de horários é de um total de **375 aulas** (15 * 25). Vale destacar que esta instância é a maior dentre as quatro, tanto em quantidade de aulas (375) quanto na quantidade de turmas (15) turmas. Foram simulados no total 20 restrições de professores e também 20 preferências, sendo que foi simulado uma restrição de horário e uma preferência aleatoriamente para cada um professor.

Figura 34 – Resultados das FOs com a melhor média da instância D.



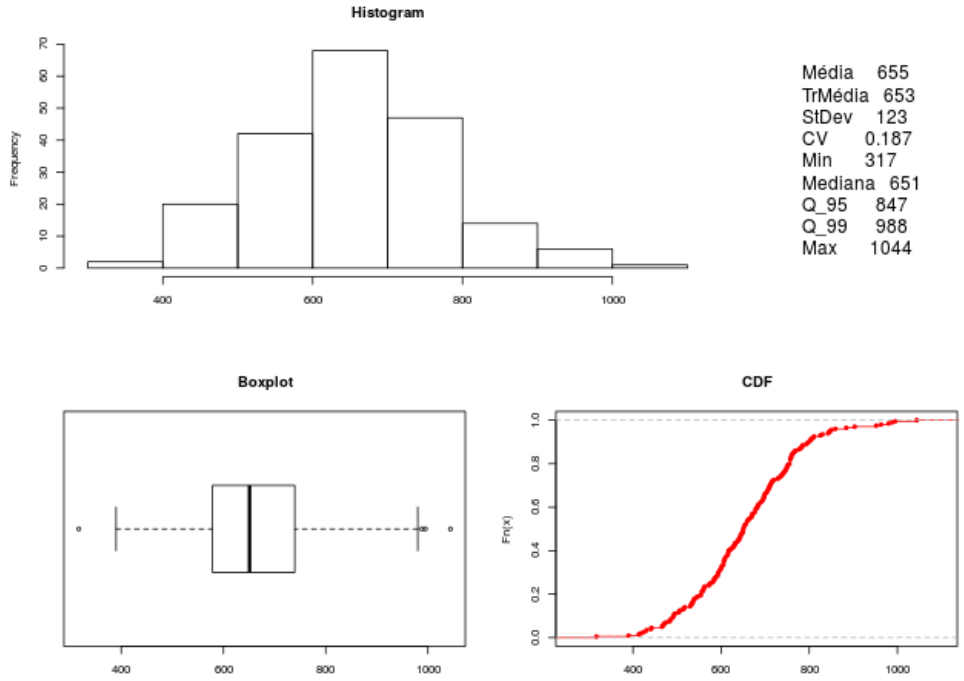
Fonte: Próprio autor

O algoritmo conseguiu alocar os horários em todas as variações da instância D, ou seja, em todas as ocasiões a solução melhor obteve um **número cromático de 25**, ou

seja, nenhuma aula ficou fora das configurações fornecidas pela instância.

Na Figura 34 podem ser observadas as estatísticas dos resultados desta instância, em relação ao cenário no qual o algoritmo obteve soluções com menor penalidade na função objetivo (FO). Esta variação da instância corresponde ao **cenário 2**, com todas as restrições mas nenhuma preferência (SR-1, WR-1 e SR-2). Esta instância é bem parecida com a instância B, não existe nenhum horário vago entre as aulas, então o algoritmo não tem muitas opções para alocar as demandas horários e conciliar as restrições. A melhor solução gerada pelas 200 execuções com esta instância foi um quadro de horários com todas as aulas alocadas como previsto, ou seja, sem nenhum aula fora do horário configurado na instância. A quantidade de restrições fracas violadas foram 24 no total, sendo 5 ocorrências de três aulas subsequentes e 19 aulas interpostas entre aulas que seriam geminâveis. Visto que a instância tem no total de 375 aulas, apenas **6,4% do total de aulas infringe** alguma restrição fraca.

Figura 35 – Resultados das FOs com a pior média da instância D.



Fonte: Próprio autor

Na Figura 35 são apresentadas as estatísticas dos resultados da instância “Escola D”, em relação ao cenário no qual o algoritmo obteve soluções com maior penalidade na função objetivo (FO). Esta variação da instância corresponde ao **cenário 3**, que inclui todas as preferências mas nenhuma restrição (SR-1, WR-1 e WR-2). Neste caso o algoritmo obteve a pior média dentre as variações propostas já que o espaço de busca está muito limitado pela quantidade de aulas e turmas, nem sempre é possível atender às preferências

de todo e cada professor. A melhor solução gerada pelas 200 execuções com esta instância foi um quadro de horários com todas as aulas alocadas como previsto, ou seja, sem nenhum aula fora do horário configurado na instância. A quantidade de restrições fracas violadas foram 19 no total, sendo 3 ocorrências de três aulas subsequentes (poligeminação) e 16 aulas interpostas entre aulas que seriam geminàveis. Visto que a instância tem no total de 375 aulas, apenas **5,06% do total de aulas infringe** alguma restrição fraca. Em tempo, este cenário continha preferências de horários e a melhor solução gerada conseguiu atender 12 das 20 preferências ou seja **60% das preferências foram atendidas**.

5.4 Comparação entre instâncias e seus cenários.

Conforme anteriormente apresentado e analisado, os experimentos de validação do algoritmo foram realizados utilizando as quatro instâncias reais de escolas da região e os 4 cenários sintéticos para cada escola (combinações dos quatro tipos de restrições fortes e fracas). Visando prover uma visão macro do comportamento da metaheurístrica GRASP+VND para cada um desses 16 experimentos de validação, compilamos os resultados em forma tabular.

Para uma melhor compreensão das Tabelas 8 e 9 e conveniência, replicamos aqui quais foram as configurações de cenários definidas na Subseção 5.3.1:

- Cenário 1: SR-1 & WR-1, *Restrições* de horário (SR-2) e **Preferências** (WR-2);
- Cenário 2: SR-1 & WR-1, *Restrições* de horário (SR-2);
- Cenário 3: SR-1 & WR-1, **Preferências** de horário (WR-2);
- Cenário 4: SR-1 & WR-1, apenas.

Bem como apresentamos um sumário das características das quatro instâncias utilizadas nos experimentos apresentados na Seção 5.3:

- Instância A: 330 aulas (11 turmas e 30 h/a), 44 restrições/preferências, sem lacunas
- Instância B: 300 aulas (12 turmas e 25 h/a), 20 restrições/preferências, sem lacunas
- Instância C: 65 aulas (4 turmas e 16 h/a +1), 27 restrições/preferências, sem lacunas
- Instância D: 375 aulas (15 turmas e 25 h/a), 20 restrições/preferências, sem lacunas

A Tabela 8 contém os valores médios das funções objetivo para as melhores soluções geradas para cada instância e cenário. Como o resultado da função objetivo (FO) é apenas um valor numérico de penalidade, sem uma unidade de medida específica, para facilitar a

comparação entre os resultados tais valores de FO foram divididos pelo maior valor de FO de cada coluna, para que os cenários de determinada instância pudessem ser mais facilmente comparados entre si.

Tabela 8 – Funções objetivos normalizadas.

FOs	Instância A	Instância B	Instância C	Instância D
Cenário 1	1	0,9258	1	0,938
Cenário 2	0,7345	**0,8894**	0,2449	**0,8748**
Cenário 3	0,9871	1	0,9816	1
Cenário 4	**0,7319**	0,9158	**0,2385**	0,9083

Fonte: Próprio autor.

Como pode ser observado na Tabela 8, em todas as quatro instâncias (A, B, C e D) os cenários 1 e 3 foram os mais penalizados pela função objetivo e os cenários 2 e 4 os menos penalizados. As instâncias A e C apresentaram comportamento similar em relação aos quatro cenários propostos, tendo lidado melhor com os cenários que não inseriam demandas de preferências de horários, conforme pode-se observar. Uma possível explicação é que, como os cenários 1 e 3 contém em seus dados as preferências de horário sugeridas pelos professores, nem sempre foi possível atender tais preferências. Conforme observado na Seção 5.3, a média de preferência de horários atendidas em todas as instâncias foi de apenas 52% fazendo com que suas funções objetivos penalizassem a solução mais significativamente pela quantidade de infrações de não atendimento a preferências, não necessariamente pelo peso da infração visto que já é o mais baixo dentre as demais infrações (vide Subseção 5.2.1).

Tabela 9 – Tempo médio (segundos) para gerar cada solução

Tempo	Instância A	Instância B	Instância C	Instância D
Cenário 1	**60,06**	**80,59**	**1,14**	**118,74**
Cenário 2	63,88	89,67	1,26	121,92
Cenário 3	70,01	**81,56**	**1,16**	**118,53**
Cenário 4	62,93	88,64	1,22	120,03

Fonte: Próprio autor.

A instância B obteve uma FO média com menor variação entre os diferentes cenários, talvez devido à própria complexidade dessa instância: muitas turmas, menos opções de horários de aula em cada turno e não poderia conter nenhum horário vago entre as aulas. Observando os resultados obtidos para a instância D, vemos que ela se assemelha com o comportamento da instância B, apresentando também uma maior complexidade. Dentre as quatro instâncias, B e C possuem as maiores quantidades de turmas e as menores ofertas de horários de aula, tendo pouco espaço de manobra para a metaheurística criar suas soluções iniciais ou gerar vizinhos a partir delas.

Após a comparação dos resultados médios das FOs, uma tabela foi criada para comparar o tempo médio de cada iteração entre as variações sugeridas de cada instância. Como pode ser observado na Tabela 9, o tempo gasto na geração de cada solução varia relativamente pouco entre os cenários de cada instância. Porém, vale a pena ressaltar que o cenário 1 foi o que demandou menos tempo na geração de cada solução, talvez pelo fato de possuir a maior quantidade de restrições fortes e fracas, reduzindo a mobilidade no algoritmo no espaço de busca pelas soluções.

6 TRABALHOS FUTUROS

Uma possível solução, para o aparente problema de os cenários com preferências de horários estarem sendo penalizados excessivamente, poderia considerar na função objetivo não a quantidade absoluta de preferências de horários não atendidas, mas talvez a proporção de não atendimento. Por exemplo, no caso da instância A, onde apenas 29,5% das preferências foram atendidas (13 em 44), ao invés de penalizar em (44-13)*peso poderia-se penalizar como (44-13)/44)*peso.

Como trabalhos futuros, sugere-se ainda explorar o comportamento de outras metaheurísticas, como o *Simulated Annealing* ou Algoritmo Genético para analisar e comparar os resultados que poderiam ser obtidos utilizando-se a modelagem do problema de *timetabling* como coloração de grafos a estrutura de vizinhança proposta. Uma outra proposta de trabalhos futuros consistiria em adaptar o protótipo de ferramenta elaborado neste trabalho de conclusão de curso para que possa também lidar com *timetabling* universitário, tanto para que possa ser validada a abordagem proposta quanto para auxiliar instituições como o próprio IFMG na recorrente tarefa de alocação de horários.

7 CONSIDERAÇÕES FINAIS

Para as instâncias e cenários utilizados nos experimentos de calibragem e validação, após comparar os gráficos analisar estatísticas de desempenho das metaheurísticas avaliadas, observamos que o GRASP+VND foi a abordagem que consistentemente propôs soluções próximas da solução ótima. Ou seja, soluções com a menor quantidade de horários utilizados, em dias úteis da semana, para encaixar a demanda de horários de aula e respeitar as restrições impostas. A otimização da alocação de horários escolares com utilizando o GRASP+VND apresentou tanto o melhor desempenho em relação à minimização da função objetivo utilizada (qualidade da solução) quanto atingiu a quantidade mínima possível de horários necessários para alocar as aulas (viabilidade da solução). Assim, dentre as abordagens analisadas recomenda-se a utilização do GRASP+VND para resolver o problema da alocação de horários escolares, através de uma transformação em um modelo de coloração de grafos.

Após o desenvolvimento e comparação de desempenho das metaheurísticas com as instâncias obtidas em escolas da região, fica claro que é possível e viável otimizar o processo de geração quadro de horários escolares, respeitando todas as restrições fortes (restrições de horário) e tentando-se ao máximo atender às restrições fracas (recomendações pedagógicas, preferências de horário, dentre outras). Algumas escolas de ensino básico da cidade de Formiga-MG gentilmente disponibilizaram algumas de suas informações sobre as demandas de turmas, professores e turnos. Em agradecimento, será cedido pelo autor deste trabalho o presente protótipo a estas escolas, para que possam experimentá-la e, assim, serem auxiliadas pela ferramenta na geração de quadro de horários factível e viável, consumindo apenas de poucos minutos a algumas horas para obter o resultado, de acordo com o tamanho da instância. Tanto a entrada quanto a saída foram modeladas de forma a mimetizar o formato de quadro de horários popularmente utilizado, facilitando o uso e preenchimento dos dados em uma planilha eletrônica tradicional.

Considerando o curto tempo investido na geração de cada solução de alocação de horários, bem como o atendimento a todos os requisitos fortes e a maximização do atendimento aos requisitos fracos, consideramos que o protótipo produzido tem potencial para crescer e amadurecer. Futuramente, a ferramenta desenvolvida neste trabalho poderia passar por novos ajustes e complementações de funcionalidades para que possa ser explorada como um produto comercial e não somente uma curiosidade acadêmica. Nesta linha, o autor deste trabalho considera a possibilidade de elaborar e integrar a ferramenta com uma interface Web, codificar uma maior gama de restrições de horários que possam ser selecionadas via interface ou personalizadas, tornando a ferramenta mais flexível e menos

dependente de alterações diretamente no código-fonte. Para tal, recomenda-se que seja feita uma pesquisa de mercado sobre as principais soluções de software do ramo educacional, bem como aprofundar-se nas peculiaridades dos horários de aula em cada nível de ensino: fundamental, médio, técnico, graduação e pós-graduação, dentre outros.

REFERÊNCIAS

BIGGS, N. L.; LLOYD, E. K.; WILSON, R. J. *Graph theory 1736-1936*. [S.l.]: Clarendon Press, 2006. Citado na página 19.

BRANDãO, M. A.; MORO, M. M.; ALMEIDA, J. M. Análise de fatores impactantes na recomendação de colaborações acadêmicas utilizando projeto fatorial. *Simpósio Brasileiro de Banco de Dados*, p. 1–6, 2013. Citado na página 36.

CHUEKE, G. V.; AMATUCCI, M. O que é bibliometria? uma introdução ao fórum. *Revista Eletrônica de Negócios Internacionais*, v. 10, n. 2, p. 1–5, 2015. Citado na página 31.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, n. 2, p. 109–133, 1995. Citado 3 vezes nas páginas 22, 23 e 24.

FOUNDATION, P. S. *The Python Profilers*. 2018. Disponível em: <https://docs.python.org/3/library/profile.html#the-python-profilers>. Citado na página 30.

GLOVER, F.; KOCHENBERGER, G. A. *Handbook of Metaheuristics*. [S.l.]: Springer US, 2003. Citado na página 22.

HANSEN et al. Variable neighborhood search and local branching. *Computers & Operations Research*, v. 33, n. 10, p. 3034–3045, 2006. Citado na página 24.

I., D. et al. Trapezoid graphs and their coloring discrete applied mathematics. p. 35–46, 1988. Citado 4 vezes nas páginas 19, 20, 21 e 22.

KARP, R. M. Reducibility among combinatorial problems. *Complexity of Computer Computations*, p. 85–103, 1972. Citado na página 21.

KUBALE, M. *Graph colorings*. [S.l.]: American Mathematical Society, 2004. Citado na página 20.

LAARHOVEN, E. A. P.J. van. *Simulated Annealing: Theory and Applications*. [S.l.]: Springer Netherlands, 1987. Citado na página 25.

LEWIS, R. Graph coloring: An ancient problem with modern applications. p. 47–50, 2016. Citado na página 19.

LUMSDAINE, A. *Sequential Vertex Coloring*. 2018. Disponível em: <https://www.boost.org/doc/libs/1_57_0/libs/graph/doc/sequential_vertex_coloring.html>. Citado na página 21.

MARTINS, A. X. et al. Uma nova estratégia de geração de rotas e um algoritmo vnd aplicado ao problema de roteirização e atribuição de comprimentos de onda. *XLI SBPO 2009 - Pesquisa Operacional na Gestão do Conhecimento*, n. 1, p. 2330–2342, 2009. Citado na página 25.

MEC. *Apresentação - Ministerio da Educação*. 2018. Disponível em: <http://portal.mec.gov.br/institucional/historia>. Citado na página 18.

PANDEY; MOHAN, H. Solving lecture time tabling problem using ga. *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, 2016. Citado na página 26.

PEREIRA, R. S.; NETTO, P. O. B.; LACRUZ, A. J. O método grasp aplicado a um problema de coloração: estudo de caso em uma instituição de ensino fundamental e médio. *X Simpósio de pesquisa operacional e logística da marinha*, n. 1, p. 1–13, 2007. Citado 3 vezes nas páginas 26, 32 e 33.

PILGRIM, M. *Dive into Python*. [S.l.]: Apress, 2008. Citado na página 28.

PINA, J. C. de; FEOFILOFF, P. *Grafos não-dirigidos bipartidos e circuitos ímpares*. 2018. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bipartite.html>. Citado na página 19.

SCHAERF, A. A survey of automated timetabling. *Artificial Intelligence Review*, v. 13, n. 2, p. 87–127, Apr 1999. Disponível em: <https://doi.org/10.1023/A:1006576209967>. Citado na página 17.

SOUZA, M. J. F.; COSTA, F. P. da; GUIMARãES, I. F. G. Um algoritmo evolutivo híbrido para o problema de programação de horários em escolas. *In XXII Encontro Nacional de Engenharia de Produção*, p. 1–8, 2002. Citado na página 26.