UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
Bacharelado em Sistemas de Informação
Natan Cordeiro de Macedo

ALOCAÇÃO DE HORÁRIOS: um algoritmo *timetabling* aplicado à realidade do Departamento de Computação da UFVJM

Diamantina, MG
2018

Natan Cordeiro de Macedo

ALOCAÇÃO DE HORÁRIOS: um algoritmo *timetabling* aplicado à realidade do Departamento de Computação da UFVJM

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação como parte dos requisitos exigidos para obtenção do título de Bacharel em Sistemas de Informação, da Universidade Federal dos Vales do Jequitinhonha e Mucuri

Orientador: Prof. Me. Eduardo Pelli
Coorientador: Prof. Dr. Alessando Vivas Andrade

Diamantina, MG
2018

Natan Cordeiro de Macedo

ALOCAÇÃO DE HORÁRIOS: um algoritmo *timetabling* aplicado à realidade do Departamento de Computação da UFVJM

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação como parte dos requisitos exigidos para obtenção do título de Bacharel em Sistemas de Informação, da Universidade Federal dos Vales do Jequitinhonha e Mucuri

Orientador: Prof. Me. Eduardo Pelli

Aprovado em: 28 / 01 / 19 .

Prof. Me. Eduardo Pelli
Orientador

Prof. Dr. Alessando Vivas Andrade
Coorientador

Profa. Dra. Caroline Queiroz Santos
UFVJM - Universidade Federal dos Vales Jequitinhonha e Mucuri

Profa. Dra. Luciana Pereira de Assis
UFVJM - Universidade Federal dos Vales Jequitinhonha e Mucuri

Diamantina, MG

Este trabalho é dedicado à minha família, em especial à minha mãe, que sempre colocou minha educação em primeiro lugar.

AGRADECIMENTOS

Agradeço primeiramente à minha mãe que sempre lutou para que eu pudesse ter a melhor educação possível dentro de todas as limitações que nos foram impostas. Agradeço também aos familiares que me apoiaram direta ou indiretamente a seguir meu caminho, àqueles que suportaram minha ausência para que eu pudesse crescer e aos meus sobrinhos que são a alegria da minha vida.

Agradeço também a pessoas especiais que mudaram minha vida e me apoiaram quando me faltava esperança e motivação para continuar, Rodrigo Miranda pelo seu companheirismo em todos os aspectos da minha vida, Fábio, Boydu, Guilherme e seus avós, vocês são irmãos para a vida e o que fizeram por mim eu nunca esquecerei. *You don't have to be blood to be family*.

Obrigado às pessoas que marcaram minha convivência em Diamantina, morar com vocês foi um aprendizado e tanto, Bruno, Mateus e Júnior, continuem lutando pelos seus sonhos, acredito no potencial de cada um de vocês.

A meus amigos da faculdade, Carol pelo apoio mútuo do início ao fim da graduação, uma irmã como você não será esquecida. Elias por nunca me deixar parar de absorver novos conhecimentos. À preciosa amizade de Magno, Pedro Estanislau, Luiz Otávio e Brian, responsáveis por sempre me apresentarem novas oportunidades e pela disposição em me ajudar sempre que possível.

Agradeço às pessoas que fizeram minha experiência de estágio na UFVJM ser muito enriquecedora, Íngrid Oliveira, Dani, Amanda, Thales, Gabi, Marina, Lucy, Rodrigo e à toda Dicom. Agradeço também à NextStep por possibilitar enriquecimento como pessoa e profissional da área, sem tais experiências eu não estaria hoje estagiando em uma empresa de tecnologia em Belo Horizonte.

Ingrid Sena, sua luta como mulher me mantém sempre atento à como o feminismo é importante e como cada um de nós pode ser ator da mudança que queremos no mundo. Obrigado pela amizade, ensinamentos e companheirismo.

Não poderia deixar de agradecer também aos professores do curso, que com profissionalismo e dinamicidade foram capazes de passar seu conhecimento em sala de aula de forma acessível e simples.

“Penso noventa e nove vezes e nada descubro; deixo de pensar, mergulho em profundo silêncio - e eis que a verdade se me revela.
(Albert Einsten)

RESUMO

A partir da observação da dificuldade enfrentada pela coordenação e corpo docente do Departamento de Computação da Universidade Federal dos Vales Jequitinhonha e Mucuri, que a cada semestre letivo se reúne para alocar manualmente os horários das aulas, os quais precisam estar adequados a várias restrições incluindo as relacionados às disponibilidades de cada docente, este trabalho propõe a criação de um algoritmo capaz de, por volta de 100ms, resolver a alocação de horários respeitando todas as restrições impostas. O *Schedule Teachers Semester Classes* foi criado usando a linguagem Java, bibliotecas para leitura e escrita em arquivos de formato aberto e manipulação de matrizes, e uma *Application Program Interface* da *International Business Machines* capaz de criar e resolver problemas de otimização matemática. O algoritmo desprende menor esforço do corpo docente para a tarefa, com configurações flexíveis que o permitem ser usado por quaisquer departamentos de quaisquer universidades enfrentando o mesmo problema.

Palavras-chave: Alocação de horários. Otimização. *Schedule Teachers Semester Classes*.

ABSTRACT

From the observation of the difficulty faced by the coordination and faculty of the Department of Computing of the Universidade Federal dos Vales Jequitinhonha e Mucuri, that each semester of school meets to manually allocate the class schedules, which must be adapted to several restrictions including the related with the availability of each teacher, this work proposes the creation of an algorithm capable of, around 100ms, solve the allocation of schedules respecting all restrictions imposed. The Schedule Teachers Semester Classes was created using the Java language, libraries for reading and writing in open format files and matrix manipulation, and an International Business Machines's Application Program Interface capable of creating and solving mathematical optimization problems. The algorithm gives less effort to faculty for the task, with flexible configurations that allow it to be used by any departments of any universities facing the same problem.

Keywords: Timetabling. Optimization. Schedule Teachers Semester Classes.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

UFVJM	Universidade Federal dos Vales Jequitinhonha e Mucuri
Decom	Departamento de Computação
PPH	Problema de Programação de Horários
PPHU	Problema de Programação de Horários em Universidades
PPHBC	Problema de Programação de Horários Baseada em Currículo
PADPH	Problema de Programação de Horários em Universidades - Professor/Disciplina $\times$ Horário
STSC	*Schedule Teachers Semester Classes*

LISTA DE SÍMBOLOS

Π	Letra grega Pi
$\in$	Pertence
$\forall$	Para todo
$\subset$	Está contido
$\sum$	Letra grega Sigma, usada para representar somatório

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho propõe melhorar a forma como é feita a alocação dos horários das aulas pelo Departamento de Computação da Universidade Federal dos Vales Jequitinhonha e Mucuri para o curso de Sistemas de Informação.

Os professores do departamento se reúnem a cada semestre letivo para alocar os horários das matérias, considerando suas restrições de disponibilidade e quantidade de aulas necessárias por disciplinas. Até então, a resolução do problema de alocação de horários dos professores era feito manualmente. A criação do algoritmo STSC capaz de resolver o problema foi proposto considerando a existência de sistemas já capazes de resolver o problema mas de forma paga [1] e considerando dimensões muito maiores do que seria necessário para o departamento. O STSC precisava ser capaz de considerar as restrições de disponibilidade dos professores além das restrições inerentes a este tipo de problema (MARTI, 2010; WILLEMEN, 2002; MÜLLER, 2005; ALMOND, 1966; NANDA; PAI; GOLE, 2012).

O desenvolvimento desse algoritmo dependeu da utilização de:

- Linguagem de programação Java, uma linguagem orientada a objetos e multi-plataforma (GOSLING; MCGILTON, 1995; DEITEL, 2009; ARNOLD; GOSLING; HOLMES, 2005; HEFFELFINGER, 2015);
- Ferramentas para medição de qualidade do código (CONTINUOUS, s.d.; ANALYZING, s.d.);
- Regras e sugestões sobre escrita de código de qualidade e de fácil leitura e manutenção (MARTIN, 2009);
- Bibliotecas para manipulação de arquivos e matrizes (ver seção 2.9)
- uma API para Java capaz de criar e resolver modelos de otimização matemática (CORPORATION, 2017a; 2017b).

Um algoritmo exato para encontrar uma solução ótima para o problema de alocação de horários, tem crescimento exponencial do tempo de execução a partir do tamanho de sua entrada, devido à característica combinatória do problema, além disso, *timetabling* é um problema NP-difícil ou NP-Completo (MÜLLER, 2005; MARTI, 2010).

Com modelos matemáticos de otimização, é possível encontrar soluções ótimas para problemas computacionais como o de *timetabling* em tempo polinomial, caso seja possível representa-lo. Para implementar uma técnica de busca baseada em modelo de otimização, a classe responsável por essa busca e as classes responsáveis por representar o resultado de uma grade de horários com aulas alocadas foram separados em pacotes diferentes, a fim de garantir

[1] ascTimetables: https://www.asctimetables.com/
Teacher Schedule Maker: https://www.humanity.com/education-scheduling-software

menor dependência entre as partes. Graças à utilização do *CPLEX Api* para Java, foi possível encontrar uma solução que respeite às restrições inerentes ao problema, além de respeitar as disponibilidades dos professores durante a semana, restrições que foram inseridas ao problema como restrições fortes.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é a criação do STSC, algoritmo capaz de automatizar através de otimização matemática, a alocação dos horários das aulas dos professores do curso de Sistemas de Informação da UFVJM.

1.1.2 Objetivos Específicos

- Estudar linguagens e ferramentas que possibilitem a criação de um algoritmo capaz de usar otimização matemática
- Estudar as variantes do problema de alocação de horários
- Facilitar o processo de definição de horários das aulas para os professores do curso

1.2 Estrutura do Trabalho

Este trabalho possui 5 capítulos, o Capítulo 1, atual, estabeleceu o atual problema enfrentado pelo departamento do curso, alocar manualmente os horários dos professores.

No Capítulo 2 são apresentados uma revisão de conceitos básicos para entendimento do trabalho, dentre eles: modelos de otimização, problemas de alocação de horários em universidades e complexidade do problema. Posteriormente, as principais ferramentas utilizadas são apresentadas: a linguagem Java, *CPLEX Api Java*, a IDE Netbeans e ferramentas e conceitos que se preocupam com a qualidade de escrita do código, focado na facilidade de leitura do mesmo.

O Capítulo 3 apresenta os métodos para a criação do algoritmo, envolvendo a modelagem do problema e a representação abstraída de objetos reais em objetos computacionais através de classes.

No Capítulo 4 são apresentados os testes de qualidade e desempenho do algoritmo com múltiplas instâncias de problemas além da análise dos mesmos.

O Capítulo 5 apresenta as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo tem como por objetivo apresentar conceitos básicos para o entendimento do trabalho. A seção 2.1 apresenta conceitos relacionados a modelos de otimização e a subseção 2.1.1 sobre a classificação de complexidade de problemas computacionais. A seção 2.2 aprofunda no que se define um problema de alocação de horários e a seção 2.3 no que se define o problema para universidades. A seção 2.4 apresenta a linguagem de programação Java, as subseções 2.4.1 e 2.4.2 aprofundam sobre a linguagem. A seção 2.5 mostra o que é a IDE NetBeans para desenvolvimento Java. A seção 2.6 apresenta um conjunto de ferramentas para implementar problemas de otimização matemática e a subseção 2.6.1 mostra uma dessas ferramentas, direcionada para Java. As seções 2.7 e 2.8 abordam conceitos de escrita de código limpo e uma ferramenta para medir qualidade do mesmo, respectivamente. Por fim, a seção 2.9 mostra as bibliotecas Java usadas para manipulação de arquivos e matrizes.

2.1 Modelo de Otimização

Um modelo não é igual à realidade, mas suficientemente similar para que as conclusões obtidas pela análise do modelo possam ser estendidas à realidade. Para a formalização de um modelo é necessário definir a estrutura relacional do sistema representado, o comportamento funcional de cada parte do sistema e os fluxos de inter-relacionamento (GOLDBARG; LUNA, 2005). A figura 1 resume essas dimensões. O eixo de domínio se refere ao tipo de complexidade e variáveis do sistema, o eixo de meio ambiente se refere à tratabilidade do problema e o eixo restante trata da determinabilidade do problema.

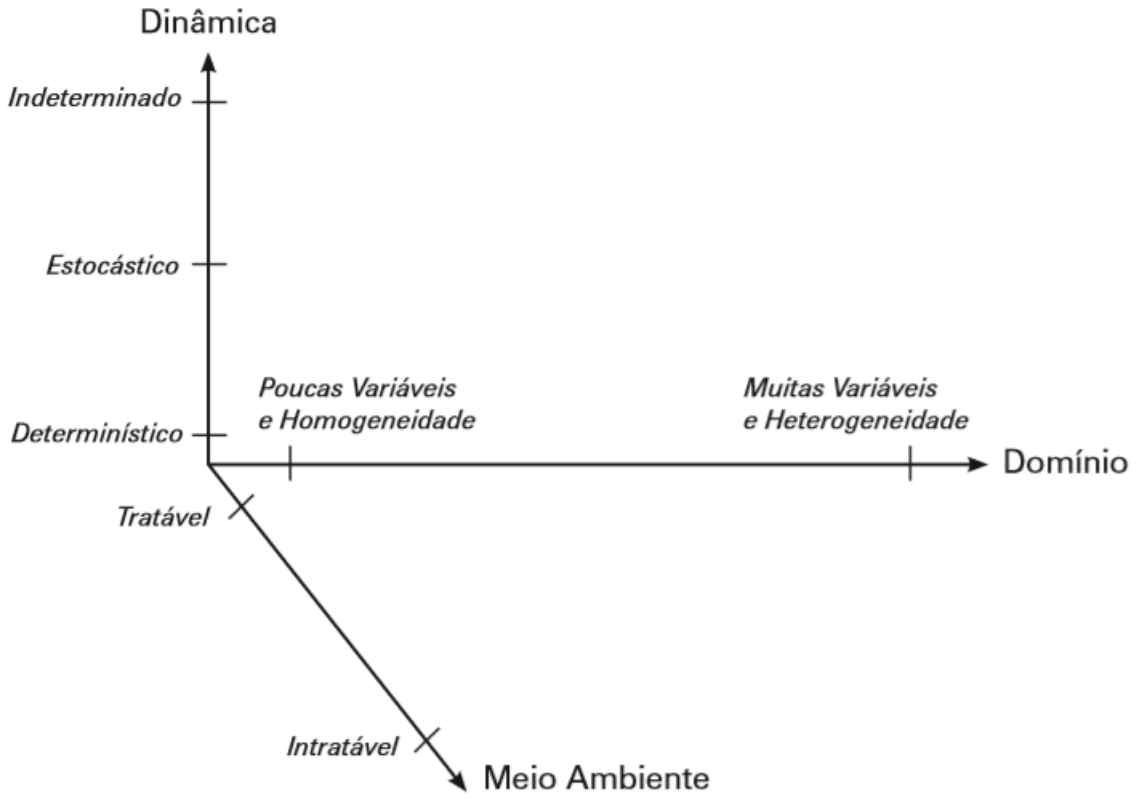
Um problema de otimização foi formalizado matematicamente em Goldbarg e Luna (2005), mas é basicamente composto por: uma função objetivo (minimizar ou maximizar), e restrições sobre as variáveis que compõe a função objetivo. O que se procura é a *configuração* (ver definição 1) que atenda ao problema (GOLDBARG; LUNA, 2005).

Definição 1 *Tendo um problema formulado* Π*, entendemos por configuração um arranjo de variáveis que atende às restrições e leva ao melhor valor que atenda ao objetivo do problema. Por exemplo:*

- *Para o problema da dieta: uma configuração é um conjunto de quantidades de alimentos, que garantem o mínimo da necessidade nutricional diária.*
- *Para o problema do sapateiro: um conjunto de quantidades inteiras de sapatos e cintos que o sapateiro deve produzir para melhor aproveitar o couro (recurso) disponível (*GOLDBARG; LUNA*, 2005).*

Para ambos os exemplos, a configuração não precisa ser a melhor, desde que se adeque às restrições. Se considerarmos a melhor configuração (e então estaremos otimizando)

Figura 1 – Dimensões da Complexidade de Modelos



Fonte: (GOLDBARG; LUNA, 2005)

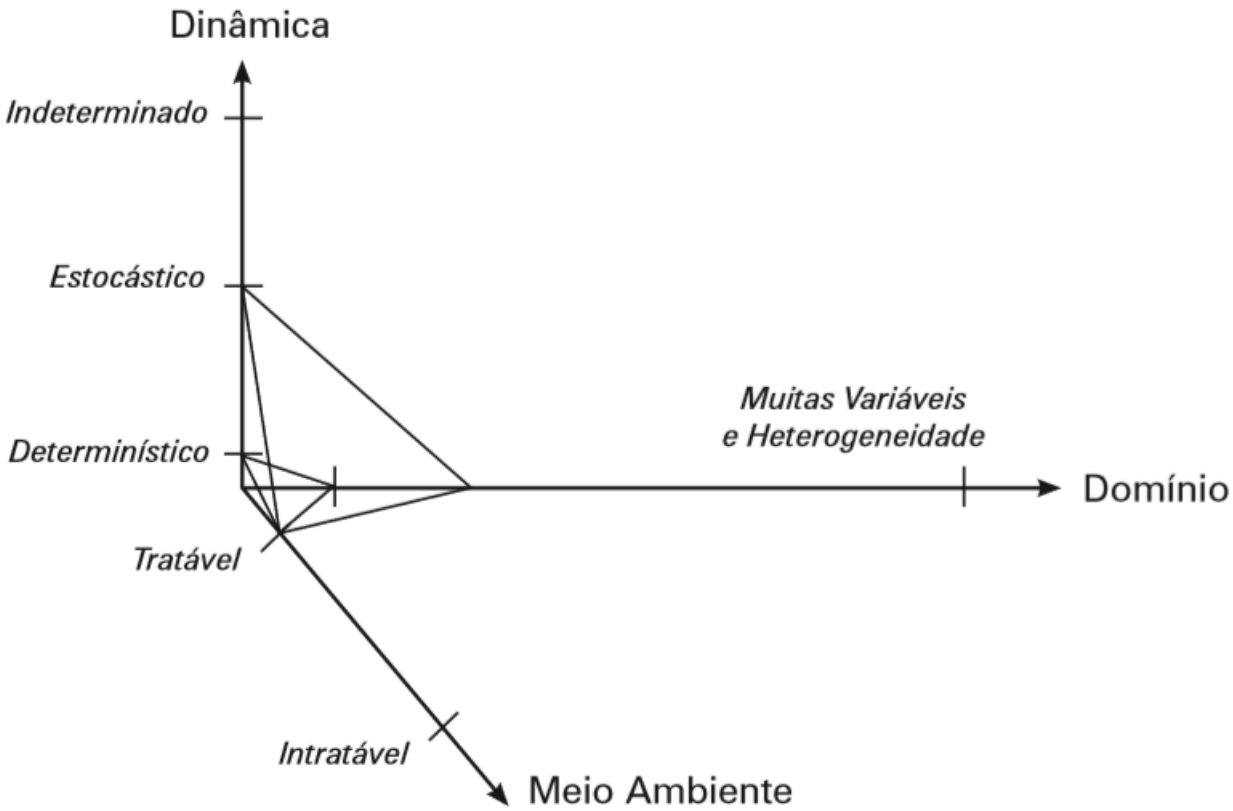
*teremos respectivamente: uma configuração que além de se adequar as necessidades nutricionais mínimas, garanta o menor gasto possível com os alimentos e uma que garanta a maximização dos lucros (*GOLDBARG; LUNA, 2005*;* WILLEMEN, 2002*).*

2.1.1 Complexidade de Problemas Computacionais

Algoritmos que resolvem problemas computacionais possem ser classificados de diversas formas, entre elas, por sua complexidade e natureza. Considerando um problema computacional Π, pela sua natureza podemos classifica-lo entre:

- **Problemas de Decisão**: quando o objetivo de Π é decidir sobre a existência de uma configuração S que atenda as propriedades exigidas em em Π . A resposta para esse tipo de problema é sim ou não (GOLDBARG; LUNA, 2005).
- **Problemas de Localização**: quando o objetivo de Π é localizar ou mostrar uma configuração S que atenda à resposta *sim* de um problema de decisão ou mostrar que a mesma não existe. A resposta para esse tipo de problema é exibir uma S válida (GOLDBARG; LUNA, 2005).

Figura 2 – Espaço de atuação dos modelos de otimização



Fonte: (GOLDBARG; LUNA, 2005)

- **Problemas de Otimização**: para esse tipo de problema, além de estarmos interessados uma configuração S que atenda as restrições do problema Π, estamos interessados em escolher S através de algum critério de otimização (GOLDBARG; LUNA, 2005).

Quanto à sua complexidade, os problemas podem ser divididos em:

- **Classe de problemas** P: são os problemas que admitem um algoritmo de resolução em tempo polinomial(GOLDBARG; LUNA, 2005).

- **Classe de problemas** NP: compreende todos os problemas de decisão Π, em que o reconhecimento de que uma entrada E é válida pode ser realizado por um algoritmo polinomial e que exista uma justificativa à resposta sim para Π (GOLDBARG; LUNA, 2005).

Observe que a classe de problemas NP não obriga a existência de uma solução polinomial para um problema Π, apenas exige que o tamanho da justificativa seja polinomial em relação a entrada E (GOLDBARG; LUNA, 2005).

Todo problema P é automaticamente NP ($P \in NP$) mas a recíproca ($NP \in P$) é um problema ainda sem resposta. Quando podemos transformar, em tempo polinomial, um problema de decisão Π_1 em Π_2, escrevemos $\Pi_1 \propto \Pi_2$. Através disso, Garey e Johnson (1979) demonstra que Π_2 é pelo menos tão difícil quanto Π_1 se:

- $\Pi_1 \in P \Rightarrow \Pi_2 \in P$
- $\Pi_1 \notin P \Rightarrow \Pi_2 \notin P$

Classe de problemas NP-Completo

O conceito de problemas NP-completo foi dado por (COOK, 1983), em seu trabalho foi demonstrado que era possível dividir problemas NP em classes equivalentes. Assim a classe NP-completo ficou definida como os problemas que atendam aos requisitos da definição 2.

Definição 2 *Um problema pertence à classe NP-completo se:*

1. $\Pi \in NP$
2. $\forall\, \Pi' \in NP \Rightarrow \Pi' \propto \Pi$

2.2 Problemas de Programação de Horários

O problema de programação de horários é um problema de otimização combinatória muito conhecido, há vários trabalhos relacionados à tentativa de resolução do mesmo desde 1960. É uma tarefa simples de explicar mas difícil de realizar devido à sua natureza. O mesmo pertence a classe NP-Completo ou NP-Difícil dos problemas computacionais a depender de sua variação (WILLEMEN, 2002; MÜLLER, 2005; SANTOS; SOUZA, 2007; ALMOND, 1966).

O trabalho de Willemen (2002) lista variantes do problema de alocação de horários e algoritmos que os resolvam, além de estudar a complexidade desses problemas em relação ao tempo computacional para que um algoritmo exato os resolva.

O trabalho de Müller (2005) estuda algoritmos aproximados para a resolução do problema de alocação de horários e pesquisa iterativa direcionada, além de formular e comparar soluções e resultados experimentais.

Santos e Souza (2007) faz adaptações de técnicas genéricas como programação linear inteira mista e meta-heurísticas para resolver o PPH, dentre elas a busca tabu, *Simulated Annealing* e algoritmos evolutivos.

Almond (1966) implementa uma heurística aproximada para o PPHU aplicado à realidade do departamento de matemática e faculdade de ciências na *University of London* no *Queen Mary College*.

Apesar de ser uma tarefa bem definida, não é simples e sua complexidade aumenta conforme o crescimento da quantidade de disciplinas, professores e restrições, o tempo computacional para encontrar a solução ótima desses problemas através de algoritmos exatos cresce

exponencialmente devido à sua característica combinatória. Um exemplo clássico do problema de *timetable* é encontrada em instituições de ensino, quanto ao *agendamento de aulas* (SANTOS; SOUZA, 2007; MARTI, 2010).

Sendo assim, são necessárias estratégias alternativas para resolução desses problemas. Entre elas estão as heurísticas, meta-heurísticas e algoritmos evolutivos que objetivam encontrar soluções boas, mas não necessariamente ótimas, e a programação linear através de modelos de otimização.

Considerando a modelagem de um Problema de Programação de Horários, primeiramente devem ser definidas as restrições. Essas restrições devem ser colocadas em termos do problema e geralmente são de dois tipos (SANTOS; SOUZA, 2007):

- **Restrições fortes:** restrições que precisam ser satisfeitas pois a implementação de um quadro de horários não é possível sem satisfaze-las. Por exemplo: a disponibilidade de um professor deve ser atendida para que o problema seja factível
- **Restrições fracas:** são restrições que a satisfação é desejável mas caso não sejam alcançadas, o quadro de horários ainda pode ser implementado. Por exemplo: uma disciplina pode ter todas as suas aulas alocadas juntas ou podem estar separadas, que a solução ainda é factível.

Diversas outras restrições podem ser consideradas, dependendo da necessidade da instituição de ensino envolvida. Em Marti (2010) são identificadas três categorias de requisitos que implicam em restrições do problema:

1. **Requisitos organizacionais**, que se referem à instituição de ensino: alocação de salas, laboratórios, legislação, carga horária semanal mínima/máxima de cada professor, carga horária total de cada disciplina
2. **Requisitos pedagógicos**, visam o bom aproveitamento das aulas: duração e distribuição das aulas ao longo da semana
3. **Requisitos pessoais**, relacionadas as preferências ou necessidades do corpo docente

Cada requisito impõe diferentes restrições ao problema. Um quadro de horários que viole alguma restrição forte não é *factível*, pois não existe solução válida que atenda às restrições da instituição e/ou corpo docente (MARTI, 2010).

2.3 Programação de Horários em Universidade

O Problema de Programação de Horários em Universidades (*University Course Timetabling*), abreviadamente PPHU, se baseia em associar um conjunto de aulas para cada disciplina, em dado número de salas e horários. O conceito de turma não é tão forte, pois os

alunos podem personalizar seus horários. O foco é dado portanto às disciplinas, visando os estudantes com disciplinas em comum e tentando evitar que tais disciplinas sejam alocadas no mesmo horário.

Outra grande diferença se dá em relação às dimensões do problema. Em universidades a restrição de disponibilidade de recursos é maior, mais disciplinas precisam ser alocadas em uma quantidade limitadas de salas.

Uma das variantes do PPHU é o **Problema de Programação de Horários Baseada em currículo (PPHBC)**, essa variante foca a alocação de horários desconsiderando a matrícula dos alunos nas disciplinas, e focando em atender às restrições do problema associadas à construção de uma grade de horários que atenda os requisitos do currículo. O PPHBC considera:

- Um conjunto de d disciplinas, $D = \{1, \ldots, d\}$, em que cada $j \in D$ deve ser associada a uma quantidade q_j de horários
- c *curricula* (ver definição 3), $C = \{1, \ldots, c\}$, sendo que disciplinas pertencentes a um curricula C_v com $v \in C$ não pode ser associadas a um mesmo horário
- h horários, $H = \{1, \ldots, h\}$, tal que cada horário $k \in H$ está associado a um valor a_k, que é o número máximo de aulas que podem ser ministradas a este horário. a_k é usado quando se considera quantidade de salas para resolução do problema, servindo para impedir que alguma turma fique sem sala.
- p professores, $P = \{1, \ldots, p\}$, em que cada professor tem um conjunto de horários em que não está disponível para dar aula.

Definição 3 *Um Curricula, neste contexto é definido por um conjunto de disciplinas que pertençam a um mesmo período de um curso.*

2.3.1 Problema de Programação de Horários em Universidades - Professor/Disciplina × Horário (PADPH)

Para resolver este problema é necessário que o Problema Professor × Disciplina (PAPD) descrito em Marti (2010) já esteja resolvido, ou seja, que todos os professores já estão associados às suas respectivas disciplinas. E em como Almond (1966) a entrada sejam os dados: quantidade de aulas para cada disciplina e disponibilidade de cada professor durante a semana.

Existem vários conflitos que são necessários serem resolvidos no PADPH, alguns deles são obrigatórios para uma solução seja factível, outros dependerão da legislação e escolha de cada universidade.

Conflitos intra-curricula: através da definição 3 sabemos que dado um curricula C_v, para cada duas disciplinas $d_1, d_2 \in C_v$ para todo $k \in H$, se d_1 está associado a k então d_2 obrigatoriamente não pode estar associado a k e a reciproca é verdadeira.

Conflitos inter-curricula: um professor b qualquer em que $b \in P$, $P = \{1, \ldots, p\}$ de professores pode ter disciplinas que pertençam a diferentes curriculas. Considerando C_b o conjunto de disciplinas que o professor b ministra, para cada duas disciplinas $d_1, d_2 \in C_b$ para todo $k \in H$, se d_1 está associado a k então d_2 obrigatoriamente não pode estar associado a k e a reciproca é verdadeira.

Indisponibilidade: por vários motivos um professor pode estar ou não disponível para dar aula em um determinado horário $k \in H$, sejam por reuniões pedagógicas ou até mesmo preferência por reunir todas suas aulas de uma disciplina em dois ou três dias da semana.

Aulas em horários consecutivos: uma universidade pode escolher limitar a quantidade de aulas consecutivas de uma mesma disciplina, por acreditar que o aproveitamento dos alunos será melhor se não forem dados quatro horários seguidos da mesma, ou pode acreditar que se uma matéria tem no máximo três aulas por semana, que as mesmas deveriam ser dadas juntas.

Para permitir que essas restrições sejam atendidas é preciso considerar essas situações na resolução do problema. Há outros conflitos, que são opcionais para a resolução do problema, e dependerão de cada universidade ou departamento aderir ou não, são eles:

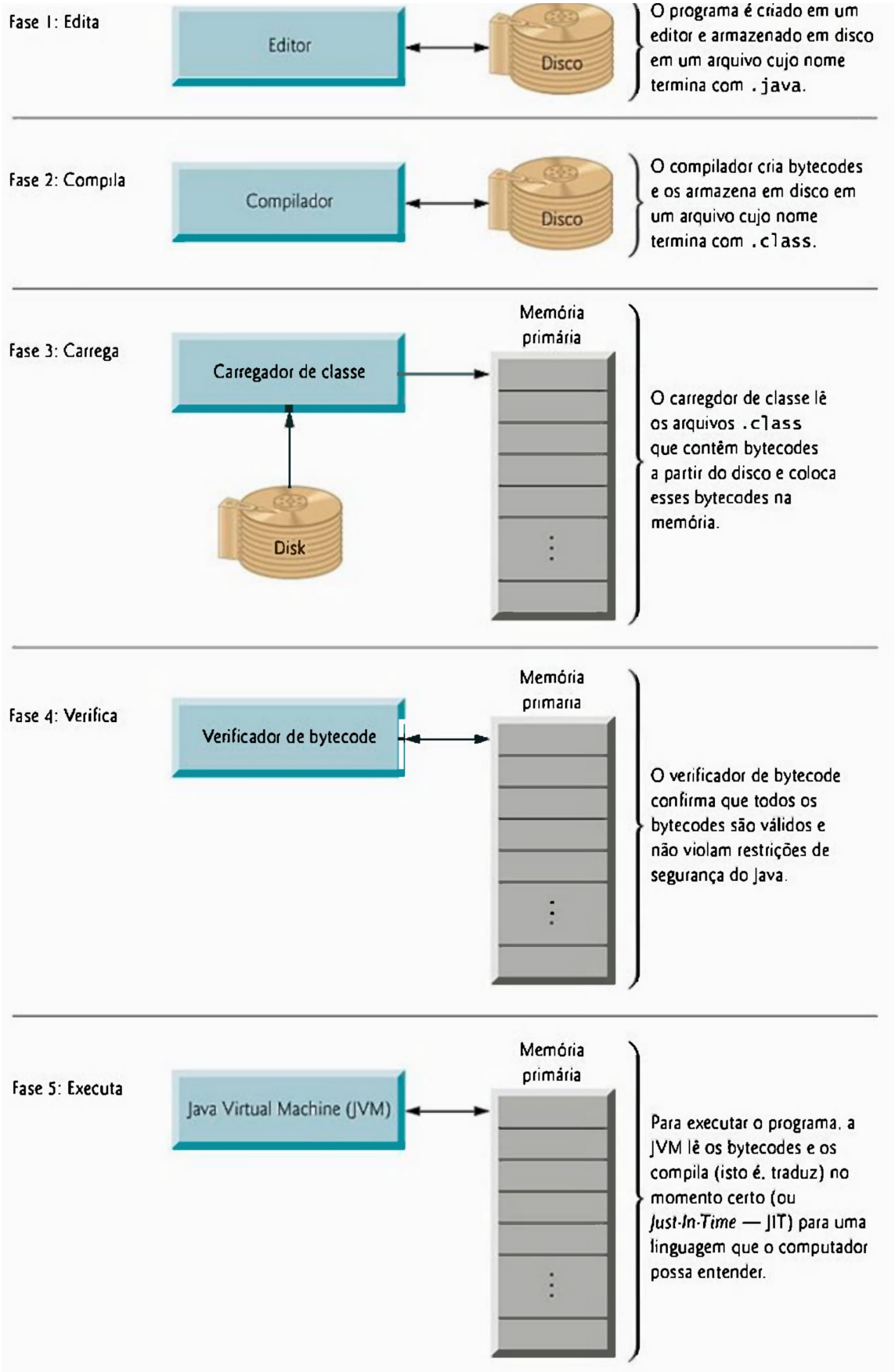
- Adequação ao turno do curso, caso o curso tenha vários turnos
- Estudantes próximos a concluir o curso, a ideia é fazer com que as aulas em que esses estudantes se matriculem não tenham conflitos para facilitar a conclusão do curso pelos mesmos

2.4 Java

Java é uma linguagem de programação de alto nível (ARNOLD; GOSLING; HOLMES, 2005; DEITEL, 2009) baseada em outra linguagem, o C++, começou a ser criada em 1991 pela *Sun Microsystems* e foi anunciada oficialmente em 1995, chamando atenção da comunidade de negócios pelo enorme interesse na Web. A linguagem é usada atualmente para desenvolver aplicativos de grande porte, aprimorar servidores da Web e outros propósitos.

Para um programa escrito em Java funcionar são necessárias várias etapas, que garantem não só o funcionamento do programa como um todo mas a portabilidade do mesmo entre plataformas, as etapas desde edição do programa até seu funcionamento estão descritos na imagem 3. Um dos principais responsáveis para garantir o funcionamento entre plataformas de programas Java é a *Java Virtual Machine* (JVM) ou Máquina Virtual Java, em tradução literal, ela é responsável por traduzir os *bytecodes* para a a linguagem da máquina em que está instalada.

Figura 3 – Funcionamento do Java



Fonte: (DEITEL, 2009)

2.4.1 Desenho da linguagem

Os requisitos de desenho da linguagem foram dirigidos pela natureza dos ambientes computacionais em que o software deveria ser colocado. O grande crescimento da internet liderou a uma nova forma de olhar o ambiente de distribuição de software, e portanto em seu ano de lançamento com o crescimento de comércio eletrônico a linguagem teve que ser capaz de oferecer um ambiente seguro, de alta performance e altamente robusto para várias plataformas e em redes distribuídas heterogêneas (GOSLING; MCGILTON, 1995).

Java se propõe a ser uma linguagem simples, fácil de aprender, familiar (desenvolvedores de outras linguagens podem aprende-la facilmente), orientada a objetos, *multi-thread* [1] e interpretada, para garantir máxima portabilidade.

2.4.2 Orientada a Objetos

De acordo com Gosling e McGilton (1995) Java é uma linguagem orientada a objetos "de cima a baixo". E essa escolha não se deu apenas para seguir a última moda de linguagem de programação, o paradigma se mistura com a necessidade de softwares distribuídos e arquitetura cliente-servidor. Os benefícios de seguir o paradigma são facilmente percebidos quanto mais as organizações mudam suas aplicações para o modelo cliente-servidor.

Para ser considerada orientada a objetos uma linguagem de programação deve ter no mínimo as características que se seguem:

1. Encapsulamento: implementa modularidade e oculta informações (abstração)
2. Polimorfismo: a mesma mensagem enviada a diferentes objetos resultarão em comportamentos que dependem da natureza dos objetos que receberam as mensagens
3. Herança: deve possível definir novas classes a partir de classes existentes a fim de reutilizar e organizar código
4. Ligação dinâmica (*dynamic binding*): objetos podem vir de qualquer lugar, por exemplo através da rede. O desenvolvedor deve ser capaz de enviar mensagens a esses objetos sem precisar saber de que tipo eles são, enquanto desenvolve.

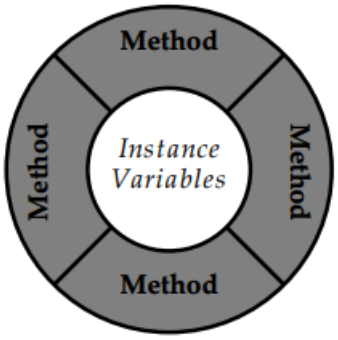
O que é um objeto?

De acordo com (GOSLING; MCGILTON, 1995) objetos são modelos de programação de software. Softwares contém objetos como por exemplo: botões, células de uma planilha, listas, o botão de fechar de uma janela qualquer, etc. Objetos em programação são abstrações[2] de objetos do mundo real, com o foco de representar suas características e comportamentos. Um objeto computacional pode ser representado como na figura 4: variáveis (características) que métodos (comportamentos) alteram.

[1] para permitir que vários processos rodem ao mesmo tempo a fim de aumentar sua performance

[2] Segundo (PRIBERAM, 2018), abstrair é separar mentalmente uma parte de um todo para a considerar independente das outras partes

Figura 4 – Representação de um objeto



Fonte: (GOSLING; MCGILTON, 1995)

2.5 NetBeans

Segundo Heffelfinger (2015, p. 7-8), NetBeans se encaixa na definição 4 e

Definição 4 *Ambiente de Desenvolvimento Integrado (Integrated Development Environment), resumidamente, uma IDE. Uma IDE contém ferramentas, extensões e funcionalidades para facilitar o desenvolvimento de programas nas linguagens em que dá suporte, por exemplo o recurso de autocompletar, sugerir alterações e refatorar do código.*

tem suporte para várias linguagens de programação como Java, C, C++, PHP, HTML e JavaScript. NetBeans é também uma plataforma, sendo possível usa-lo para criar plugins e aplicações autônomas.

Como IDE de Java, NetBeans tem suporte para aplicações Java SE (Standard Edition), ME (Micro Edition) e EE (Enterprise Edition) que geralmente rodam em grandes servidores e suportam milhares de usuários ao mesmo tempo (HEFFELFINGER, 2015).

Assim como a linguagem Java, a IDE é facilmente instalável em sistemas operacionais diversos, e tem suporte oficial para Windows, Linux e Mac OS X. A forma de baixar, instalar e usar o NetBeans e como tirar vantagens de suas ferramentas e funcionalidades é descrita com detalhes em Heffelfinger (2015).

2.6 *IBM ILOG CPLEX Optimization Studio*

O *IBM ILOG CPLEX Optimization Studio*, vulgarmente referido como simplesmente CPLEX é uma caixa de ferramentas, entre elas uma IDE (ver definição 4), que permite construir modelos de otimização. É a consolidação da Linguagem de Programação Linear e o motor de uma biblioteca que oferece robustez para resolver problemas expressados em modelos matemáticos. Ele pode ser baixado gratuitamente caso o usuário tenha uma conta de e-mail institucional, podendo ser usado para propósitos educacionais por até um ano (CORPORATION, 2017a).

2.6.1 CPLEX Api para Java

A Api[3] para Java é uma das ferramentas do CPLEX, para utiliza-la basta importar o arquivo cplex.jar para o projeto Java no NetBeans e começar a usar. O arquivo fica disponível na pasta:

```
C:\ProgramFiles\IBM\ILOG\CPLEX_Studio128\cplex\lib\
```

Observando que a pasta CPLEX_Studio128 muda conforme a versão instalada, neste caso foi a versão 12.8, e o CPLEX Studio foi instalado no local padrão. Documentação e exemplos de como usar estão disponíveis em (CORPORATION, 2017b).

2.7 Clean Code

O livro (MARTIN, 2009) é um livro importante para qualquer programador que queira se tornar ainda melhor. O livro não trata de questões técnicas, pois se o fizesse nunca estaria atual, mas trata de leitura e escrita de código. Mas o que seria escrever código limpo?

> Eu gosto que meu código seja elegante e eficiente. A lógica deve ser clara e direta para que seja difícil bugs se esconderem, dependências devem ser mínimas para fácil manutenção, tratamento de erros completa de acordo com alguma estratégia articulada, e performance próxima do ótimo para que outras pessoas não se sintam tentadas a mexer no código com otimizações sem sentido. Código limpo faz uma coisa e a faz muito bem [4] (MARTIN, 2009, tradução nossa, p. 9 apud STROUSTRUP, s.d.).

Dentre os vários princípios do Clean Code, se destacam: escrever funções que fazem somente uma coisa, e que a façam com a maior clareza possível; dar nomes simples e com significado às variáveis; lidar com erros através de lançamento de exceções e não de código de erros, escrever interfaces que façam poucas coisas, escrever classes com menos linhas possíveis e diminuir ao máximo o acoplamento de código e um dos princípios mais extremos: evitar ao máximo escrever comentários de código, pois o código deve ser simples o suficiente para falar por si mesmo, e se não o é, precisa ser refatorado.

O Clean Code vem para orientar programadores a escreverem seus códigos cada vez mais limpos e fáceis de ler. Segundo Martin (2009), a proporção de ler código para escrever é de 10:1, pois conferimos o tempo todo outras partes do código para escrevermos novas partes. O que o autor sugere é que passemos mais tempo escrevendo código de qualidade, para o mesmo fique simples para nós e outras pessoas entenderem, assim que tiverem contato com ele.

---

[3] *Application Program Interface*

[4] I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well.

2.8 SonarQube

SonarQube é uma ferramenta capaz de escanear e inspecionar código, através de heurísticas com a finalidade de encontrar bugs, vulnerabilidades, code smells e duplicações de código, além de verificar se bons padrões de código estão sendo seguidos. O

1. Bugs [5]: um erro ou problema em no código
2. Vulnerabilidades: alguma característica do código que pode significa uma falha de segurança
3. Code Smells: alguma característica do código que pode indicar um problema mais profundo
4. Duplicações: código que está repetido em um programa e que pode ser reaproveitado
5. Falta de aderência aos bons padrões de código: nomes de variáveis, nomes de métodos, nomes de classes, tamanho das classes e métodos.

Muitos code smells são na verdade problemas escrita de código, e o SonarQube verifica muitos desses problemas considerando vários princípios do Clean Code. O SonarQube aponta vários problemas de qualidade do código a fim de que o mesmo esteja em constante melhoria e que nenhum dos aspectos procurados pela ferramenta, aumentem conforme o código evolua. Dentre itens de qualidade de um código estão: tamanho das classes (em linhas) e métodos, nomes de métodos, classes e variáveis. Para o NetBeans existe uma adaptação do SonarQube em forma de plugin, chamado radar-netbeans [6].

2.9 Bibliotecas

Uma biblioteca é um conjunto de classes que são organizadas dentro de um pacote de modo a promover reuso. Quando queremos representar um objeto do mundo real em código precisamos abstrai-lo para imitar seus comportamentos e características, mas a depender do tipo de objeto, ele já pode ter sido abstraído, não precisamos então fazer uma nova abstração do mesmo objeto, bastando utilizar o código de terceiros, quando possível. Duas bibliotecas merecem destaque por representarem bem comportamentos e características de objetos que precisarão ser usados neste trabalho:

1. jOpenDocument[7] A biblioteca *jOpenDocument* é gratuita e *OpenSource*[8]. Permite ler e modificar arquivos em formatos abertos (opt, etc). Foi necessário ao trabalho pois é preciso ler a disponibilidade dos professores em cada par dia-horário para a modelagem

[5] (DICTIONARY, 2018)
[6] http://plugins.netbeans.org/plugin/51532/radar-netbeans
[7] (http://www.jopendocument.org/)
[8] Código fonte disponibilizado com licença de código aberto, em que qualquer um pode utilizar e modificar

do problema, para isso foi utilizada a biblioteca e um arquivo de tabelas ods com estes dados e os dados das disciplinas (nome, código, período (ou curricula) e créditos)

2. JAMA: A Java Matrix Package (Um pacote de matriz java) [9]

   A biblioteca JAMA é uma biblioteca de operações algébricas lineares básicas para Java. Permite construção e manipulação de matrizes densas. Foi criado para permitir funcionalidades rotineiras relacionadas a matrizes, criada em uma forma que é natural e de fácil entendimento para leigos.

[9] https://math.nist.gov/javanumerics/jama/

3 METODOLOGIA

O presente capítulo objetiva abordar os passos necessários para que a criação do algoritmo capaz de resolver o PADPH (ver subseção 2.3.1) fosse possível. A seção 3.1 mostra como o problema foi modelado matematicamente. A seção 3.2 mostra: como os dados de uma planilha foram abstraídos classes de objetos para a linguagem Java e questões arquiteturais do algoritmo que possibilitam sua reutilização.

3.1 Modelagem do problema

Através das restrições do problema e dos conflitos que precisamos resolver poderemos modelar matematicamente o problema. Diferente do proposto em Marti (2010) não usaremos um coeficiente nas variáveis do problema, mas em comum ao trabalho não foram usadas restrições de salas e/ou turmas. Além disso, foi necessário separar em duas restrições a restrição de aulas consecutivas descrita no trabalho de Marti (2010), devido à forma como o STSC aceita horários fixos na grade.

Vamos definir do que é composto nosso problema:

1. Um conjunto de h horários, $H = \{1, \ldots, h\}$, sendo que cada $k \in H$ um par dia-hora.
2. Um conjunto de d disciplinas, $D = \{1, \ldots, d\}$, cada $j \in D$ está associada a uma $q_j \in N$, representando o número de aulas que precisam ser ministradas relacionadas à disciplina
3. Um conjunto de c *curriculas*, $C = \{1, \ldots, c\}$, cada curricula r representa um período do curso.
4. Um conjunto de p professores, $P = \{1, \ldots, p\}$, cada professor n tem um conjunto $D_n \subset D$ que representa as disciplinas que o mesmo é responsável.
5. Uma matriz $I_{D \times H}$ em que cada entrada é definida pela equação 1. Essa matriz guardará os dados de entrada de indisponibilidade dos professores.
6. Uma matriz $X_{D \times H}$ em que cada entrada é definida pela equação 2. Essa matriz guardará as variáveis do nosso problema de otimização.

$$i_{jk} = \begin{cases} 0 & \text{se o professor associado à disciplina } j \text{ está disponível no horário } k, \\ 1 & \text{caso contrário} \end{cases} \tag{1}$$

$$x_{jk} = \begin{cases} 0 & \text{se s disciplina } j \text{ está associada ao horário } k, \\ 1 & \text{caso contrário} \end{cases} \tag{2}$$

Vamos agora construir nossas restrições a partir do que temos:

- É preciso garantir que nenhuma disciplina j de nenhum professor n seja colocada em horário em que ele não está disponível. A restrição 4 existe para garantir isso. O conjunto $D_n \subset D$ representa as disciplinas do professor n (ver equações 1 e 2).

- Um professor $n \in P$ não pode ter suas aulas de quaisquer disciplinas $j \in D_n \subset D$ no mesmo horário e isso é uma restrição que deve existir para cada professor, a partir disso temos a restrição 5.

- O conjunto de disciplinas $D_r \subset D$ sendo $r \in C$ um curricula, não pode ter aulas no mesmo horário. Essa é uma restrição válida para qualquer curricula, a partir disso temos a restrição 6.

- Cada disciplina j deve ter uma quantidade q_j de aulas durante a semana, a restrição 7 garante isso.

- Cada disciplina j tem um número inteiro s_j que limita a quantidade de aulas dadas por dia, para que não ocorra casos em que uma matéria tenha, por exemplo suas quatro aulas dadas em apenas um dia da semana. Cada $H_v \subset H$ representa os pares dia-hora de um dia $v \in W = \{1, \ldots, w\}$ da semana. Considere também os conjunto W_j e W_j^c tal que $W_j + W_j^c = W$.

  W_j é o conjunto dos dias que já tem aula da disciplina j, fixada na grade e W_j^c é o conjunto dos dias que não tem nenhuma aula da disciplina. A partir disso temos as restrição 8 e 9.

Nosso objetivo é minimizar x_{jk} que tem valor 0 caso uma disciplina esteja associada a um horário, ao minimizar essa função estamos tentando associar as disciplinas j a um horário k para que o valor da função objetivo (equação 3) diminua (ver equação 2).

$$\text{Minimizar} \sum_{j=1}^{d} \sum_{k=1}^{h} x_{jk} \tag{3}$$

Sujeito a:

$$\sum_{j \in D_n} i_{jk} x_{jk} = 0, \qquad \forall\, n \in P, \forall\, k \in H; \tag{4}$$

$$\sum_{j \in D_n} x_{jk} \leq 1, \qquad \forall\, n \in P, \forall\, k \in H; \tag{5}$$

$$\sum_{j \in D_r} x_{jk} \leq 1, \qquad \forall\, r \in C, \forall\, k \in H; \tag{6}$$

$$\sum_{k=1}^{h} x_{jk} = q_j, \qquad \forall\, j \in D; \tag{7}$$

$$\sum_{v \in H_v} x_{jv} = 0, \qquad \forall\, j \in D, \forall\, v \in W_j \tag{8}$$

$$\sum_{v \in H_v} x_{jv} \leq s_j, \qquad \forall\, j \in D, \forall\, v \in W_j^c \tag{9}$$

3.2 Classes do STSC

Usando o Java, CPLEX, as bibliotecas Jama e JOpenDocument, o plugin radar-netbeans e princípios do trabalho de Martin (2009) o STSC foi construído, com o objetivo de além de resolver o problema proposto, resolve-lo com o código mais limpo e com qualidade possível, utilizando bibliotecas para auxiliar a manipulação de matrizes e de documentos em formato aberto. Primeiro foi necessário fazer uma representação, a partir dos dados de entrada da planilha do que seria uma classe de disponibilidade do professor, uma de disciplina e uma que representasse o professor em si. Os dados de entrada da planilha são:

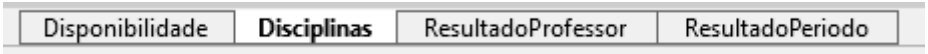
- Abas de dados de entrada e de saída, mostrados na figura 5
- Nomes dos professores e disponibilidades, na aba "Disponibilidade", e primeira coluna dessa planilha, como mostrado na figura 6. A figura mostra apenas uma parte das disponibilidades, que continua à direita até o último par dia-hora, e apenas alguns professores. Quanto à representação dos dados:

  1. 0 representa "não estou disponível"
  2. 1 representa "estou disponível"
  3. Caso contenha um código como "MAT001"significa que o professor quer fixar uma aula da disciplina naquele horário. Isso é necessário por, geralmente duas razões:
     a) O professor responsável pela disciplina vem de outro departamento, e portanto seu horário já está fixado na grade
     b) O professor tem uma preferência pedagógica por, dentre suas 4 aulas, deixar uma delas no sábado, ou no caso das matérias de Projeto Orientado I e II e estágio curricular supervisionado, já são geralmente colocados no sábado pela necessidade mínima de encontros entre os estudantes e os professores responsáveis pela disciplina.

- Dados das disciplinas, período a que pertencem, código, quantidade de aulas a serem ministradas e professor associado, como mostrado na figura 7

As representações dos dados do arquivo juntamente com o objetivo final: representar uma grade curricular com disciplina em cada período respeitando as disponibilidades dos professores pode ser melhor observada no código disponibilizado no GitHub [1]. Os pacotes criados durante o desenvolvimento do STSC são mostrados na figura 8, e a lista de classes por cada pacote é mostrada na figura 9, os nomes dos professores não refletem a realidade.

Figura 5 – Abas mostrando a estrutura da planilha



Fonte: Elaborada pelo autor

Figura 6 – Dados de disponibilidade dos professores

Professor	Segunda18	Segunda19	Segunda20	Segunda21	Segunda22	Terça18
Matemática	MAT001	MAT001	0	0	0	1
Andre	1	1	1	1	1	1
Athila	1	1	1	1	1	1

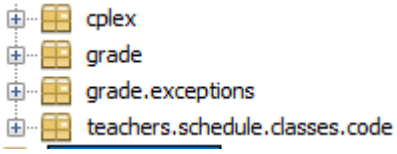
Fonte: Elaborada pelo autor

Figura 7 – Dados de disciplinas

Nome da Matéria	Código	Período	Créditos	Professor
Fundamentos de Matemática	MAT001	1	4	Matemática
Fundamentos de Sistemas de Informação	COM040	1	4	Claudia
Inglês Instrumental	COM043	1	3	Inglês
Introdução à Lógica Computacional	MAT007	1	4	Matemática
Leitura e Produção de Textos	COM059	1	4	Andre
Algoritmos e Estrutura de Dados I	COM001	2	5	Leonardo

Fonte: Elaborada pelo autor

Figura 8 – Pacotes do Projeto

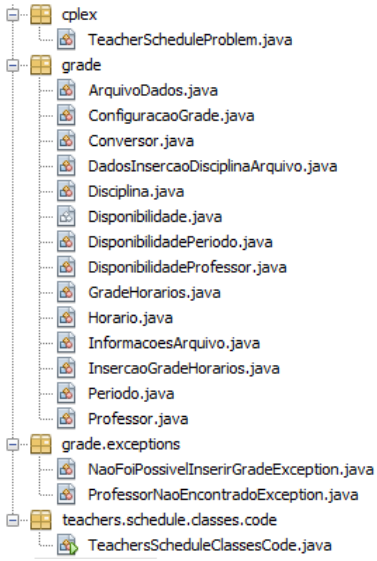


Fonte: Elaborada pelo autor

O pacote "grade"e o subpacote "grade.exceptions"são independentes dos demais pacotes. Esses dois pacotes representam o mínimo que uma grade de horários precisa: períodos, lista de professores, lista de disciplinas e disponibilidades, e tem mecanismos para permitir a alocação das disciplinas sem perder a consistência dos dados, garantida por mecanismos que estão nos trechos de código 1 da classe "GradeHorarios"e no trecho de código de exemplo 2

[1] A versão do código do STSC deste trabalho está disponível pelo link: https://github.com/natancmacedo/teachers-schedule-classes-semester/tree/v1.0.0-TCC

Figura 9 – Classes por pacote



Fonte: Elaborada pelo autor

sobre como utilizar o objeto para fazer inserção da disciplina.

Código 1 – Trecho de código da classe GradeHorarios mostrando como é garantida a consistência dos dados

```
1  public void inserirDisciplinaNaGrade( InsercaoGradeHorarios
       novaInsercao ) {
2          if (this.ehPossivelInserirDisciplina(novaInsercao)) {

4              List<Horario> horarios = novaInsercao.getHorarios();

6              for (Horario horario : horarios) {
7                  novaInsercao.getProfessor()
8                          .setDisciplinaHorarioPessoal(novaInsercao
                               .getDisciplina(), horario);

10                 novaInsercao.getPeriodo()
11                         .inserirDisciplinaNoHorario(novaInsercao.
                               getDisciplina(), horario);

13                 novaInsercao.getDisciplina()
14                         .addCreditosAlocados();
15             }
```

```
17          } else {
18              try {
19                  throw new NaoFoiPossivelInserirGradeException("ãNo
                        éí possvel inserir "
20                          + novaInsercao);
21              } catch (NaoFoiPossivelInserirGradeException ex) {
22                  Logger.getLogger(GradeHorarios.class.getName()).
                        log(Level.SEVERE, null, ex);
23              }
24          }
25      }

27  public Boolean ehPossivelInserirDisciplina( InsercaoGradeHorarios
        tentativa ) {
28          Boolean disciplinaTotalmenteAlocada = tentativa.
                getDisciplina().estaTotalmenteAlocada();

30          List<Horario> horarios = tentativa.getHorarios();

32          Boolean professorDisponivelNosHorarios = tentativa.
                getProfessor()
33                  .estaDisponivelNosHorarios(horarios);

35          Boolean periodoTemDisciplinaNosHorarios = tentativa.
                getPeriodo()
36                  .horariosEstaoOcupados(tentativa.getHorarios());

38          return !disciplinaTotalmenteAlocada
39                  && professorDisponivelNosHorarios
40                  && !periodoTemDisciplinaNosHorarios;
41      }
```

Fonte: Autor

Código 2 – Trecho de código exemplo de como inserir uma disciplina na grade

```
1  List<Horario> horarios = new ArrayList<>();
2  horarios.add(new Horario(0, 0));
3  horarios.add(new Horario(0, 1));

5  Disciplina disciplina = grade.getDisciplinaPorCodigo("MAT001");
6  Periodo periodo = grade.getPeriodoPorNumero(disciplina.getPeriodo
       ());
```

```
7  Professor professor = grade.getProfessorDaDisciplina(disciplina);

9  grade.inserirDisciplinaNaGrade(new InsercaoGradeHorarios(
       professor, disciplina, horarios, periodo));
```

Fonte: Autor

Com esses mecanismos, não importa a estratégia de alocação de horários que o usuário ou desenvolvedor queira usar, desde que o mesmo tenha os pacotes é possível que ele defina a própria estratégia e faça suas próprias inserções. Claro que isso dependerá também da utilização correta do pacote e que o arquivo de dados esteja devidamente preenchido. Um exemplo sobre a utilização básica dos pacotes é mostrada no código 3.

Código 3 – Uso básico para inicialização de um problema

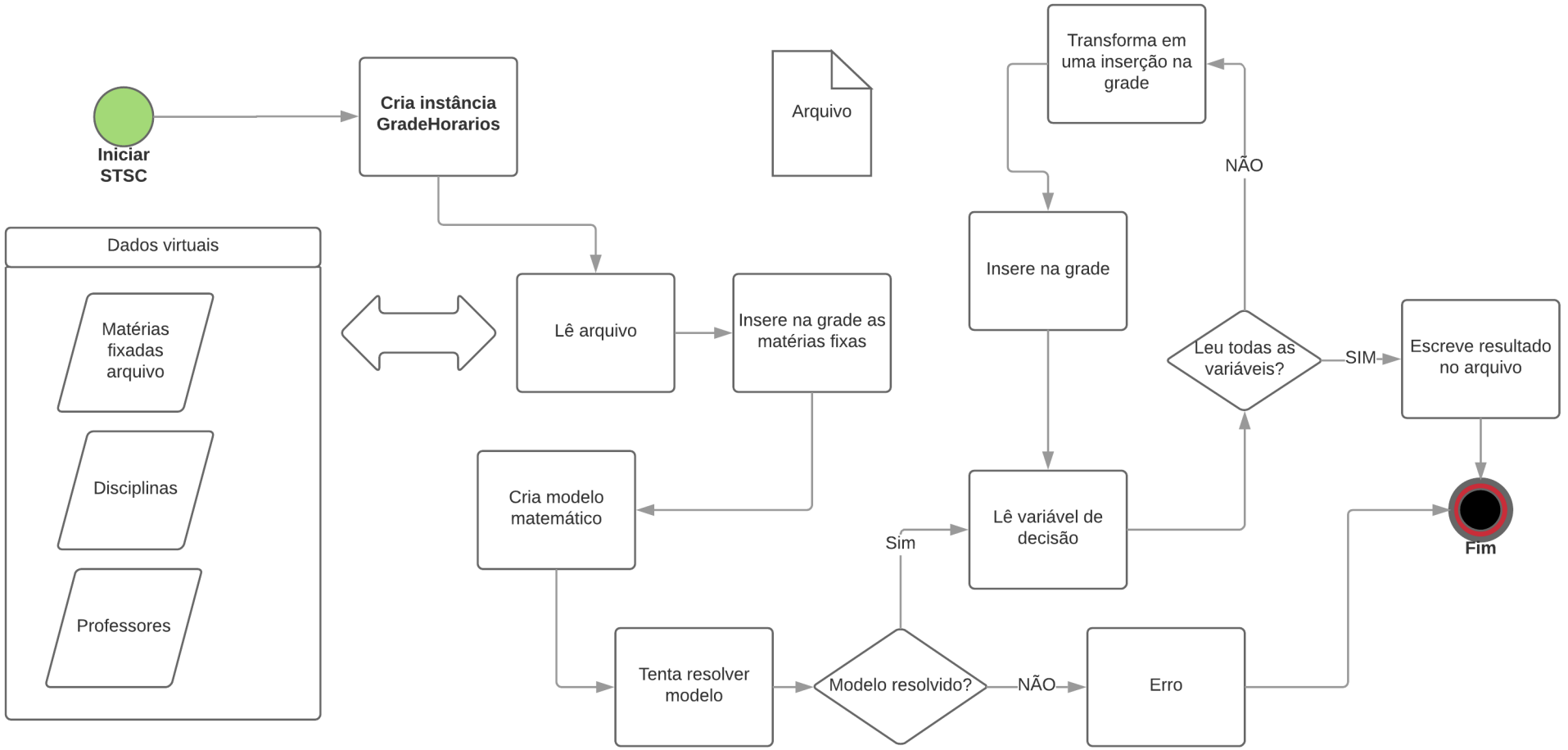
```
1          ConfiguracaoGrade config = new ConfiguracaoGrade(9, 6, 5)
               ;
2          GradeHorarios grade = new GradeHorarios(config);

4          InformacoesArquivo infoFile = new InformacoesArquivo("../
               dados/DadosReunidos.ods",
5                "Disponibilidade",
6                "Disciplinas",
7                "ResultadoProfessor",
8                "ResultadoPeriodo");

10         grade.preencherDadosProblema(infoFile);
```

Fonte: Autor

No problema específico do Decom na UFVJM, são seis dias letivos: de segunda à sábado, sendo que de segunda à sexta as aulas do curso de Sistemas de Informação vão de 18 às 23h, e no sábado de 8 às 13h, com duração de uma hora cada aula. O algoritmo não é capaz de fazer diferenciação entre os dias da semana que precisem ter menos aula, como pode ser visto na classe "ConfiguracaoGrade"e (representação UML em 22), é determinado quantas aulas por dia, quantos dias da semana são válidos (começando da segunda feira) e quantos períodos existem para a grade. Caso seja preciso limitar que um dia não tenha mais do que, por exemplo, três horários no sábado em uma grade que tem cinco horas de aula, basta colocar 0 nas duas últimas horas do sábado de todos os professores na aba de Disponibilidade do arquivo de dados.

O funcionamento geral do STSC como um todo segue o fluxo descrito na figura 10.

Figura 10 – Fluxograma geral do STSC



Fonte: Elaborada pelo autor

4 TESTES E RESULTADOS

O presente capítulo tem por objetivo mostrar os passos para utilização do STSC, na prática, além de mostrar os resultados alcançados referentes à tempo, desempenho e alocação de horários detalhada de aulas em um teste com dois períodos, onze disciplinas e seis professores. Os nomes dos professores são fictícios, em contrapartida os dados referentes às disciplinas são os atuais no curso de Sistemas de Informação da UFVJM.

O STSC é composto pelo CPLEX e um conjunto de classes responsáveis por representar uma grade de horários. O tempo de execução do STSC como um todo pode ser calculado diferentemente do tempo que a estratégia (CPLEX) demora para criar um modelo e resolvê-lo para encontrar uma solução ótima para uma determinada instância do problema.

A execução dos testes do STSC contou com variações na entrada dos dados, derivados dos do arquivo com os dados dos nove períodos. Para os primeiros testes foram feitos cortes com o seguinte critério: criação de um novo arquivo por quantidade de períodos, cada arquivo contendo as disciplinas de um a nove períodos e com os professores dessas disciplinas. Todos os professores com disponibilidade total em todos os dias da semana, e o professor "Matemática"que representa o departamento responsável pelas disciplinas de matemática do curso, com horários fixos da matéria de código "MAT001"na segunda feira de 18 às 20h. A quantidade de horários e dias ficou fixa (5 e 6 respectivamente) os resultados desses testes estão na tabela 1, a primeira coluna se refere a quantidade de períodos, a segunda à quantidade de professores, a terceira à quantidade de disciplinas, a quarta ser refere ao tempo gasto total do STSC e a última coluna o tempo gasto para o CPLEX criar o modelo e resolver o problema. O segundo teste teve as mesmas configurações do primeiro, diferindo apenas na disponibilidade de todos os professores nos sábados que foi modificada para 0 em todos os horários, os resultados deste teste estão na tabela 2.

Tabela 1 – Resultados dos testes com disponibilidade todos os dias

Períodos	Professores	Disciplinas	STSC	CPLEX
1	4	5	863ms	0ms
2	6	11	331ms	20ms
3	9	17	238ms	30ms
4	12	23	1467ms	30ms
5	15	29	239ms	30ms
6	16	35	315ms	80ms
7	18	41	248ms	60ms
8	18	47	2307ms	50ms
9	18	49	554ms	130ms

Fonte: Produzido pelo autor.

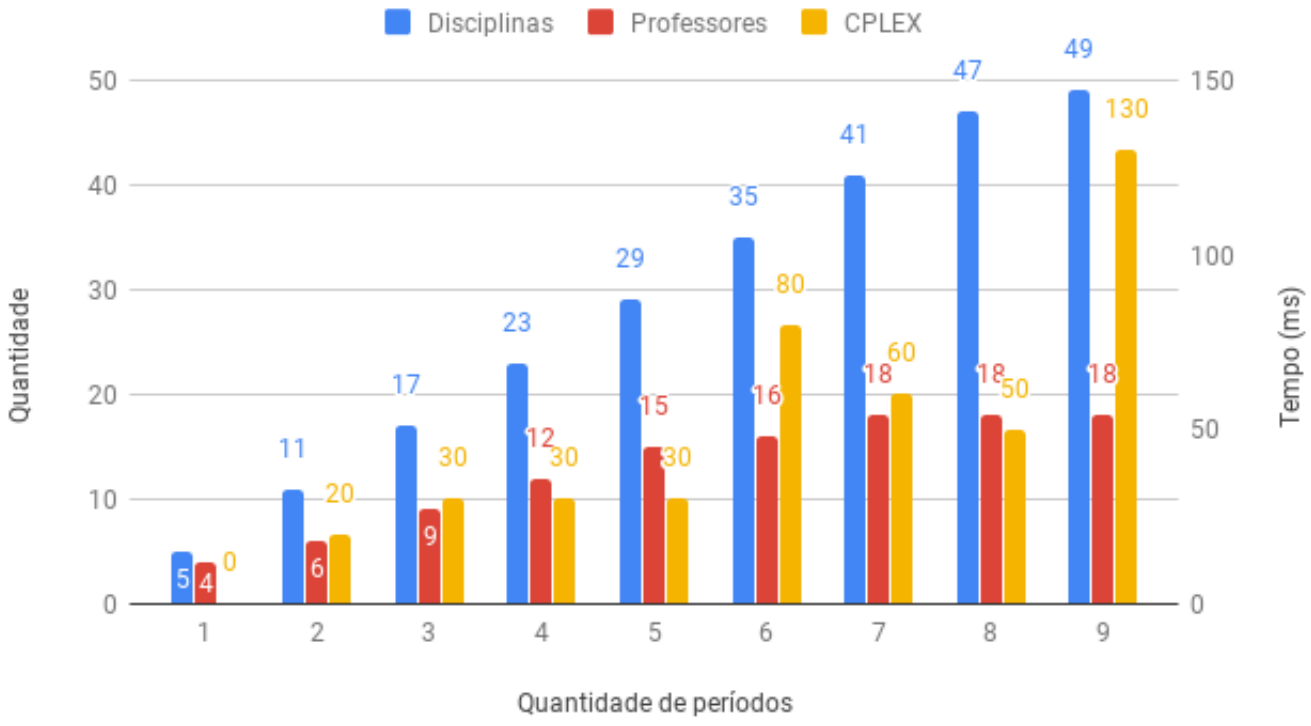
Para entendermos melhor o desempenho do algoritmo podemos ver na figura 11 o gráfico comparando o tempo do algoritmo em relação ao crescimento das disciplinas, períodos e professores, considerando os testes da tabela 1.

Tabela 2 – Resultados dos testes com disponibilidade todos os dias exceto sábados

Períodos	Professores	Disciplinas	STSC	CPLEX
1	4	5	784ms	10ms
2	6	11	310ms	10ms
3	9	17	370ms	10ms
4	12	23	1407ms	30ms
5	15	29	247ms	30ms
6	16	35	307ms	80ms
7	18	41	303ms	80ms
8	18	47	2428ms	80ms
9	18	49	954ms	90ms

Fonte: Produzido pelo autor.

Figura 11 – Desempenho do algoritmo em relação a quantidade de disciplinas e professores



Fonte: Elaborada pelo autor

Um terceiro teste foi feito, usando apenas dois períodos e as disciplinas pertencentes aos mesmos, além dos professores responsáveis por elas. A figura 12 mostra o resultado da alocação das disciplinas nos dois períodos. Os números representam a quantidade de professores disponíveis no horário, caso não exista uma matéria alocada no mesmo.

A figura 13 mostra os dados das disciplinas que foram alocadas no teste, com seu nome, código, período a que pertencem, créditos (quantidade de aulas que devem ser ministradas) e professor responsável.

Figura 12 – Resultado de alocação de disciplinas em dois períodos

1º	SEG	TER	QUAR	QUI	SEXT	SAB	
	MAT001	COM059	COM040	MAT007	COM043		0
	MAT001	COM040	2	COM040	COM043		0
	COM059	COM040	2	2	COM043		0
	MAT007	MAT001	MAT001	2	3		0
	COM059	COM059	MAT007	2	MAT007		0

2º	SEG	TER	QUAR	QUI	SEXT	SAB	
	COM002	MAT006	MAT003	COM002	COM001		0
	COM002	COM060	MAT006	COM002	COM001		0
	MAT006	COM060	MAT003	MAT006	COM001		0
	COM003	COM060	COM003	COM001	MAT003		0
	COM003	MAT003	2	COM001	COM003		0

Fonte: Elaborada pelo autor

Figura 13 – Dados das disciplinas alocadas no teste

Nome da Matéria	Código	Período	Créditos	Professor
Fundamentos de Matemática	MAT001	1	4	Matemática
Fundamentos de Sistemas de Informação	COM040	1	4	Claudia
Inglês Instrumental	COM043	1	3	Inglês
Introdução à Lógica Computacional	MAT007	1	4	Matemática
Leitura e Produção de Textos	COM059	1	4	Andre
Algoritmos e Estrutura de Dados I	COM001	2	5	Leonardo
Cálculo Diferencial e Integral I	MAT003	2	4	Matemática
Matemática Discreta	MAT006	2	4	Matemática
Metodologia do Trabalho e da Pesquisa Científica e Tecnológica	COM060	2	3	Andre
Sistemas de Computação	COM002	2	4	Rafael
Teoria Geral dos Sistemas	COM003	2	4	Claudia

Fonte: Elaborada pelo autor

As figuras 14, 15, 16, 17, 18 e 19 mostram os resultados de alocação para os professores mostrados na figura 13. É possível perceber que mesmo cumprindo todas as restrições impostas ao problema, o algoritmo ainda assim separa as disciplinas em algumas formas não muito aceitáveis, como por exemplo para o segundo período, a disciplina de código "MAT003"na quarta-feira, que teve as duas aulas da disciplina no dia separadas pela disciplina de código "MAT006". Tais resultados podem ser melhorados de duas formas: através da correção da

disponibilidade, por exemplo: observando o resultado da alocação do professor André, na figura 14, o mesmo está disponível em toda a segunda-feira mas ainda assim é obtido esse comportamento indesejável na alocação, pode-se perguntar ao mesmo sua preferência em relação às duas aulas da disciplina de código "COM059"na segunda-feira, sua disponibilidade é corrigida neste dia e o STSC é executado mais uma vez, caso a solução ainda seja alcançável, tem-se um resultado mais satisfatório. A segunda forma é criar uma restrição que force, em caso de existência de mais de uma aula de uma disciplina no mesmo dia, sua alocação conjunta (uma aula seguida da outra).

Figura 14 – Resultado da alocação para o professor André

Andre	SEG	TER	QUAR	QUI	SEXT	SAB	
		1	COM059	0	0	0	0
		1	COM060	0	0	0	0
	COM059		COM060	0	0	0	0
		1	COM060	0	0	0	0
	COM059		COM059	0	0	0	0

Fonte: Elaborada pelo autor

Figura 15 – Resultado da alocação para a professora Cláudia

Claudia	SEG	TER	QUAR	QUI	SEXT	SAB	
		1		1	COM040	1	1	0
		1	COM040		1	COM040	1	0
		1	COM040		1	1	1	0
	COM003			1	COM003	1	1	0
	COM003		1		1	1	COM003	0

Fonte: Elaborada pelo autor

O último teste se refere à medidas de qualidade do código, usando a ferramenta para NetBeans descrita na seção 2.8 para garantir a qualidade do código e sua aderência aos princípios descritos na seção 2.7. O resultado da ferramenta quanto ao código está na figura , como é possível verificar, o código passou em todas as medidas de qualidade da ferramenta, o que não garante que o código está perfeito no que se propõe, mas que o mesmo tem o mínimo de qualidade esperado no requisito de segurança e legibilidade.

Podemos perceber que através do STSC é possível alcançar boas soluções para o PADPH aplicado ao Decom, embora exista a possibilidade de melhoria do resultado em relação as alocações de aulas que não ficam consecutivas.

Figura 16 – Resultado da alocação para o professor responsável por ministrar Inglês Instrumental

Inglês	SEG	TER	QUAR	QUI	SEXT	SAB	
	0	0	0	0	COM043		0
	0	0	0	0	COM043		0
	0	0	0	0	COM043		0
	0	0	0	0	1		0
	0	0	0	0	1		0

Fonte: Elaborada pelo autor

Figura 17 – Resultado da alocação para o professor Leonardo

Leonardo	SEG	TER	QUAR	QUI	SEXT	SAB	
	0	0	0	1	COM001		0
	0	0	0	1	COM001		0
	0	0	0	1	COM001		0
	0	0	0	COM001		0	0
	0	0	0	COM001		0	0

Fonte: Elaborada pelo autor

Figura 18 – Resultado da alocação para o departamento de matemática (que no contexto do curso deve já vir definido, devido à natureza organizacional da UFVJM e da relação entre os departamentos)

Matemática	SEG	TER	QUAR	QUI	SEXT	SAB	
	MAT001	MAT006	MAT003	MAT007		1	0
	MAT001		1	MAT006	1	1	0
	MAT006		1	MAT003	MAT006	1	0
	MAT007	MAT001	MAT001		1	MAT003	0
		1	MAT003	MAT007		1	MAT007	0

Fonte: Elaborada pelo autor

Comparando os resultados, em relação a desempenho, com o trabalho de Marti (2010) que tem o modelo mais próximo do que foi implementado neste trabalho. Temos os seguintes cenários:

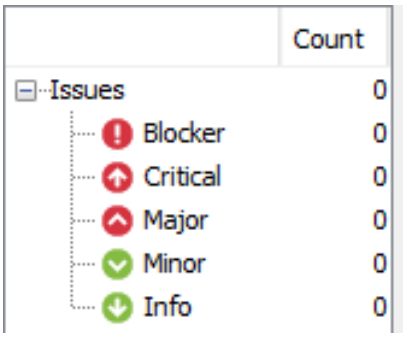
- Em Marti (2010): Uma instância com 54 professores, 103 disciplinas, usando a versão 9.0 do CPLEX, algoritmo *Branch-and-Cut*, e linguagem C++, o tempo para alcançar a solução foi de 62ms;

Figura 19 – Resultado da alocação para o professor Rafael

Rafael	SEG	TER	QUAR	QUI	SEXT	SAB	
	COM002	0	0	COM002	0	0	
	COM002	0	0	COM002	0	0	
	0	0	0	0	0	0	
	0	0	0	0	0	0	
	0	0	0	0	0	0	

Fonte: Elaborada pelo autor

Figura 20 – Resultado da análise de qualidade do código, pelo SonarQube



Fonte: Elaborada pelo autor

- Instância deste trabalho: 17 professores, 49 disciplinas, usando a versão 12.8 do CPLEX e linguagem Java o tempo para alcançar a solução foi em média de 110ms.

É possível que a diferença de design da linguagem Java comparada a do C++ a torne mais lenta além de termos a diferença do uso de um algoritmo *Branch-and-cut* no trabalho de Marti (2010).

5 CONCLUSÃO E TRABALHOS FUTUROS

O processo de alocação de horários em universidades, quando feito manualmente é um trabalho tedioso e que exige grande esforço para que as restrições do problema não sejam quebradas. Visando automatizar esse processo existem vários trabalhos estudando a complexidade do problema, propondo algoritmos exatos, heurísticas e meta-heurísticas capazes de alcançar soluções para o mesmo.

O presente trabalho objetivou criar o algoritmo STSC usando otimização matemática para alcançar uma solução viável para o problema do Departamento de Computação na UFVJM.

O algoritmo foi criado usando a linguagem Java, considerando princípios de boa escrita de código, usando ferramentas para análise da qualidade do mesmo, além da leitura das documentações das bibliotecas usadas para manipular arquivos, matrizes e para criação de modelos de otimização matemática; a fim de poder ler os dados e representa-los em forma de objetos computacionais, manipula-los, buscar uma solução e escrevê-la de volta no arquivo em formato aberto, com foco em fácil entendimento para o usuário.

O STSC criado permite a futura utilização de quaisquer outras técnicas para alcançar uma solução viável, devido suas características de arquitetura e divisão dos pacotes de classes Java.

Para que seja possível alcançar melhores resultados e facilitar o uso do algoritmo por usuários de quaisquer níveis, são descritos os possíveis trabalhos futuros:

- Criar/Alterar as restrições do algoritmo para que as mesmas aulas em um dia possam ser alocadas uma após a outra, sem precisar alterar a disponibilidade do professores para isso;
- Desenvolver análise de inconsistências capaz de mostrar o porquê de um conjunto de disponibilidades, disciplinas e professores não gerarem uma solução viável e o que pode ser feito para alcançar a solução;
- Desenvolver uma interface Web para que os professores possam entrar com suas disponibilidades em qualquer momento e lugar.

BIBLIOGRAFIA

ALMOND, Mary. An algorithm for constructing university timetables. *The Computer Journal*, The British Computer Society, v. 8, n. 4, p. 331–340, 1966.

ANALYZING Source Code - Scanners. Disponível em: <https://docs.sonarqube.org/display/SCAN/Analyzing%20Source%20Code>.

ARNOLD, Ken; GOSLING, James; HOLMES, David. *The Java programming language*. Addison Wesley Professional, 2005.

CONTINUOUS Inspection. Disponível em: <https://www.sonarqube.org/features/clean-code/>.

COOK, Stephen A. The classification of problems which have fast parallel algorithms. In: SPRINGER. *International Conference on Fundamentals of Computation Theory*. 1983. p. 78–93.

CORPORATION, IBM. *IBM ILOG CPLEX Optimization Studio V12.8.0 documentation*. IBM Corporation, 2017. Disponível em: <https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html>.

CORPORATION, IBM. *ilog.cplex (CPLEX Java API Reference Manual)*. IBM Corporation, 2017. Disponível em: <https://www.ibm.com/support/knowledgecenter/SSSA5P_12.5.0/ilog.odms.cplex.help/refjavacplex/html/ilog/cplex/package-summary.html>.

DEITEL, Paul. *Java Como Programar*. Pearson, 2009. ISBN: 8576055635. Disponível em: <https://www.amazon.com/Java-Como-Programar-Portuguese-Brasil/dp/8576055635?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=8576055635>.

DICTIONARY, Cambrigde. *BUG | eaning in Cambrigde English Dictionary*. 2018. Disponível em: <https://dictionary.cambridge.org/dictionary/english/bug#dataset-cbed>.

GAREY, Michael R; JOHNSON, David S. Computers and intractability: a guide to NP-completeness. WH Freeman e Company, San Francisco, 1979.

GOLDBARG, Marco Cesar; LUNA, Henrique Pacca L. *Otimização combinatória e programação linear: modelos e algoritmos*. Elsevier, 2005.

GOSLING, James; MCGILTON, Henry. The Java language environment. *Sun Microsystems Computer Company*, v. 2550, 1995.

HEFFELFINGER, David R. *Java EE 7 Development with NetBeans 8*. Packt Publishing Ltd, 2015.

MARTI, Jean Paulo. *O Problema do agendamento semanal de aulas*. 2010. Tese (Mestrado). Ciências Exatas e da Terra - Ciências da Computação. Disponível em: <http://repositorio.bc.ufg.br/tede/handle/tde/502>.

MARTIN, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

MÜLLER, Tomáš. Constraint-based timetabling. Citeseer, 2005.

NANDA, Anirudha; PAI, Manisha P; GOLE, Abhijeet. An algorithm to automatically generate schedule for school lectures using a heuristic approach. *International journal of machine learning and computing*, IACSIT Press, v. 2, n. 4, p. 492, 2012.

PRIBERAM. *Definição Abstrair*. 2018. Disponível em: <https://dicionario.priberam.org/abstrair>.

SANTOS, Haroldo Gambini; SOUZA, Marcone Jamilson Freitas. Programação de horários em instituições educacionais: formulações e algoritmos. *XXXIX SBPO-Simpósio Brasileiro de Pesquisa Operacional*, n. 1, p. 2827–2882, 2007.

STROUSTRUP, Bjarne. *Inventor of C++*.

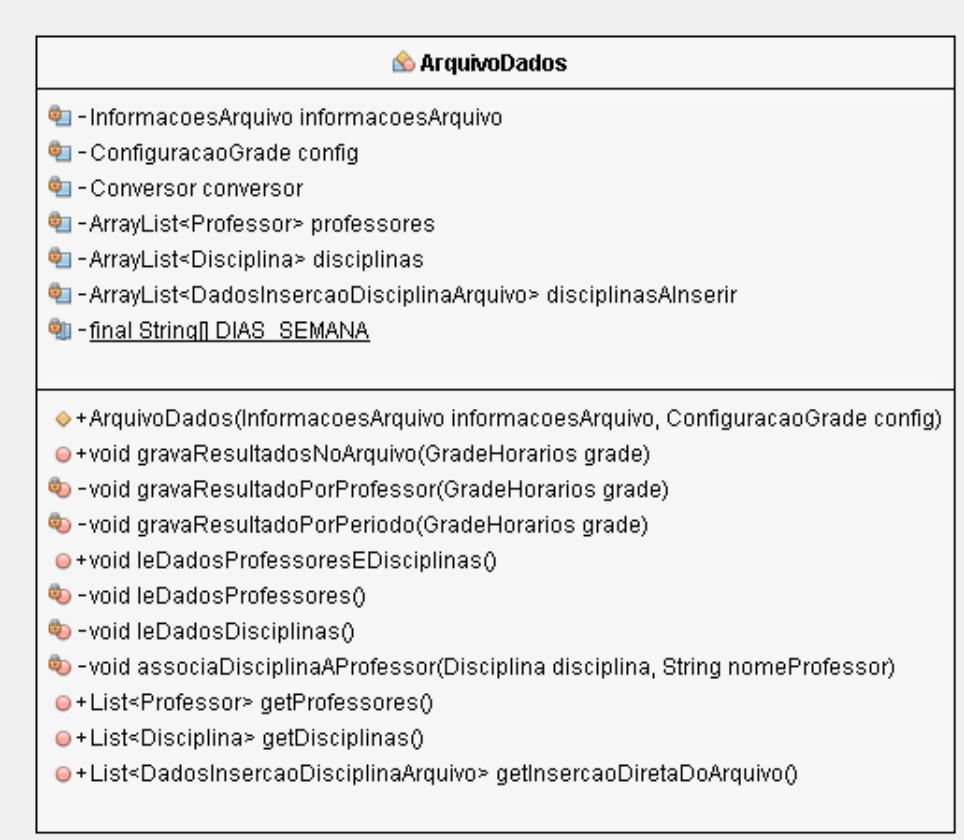
WILLEMEN, RJ. School timetable construction : algorithms and complexity. *Production Planning Control - PRODUCTION PLANNING CONTROL*, Citeseer, jan. 2002.

APÊNDICE A – CÓDIGO

O código do STSC na versão deste trabalho pode ser acessado no repositório público no GitHub através do link: https://github.com/natancmacedo/teachers-schedule-classes-semester/tree/v1.0.0-TCC.

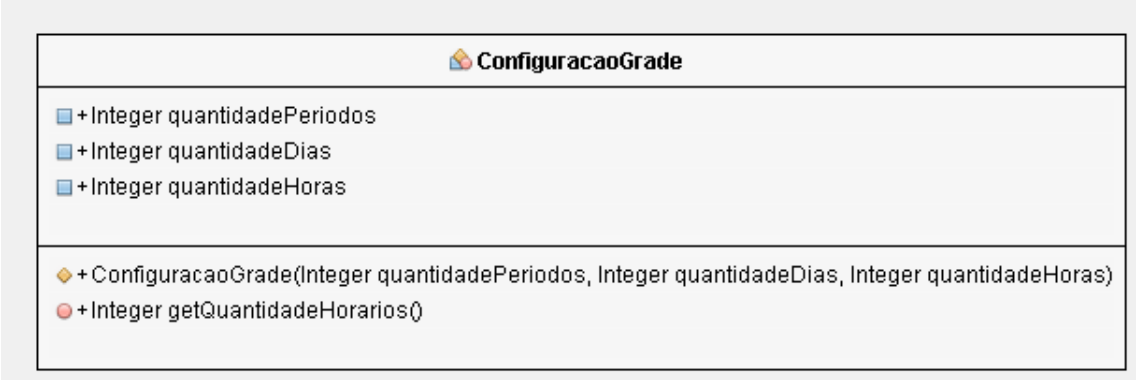
APÊNDICE B – REPRESENTAÇÃO UML DAS CLASSES

Figura 21 – A classe Arquivo Dados



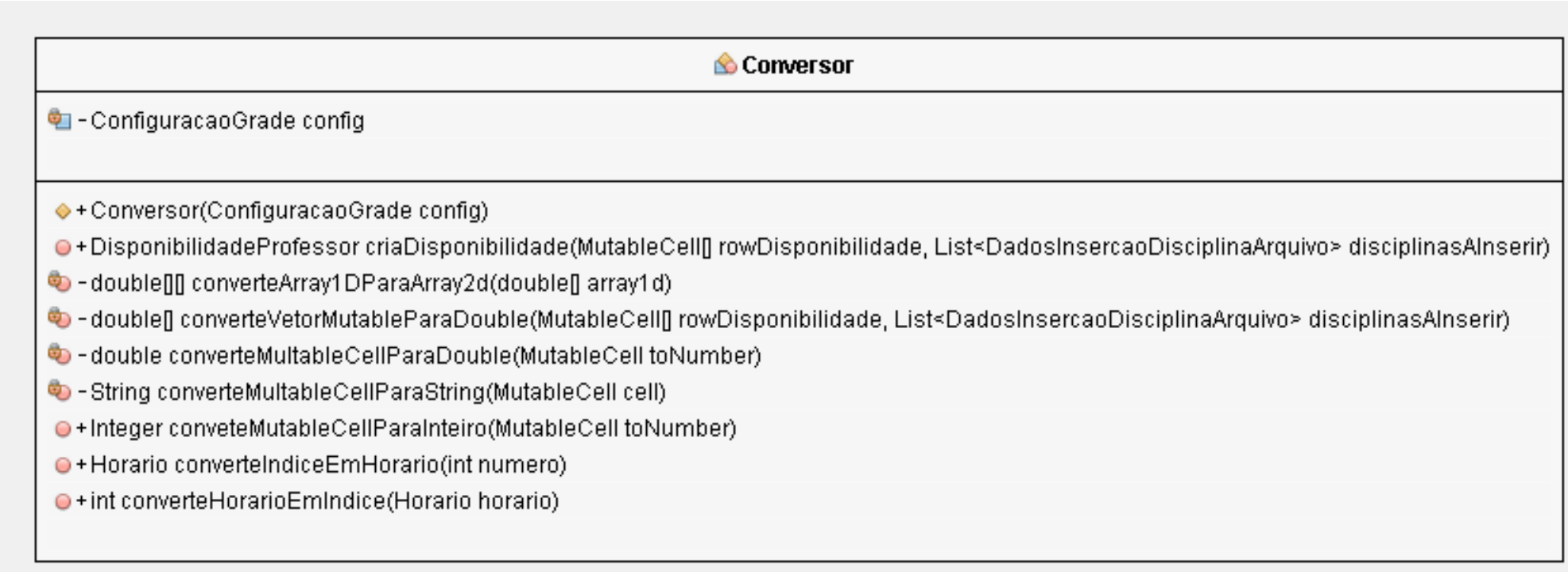
Fonte: Elaborada pelo autor

Figura 22 – A classe ConfiguracaoGrade



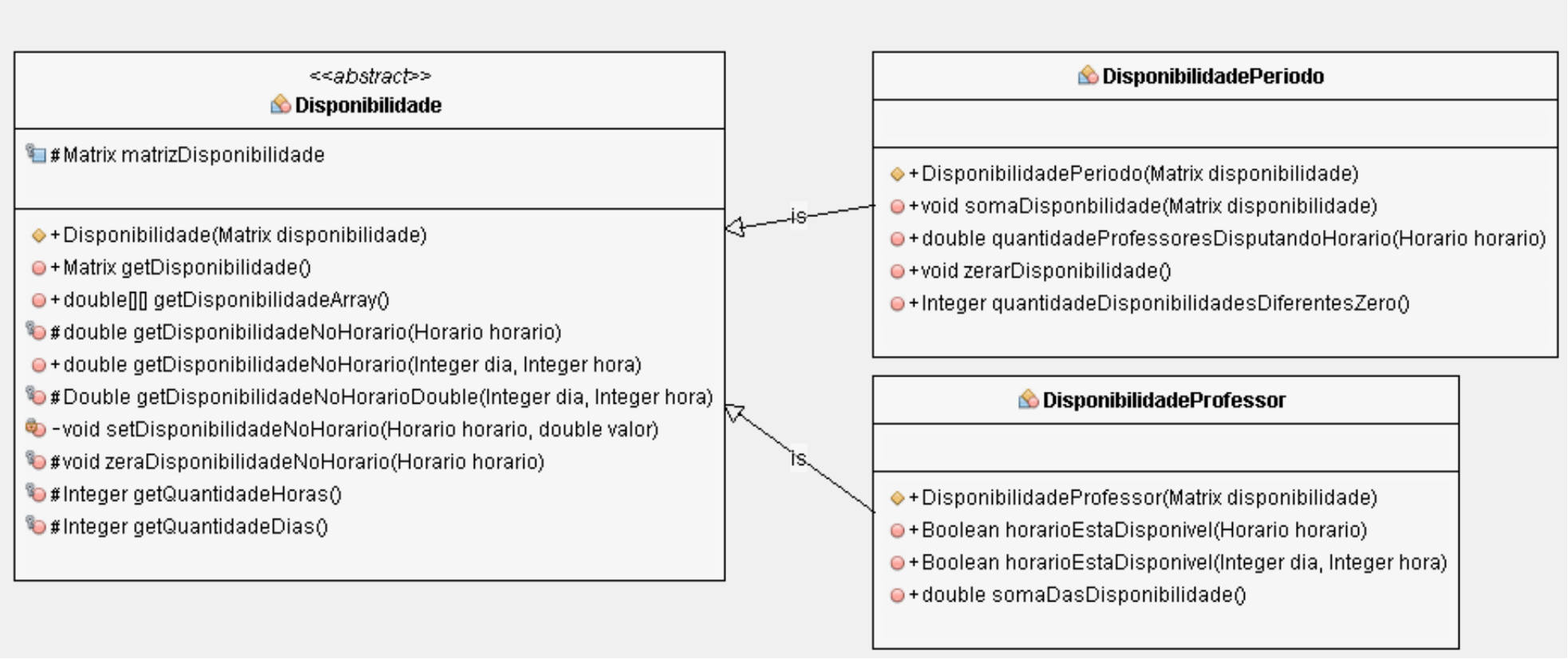
Fonte: Elaborada pelo autor

Figura 23 – A classe Conversor



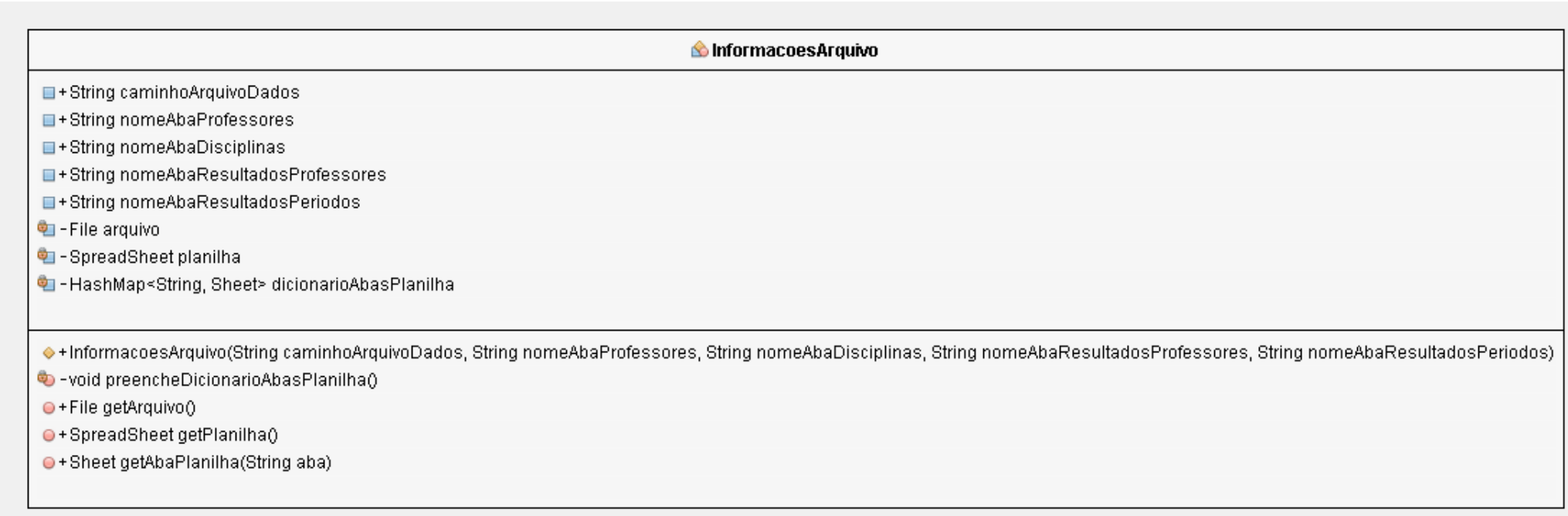
Fonte: Elaborada pelo autor

Figura 24 – As classes e relacionamento entre Disponibilidades



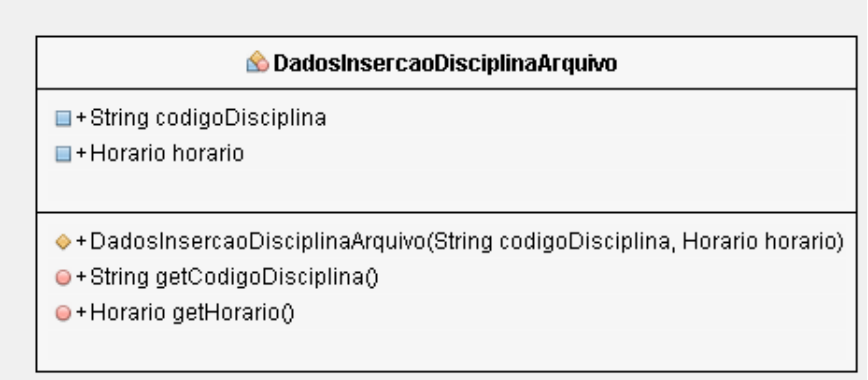
Fonte: Elaborada pelo autor

Figura 25 – A classe InformacoesArquivo



Fonte: Elaborada pelo autor

Figura 26 – A classe DadosInsercaoDisciplinaArquivo

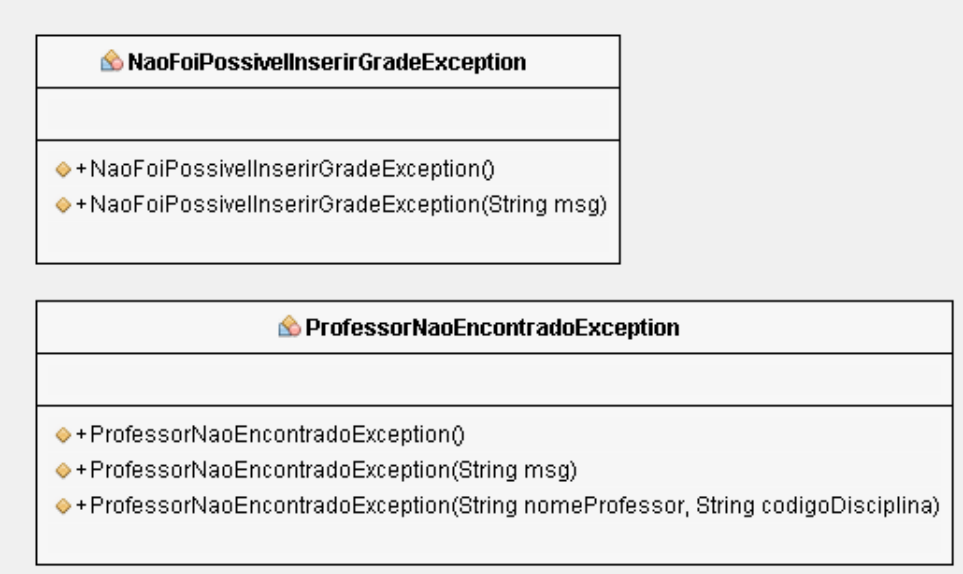


Fonte: Elaborada pelo autor

Figura 27 – A classe Disciplina

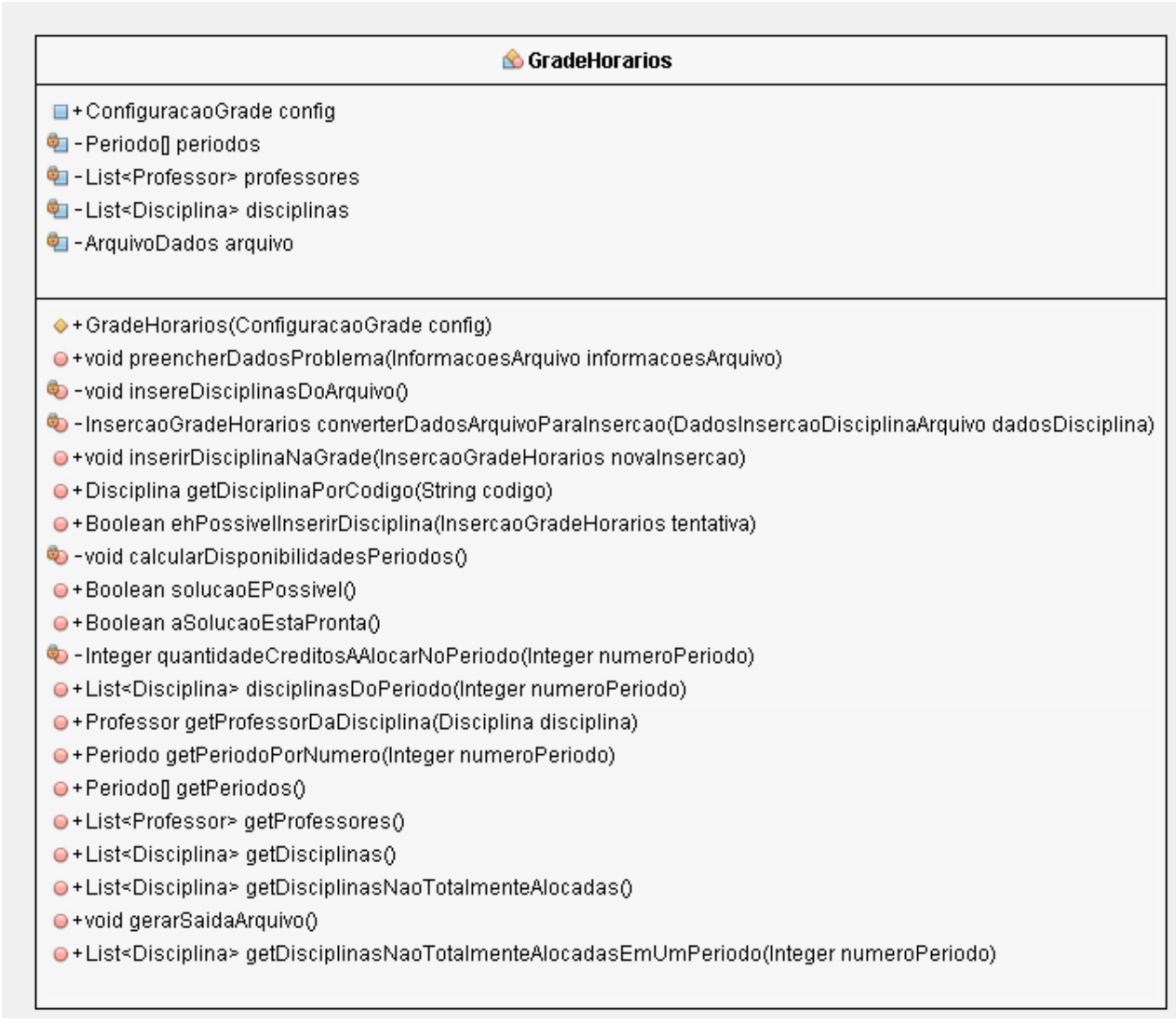
Fonte: Elaborada pelo autor

Figura 28 – As classes de exceção



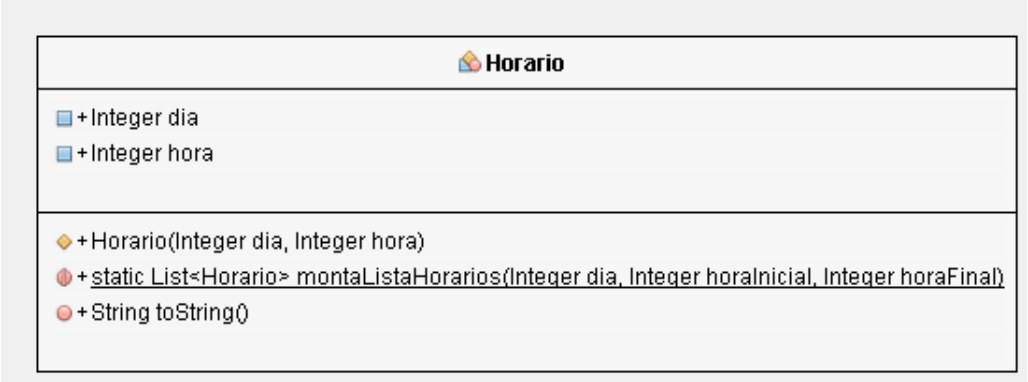
Fonte: Elaborada pelo autor

Figura 29 – A classe GradeHorarios



Fonte: Elaborada pelo autor

Figura 30 – A classe Horario

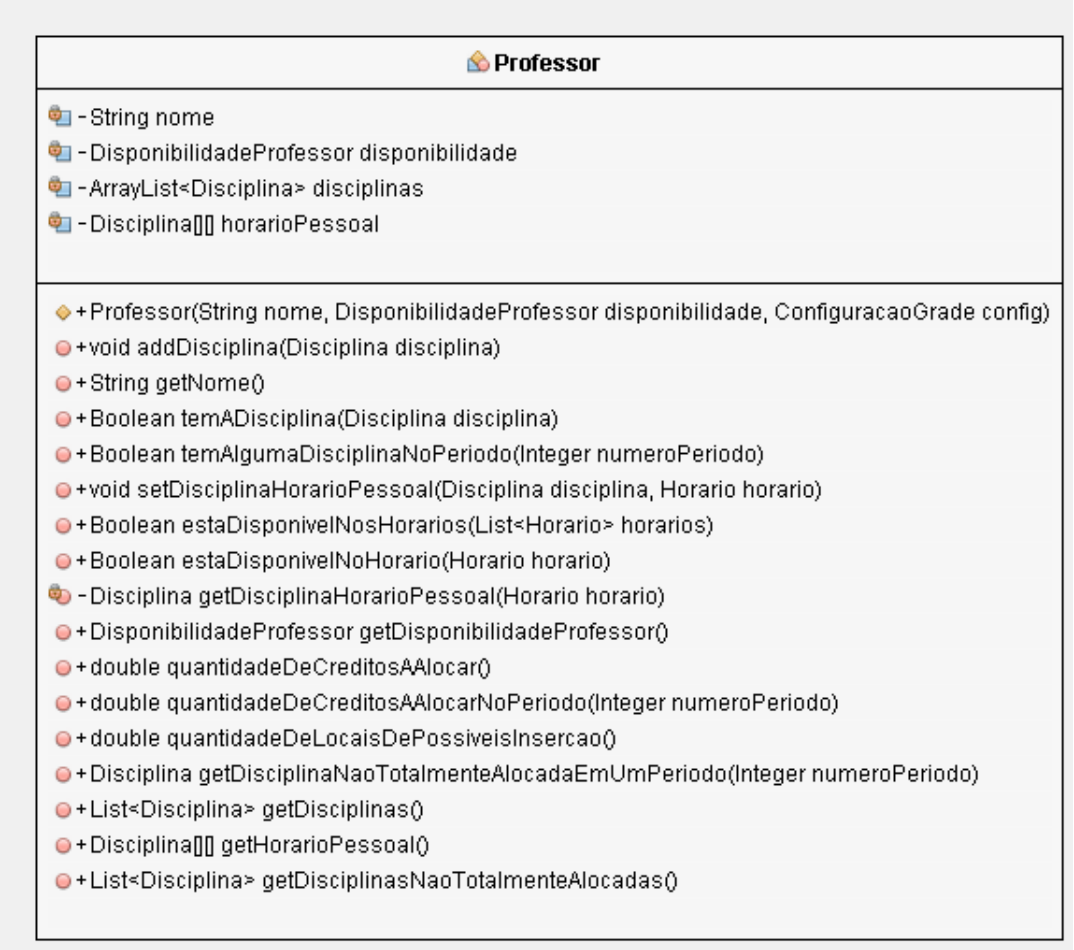


Fonte: Elaborada pelo autor

Figura 31 – A classe Periodo

Fonte: Elaborada pelo autor

Figura 32 – A classe Professor



Fonte: Elaborada pelo autor

Figura 33 – A classe TeacherScheduleProblem

Fonte: Elaborada pelo autor