UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Eduardo Henrique Hoffmann Kalfels

ALGORITMOS GENÉTICOS COMO UMA ABORDAGEM PARA A ALOCAÇÃO DE GRADE DE HORÁRIOS DA UFSC

Florianópolis

2017

Eduardo Henrique Hoffmann Kalfels

ALGORITMOS GENÉTICOS COMO UMA ABORDAGEM PARA A ALOCAÇÃO DE GRADE DE HORÁRIOS DA UFSC

Trabalho de Conclusão de Curso submetido ao curso Sistemas de Informação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Sistemas de Informação. Orientador: Prof. Elder Rizzon Santos

Florianópolis

2017

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Kalfels, Eduardo Henrique Hoffmann
Algoritmos Genéticos como uma abordagem para a alocação de grade de horários da UFSC / Eduardo Henrique Hoffmann Kalfels ; orientador, Elder Rizzon Santos, 2017.
131 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Sistema de Informação, Florianópolis, 2017.

Inclui referências.

1. Sistema de Informação. 2. Inteligência artificial. 3. Algoritmos genéticos. 4. Conexionismo. 5. Timetabling. I. Santos, Elder Rizzon. II. Universidade Federal de Santa Catarina. Graduação em Sistema de Informação. III. Título.

Eduardo Henrique Hoffmann Kalfels

ALGORITMOS GENÉTICOS COMO UMA ABORDAGEM PARA A ALOCAÇÃO DE GRADE DE HORÁRIOS DA UFSC

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Sistemas de Informação", e aprovado em sua forma final pelo curso Sistemas de Informação da Universidade Federal de Santa Catarina.

Florianópolis, 29 de junho 2017.

Prof. Elder Rizzon Santos
Orientador

Banca Examinadora:

Prof. João Cândido Dovicchi

Prof. Ricardo Azambuja Silveira

RESUMO

Em diversas situações precisa-se de soluções para problemas que são de difícil manipulação através de técnicas tradicionais de computação, devido à alta complexidade e ao grande tamanho do espaço de busca. Com o objetivo de facilitar a resolução destes problemas, de forma eficiente e flexível, são propostos os Algoritmos Genéticos, os quais utilizam regras de transição probabilísticas e não regras determinísticas, para encontrar uma solução ótima - ou quase ótima - para o problema. Este trabalho propõe o estudo de diferentes técnicas, perspectivas e abordagens de algoritmos genéticos, além da pesquisa sobre o problema de *timetabling* e as abordagens encontradas para solucionar este problema.

Com o estudo de diferentes aplicações de algoritmos genéticos, e da complexidade da solução para o problema de *timetabling*, este trabalho propõe um modelo de algoritmo genético para a aplicação no contexto de geração de grade de horários para as aulas da UFSC, alocando turmas e professores.

Ao final do trabalho são apresentados os resultados obtidos com o algoritmo desenvolvido e uma análise sobre a abordagem utilizada. São feitas também considerações sobre o desenvolvimento do trabalho, possíveis melhorias e trabalhos futuros.

Palavras-chave: inteligência artificial, algoritmo genético, conexionismo, timetabling

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Além de escolas e universidades, empresas e outras organizações enfrentam problemas para gerar grades de horários que exigem que diversas regras sejam respeitadas, consideram preferências das pessoas envolvidas, disponibilidade de horários, e diversos outros fatores que tornam a tarefa bastante complexa.

Os resultados gerados através de abordagens manuais ou algoritmos determinísticos são ruins do ponto de vista econômico, assim como do ponto de vista das pessoas envolvidas, as quais são fortemente afetadas pelas soluções geradas (BURKE; CAUSMAECKER; BERGHE, 2004).

Dado este problema, uma abordagem alternativa dá-se através dos Algoritmos Genéticos, que estão inseridos no campo de estudo de Inteligência Artificial (IA). Uma popular e inicial definição de IA foi introduzida por John McCarthy na famosa conferência de Dartmouth em 1956: "fazer a máquina comportar-se de tal forma que seja chamada inteligente caso fosse este o comportamento de um ser humano."Porém, ao apenas imitar o comportamento do ser humano, esta definição parece ignorar a existência da IA forte: proposta em que busca-se fazer com que a máquina raciocine de forma autônoma, igual ou melhor que a mente humana.

Duas abordagens comuns no campo de IA são a simbólica - que envolve a manipulação de símbolos e conceitos abstratos - e a conexionista - onde o raciocínio não é realizado através de símbolos, mas de conexões.

É nesta abordagem conexionista, ou sub-simbólica, em que aparecem os algoritmos genéticos. De modo geral, nesta abordagem a semântica do domínio não precisa ser introduzida explicitamente, e o sistema pode inferir o conhecimento, através de um processo de aprendizagem. Este conhecimento aprendido não costuma ser facilmente interpretável pelo usuário.

Para diversos problemas, em diferentes ambientes, podemos utilizar algoritmos genéticos para buscar a melhor solução possível. Algoritmo Genético (AG), segundo (MITCHELL, 1996), é uma técnica de busca utilizada na ciência da computação para encontrar soluções aproximadas em problemas de otimização e busca, fundamentado principalmente pelo americano John Henry Holland. Algoritmos genéticos são uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva como hereditariedade, mutação, seleção natural e recombinação (ou crossing over) (GOLDBERG, 1989).

Existem diversas situações possíveis em que pode-se fazer uso de um algoritmo genético, incluindo: otimização de rotas, alocação de recursos em uma grade de horários, maximização da eficiência de uma termoelétrica.

Este trabalho propõe elaborar um algoritmo genético para lidar com a otimização na alocação de turmas e professores na grade de horários das turmas das disciplinas (do departamento INE) obrigatórias do curso de Sistemas de Informação da UFSC, a fim de encontrar uma solução satisfatória para o cenário.

1.1 MOTIVAÇÃO

A partir da observação pessoal das alocações de disciplinas e professores na universidade, criou-se o interesse em entender melhor o processo realizado para a geração das grades de horários do curso de Sistemas de Informação.

Com a pesquisa realizada sobre o assunto, constatou-se que trata-se de um trabalho manual e que demanda grande esforço do responsável pelas alocações. Diversas regras e preferências foram identificadas no ambiente da UFSC, o que implica em muitas combinações possíveis de alocações e, ao mesmo tempo, grande dificuldade em avaliar as alocações de forma quantitativa: através de índices e números.

Dado isso, este trabalho é motivado pelos fatores descritos acima, somados à busca de uma abordagem que automatize a solução do problema de *timetabling* e atinja resultados satisfatórios e passíveis de avaliação objetiva.

1.2 OBJETIVOS

O objetivo deste trabalho será apresentar uma solução para o problema de geração automatizada de grades de horários de turmas do curso de Sistemas de Informação da UFSC, considerando diversos critérios observados na prática como disponibilidade de horários de professores, diversas turmas por disciplina e densidade de aulas de uma mesma fase do curso.

1.2.1 Objetivos específicos

- Analisar o estado da arte em algoritmos genéticos e áreas relacionadas;
- Levantar as principais áreas de aplicação e métodos para a construção de algoritmos genéticos
- Construir um algoritmo genético para solucionar o problema proposto
- Utilizar este algoritmo a partir de testes e analisar os resultados obtidos

1.3 METODOLOGIA DE PESQUISA

A pesquisa deste trabalho será feita através de livros, fontes na internet e trabalhos referentes ao assunto.

Para atingir os objetivos definidos, será realizado um estudo do trabalho teórico e prático realizado até hoje com os algoritmos genéticos. Serão estudadas as propriedades, particularidades e características dos algoritmos genéticos, através da leitura de artigos científicos publicados em periódicos e conferências com boa reputação na área de estudo.

Além disso, serão analisados os requisitos do problema, afim de direcionar o algoritmo para que este gere soluções adequadas para o uso real.

1.4 ORGANIZAÇÃO DO TEXTO

Este trabalho está organizado em cinco capítulos, sendo o presente capítulo responsável por apresentar as informações gerais sobre o trabalho, com objetivo introdutório, descrevendo os objetivos definidos e a metodologia de pesquisa que será utilizada.

O segundo capítulo busca apresentar conceitos importantes sobre os temas abordados ao longo do trabalho, algoritmos genéticos e o problema de *timetabling*. Além disso, o trabalho é contextualizado dentre outras aplicações conhecidas para os algoritmos genéticos.

No terceiro capítulo é apresentada a solução proposta - através de um algoritmo genético - para o problema encontrado, a arquitetura

base, características e detalhes da implementação do algoritmo genético.

No quarto capítulo, são apresentados os resultados obtidos com execuções de testes da solução, descrevendo características da implementação e seu funcionamento.

Por fim, no quinto capítulo são apresentadas as conclusões e considerações finais sobre a solução implementada, além de sugestões de trabalhos futuros na solução implementada.

2 REFERENCIAL TEÓRICO

2.1 PROBLEMA DE ALOCAÇÃO DE HORÁRIOS E RECURSOS

A geração de grades de horários em cursos da UFSC, considerando diversas restrições e a disponibilidade de diferentes recursos, é um problema de *timetabling*, definido por (WREN, 2006). Esse autor define *timetabling* como a alocação, sujeita a restrições, de dados a objetos distribuídos no tempo e no espaço, de maneira a satisfazer da melhor possível um conjunto de objetivos desejáveis. Outra definição para *timetabling*, segundo (MÜLLER, 2005), é o procedimento de alocar atividades no tempo e espaço de forma a atingir objetivos desejáveis assim como considerar restrições conhecidas de acordo com o contexto.

Estes objetivos podem ser: maximizar a eficiência no uso de recursos, considerar os desejos e preferências de recursos humanos (como médicos, professores, funcionários, que desejam trabalhar mais ou menos em determinados dias ou horários). Enquanto que restrições podem ser exemplificadas com: recursos que não podem ser compartilhados em um mesmo período de tempo, outros recursos que podem ou devem ser compartilhados por uma quantidade de atividades simultaneamente.

Abordagens sistemáticas com o objetivo de gerar timetables satisfatórios são vistas com grande importância por (BURKE; CAUSMAECKER; BERGHE, 2004). Uma aplicação importante citada pelo autor é a área médica, cujos pacientes com necessidades de cuidados médicos devem ser atendidos com agilidade e eficiência, assim como as demandas gerenciais da organização. Um ponto muito importante na geração automatizada de alocações de horários e recursos é o impacto significativo no uso do tempo do(s) responsável(eis) pelo planejamento deste cenários. Além disso, as abordagens automatizadas são caracterizadas pela potencialização da melhoria e otimização do processo de *timetabling* e pela qualidade dos timetables gerados. As abordagens matemáticas ou heurísticas permitem - além da geração de diversas soluções com variadas características - a criação e análise de métricas quantitativas e qualitativas das soluções geradas, assim como a inclusão de preferências, conforme dito anteriormente, que atendam os desejos da organização ou dos próprios funcionários.

2.2 INTELIGÊNCIA COMPUTACIONAL

Inteligência Computacional é uma área da ciência da computação que busca, através de técnicas inspiradas na Natureza, o desenvolvimento de sistemas inteligentes que imitam aspectos do comportamento humano, tais como: aprendizado, percepção, raciocínio, evolução e adaptação. Algumas abordagens da inteligência computacional são descritas a seguir:

- Redes Neurais: modelos computacionais não lineares, desenvolvidos baseados na concepção da estrutura e da metodologia operacional do cérebro humano, e procurando reproduzir características humanas, dentre as quais: aprendizado, associação, generalização e abstração. Algumas aplicações para esta técnica computacional são: aprendizado de padrões a partir de dados não lineares, incompletos, com ruído ou compostos de exemplos contraditórios.

- Algoritmos Genéticos: algoritmos matemáticos, cuja concepção inspira-se nos mecanismos de evolução natural e recombinação genética, baseados no princípio de reprodução e sobrevivência dos mais aptos de Darwin, fornecendo uma forma de busca adaptativa.

- Lógica Nebulosa (Fuzzy): pretende modelar o modo aproximado do raciocínio humano, permitindo a tomada de decisões racionais em um ambiente de incerteza e imprecisão, aplicadas ao desenvolvimento de sistemas computacionais. Esta lógica oferece um mecanismo para manipulação de informações imprecisas, obtendo respostas aproximadas a partir de conhecimento inexato, incompleto ou não totalmente confiável.

- Sistemas Especialistas: programas computacionais com o objetivo de solucionar problemas em um campo especializado do conhecimento humano. Utiliza algumas técnicas de IA, base de conhecimento e raciocínio inferencial para o seu processamento.

Dado isso, este trabalho tem como objetivo estudar as abordagens e detalhes dos Algoritmos Genéticos para utilizar na solução do problema que será descrito mais à frente. Mais detalhes sobre os Algoritmos Genéticos serão explicados na seção seguinte.

2.3 ALGORITMOS GENÉTICOS

Esta seção deste documento irá apresentar o conceito de um Algoritmo Genético, apresentando analogias e definições de forma a tornar compreensível a aplicabilidade desta abordagem para o problema definido. Além disso, serão explicados os detalhes de sua estrutura, diferentes formas de implementação e aspectos que devem ser considerados ao desenvolver um AG.

2.3.1 Princípios e Aplicações de AGs

A abordagem através de Algoritmos Genéticos é uma técnica de busca e otimização, altamente paralela, inspirada no princípio Darwiniano de seleção natural e reprodução genética. De acordo com a teoria de Darwin, o princípio do processo de seleção permite maior longevidade aos indivíduos mais aptos, ou seja, maior probabilidade de reprodução. Estes indivíduos têm maior chance de perpetuar os seus códigos genéticos para as próximos gerações. Os códigos genéticos constituem a identidade de cada indivíduo e são representados nos cromossomos.

Os AGs são construídos imitando estes princípios, buscando a melhor solução para um determinado problema, por meio da evolução de conjuntos (populações) de soluções (indivíduos) codificados como cromossomos artificiais. Cromossomo é implementado como uma estrutura de dados para representar uma possível solução no espaço de busca do problema. Submete-se, então, estes cromossomos a um processo evolucionário, constituído pela avaliação, seleção, crossover e mutação. Ao longo da execução de vários ciclos deste processo a população resultante tende a conter indivíduos cada vez mais aptos, ou seja, soluções melhores.

Podemos representar uma analogia entre AGs e o sistema natural através da seguinte tabela:

Tabela 1 – Tabela analogia AGs

Natureza	Algoritmos Genéticos
Cromossomo	Palavra binária, vetor, etc
Gene	Característica do problema
Alelo	Valor da característica
Loco	Posição na palavra, vetor
Genótipo	Estrutura
Fenótipo	Estrutura submetida ao problema
Indivíduo	Solução
Geração	Ciclo

Os componentes dos Algoritmos Genéticos são apresentados da seguinte forma:

1. Problema a ser otimizado
2. Representação das Soluções do Problema
3. Decodificação do cromossomo
4. Avaliação
5. Seleção
6. Operadores Genéticos
7. Inicialização da População
8. Parâmetros e Critérios de Parada

2.3.2 Problema

Problema complexo de otimização: com diversos parâmetros ou combinações de características visando a melhor solução; com muitas restrições ou condições que não podem ser representadas matematicamente; com grandes espaços de busca. Ex.: rota a ser seguida por um caminhão de lixo, que deve percorrer todas as ruas de um bairro, em um tempo satisfatório e percorrendo a menor distância possível; alocação de médicos e enfermeiros na grade horária de um hospital, considerando recursos (salas, equipamentos, ferramentas), restrições e preferências.

2.3.3 Representação das soluções do problema

É constituída pela definição da estrutura do cromossomo que será manipulado pelo algoritmo. Esta etapa depende do tipo do problema a ser representado e do que pretende-se manipular geneticamente. Alguns dos principais tipos de representação são: binária (problemas numéricos, inteiros), números reais (problemas numéricos), permutação de símbolos (baseados em ordem), símbolos repetidos (grupamento).

A representação através de uma sequência de bits e o indivíduo como a concatenação dessas sequências de todos os atributos é uma das principais formas. Variação de codificações binárias podem ser encontradas em (CARUANA; SCHAFFER, 1988; HOLLAND, 1975).

É também muito utilizada a codificação usando o próprio alfabeto do atributo de um indivíduo que deve ser representado (letras, códigos, números reais, etc.). Alguns exemplos disto podem ser encontrados em (MEYER; PACKARD, 1992; KITANO, 1994).

Existem diversas outras formas de representar um indivíduo, de modo geral a escolha da forma mais adequado está fortemente ligada ao problema que o algoritmo visa resolver.

2.3.4 Decodificação do cromossomo

Esta etapa compreende a proposição da solução real do problema, a partir da decodificação do cromossomo gerado pelo algoritmo. Neste momento, a escolha pela representação binária mostra-se vantajosa devido à facilidade de transformação para um número inteiro ou real.

2.3.5 Função de avaliação

Componente essencial na ligação entre os algoritmos genéticos e o "mundo real", a avaliação é formada pela aplicação de uma função sobre cada indivíduo da população corrente, com o objetivo de medir a aptidão destes em relação ao problema. Esta função deve ser elaborada de forma a atingir a melhor representação do problema, já que a sua aplicação dirigirá o processo de busca pela melhor solução. A função de avaliação é para um AG o que o meio ambiente é para seres humanos. É uma etapa muito especializada do algoritmo, pois depende essencialmente das características de cada problema.

2.3.6 Métodos de seleção

Processo que seleciona quais indivíduos da população corrente realização a reprodução. Neste momento utiliza-se o valor de aptidão, medido pelo processo anterior, para que indivíduos mais aptos tenham maior probabilidade de participarem da reprodução. Podemos utilizar a seguinte função para representar a probabilidade de cada indivíduo ser selecionado:

(PACHECO, 1999) apresenta a seguinte equação para demonstrar a probabilidade pi do indivíduo i ser selecionado, se fi é a avaliação do indivíduo i na população corrente, proporcional a

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i}$$

Figura 1 Onde N é o número de indivíduos na população.

A seleção é tipicamente implementada utilizando o algoritmo de "roleta", onde cada indivíduo é representado por uma fatia proporcional à sua aptidão, relativa ao resto da população.

O operador utilizado na seleção é um componente muito importante na elaboração do algoritmo genético. Os cinco principais mecanismos identificados pela literatura são: proporcional, por torneios, com truncamento, por normalização linear e por normalização exponencial (BLICKLE, 1996). Cada um destes mecanismos é caracterizado pela pressão seletiva ou intensidade de seleção que ele introduz no algoritmo genético. O termo "pressão seletiva" possui diferentes conotações na literatura de computação evolucionária. A "intensidade de seleção" empregada é definida pela variação na aptidão média de determinada população, induzida pelo método de seleção (BLICKLE, 1996).

Na seleção proporcional, a probabilidade de um indivíduo ser selecionado é proporcional ao valor da sua aptidão. Sendo assim, a pressão seletiva é caracterizada pela razão entre o desvio padrão das aptidões e a médias destas. Identificam-se dois principais problemas no método de seleção proporcional: a existência de super-indivíduos, que ocorre quando um indivíduo detém um valor de aptidão muito maior que dos demais, fazendo com que o algoritmo convirja prematuramente; a competição próxima, esta ocorrendo na situação oposta em que indi-

víduos apresentam aptidões muito semelhantes, porém diferentes. Isto faz com que a intensidade de seleção seja menor do que a desejável.

O método de seleção por torneios escolhe um grupo de t indivíduos aleatoriamente, e a partir deste grupo o indivíduo com maior aptidão é selecionado. Assumindo uma distribuição Gaussiana de valores de aptidão por indivíduo, têm-se a seguinte equação integral como representação da intensidade de seleção (BLICKLE, 1996):

$$I = \int_{-\infty}^{\infty} t \cdot x \cdot \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{y^2}{2}} dy \right)^{t-1} dx$$

Figura 2 Integral que representa a intensidade da seleção pelo método de torneios.

Com isso, observa-se que a pressão seletiva, utilizando este método de seleção, aumenta ao passo em que o número de indivíduos envolvidos no torneio aumenta.

Já o mecanismo de seleção por truncamento define um limiar T, onde somente os T melhores indivíduos terão chance de serem selecionados, sendo que cada um destes terá a mesma probabilidade de ser selecionado. Desta forma, a pressão seletiva diminui a medida em que T aumenta. Isso pode ser observado através de um gráfico traçado da intensidade de seleção em função do limiar T, conforme a fórmula de intensidade abaixo (BLICKLE, 1996):

$$I = \frac{1}{T} \cdot \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{f_c^2}{2}}$$

Figura 3 Equação que representa a intensidade da seleção pelo método de truncamento.

Sendo que fc é o valor de menor aptidão dos T melhores indivíduos pré-selecionados.

O método de seleção por normalização linear funciona da seguinte forma: ordena-se os indivíduos pelo seu valor de aptidão; valores de aptidão são alterados de acordo com a posição relativa de cada indivíduo; o indivíduo de melhor aptidão recebe um valor max e o de pior aptidão recebe um valor min, sendo que os valores max e min devem ser

definidos de forma a satisfazer as condições max = 2 - min e min >= 0; os demais indivíduos têm seus valores de aptidão linearmente distribuídos entre min e max, de acordo com a sua posição determinada na ordenação (i=1 corresponde ao pior elemento). Neste método a pressão seletiva diminui a medida em que min aumenta, conforme observa-se na fórmula da intensidade de seleção abaixo (PACHECO, 1999):

$$I = (1 - mín)\frac{1}{\sqrt{\pi}}$$

Figura 4 Equação que representa a intensidade da seleção por normalização.

A diferença do método de seleção por normalização exponencial diferencia-se da normalização linear pela forma que são atribuídas as probabilidades de seleção de cada indivíduo, seguindo uma função exponencial. Neste caso a pressão seletiva diminui a medida em que o grau de 'exponencialidade' da função aumenta (BLICKLE, 1996).

2.3.7 Operadores genéticos

Após a seleção de um conjunto de indivíduos, realiza-se a recombinação dos seus materiais genéticos, através do operador de crossover. Este operador é considerado a característica fundamental dos algoritmos genéticos. Indivíduos selecionados para reprodução são combinados em pares aleatoriamente, baseados na aptidão, e novos indivíduos são criados a partir da troca do material genético. Isso faz com que os descendentes sejam diferentes dos pais, porém com características de ambos os genitores.

Uma forma muito simples de crossover é através de um ponto de corte, onde o material genético dos genitores é cortado em uma posição escolhida de forma aleatória, trocando um "pedaço" de um genitor, pelo pedaço correspondente do outro genitor.

Os cromossomos gerados pelo operador de crossover são submetidos a operação de mutação, sendo este um operador exploratório cujo objetivo é aumentar a diversidade da população. Este operador efetua a troca do valor de uma posição do cromossomo, com uma probabilidade geralmente baixa (<1%).

2.3.8 Inicializador da população

Este processo consiste na criação de indivíduos - a população inicial, tipicamente de forma aleatória, para formar a população que será submetida no primeiro ciclo do algoritmo genético. Dependendo do contexto do problema, pode-se conhecer boas "sementes" para que a população inicial aleatória seja formada por indivíduos com bons cromossomos, permitindo uma evolução mais rápida.

Boas soluções para um problema costumam ser encontradas de forma eficiente através da execução de evoluções sucessivas, ou seja, rodadas do algoritmo, semeando-se a população inicial da evolução seguinte com as melhores soluções encontradas na anterior.

2.3.9 Parâmetros e critérios de parada

A seguir são apresentados os parâmetros de controle do processo evolucionário em um algoritmo genético:

- Tamanho da população: número de pontos do espaço de busca sendo considerados em paralelo a cada ciclo.
- Taxa de Crossover: probabilidade (pc) de um indivíduo ser recombinado com outro.
- Taxa de Mutação: probabilidade (pm) do conteúdo de uma posição/gene do cromossomo ser alterado.
- Número de Gerações: total de ciclos de evolução de um AG.
- Total de Indivíduos: total de tentativas em um experimento (tamanho da população x número de gerações).

Resumidamente, um algoritmo genético pode ser descrito como um processo contínuo de repetição de ciclos evolucionários, encerrados por um critério de parada, que costuma ser composto pelos dois últimos parâmetros apresentados.

2.4 APLICAÇÕES

Nesta seção será apresentado, primeiramente, um trabalho realizado sobre o problema de *timetabling* utilizando também a abordagem

de algoritmos genéticos. Além disso, serão apresentadas outras duas aplicações de algoritmos genéticos encontradas durante a pesquisa.

2.4.1 Geração de grades de horário para escalas de trabalho utilizando algoritmos genéticos

O Trabalho de Conclusão de Curso (CAMPOS; RIBEIRO, 2012) serviu de apoio para a pesquisa do problema de *timetabling* e a solução através da elaboração de um algoritmo genético.

Os autores buscam apresentar uma solução para o problema de geração automatizada de grades de horários de escala trabalho para empresas, considerando diversos critérios observados na prática como disponibilidade e aptidão de recursos, legislação trabalhista, preferências pessoais dos funcionários e distribuição da demanda de trabalho ao longo de determinado período.

Durante o trabalho, são utilizadas como referências as necessidades de uma empresa de *call center* que precisa planejar as jornadas de trabalho de acordo com as exigências contratuais e com a distribuição da demanda de trabalho atendendo da melhor forma possível estes critérios.

2.4.2 Metodologia VeriSC com Algoritmos Genéticos

No trabalho desenvolvido na dissertação de mestrado (FRANCO, 2014) o autor implementa um algoritmo que se baseia no conceito de algoritmos genéticos visando otimizar a geração de dados da etapa de verificação funcional na metodologia VeriSC.

A metodologia VeriSC é uma metodologia de verificação funcional que guia a verificação funcional, definindo a geração do testbench que pode ser gerado antes mesmo do DUV (Device Under Verification). O ambiente de testes pode ser gerado através da ferramenta eTBc (Easy Testbench Creator) criando automaticamente parte desse ambiente necessitando, apenas, que o engenheiro de verificação descreva algumas funcionalidades do dispositivo e das variáveis de verificação. A geração de dados utilizada por essa metodologia é realizada pela biblioteca SystemC Verification Library (SILVA, 2007), que gera dados pseudo aleatórios. Portanto, se faz necessária uma análise sobre quais dados devem ser gerados, pretendendo estimular as funcionalidades do dispositivo de forma otimizada.

A implementação do AG foi desenvolvida na linguagem de programação C++. Essa linguagem foi escolhida porque é possível sua comunicação com a linguagem utilizada na implementação dos módulos, a linguagem SystemC. A linguagem C++ possibilita um nível de abstração maior, em comparação com a linguagem SystemC, permitindo que a implementação possa ocorrer no nível de transações (alto nível) ao invés de ser realizada no nível de sinais (baixo nível). Isto proporciona maior facilidade na implementação das funcionalidades do AG.

Duas implementações do AG foram realizadas, na qual uma implementação utilizou a estrutura de dados vetor, à medida que a outra utilizou a estrutura de dados lista encadeada. Essas estruturas de dados definem como o cromossomo de cada indivíduo será implementado. Na implementação que utiliza a estrutura vetor, cada vetor corresponde a um indivíduo e cada posição desse vetor equivale a um gene. Já na estrutura lista encadeada, cada gene corresponde a uma célula e a lista corresponde a um indivíduo.

2.4.2.1 Avaliação

A função de aptidão do algoritmo é definida pelas condições presentes nos buckets do experimento. Os buckets descrevem os critérios de cobertura para cada experimento e são esses critérios que deverão ser avaliados pela função de aptidão. A função de aptidão poderá avaliar os dados de entrada e/ou de saída da simulação, de acordo com os critérios de cobertura. Para isto, esses dados deverão ser encaminhados para o módulo Source por meio do sinal de realimentação. Caso os dados estejam em módulos diferentes do Checker, deverá ser criado um sinal interligando esses módulos com o objetivo de enviar os dados para o Checker e, em seguida, enviá-los para o Source através do sinal de realimentação. É importante observar que a função de aptidão irá atribuir uma nota ao indivíduo de entrada, avaliando seus dados de entrada e/ou saída da simulação.

1. Função de aptidão do bucket 1 :
   (a) variável de saída >= 30 por 6.000 vezes;
   (b) variável de saída <= -50 por 9.000 vezes;
2. Função de aptidão do bucket 2 :
   (a) variável de entrada >= 10 e <= 80 por 200.000 vezes;

(b) variável de entrada >= 150 e <= 160 por 50.000 vezes.

Se o dado avaliado estiver dentro da faixa de valores determinados pela função de aptidão, o indivíduo receberá +1 em sua nota. Por outro lado, a nota não será modificada caso o dado avaliado não obedeça às condições da função de aptidão.

O valor padrão para pontuação de um indivíduo é +1, contudo, existem algumas condições mais difíceis de serem satisfeitas e podem ser atribuídos pesos para essas condições.

A segunda condição da função de aptidão do bucket 2, por exemplo, requer que a variável de entrada possua um valor entre 150 e 160 (intervalo de 10 números inteiros). É possível, então, aumentar o peso da nota desta condição (aumentar +2 ou +3 no valor atribuído na nota) para que os indivíduos que a satisfizerem, recebam uma nota maior e ter mais chances de terem seus genes propagados para as próximas gerações.

2.4.2.2 Seleção

Para a seleção, foram utilizados os métodos: roleta viciada e torneio. O método da roleta utiliza o valor de aptidão dos indivíduos para calcular a porção da roleta que cada um possuirá. A porção da roleta substituirá o valor de aptidão em sua devida localização (primeira posição no vetor da estrutura população ou no vetor auxiliar). É gerado um valor aleatoriamente, entre o intervalo 0 - 359 (considerando a circunferência de uma roleta subtraídos de um), que determinará qual foi a porção escolhida da roleta e, consequentemente, qual indivíduo foi selecionado.

O método do torneio seleciona dois indivíduos de forma aleatória para participar do torneio. Será comparado o valor de aptidão desses dois indivíduos e aquele que possuir o maior valor, será o escolhido para gerar a nova população.

Os dois métodos foram escolhidos devido a diferença na abordagem da convergência genética, na qual o primeiro método possui alta probabilidade de ocorrer a convergência, enquanto que o segundo, possui baixa probabilidade. Selecionados os indivíduos pais, serão aplicados sobre eles os operadores genéticos. Serão gerados, então, os indivíduos filhos que deverão ser melhores do que os indivíduos pais.

2.4.2.3 Operadores Genéticos

Os operadores genéticos utilizados foram os operadores de recombinação e mutação.

No trabalho (FRANCO, 2014), a combinação do cromossomo dos indivíduos pais foi realizada por meio da recombinação de um ponto. Essa recombinação divide o cromossomo dos indivíduos pais em dois e os alterna, produzindo os cromossomos dos indivíduos filhos. Uma segunda variável aleatória é necessária para determinar a posição do ponto de corte nos cromossomos. O intervalo desta variável corresponde ao início e fim do cromossomo dos indivíduos pais, podendo ocorrer entre indivíduos que possuam o tamanho de seus cromossomos distintos. Após a geração dos indivíduos filhos, eles serão encaminhados ao operador de mutação.

A implementação do operador de recombinação se diferencia para as duas estruturas de dados utilizadas na codificação do algoritmo. Foi necessário a modificação do modo em que o vetor população foi manipulado. Para a estrutura de dados vetor, a recombinação foi realizada conforme descrito na figura abaixo.

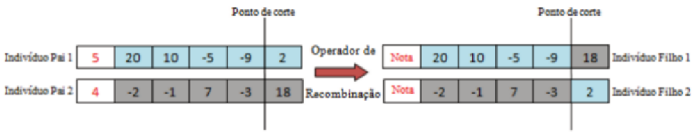

Figura 5 Operador de recombinação sendo aplicado na codificação realizada por meio da estrutura de dados vetor.

A recombinação de indivíduos que implementaram a estrutura de dados lista encadeada, foi realizada conforme descrito na figura abaixo.

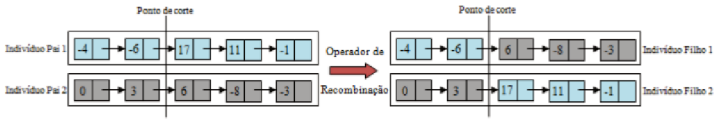

Figura 6 Operador de recombinação sendo aplicado na codificação realizada por meio da estrutura de dados lista encadeada.

2.4.2.4 Resultados Obtidos

Os resultados iniciais do trabalho (FRANCO, 2014) começaram a ser obtidos durante o período de desenvolvimento da nova geração de dados para a metodologia VeriSC. Os resultados podem ser listados da seguinte forma: 1) como as etapas de estudo aprofundado da metodologia VeriSC e dos algoritmos genéticos; 2) implementação do algoritmo na geração de dados; 3) estudo e compreensão dos problemas a serem otimizados; 4) introdução dessa geração de dados na metodologia VeriSC. Somente após essas etapas, foi possível obter os resultados experimentais.

Após diversos experimentos, testando as possíveis combinações de configuração do AG (Lista ou Vetor x Roleta ou Torneio), além da própria implementação da biblioteca SCV, concluiu-se que a geração de dados utilizando os Algoritmos Genéticos reduziu o tempo de verificação funcional dentro da metodologia VeriSC, obtendo sucesso nos objetivos definidos. Com essa nova abordagem, além de reduzir o tempo de verificação e, consequentemente, de todo o projeto de desenvolvimento de hardware, os dados utilizados na verificação são distintos, tornando a verificação mais robusta e aumentando a confiabilidade do dispositivo.

2.4.3 Seleção de Atributos Utilizando Algoritmos Genéticos

O artigo (SUMBANA et al., 2012b) apresenta uma abordagem baseada em algoritmos genéticos para reduzir o número de atributos utilizados na mineração de textos elaborada para a detecção automática de vandalismo na Wikipedia. Os resultados experimentais mostram que a abordagem proposta é capaz de reduzir o número de atributos em até 83% sem alteração significativa na efetividade da detecção. Mostra-se também que a técnica proposta é capaz de selecionar um número reduzido de atributos que produzem uma efetividade na classificação superior ao conjunto selecionado por uma técnica de seleção de atributos considerada estado-da-arte, o Information Gain.

A motivação da abordagem de classificação binária para a detecção de vandalismo na Wikipedia se dá principalmente pela alta ocorrência de inserções ou alterações maliciosas de conteúdo, gerando uma forte demanda por abordagens mais eficazes do que uma simples detecção manual realizada por voluntários.

O processo de seleção de atributos é estudado há muitos anos nas

áreas de estatística e reconhecimento de padrões (JAIN; DUIN; MAO, 2000) e posteriormente na área de aprendizado de máquina (PAPPA; FREITAS; KAESTNER, 2002). A seleção de atributos tem como objetivo descobrir um subconjunto de atributos mais relevantes para uma tarefa alvo, considerando os atributos iniciais, sendo importante por tornar o processo de aprendizagem mais eficiente, entre outras coisas. Segundo (JAIN; DUIN; MAO, 2000), a redução da dimensionalidade do espaço de atributos pode diminuir o custo de processamento, aumentar a acurácia da classificação, além de diminuir a chance de super-especialização do modelo gerado, que não generaliza para novos conjuntos de dados.

Neste artigo, os autores buscaram maximizar a eficácia e a eficiência do processo de detecção de vandalismo na Wikipedia. Isso foi possível através do método de detecção proposto em (SUMBANA et al., 2012a), focando agora em reduzir o número de atributos usados, mantendo resultados em termos de MacroF1 (métrica obtida através de um cálculo sobre a precisão e o recall) comparáveis aos obtidos quando todos os atributos originalmente definidos na coleção de dados adotada são utilizados. Para atingir esta redução no conjunto de atributos utilizou-se a técnica de algoritmos genéticos (AG), a qual tem se mostrado eficaz na seleção de atributos (PAPPA; FREITAS; KAESTNER, 2002), para selecionar o subconjunto que produz melhores resultados. Nos experimentos realizados obtiveram reduções de até 83% no número de atributos ao passo em que mantiveram a efetividade da detecção em níveis similares aos obtidos com o uso de todos os atributos (ou até mesmo superando-os). Sobretudo, mostrou-se que a técnica proposta é capaz de selecionar um número reduzido de atributos que produzem uma efetividade na classificação superior ao conjunto selecionado por uma técnica do estado-da-arte de seleção de atributos, o Information Gain (MOORE, 2012), com o mesmo número de atributos.

3 PROPOSTA

Neste trabalho será apresentada uma proposta de um modelo de algoritmo genético a fim de buscar uma solução para um problema de difícil tratamento sem a intervenção humana: alocação de professores e salas, em disciplinas de cursos da UFSC (Universidade Federal de Santa Catarina).

Na UFSC existem dois processos distintos e independentes, um para cada alocação citada: alocação de professores para todas as turmas de cada disciplina e alocação destas turmas em uma ou mais salas ou laboratórios - tendo em vista que as turmas, em geral, são ofertadas em dois ou três dias, cada dia com seu respectivo horário.

Uma turma é uma oferta de determinada disciplina, em determinados horários (cumprindo a carga horária da disciplina), ministrada por um ou mais professores em uma determinada sala/laboratório.

Os horários em que cada turma pode ser alocada devem pertencer ao seguinte grupo: 07:30, 08:20, 9:10, 10:10, 11:00, 13:30, 14:20, 15:10, 16:20, 17:10, 18:30, 19:20, 20:20 e 21:10, sendo que cursos de período noturno oferecem disciplinas apenas nos últimos 4 horários citados (a partir das 18:30).

A identificação de uma turma é realizada através da combinação entre o código da disciplina e o código da turma - composto pelo identificador da fase e o identificador do curso -, ex.: disciplina INE5660, turma 09238A, sendo que 09 identifica que a disciplina é da 9ª fase do curso do código 238, e o último caractere é opcional (A), sendo utilizado apenas quando há mais de uma turma para a mesma disciplina. As disciplinas podem ser ofertadas em mais de uma turma, caso julgue-se necessário.

Cada professor está alocado em um determinado departamento dentro da Universidade e possui disponibilidades de horários para diferentes funções: ensino, pesquisa e extensão. Cada professor possui conhecimento específico em uma ou mais áreas de ensino, por exemplo: Banco de Dados, Fundamentos de Programação, Teoria da Computação e Linguagens. Cada disciplina está contida em uma área de ensino, cujo professor deve conhecê-la.

3.1 REQUISITOS DO PROBLEMA

Para o problema descrito, identificaram-se dois tipos de requisitos diferentes: principais e desejáveis. Os requisitos principais possuem maior peso na avaliação de cada indivíduo (solução) e são conceitos fundamentais para uma alocação satisfatória. Já os requisitos desejáveis são preferências - nelas podem ser incluídas diferentes desejos do usuário do algoritmo, de acordo com acordos e protocolos da organização -, estes tendo um peso menor na avaliação de um indivíduo.

A tabela abaixo identifica e descreve os requisitos principais.

Tabela 2 – Tabela de Requisitos Principais

ID	Descrição do requisito
RP.1	O algoritmo deve alocar turmas de todas as disciplinas obrigatórias do curso de Sistemas de Informação, que sejam do departamento INE. Neste caso não são alocadas as turmas de disciplinas de diferentes departamentos (CAD, EGC, etc), assim como de disciplinas optativas e de projetos (Introdução a Projetos, Projetos I e Projetos II)
RP.2	As turmas devem ser alocadas de acordo com o período do curso: turmas de cursos noturnos são alocadas apenas nos últimos 4 horários possíveis (a partir de 18:30)
RP.3	As disciplinas devem ser ministradas por professores da mesma área de ensino
RP.4	Não deve haver choques de horário de turmas de disciplinas de uma mesma fase de um mesmo curso
RP.5	O professor é também um recurso, logo, não deve haver choques de horário de aulas ministradas por um mesmo professor
RP.6	Turma alocada na quantidade de horários necessária (horas/aula da disciplina)
RP.7	O mesmo professor deve ser alocado para os diferentes horários de uma mesma turma

Enquanto isso, são identificados e descritos os requisitos desejáveis do problema na tabela abaixo, a qual também indica se o requisito é atendido na implementação do algoritmo desenvolvido neste trabalho.

Tabela 3 – Tabela de Requisitos Desejáveis

ID	Descrição do requisito	Atendido?
RD.1	É preferível que as aulas de uma mesma turma sejam agrupadas (2, 3, 4 aulas seguidas)	Sim
RD.2	O algoritmo deve permitir que o usuário dê os horários das turmas já definidos como entrada. Neste caso, o algoritmo não altera estes horários e aloca apenas os professores.	Sim
RD.3	Os professores que já ministraram a disciplina recebem prioridade de alocação	Não
RD.4	Os horários de aulas de um professor devem ser "densos", visando evitar poucas aulas dispersas em todos os dias da semana	Não
RD.5	Uma mesma turma pode ser divida por dois ou mais professores	Não

3.2 ALGORITMO GENÉTICO PARA A ALOCAÇÃO DE TURMAS E PROFESSORES

Nesta seção serão identificados os detalhes de implementação do algoritmo genético desenvolvido neste trabalho, com o objetivo de gerar soluções satisfatórias para o problema de alocação de turmas e professores dentro da grade horária da UFSC.

3.2.1 Detalhes da Implementação

O algoritmo foi desenvolvido na linguagem Java, sem a utilização de nenhuma biblioteca ou *framework* específico para o desenvolvimento de AGs.

A estrutura básica da camada de modelo do algoritmo é representada pela figura abaixo:

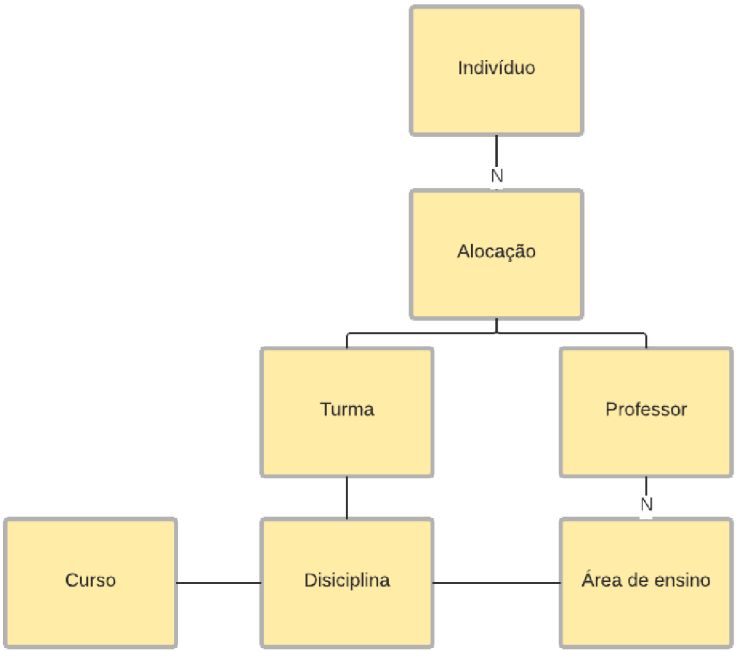

Figura 7 Modelo do algoritmo.

Cada objeto da classe Curso possui os atributos: codigo (String); nome (String); periodo (Enum com dois valores possíveis: INTEGRAL ou NOTURNO); fases (Integer).

Objetos da classe AreaEnsino possuem os atributos: codigo (String); nome (String).

Enquanto isso, a classe Disciplina é definida por: codigo (String); nome (String); curso (Curso); areaEnsino (AreaEnsino); fase (Integer); horasAula (Integer).

Um objeto do tipo Turma possui os atributos: codigo (String); disciplina (Disciplina).

E um objeto do tipo Professor possui os atributos: codigo (String); nome (String); areasEnsino (vetor de AreaEnsino).

Dado isso, cada objeto da classe Individuo possui um vetor de N objetos da classe Alocacao, sendo N o parâmetro de entrada "tamanho da população"que será apresentado na próxima seção.

Por fim, a classe Alocacao possui 4 atributos:

1. diaSemana: discretização do dia da semana da alocação, podendo receber valores inteiros entre 0 (domingo) e 6 (sábado).
2. horario: discretização do horário dentro do dia, podendo receber valores inteiros entre 0 (07:30-08:20) e 13 (21:20-22:00).
3. turma: objeto da classe Turma, que representa a turma alocada.
4. professor: objeto da classe Professor, que representa o professor alocado.

3.2.2 Entrada

O algoritmo para alocação de turmas e professores recebe dois tipos de parâmetros de entrada: parâmetros de configuração do AG, e dados em arquivos no formato CSV.

Os parâmetros de configuração do AG são identificados a seguir:

- Tamanho da população
- Tipo de seleção (torneio ou truncamento)
- Probabilidade de crossover
- Probabilidade de mutação
- Taxa de mutação
- Elitismo (quantidade)
- Máximo de gerações

Os arquivos CSV são estruturados da seguinte forma:

Tabela 4 – Cursos

Atributo	Codificação	Exemplo
Código do curso	Int	238
Nome	String	Sistemas de Informação
Período	Char	‘i’ = integral ‘n’ = noturno

Tabela 5 – Áreas de ensino

Atributo	Codificação	Exemplo
Código da área de ensino	Int	7
Nome da área de ensino	String	Introdução à Computação (ICC)

Tabela 6 – Professores

Atributo	Codificação	Exemplo
Código do professor	Int	5
Nome do professor	String	Elder Rizzon Santos
Áreas de ensino	Int[]	5,7,12

Tabela 7 – Disciplinas

Atributo	Codificação	Exemplo
Fase	Int	5
Código da disciplina	String (alfanumérico)	INE5601
Nome da disciplina	String	Fundamentos Matemáticos da Informática
Código do curso	Int	238
Código da área de ensino	Int	7
Horas/aula (por semana)	Int	4

Tabela 8 – Turmas

Atributo	Codificação	Exemplo
Disciplina + código da turma	Disciplina | "-" | Fase | Curso | (Turma)	INE5601-01238A
Alocações (opcional)	Dia | "." | Horário | ("," | Dia | "." | Horário)*	2.18:30,2.19:10, 4.20:20,4.21:10

3.2.3 Indivíduo

No modelo implementado, o indivíduo é representado pela classe Individuo, que é composta por um conjunto (array) de objetos da classe Alocacao. A classe Alocacao representa um gene, e é composta pelos atributos representados na tabela abaixo:

Tabela 9 – Alocacao

Atributo	Codificação	Exemplo
Dia da semana	Int	0 = Domingo 1 = Segunda-feira ... 6 = Sábado
Horário	Int	0 = 7:30 1 = 8:20 ... 13 = 21:20
Turma	Turma	(descrito posteriormente)
Professor	Professor	(descrito posteriormente)

As demais classes são representadas pelas tabelas abaixo:

Tabela 10 – Turma

Atributo	Codificação	Exemplo
Código	String	01238A
Disciplina	Disciplina	(descrito posteriormente)

Tabela 11 – Disciplina

Atributo	Codificação	Exemplo
Código	String	INE5601
Nome	String	Fundamentos Matemáticos da Informática
Curso	Curso	(descrito posteriormente)
Área de ensino	AreaEnsino	(descrito posteriormente)
Fase	Int	5
Horas/aula (por semana)	Int	4

Tabela 12 – Curso

Atributo	Codificação	Exemplo
Código	String	INE5601
Código	String	218
Período do curso	PeriodoCurso Enum	INTEGRAL ou NOTURNO
Quantidade de fases	Int	9

Tabela 13 – AreaEnsino

Atributo	Codificação	Exemplo
Código	String	INE5601
Código	String	7
Nome	String	Introdução à Computação (ICC)

Tabela 14 – Professor

Atributo	Codificação	Exemplo
Código	String	5
Nome	String	Elder Rizzon Santos
Áreas de ensino	AreaEnsino[]	(descrito acima)

3.2.4 População

A População do AG é representada pela classe Populacao, que é composta por um array de objetos da classe Individuo e um atributo do tipo Int que identifica o tamanho da população. A classe pode ser instanciada de forma a criar uma lista vazia de indivíduos, para serem preenchidos posteriormente, ou pode ser instanciada de forma a gerar aleatoriamente indivíduos até completar o array. Além disso, possui um método para ordenação da população, de acordo com o fitness dos indivíduos (decrescente).

3.2.5 Geração de indivíduos

A partir da lista de turmas e, opcionalmente, os seus horários, o algoritmo guarda uma lista base de objetos da classe Alocacao, contendo um elemento para cada alocação necessária de cada turma (ex.: se foi informada uma turma de uma disciplina de 4 horas/aula e outra turma de uma disciplina de 6 horas/aula, serão instanciados 10 objetos de alocação), com os atributos dia e horário preenchidos caso a entrada tenha informado estes atributos. O preenchimento aleatório de um indivíduo consiste na inserção aleatória dos seus atributos:

- Dia (quando não informado)
- Horário (quando não informado, sendo que gera apenas horários de acordo com o período do curso da disciplina desta turma, atendendo assim o requisito RP.2)
- Professor (qualquer professor cuja sua lista de áreas de ensino contenha a área de ensino da disciplina, atendendo aqui o requisito RP.3)

3.2.6 Métodos de seleção

Atualmente estão implementados dois métodos clássicos de seleção: Torneio e Truncamento.

O método de torneio é realizado da seguinte forma: são selecionados 3 indivíduos aleatoriamente da população anterior, destes indivíduos são selecionados os 2 com maior valor de aptidão.

Já o método de truncamento seleciona 2 pais a partir de outro processo: a partir de uma taxa de truncamento T (informada como parâmetro do algoritmo) são "cortados" os T indivíduos com maior aptidão da população anterior, e a partir deste subconjunto são selecionados aleatoriamente 2 indivíduos.

Ambos os métodos retornam 2 indivíduos para serem usados como pais na entrada dos métodos de crossover e mutação, gerando 2 filhos para a nova população da geração seguinte.

3.2.7 Crossover

O crossover é realizado da seguinte forma: após a escolha de dois pais para realizarem o cruzamento, os seus genes são recombinados com o objetivo de gerar dois novos filhos. Para isso, os seus genes são iterados simultaneamente (array de Alocacao), sendo que a cada iteração, para cada atributo é sorteado um valor booleano, caso resulte em true o valor do atributo do P2 (pai 2) será atribuído ao atributo do F1 (filho 1) e o valor do mesmo atributo do P1 será atribuído ao atributo do F2. Caso resulte em false é feito a cópia do atributo de P1 para F1 e P2 para F2.

Ex.:

```
P1.horario = 5
P2.horario = 7
se trocar()
        então F1.horario = P2.horario
              F2.horario = P1.horario
senão
              F1.horario = P1.horario
              F2.horario = P2.horario
```

3.2.8 Mutação

A mutação de um indivíduo é realizada, caso o indivíduo seja escolhido (parâmetro de percentual de mutação), da seguinte forma: sorteia-se quais dos 4 atributos serão mutados (a quantidade varia entre 1 e 4, de acordo com o parâmetro de taxa de mutação), itera-se o array de Alocacao do indivíduo, atribuindo valores aleatórios para cada atributo escolhido inicialmente. Ex.: taxa de mutação = 3 e foram sorteados os atributos "turma", "horario" e "professor", e então são gerados valores aleatórios para estes atributos para cada Alocacao do indivíduo, mantendo os valores antigos do atributo "diaSemana".

3.2.9 Elitismo

O elitismo está presente neste algoritmo através de um parâmetro de entrada que identifica a quantidade de indivíduos mais aptos que serão "replicados" da população anterior para a nova. Ou seja, no início da geração de uma nova população, os E (parâmetro de entrada) primeiros indivíduos são copiados dos melhores indivíduos da população anterior.

3.2.10 Aptidão

A aptidão do indivíduo (ou fitness) é calculada pela função da classe IndividuoUteis, que recebe um Individuo como parâmetro, e retorna um objeto da classe Aptidao (utilizada apenas para guardar separadamente os scores de cada critério de aptidão separadamente). O valor final da aptidão do indivíduo é calculado através da soma dos resultados de 4 avaliações: avaliação de turmas alocadas (que atende o requisito RP.6); avaliação de choques de horários (que atende os requisitos RP.4 e RP.5); avaliação de professores por turma (que atende o requisito RP.7); avaliação de agrupamento de aulas (que atende o requisito RD.1). Estas avaliações são descritas a seguir.

3.2.10.1 Avaliação de turmas alocadas

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir do somatório de horas/aula de cada turma), iniciando com o

seu valor máximo e sendo decrementado a cada diferença de horários alocados por turma em relação a sua necessidade (horas/aulas da disciplina da turma). Ex.: turma da disciplina INE5601 (4 horas/aula) foi alocada em 5 horários na semana = -1; turma da mesma disciplina foi alocada em 2 horários na semana = -2.

3.2.10.2 Avaliação de choques de horários

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir da soma de fases que possuem disciplinas com a quantidade de professores disponíveis, multiplicando este valor pela quantidade de horários disponíveis para alocação (sendo 14 no contexto da UFSC), multiplicando também pela quantidade de dias na semana), iniciando com seu valor máximo e podendo ser decrementado por dois fatores: o horário de uma turma já está alocado por outra turma, de outra disciplina, da mesma fase do mesmo curso; o horário de uma turma de um professor já está alocado por outra turma, com o mesmo professor.

3.2.10.3 Avaliação de professores por turma

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir do somatório de horas/aula de cada turma), iniciando com o seu valor máximo e podendo ser decrementado a cada divergência entre o professor alocado para uma turma, e o professor alocado para a mesma turma em outro horário.

3.2.10.4 Avaliação de agrupamento de aulas

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir do somatório de horas/aula de cada turma), iniciando com o seu valor máximo e sendo decrementado caso uma alocação de uma turma esteja isolada não contenha outra alocação da mesma turma no horário anterior ou seguinte.

3.2.10.5 Normalização e pesos

Todos esses valores de cada avaliação são normalizados antes de serem somados, visando equilibrar o peso de cada fator para o valor

final da aptidão do indivíduo. Esta normalização é feita através de uma função que recebe: (i) o valor, (ii) o valor máximo possível, (iii) peso da avaliação. Esta função divide o valor pelo seu valor máximo possível e então multiplica o resultado desta divisão pelo peso da avaliação, a partir disso é obtido um valor proporcional de cada avaliação, que então é convertido para um número inteiro para facilitar o tratamento do fitness.

Os pesos são definidos a partir da importância do requisito que a avaliação atende. Por padrão, as avaliações de requisitos principais (RPs) possuem peso 10.000, enquanto que a avaliação de requisito desejado (RD) possui peso 200. Diferentes valores de pesos de RDs foram testados a fim de encontrar resultados mais satisfatórios e de forma mais rápida, conforme descrito no próximo capítulo deste documento.

3.2.11 Critérios de parada

Após o início da execução do algoritmo genético implementado, existem dois critérios analisados ao final de cada nova geração de indivíduos, para que então o algoritmo seja interrompido e apresente uma saída (solução):

- Número máximo de gerações, o qual é definido internamente, variando de acordo com o tamanho da população informado.
- O melhor indivíduo da nova geração atingiu os valores máximos de aptidão nas avaliações de Requisitos Principais, descritos na seção 3.2.10 deste documento.

4 ANÁLISE DO ALGORITMO

Foram realizados diversos testes para analisar a qualidade das soluções produzidas pelo algoritmo proposto, com diversas variações de parâmetros de entrada, buscando a melhor configuração para atingir resultados satisfatórios.

Neste capítulo, serão descritas as métricas utilizadas na avaliação dos resultados obtido, assim como os alguns dos principais testes em termos de variação de configuração e qualidade de soluções.

Nas seções 4.3 e 4.4 são descritos os principais testes realizados, representados por tabelas onde cada linha é uma execução do algoritmo, e suas configurações e resultados são estruturados através das seguintes colunas:

- Configuração AG:
  - Pop: tamanho da população
  - Sel: tipo de seleção utilizado, Torneio ou Truncamento, sendo que no segundo é informada também a taxa de truncamento (limiar)
  - PbC: probabilidade de realização do crossover dos pais selecionados
  - PbM: probabilidade de realização da mutação nos filhos gerados
  - TxM: taxa de mutação, ou seja, quantos atributos da alocação serão mutados
  - Elit: taxa de elitismo, ou seja, a quantidade de indivíduos que são replicados de uma geração à próxima
- Resultado:
  - Tempo: tempo de execução do algoritmo, até ser interrompido por algum dos critérios de parada
  - Crit. Parada: critério de parada que foi atingido, "Aptidão máxima"quando o algoritmo gerou um indivíduo cujas avaliações principais (seções 3.2.10.1, 3.2.10.2 e 3.2.10.3) obtiveram nota máxima ou "Máx gerações"quando o algoritmo atingiu o número máximo de gerações definidas
  - Turmas Al.: "nota / peso"da avaliação de turmas alocadas (seção 3.2.10.1)

  - Choq.: "nota / peso"da avaliação de choques de horários (seção 3.2.10.2)
  - Prof T.: "nota / peso"da avaliação de professores por turma (seção 3.2.10.3)
  - Agrup.: "nota / peso"da avaliação de aulas agrupadas (seção 3.2.10.4)

Dois exemplos de soluções geradas a partir dos testes podem ser observados no capítulo de anexos.

4.1 MÉTRICAS DE AVALIAÇÃO

Para a avaliação dos resultados obtidos em cada teste foram consideradas as seguintes métricas:

- Solução com a melhor aptidão encontrada após atingir o critério de parada, com a sua aptidão composta por cada valor de aptidão parcial, descrito na seção 3.2.10 deste documento, em relação ao valor máximo de cada avaliação (pesos).
  - Turmas alocadas
  - Choque de horários
  - Professores por turma
  - Agrupamento de aulas
- Critério de parada: se foi atingido o número máximo de gerações ou foi atingiu a aptidão satisfatória dos Requisitos Principais (aptidão máxima)
- Tempo de execução

4.2 DADOS DE ENTRADA

Para a entrada dos diversos testes, foram utilizados os seguintes dados, na estrutura de arquivos CSV:

- Áreas de Ensino: áreas do departamento INE, no total são 14 Áreas de ensino, como: Banco de Dados, Computação Numérica, Fundamentos de Programação, entre outras.

- Cursos: Sistemas de Informação, seu código e período (noturno).
- Disciplinas: todas as disciplinas do departamento INE, obrigatórios no currículo do curso de Sistemas de Informação, assim como os seus atributos (fase, código, nome, código do curso, área de ensino e horas/aula)
- Professores: foram informoados 38 professores do INE - sendo o suficiente para alocar todas as turmas - contendo o nome e as suas respectivas áreas de ensino.

Os dados de entrada para as turmas variam de acordo com os tipos de testes que foram realizados e serão descritos na próxima seção.

4.3 TESTES SEM HORÁRIOS PRÉ DEFINIDOS

Os testes realizados para o cenário em que o usuário não conhece os horários das turmas - ou prefira que o próprio algoritmo encontre novas possibilidades - receberam como entrada apenas os códigos das turmas (além dos dados de entrada descritos na seção 4.2).

4.3.1 Testes com todas as turmas

A partir dos dados de entrada citados, assim como os dados de todas as turmas das disciplinas também informadas - do departamento INE, obrigatórias do curso de Sistemas de Informação da UFSC -, foram realizados os seguintes testes para obter soluções de alocação:

Configuração AG						Resultado					
Pop	Sel	PbC	PbM	TxM	Elit	Tempo	Crit. Parada	Turmas Al.	Choq.	Prof T.	Agrup.
300	Torneio	80,00%	15,00%	4	8	07:23:46	Máx gerações: 250.000	10.000 / 10.000	9.803 / 10.000	10.000 / 10.000	1.939 / 2.000
300	Torneio	85,00%	14,00%	4	8	08:18:05	Máx gerações: 250.000	10.000 / 10.000	9.939 / 10.000	10.000 / 10.000	2.000 / 2.000
500	Torneio	85,00%	16,00%	2	20	06:17:23	Máx gerações: 100.000	10.000 / 10.000	9.950 / 10.000	10.000 / 10.000	500 / 500
500	Truncamento (150)	85,00%	16,00%	4	18	04:55:51	Máx gerações: 100.000	10.000 / 10.000	9.900 / 10.000	8.734 / 10.000	445 / 500
500	Truncamento (250)	81,00%	18,00%	2	8	04:53:23	Máx gerações: 100.000	10.000 / 10.000	9.917 / 10.000	9.337 / 10.000	481 / 500

Figura 8 Testes com todas as turmas, sem horários pré definidos

Nestes testes, não foi possível atingir aptidão máxima nas avaliações dos requisitos principais. Porém chegou-se muito próximo disso com a terceira configuração, onde foi informada uma maior população, utilizando o método de seleção de torneio, e após o número máximo de gerações ser atingido a solução encontrada teve aptidão máxima em todos os critérios, exceto o de choque de horários (9.950/10.000).

Levando em consideração estes principais testes, assim como todos os outros executados ao longo do trabalho, foi possível identificar maior convergência na utilização do método de seleção por torneio, assim como outras faixas de valores para cada parâmetro.

4.3.2 Testes com apenas uma turma por disciplina

Os testes com apenas uma turma por disciplina possuem as mesmas entradas dos testes da seção anterior, porém removendo as turmas de código com finais "B" e "C".

Foram realizados os seguintes testes:

Configuração AG						Resultado					
Pop	Sel	PbC	PbM	TxM	Elit	Tempo	Crit. Parada	Turmas Al.	Choq.	Prof T.	Agrup.
300	Torneio	85,00%	15,00%	4	12	00:31:43	Aptidão máxima	10.000 / 10.000	10.000 / 10.000	10.000 / 10.000	500 / 500
500	Torneio	85,00%	15,00%	4	20	04:14:09	Aptidão máxima	10.000 / 10.000	10.000 / 10.000	10.000 / 10.000	2.000 / 2.000
100	Truncamento(85)	72,00%	28,00%	2	4	02:32:50	Máx gerações: 300.000	7.288 / 10.000	9.821 / 10.000	5.677 / 10.000	338 / 500
100	Truncamento(75)	75,00%	25,00%	2	2	02:31:59	Máx gerações: 300.000	10.000 / 10.000	9.810 / 10.000	6.271 / 10.000	322 / 500

Figura 9 Testes com uma turma por disciplina, sem horários pré definidos

Neste caso, como o número de combinações possíveis para gerar soluções com aptidão máxima era maior em relação aos testes da subseção anterior, onde haviam mais turmas de uma mesma disciplina, conseguiu-se obter soluções de maior qualidade, também através do método de seleção de pais por torneio.

4.4 TESTES COM HORÁRIOS PRÉ DEFINIDOS

Para simular o cenário em que o usuário deseja apenas a alocação dos professores para cada turma, pré definindo os horários em que cada turma será oferecida, os testes realizados receberam como entrada os códigos das turmas e uma lista de horários (de acordo com o padrão descrito na seção 3.2.2 deste documento).

4.4.1 Testes com todas as turmas

A partir dos dados de entrada citados, assim como os dados de todas as turmas das disciplinas informadas, e seus respectivos horários pré definidos, foram realizados os seguintes testes para obter soluções de alocação de professores:

Configuração AG						Resultado					
Pop	Sel	PbC	PbM	TxM	Elit	Tempo	Crit. Parada	Turmas Al.	Choq.	Prof T.	Agrup.
100	Torneio	85,00%	21,00%	4	8	03:24:35	Máx gerações: 300.000	10.000 / 10.000	9.935 / 10.000	10.000 / 10.000	2.000 / 2.000
150	Torneio	85,00%	17,00%	4	6	04:07:28	Máx gerações: 200.000	10.000 / 10.000	9.950 / 10.000	10.000 / 10.000	2.000 / 2.000
100	Torneio	84,00%	18,00%	4	6	02:34:10	Máx gerações: 300.000	10.000 / 10.000	9.985 / 10.000	10.000 / 10.000	2.000 / 2.000
500	Truncamento (350)	85,00%	21,00%	4	22	04:17:18	Máx gerações: 100.000	10.000 / 10.000	9.921 / 10.000	9.277 / 10.000	500 / 500
500	Truncamento (150)	85,00%	16,00%	4	18	04:17:27	Máx gerações: 100.000	10.000 / 10.000	9.957 / 10.000	8.915 / 10.000	500 / 500

Figura 10 Testes com todas as turmas, com horários pré definidos

4.4.2 Testes com apenas uma turma por disciplina

Os testes com apenas uma turma por disciplina possuem as mesmas entradas dos testes do item anterior (4.4.1), porém removendo as turmas de código com finais “B” e “C”.

Foram realizados os seguintes testes:

Configuração AG						Resultado					
Pop	Sel	PbC	PbM	TxM	Elit	Tempo	Crit. Parada	Turmas Al.	Choq.	Prof T.	Agrup.
100	Torneio	84,00%	20,00%	4	6	00:09:14	Aptidão máxima	10.000 / 10.000	10.000 / 10.000	10.000 / 10.000	2.000 / 2.000
100	Torneio	84,00%	18,00%	4	6	02:34:10	Máx gerações: 300.000	10.000 / 10.000	9.985 / 10.000	10.000 / 10.000	2.000 / 2.000
500	Truncamento (350)	82,00%	23,00%	4	20	03:18:12	Máx gerações: 100.000	10.000 / 10.000	9.992 / 10.000	10.000 / 10.000	500 / 500
300	Truncamento (100)	87,00%	25,00%	4	12	03:46:49	Máx gerações: 200.000	10.000 / 10.000	9.975 / 10.000	9.322 / 10.000	500 / 500

Figura 11 Testes com todas as turmas, com horários pré definidos

5 CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo irá apresentar as conclusões obtidas ao término do trabalho, assim como indicações de trabalhos futuros que podem ser realizados a partir deste.

5.1 CONCLUSÕES

Este trabalho apresentou a teoria a respeito de algoritmos genéticos, *timetabling* e outros aspectos importantes para o embasamento teórico necessário na construção do algoritmo genético proposto e implementado como principal objetivo deste trabalho.

A utilização da abordagem de algoritmos genéticos possibilitou a solução do problema identificado, com características e restrições complexas, sem a utilização de técnicas especificas do domínio abordado. Ao invés disso, o algoritmo demandou apenas de meios para a avaliação da qualidade das soluções geradas.

A partir disso, podemos concluir que a utilização de algoritmos genéticos oferece uma alternativa viável para solucionar problemas de *timetabling*, como é o caso da geração de grade de horárias da UFSC, onde algumas regras e preferências internas dificultam a implementação de métodos determinísticos para a geração de soluções satisfatórias.

O uso de algoritmos genéticos em domínios complexos também foi abordado nos trabalhos de (FRANCO, 2014) e (SUMBANA et al., 2012b), tomados como principais referências neste trabalho.

Embora os resultados obtidos neste trabalho possam confirmar a viabilidade de aplicação da técnica, a implementação apresentada, no seu estado atual, requer importantes melhorias para a sua utilização em cenários reais. Entre essas melhorias, podem ser citados os requisitos desejáveis (DP) não atendido pelo algoritmo, conforme descrito na seção 3.1.

Espera-se que este trabalho auxilie novas iniciativas na área de geração de grades horárias para escolas e universidades.

5.2 TRABALHOS FUTUROS

Nesta seção, são reunidas sugestões de melhorias e continuidade na busca por soluções para a automatização da geração de grades de

horários das turmas UFSC.

- Implementação dos requisitos desejáveis que não foram considerados no algoritmo, e que estão descritos na seção 3.1.
- Implementação de outro algoritmo genético para a alocação de salas ou laboratórios para as turmas, podendo trabalhar em sincronia com o algoritmo desenvolvido neste trabalho.
- Melhorar a estrutura do algoritmo, de forma mais genérica, para permitir a aplicação em diferentes domínios.
- Avaliação da utilização de outros operadores de seleção, com o objetivo de aumentar ou diminuir a pressão seletiva, ou ainda para contribuir na distribuição das chances por faixas de valores de aptidão.
- Estudo de diferentes bibliotecas que possam facilitar a implementação do algoritmo, e de diferentes técnicas para melhorar o seu resultado. Algumas das bibliotecas encontradas durante este trabalho foram: JGAP, Jenetics, GACS e PyBrain.
- Avaliação da utilização de diferentes critérios de parada para o algoritmo genético, como estagnação do valor de aptidão, tempo de execução, entre outros.
- Avaliação dos benefícios do uso de técnicas e estratégias que auxiliem na fuga de eventuais mínimos locais atingidos pelo algoritmo.
- Identificar outros aspectos que possam ser introduzidos como requisitos desejáveis para este algoritmo, de forma a aumentar a sua aplicabilidade em problemas reais.

REFERÊNCIAS

BLICKLE, T. Theory of evolutionary algorithms and application to system synthesis. 1996. <http://www.handshake.de/user/blickle/publications/diss.pdf>. Acessado em 21/11/2016.

BURKE, E. K.; CAUSMAECKER, P. D.; BERGHE, G. V. The state of the art of nurse rostering. *Journal of Scheduling*, v. 7, p. 441–499, 2004.

CAMPOS, A. J. R. de; RIBEIRO, G. O. Geração de grades de horário para escalas de trabalho utilizando algoritmos genéticos. 2012.

CARUANA, R. A.; SCHAFFER, J. D. Representation and hidden bias: Gry vs. binary coding for genetic algorithms. *Proceedings INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, v. 5, p. 441–499, 1988.

FRANCO, R. A. P. Verificação funcional de sistemas digitais utilizando algoritmos genéticos na geração de dados aplicada à metodologia verisc. 2014. <https://repositorio.bc.ufg.br/tede/handle/tede/5028>. Acessado em 21/11/2016.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. [S.l.: s.n.], 1989.

HOLLAND, J. H. *Adaptation in natural and artificial systems*. [S.l.]: The University of Michigan Press, Ann Arbor, MI, 1975.

JAIN, A. K.; DUIN, R. P. W.; MAO, J. Statistical pattern recognition: A review. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 22*, 2000.

KITANO, H. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D, Amsterdam*, v. 75, p. 225–238, 1994.

MEYER, T. P.; PACKARD, N. H. Local forecasting of high-dimensional chaotic dynamics. *CASDAGLI, N.; EUBANK, S. (Eds.), Nonlinear Modeling and Forecasting*, 1992.

MITCHELL, M. *An Introduction to Genetic Algorithms, Complex Adaptive Systems*. [S.l.: s.n.], 1996.

MOORE, A. W. Information gain. 2012.

MÜLLER, T.

Constraint-based Timetabling. — Charles University in Prague Faculty of Mathematics and Physics, 2005.

PACHECO, M. A. C. Algoritmos genéticos: Princípios e aplicações. 1999.

PAPPA, G. L.; FREITAS, A. A.; KAESTNER, C. A. A. Attribute selection with a multi-objective genetic algorithm. In: *In Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*. [S.l.: s.n.], 2002.

SILVA, K. R. G. *Uma Metodologia de Verificação Funcional para Circuitos Digitais*. Tese (Doutorado) — Universidade Federal de Campina Grande, 2007.

SUMBANA, M. I. M. et al.

Automatic vandalism detection in wikipedia with active associative classification, 2012.

SUMBANA, M. I. M. et al. Seleção de atributos utilizando algoritmos genéticos para detecção do vandalismo na wikipedia. *Simpósio Brasileiro de Bancos de Dados*, 2012.

WREN, A. Scheduling, timetabling and rostering – a special relationship? *Springer, Berlin, Heidelberg*, p. 46–75, 2006.

APÊNDICE A – Artigo

Algoritmos Genéticos como uma abordagem para a alocação de grade de horários da UFSC

Eduardo H. H. Kalfels

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

eduardo.kalfels@grad.ufsc.br

Abstract. *This article discusses the development of a genetic algorithm for UFSC timetables generation. Additionally, a brief review of the area literature is introduced, where concepts related to genetic algorithms and timetabling problems are described. At the end, the results obtained in the test involving real data are exposed.*

Resumo. *Este artigo aborda o desenvolvimento de um algoritmo genético para geração de grades de horário da UFSC. Adicionalmente, é realizada uma breve revisão da literatura referente a área, onde são apresentados conceitos relacionados a algoritmos genéticos e problemas de timetabling. Ao final, são expostos os resultados obtidos no teste envolvendo dados reais.*

1. Introdução

Além de escolas e universidades, empresas e outras organizações enfrentam problemas para gerar grades de horários que exigem que diversas regras sejam respeitadas, consideram preferências das pessoas envolvidas, disponibilidade de horários, e diversos outros fatores que tornam a tarefa bastante complexa.

Os resultados gerados através de abordagens manuais ou algoritmos determinísticos são ruins do ponto de vista econômico, assim como do ponto de vista das pessoas envolvidas, as quais são fortemente afetadas pelas soluções geradas [BURKE et al. 2004].

Dado este problema, uma abordagem alternativa dá-se através dos Algoritmos Genéticos, que estão inseridos no campo de estudo de Inteligência Artificial (IA). Uma popular e inicial definição de IA foi introduzida por John McCarthy na famosa conferência de Dartmouth em 1956: ”fazer a máquina comportar-se de tal forma que seja chamada inteligente caso fosse este o comportamento de um ser humano.”Porém, ao apenas imitar o comportamento do ser humano, esta definição parece ignorar a existência da IA forte: proposta em que busca-se fazer com que a máquina raciocine de forma autônoma, igual ou melhor que a mente humana.

Para diversos problemas, em diferentes ambientes, podemos utilizar algoritmos genéticos para buscar a melhor solução possível. Algoritmo Genético (AG), segundo [Mitchell 1996], é uma técnica de busca utilizada na ciência da computação para encontrar soluções aproximadas em problemas de otimização e busca, fundamentado principalmente pelo americano John Henry Holland. Algoritmos genéticos são uma classe

particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva como hereditariedade, mutação, seleção natural e recombinação (ou crossing over) [Goldberg 1989].

Existem diversas situações possíveis em que pode-se fazer uso de um algoritmo genético, incluindo: otimização de rotas, alocação de recursos em uma grade de horários, maximização da eficiência de uma termoelétrica.

Este artigo mostra a elaboração de um algoritmo genético para lidar com a otimização na alocação de turmas e professores na grade de horários das turmas das disciplinas (do departamento INE) obrigatórias do curso de Sistemas de Informação da UFSC, a fim de encontrar uma solução satisfatória para o cenário.

2. Timetabling

A geração de grades de horários em cursos da UFSC, considerando diversas restrições e a disponibilidade de diferentes recursos, é um problema de *timetabling*, definido por [Wren 2006]. Esse autor define *timetabling* como a alocação, sujeita a restrições, de dados a objetos distribuídos no tempo e no espaço, de maneira a satisfazer da melhor possível um conjunto de objetivos desejáveis. Outra definição para *timetabling*, segundo [MÜLLER 2005], é o procedimento de alocar atividades no tempo e espaço de forma a atingir objetivos desejáveis assim como considerar restrições conhecidas de acordo com o contexto.

Estes objetivos podem ser: maximizar a eficiência no uso de recursos, considerar os desejos e preferências de recursos humanos (como médicos, professores, funcionários, que desejam trabalhar mais ou menos em determinados dias ou horários). Enquanto que restrições podem ser exemplificadas com: recursos que não podem ser compartilhados em um mesmo período de tempo, outros recursos que podem ou devem ser compartilhados por uma quantidade de atividades simultaneamente.

Abordagens sistemáticas com o objetivo de gerar timetables satisfatórios são vistas com grande importância por [BURKE et al. 2004]. Uma aplicação importante citada pelo autor é a área médica, cujos pacientes com necessidades de cuidados médicos devem ser atendidos com agilidade e eficiência, assim como as demandas gerenciais da organização. Um ponto muito importante na geração automatizada de alocações de horários e recursos é o impacto significativo no uso do tempo do(s) responsável(eis) pelo planejamento deste cenários. Além disso, as abordagens automatizadas são caracterizadas pela potencialização da melhoria e otimização do processo de *timetabling* e pela qualidade dos timetables gerados. As abordagens matemáticas ou heurísticas permitem - além da geração de diversas soluções com variadas características - a criação e análise de métricas quantitativas e qualitativas das soluções geradas, assim como a inclusão de preferências, conforme dito anteriormente, que atendam os desejos da organização ou dos próprios funcionários.

3. Algoritmos genéticos

A abordagem através de Algoritmos Genéticos é uma técnica de busca e otimização, altamente paralela, inspirada no princípio Darwiniano de seleção natural e reprodução genética. De acordo com a teoria de Darwin, o princípio do processo de seleção permite maior longevidade aos indivíduos mais aptos, ou seja, maior probabilidade de

reprodução. Estes indivíduos têm maior chance de perpetuar os seus códigos genéticos para as próximos gerações. Os códigos genéticos constituem a identidade de cada indivíduo e são representados nos cromossomos.

Os AGs são construídos imitando estes princípios, buscando a melhor solução para um determinado problema, por meio da evolução de conjuntos (populações) de soluções (indivíduos) codificados como cromossomos artificiais. Cromossomo é implementado como uma estrutura de dados para representar uma possível solução no espaço de busca do problema. Submete-se, então, estes cromossomos a um processo evolucionário, constituído pela avaliação, seleção, crossover e mutação. Ao longo da execução de vários ciclos deste processo a população resultante tende a conter indivíduos cada vez mais aptos, ou seja, soluções melhores.

Os componentes dos Algoritmos Genéticos são apresentados da seguinte forma:

1. Problema a ser otimizado
2. Representação das Soluções do Problema
3. Decodificação do cromossomo
4. Avaliação
5. Seleção
6. Operadores Genéticos
7. Inicialização da População
8. Parâmetros e Critérios de Parada

4. Algoritmo desenvolvido

Foi desenvolvido um modelo de algoritmo genético a fim de buscar uma solução para um problema de difícil tratamento sem a intervenção humana: alocação de professores e salas, em disciplinas de cursos da UFSC (Universidade Federal de Santa Catarina).

Uma turma é uma oferta de determinada disciplina, em determinados horários (cumprindo a carga horária da disciplina), ministrada por um ou mais professores em uma determinada sala/laboratório.

Os horários em que cada turma pode ser alocada devem pertencer ao seguinte grupo: 07:30, 08:20, 9:10, 10:10, 11:00, 13:30, 14:20, 15:10, 16:20, 17:10, 18:30, 19:20, 20:20 e 21:10, sendo que cursos de período noturno oferecem disciplinas apenas nos últimos 4 horários citados (a partir das 18:30).

A identificação de uma turma é realizada através da combinação entre o código da disciplina e o código da turma - composto pelo identificador da fase e o identificador do curso -, ex.: disciplina INE5660, turma 09238A, sendo que 09 identifica que a disciplina é da 9ª fase do curso do código 238, e o último caractere é opcional (A), sendo utilizado apenas quando há mais de uma turma para a mesma disciplina. As disciplinas podem ser ofertadas em mais de uma turma, caso julgue-se necessário.

Cada professor está alocado em um determinado departamento dentro da Universidade e possui disponibilidades de horários para diferentes funções: ensino, pesquisa e extensão. Cada professor possui conhecimento específico em uma ou mais áreas de ensino, por exemplo: Banco de Dados, Fundamentos de Programação, Teoria da Computação e Linguagens. Cada disciplina está contida em uma área de ensino, cujo professor deve conhecê-la.

4.1. Requisitos do problema

Para o problema descrito, identificaram-se dois tipos de requisitos diferentes: principais e desejáveis. Os requisitos principais possuem maior peso na avaliação de cada indivíduo (solução) e são conceitos fundamentais para uma alocação satisfatória. Já os requisitos desejáveis são preferências - nelas podem ser incluídas diferentes desejos do usuário do algoritmo, de acordo com acordos e protocolos da organização -, estes tendo um peso menor na avaliação de um indivíduo.

A tabela abaixo identifica e descreve os requisitos principais.

Tabela 1. Tabela de Requisitos Principais

ID	Descrição do requisito
RP.1	O algoritmo deve alocar turmas de todas as disciplinas obrigatórias do curso de Sistemas de Informação, que sejam do departamento INE. Neste caso não são alocadas as turmas de disciplinas de diferentes departamentos (CAD, EGC, etc), assim como de disciplinas optativas e de projetos (Introdução a Projetos, Projetos I e Projetos II)
RP.2	As turmas devem ser alocadas de acordo com o período do curso: turmas de cursos noturnos são alocadas apenas nos últimos 4 horários possíveis (a partir de 18:30)
RP.3	As disciplinas devem ser ministradas por professores da mesma área de ensino
RP.4	Não deve haver choques de horário de turmas de disciplinas de uma mesma fase de um mesmo curso
RP.5	O professor é também um recurso, logo, não deve haver choques de horário de aulas ministradas por um mesmo professor
RP.6	Turma alocada na quantidade de horários necessária (horas/aula da disciplina)
RP.7	O mesmo professor deve ser alocado para os diferentes horários de uma mesma turma

Enquanto isso, são identificados e descritos os requisitos desejáveis do problema na tabela abaixo, a qual também indica se o requisito é atendido na implementação do algoritmo desenvolvido neste trabalho.

Tabela 2. Tabela de Requisitos Desejáveis

ID	Descrição do requisito	Atendido?
RD.1	É preferível que as aulas de uma mesma turma sejam agrupadas (2, 3, 4 aulas seguidas)	Sim
RD.2	O algoritmo deve permitir que o usuário dê os horários das turmas já definidos como entrada. Neste caso, o algoritmo não altera estes horários e aloca apenas os professores.	Sim
RD.3	Os professores que já ministraram a disciplina recebem prioridade de alocação	Não
RD.4	Os horários de aulas de um professor devem ser "densos", visando evitar poucas aulas dispersas em todos os dias da semana	Não
RD.5	Uma mesma turma pode ser divida por dois ou mais professores	Não

4.2. Detalhes da Implementação

O algoritmo foi desenvolvido na linguagem Java, sem a utilização de nenhuma biblioteca ou *framework* específico para o desenvolvimento de AGs.

Cada objeto da classe Curso possui os atributos: codigo (String); nome (String); periodo (Enum com dois valores possíveis: INTEGRAL ou NOTURNO); fases (Integer).

Objetos da classe AreaEnsino possuem os atributos: codigo (String); nome (String).

Enquanto isso, a classe Disciplina é definida por: codigo (String); nome (String); curso (Curso); areaEnsino (AreaEnsino); fase (Integer); horasAula (Integer).

Um objeto do tipo Turma possui os atributos: codigo (String); disciplina (Disciplina).

E um objeto do tipo Professor possui os atributos: codigo (String); nome (String); areasEnsino (vetor de AreaEnsino).

Dado isso, cada objeto da classe Individuo possui um vetor de N objetos da classe Alocacao, sendo N o parâmetro de entrada "tamanho da população".

Por fim, a classe Alocacao possui 4 atributos:

1. diaSemana: discretização do dia da semana da alocação, podendo receber valores inteiros entre 0 (domingo) e 6 (sábado).
2. horario: discretização do horário dentro do dia, podendo receber valores inteiros entre 0 (07:30-08:20) e 13 (21:20-22:00).
3. turma: objeto da classe Turma, que representa a turma alocada.

4. professor: objeto da classe Professor, que representa o professor alocado.

4.3. Geração de indivíduos

A partir da lista de turmas e, opcionalmente, os seus horários, o algoritmo guarda uma lista base de objetos da classe Alocacao, contendo um elemento para cada alocação necessária de cada turma (ex.: se foi informada uma turma de uma disciplina de 4 horas/aula e outra turma de uma disciplina de 6 horas/aula, serão instanciados 10 objetos de alocação), com os atributos dia e horário preenchidos caso a entrada tenha informado estes atributos. O preenchimento aleatório de um indivíduo consiste na inserção aleatória dos seus atributos:

- Dia (quando não informado)
- Horário (quando não informado, sendo que gera apenas horários de acordo com o período do curso da disciplina desta turma, atendendo assim o requisito RP.2)
- Professor (qualquer professor cuja sua lista de áreas de ensino contenha a área de ensino da disciplina, atendendo aqui o requisito RP.3)

4.4. Métodos de seleção

Atualmente estão implementados dois métodos clássicos de seleção: Torneio e Truncamento.

O método de torneio é realizado da seguinte forma: são selecionados 3 indivíduos aleatoriamente da população anterior, destes indivíduos são selecionados os 2 com maior valor de aptidão.

Já o método de truncamento seleciona 2 pais a partir de outro processo: a partir de uma taxa de truncamento T (informada como parâmetro do algoritmo) são "cortados" os T indivíduos com maior aptidão da população anterior, e a partir deste subconjunto são selecionados aleatoriamente 2 indivíduos.

Ambos os métodos retornam 2 indivíduos para serem usados como pais na entrada dos métodos de crossover e mutação, gerando 2 filhos para a nova população da geração seguinte.

4.5. Crossover

O crossover é realizado da seguinte forma: após a escolha de dois pais para realizarem o cruzamento, os seus genes são recombinados com o objetivo de gerar dois novos filhos. Para isso, os seus genes são iterados simultaneamente (array de Alocacao), sendo que a cada iteração, para cada atributo é sorteado um valor booleano, caso resulte em true o valor do atributo do P2 (pai 2) será atribuído ao atributo do F1 (filho 1) e o valor do mesmo atributo do P1 será atribuído ao atributo do F2. Caso resulte em false é feito a cópia do atributo de P1 para F1 e P2 para F2.

4.6. Mutação

A mutação de um indivíduo é realizada, caso o indivíduo seja escolhido (parâmetro de percentual de mutação), da seguinte forma: sorteia-se quais dos 4 atributos serão mutados (a quantidade varia entre 1 e 4, de acordo com o parâmetro de taxa de mutação), itera-se o array de Alocacao do indivíduo, atribuindo valores aleatórios para cada atributo escolhido inicialmente. Ex.: taxa de mutação = 3 e foram sorteados os atributos "turma", "horario" e "professor", e então são gerados valores aleatórios para estes atributos para cada Alocacao do indivíduo, mantendo os valores antigos do atributo "diaSemana".

4.7. Elitismo

O elitismo está presente neste algoritmo através de um parâmetro de entrada que identifica a quantidade de indivíduos mais aptos que serão "replicados" da população anterior para a nova. Ou seja, no início da geração de uma nova população, os E (parâmetro de entrada) primeiros indivíduos são copiados dos melhores indivíduos da população anterior.

4.8. Aptidão

A aptidão do indivíduo (ou fitness) é calculada pela função da classe IndividuoUteis, que recebe um Individuo como parâmetro, e retorna um objeto da classe Aptidao (utilizada apenas para guardar separadamente os scores de cada critério de aptidão separadamente). O valor final da aptidão do indivíduo é calculado através da soma dos resultados de 4 avaliações: avaliação de turmas alocadas (que atende o requisito RP.6); avaliação de choques de horários (que atende os requisitos RP.4 e RP.5); avaliação de professores por turma (que atende o requisito RP.7); avaliação de agrupamento de aulas (que atende o requisito RD.1). Estas avaliações são descritas a seguir.

4.8.1. Avaliação de turmas alocadas

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir do somatório de horas/aula de cada turma), iniciando com o seu valor máximo e sendo decrementado a cada diferença de horários alocados por turma em relação a sua necessidade (horas/aulas da disciplina da turma). Ex.: turma da disciplina INE5601 (4 horas/aula) foi alocada em 5 horários na semana = -1; turma da mesma disciplina foi alocada em 2 horários na semana = -2.

4.8.2. Avaliação de choques de horários

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir da soma de fases que possuem disciplinas com a quantidade de professores disponíveis, multiplicando este valor pela quantidade de horários disponíveis para alocação (sendo 14 no contexto da UFSC), multiplicando também pela quantidade de dias na semana), iniciando com seu valor máximo e podendo ser decrementado por dois fatores: o horário de uma turma já está alocado por outra turma, de outra disciplina, da mesma fase do mesmo curso; o horário de uma turma de um professor já está alocado por outra turma, com o mesmo professor.

4.8.3. Avaliação de professores por turma

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir do somatório de horas/aula de cada turma), iniciando com o seu valor máximo e podendo ser decrementado a cada divergência entre o professor alocado para uma turma, e o professor alocado para a mesma turma em outro horário.

4.8.4. Avaliação de agrupamento de aulas

Calcula um valor inteiro entre 0 e o seu valor máximo (calculado a partir do somatório de horas/aula de cada turma), iniciando com o seu valor máximo e sendo decrementado caso uma alocação de uma turma esteja isolada não contenha outra alocação da mesma turma no horário anterior ou seguinte.

4.8.5. Normalização e pesos

Todos esses valores de cada avaliação são normalizados antes de serem somados, visando equilibrar o peso de cada fator para o valor final da aptidão do indivíduo. Esta normalização é feita através de uma função que recebe: (i) o valor, (ii) o valor máximo possível, (iii) peso da avaliação. Esta função divide o valor pelo seu valor máximo possível e então multiplica o resultado desta divisão pelo peso da avaliação, a partir disso é obtido um valor proporcional de cada avaliação, que então é convertido para um número inteiro para facilitar o tratamento do fitness.

Os pesos são definidos a partir da importância do requisito que a avaliação atende. Por padrão, as avaliações de requisitos principais (RPs) possuem peso 10.000, enquanto que a avaliação de requisito desejado (RD) possui peso 200. Diferentes valores de pesos de RDs foram testados a fim de encontrar resultados mais satisfatórios e de forma mais rápida, conforme descrito no próximo capítulo deste documento.

4.9. Critérios de parada

Após o início da execução do algoritmo genético implementado, existem dois critérios analisados ao final de cada nova geração de indivíduos, para que então o algoritmo seja interrompido e apresente uma saída (solução):

- Número máximo de gerações, o qual é definido internamente, variando de acordo com o tamanho da população informado.
- O melhor indivíduo da nova geração atingiu os valores máximos de aptidão nas avaliações de Requisitos Principais.

5. Resultados

Foram realizados diversos testes para analisar a qualidade das soluções produzidas pelo algoritmo proposto, com diversas variações de parâmetros de entrada, buscando a melhor configuração para atingir resultados satisfatórios.

Levando em consideração os principais testes, assim como todos os outros executados ao longo do trabalho, foi possível identificar maior convergência na utilização do método de seleção por torneio, assim como outras faixas de valores para cada parâmetro.

6. Conclusões e trabalhos futuros

Este trabalho apresentou a teoria a respeito de algoritmos genéticos, *timetabling* e outros aspectos importantes para o embasamento teórico necessário na construção do algoritmo genético proposto e implementado como principal objetivo deste trabalho.

A utilização da abordagem de algoritmos genéticos possibilitou a solução do problema identificado, com características e restrições complexas, sem a utilização de técnicas especificas do domínio abordado. Ao invés disso, o algoritmo demandou apenas de meios para a avaliação da qualidade das soluções geradas.

A partir disso, podemos concluir que a utilização de algoritmos genéticos oferece uma alternativa viável para solucionar problemas de *timetabling*, como é o caso da geração de grade de horárias da UFSC, onde algumas regras e preferências internas dificultam a implementação de métodos determinísticos para a geração de soluções satisfatórias.

O uso de algoritmos genéticos em domínios complexos também foi abordado nos trabalhos de [Franco 2014] e [Sumbana et al. 2012], tomados como principais referências neste trabalho.

Embora os resultados obtidos neste trabalho possam confirmar a viabilidade de aplicação da técnica, a implementação apresentada, no seu estado atual, requer importantes melhorias para a sua utilização em cenários reais. Entre essas melhorias, podem ser citados os requisitos desejáveis (DP) não atendido pelo algoritmo.

Espera-se que este trabalho auxilie novas iniciativas na área de geração de grades horárias para escolas e universidades.

Por fim, são reunidas sugestões de melhorias e continuidade na busca por soluções para a automatização da geração de grades de horários das turmas UFSC.

- Implementação dos requisitos desejáveis que não foram considerados no algoritmo.
- Implementação de outro algoritmo genético para a alocação de salas ou laboratórios para as turmas, podendo trabalhar em sincronia com o algoritmo desenvolvido neste trabalho.
- Melhorar a estrutura do algoritmo, de forma mais genérica, para permitir a aplicação em diferentes domínios.
- Avaliação da utilização de outros operadores de seleção, com o objetivo de aumentar ou diminuir a pressão seletiva, ou ainda para contribuir na distribuição das chances por faixas de valores de aptidão.
- Estudo de diferentes bibliotecas que possam facilitar a implementação do algoritmo, e de diferentes técnicas para melhorar o seu resultado. Algumas das bibliotecas encontradas durante este trabalho foram: JGAP, Jenetics, GACS e PyBrain.
- Avaliação da utilização de diferentes critérios de parada para o algoritmo genético, como estagnação do valor de aptidão, tempo de execução, entre outros.
- Avaliação dos benefícios do uso de técnicas e estratégias que auxiliem na fuga de eventuais mínimos locais atingidos pelo algoritmo.
- Identificar outros aspectos que possam ser introduzidos como requisitos desejáveis para este algoritmo, de forma a aumentar a sua aplicabilidade em problemas reais.

Referências

BURKE, E. K., CAUSMAECKER, P. D., and BERGHE, G. V. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, 7:441–499.

Franco, R. A. P. (2014). Verificação funcional de sistemas digitais utilizando algoritmos genéticos na geração de dados aplicada à metodologia verisc.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms, Complex Adaptive Systems*.

MÜLLER, T. (2005). Constraint-based timetabling.

Sumbana, M. I. M., Silva, A. J. C., Gonçalves, M. A., Almeida, J., and Pappa, G. (2012). Seleção de atributos utilizando algoritmos genéticos para detecção do vandalismo na wikipedia. *Simpósio Brasileiro de Bancos de Dados*.

Wren, A. (2006). Scheduling, timetabling and rostering – a special relationship? *Springer, Berlin, Heidelberg*, pages 46–75.

APÊNDICE B – Código do algoritmo (Java)

```
// AG.java
package br.ufsc.ag.alocacao.professores.controle;

import java.io.FileNotFoundException;
import java.util.Calendar;
import java.util.Date;

import br.ufsc.ag.alocacao.professores.modelo.Aptidao;
import br.ufsc.ag.alocacao.professores.modelo.Individuo;
import br.ufsc.ag.alocacao.professores.uteis.ArquivoUteis;
import br.ufsc.ag.alocacao.professores.uteis.Constantes;
import br.ufsc.ag.alocacao.professores.uteis.IndividuoUteis;
import
    br.ufsc.ag.alocacao.professores.uteis.enums.TipoCrossoverEnum;

public class AG {

   private static final StringBuilder LOG_SB = new
       StringBuilder();

   private static void logInfo(String log) {
      LOG_SB.append("\n" + log);
      System.out.println(log);
   }

   private static void logError(String log) {
      LOG_SB.append(log);
   }

    public static void main(String[] args) {

      long t0 = Calendar.getInstance().getTimeInMillis();

        // taxa de crossover (0 - 1)
        Algoritmo.setTaxaDeCrossover(0.87);
        Algoritmo.setTipoCrossover(TipoCrossoverEnum.DISTRIBUIDO);
        //percentual de indivduos da  populao  que  sero  mutados
        Algoritmo.setPercentualIndividuosMutados(0.25);
        // taxa de  mutao  (quantidade de genes que sero mutados)
        Algoritmo.setTaxaDeMutacao(4);
        // tipo de  seleo  dos pais
        Algoritmo.setTipoSelecaoTorneio();
        // elitismo
        int elitismo = 12;
```

```
  // tamanho da  populao
  int tamPop = 300;
  // numero mximo de  geraes
  int numMaxGeracoes = (tamPop <= 100 ? 300 : tamPop <=
      300 ? 200 : tamPop <= 400 ? 150 : 100) * 1000;

  // quantas  solues  (melhores indivduos) devem compor a
      sada do algoritmo
  int limitSolucoes = 5;

  logInfo("--------------- Alogitmo iniciado com as
      seguintes configuraes ---------------");
  logInfo(String.format("Crossover [%s - %s]",
      Algoritmo.getTipoCrossover().name(), 100 *
      Algoritmo.getTaxaDeCrossover() + "%"));
  logInfo(String.format(" Mutao  [%s - %s atributos]", 100 *
      Algoritmo.getPercentualIndividuosMutados() + "%",
      Algoritmo.getTaxaDeMutacao()));
logInfo(String.format(" Seleo  [%s - %s]",
    Algoritmo.getTipoSelecaoPais(),
    Algoritmo.getTaxaDeTruncamento()));
logInfo(String.format("Elitismo [%s]", elitismo));
logInfo(String.format("Tamanho da populao [%s]", tamPop));
logInfo(String.format("Mximo de  geraes  [%s]",
    numMaxGeracoes));
logInfo(String.format("Horrios informados [%s]",
    IndividuoUteis.HORARIOS_INFORMADOS));
logInfo("\n");

  //cria a primeira  populao  aleatrioa
  Populacao populacao = new Populacao(tamPop, true);

  int geracao = 0;

  //loop  at  o  critrio  de parada
  while (geracao < numMaxGeracoes) {
      geracao++;

      //cria nova populacao
      populacao = Algoritmo.novaGeracao(populacao,
          elitismo);

      Aptidao aptidao =
          IndividuoUteis.getAptidao(populacao.getIndividuo(0));
```

```
    logInfo(String.format(" Gerao  %s | Aptido: %s
        (turmas alocadas: %s | choques: %s |
        professorTurma: %s | aulasAgrupadas: %s)",
         geracao,
         aptidao,
         aptidao.getScoreTurmasAlocadas(),
         aptidao.getScoreChoqueHorarios(),
         aptidao.getScoreProfessoresTurma(),
         aptidao.getScoreAulasPares()));
    if(IndividuoUteis
    .isAptidaoObrigatoriaCumprida(populacao.getIndividuo(0)))
        {
       break;
    }
    for (int i = 0; i < populacao.getNumIndividuos();
        i++) {
    populacao.getIndividuo(i).setAptidao(null);;
 }
}

if (geracao == numMaxGeracoes) {
 logInfo("Nmero Maximo de  Geraes  | " +
     populacao.getIndividuo(0) + " " +
     IndividuoUteis.getAptidao(populacao.getIndividuo(0)));
}

logInfo(String.format("Tempo gasto: %s segundos", new
    Double((Calendar.getInstance().getTimeInMillis() -
    t0)) / 1000));

logInfo("\nMelhores  alocaes  geradas:\n");
Individuo[] melhoresIndividuos = new
    Individuo[limitSolucoes];
for (int i = 0; i < limitSolucoes; i++) {
 melhoresIndividuos[i] = populacao.getIndividuo(i);
 Aptidao aptidao =
     IndividuoUteis.getAptidao(populacao.getIndividuo(i));
 logInfo(String.format("%s melhor indivduo [aptido=%s,
     turmasAlocadas=%s, choqueHorarios=%s,
     professoresTurma=%s, aulasAgrupadas=%s]]",
      i+1,
      aptidao.getAptidao(),
      aptidao.getScoreTurmasAlocadas(),
      aptidao.getScoreChoqueHorarios(),
      aptidao.getScoreProfessoresTurma(),
```

```
                aptidao.getScoreAulasPares()));
        }
          Date t1 = Calendar.getInstance().getTime();
          try {
           ArquivoUteis.escreverResultadoCsv(t1,
                Constantes.ARQUIVOS.CSV.CAMINHO_BASE_ESCRITA,
                Constantes.ARQUIVOS.CSV.NOME_ARQUIVO_ESCRITA,
                melhoresIndividuos);
        } catch (FileNotFoundException e) {
           logError("Erro ao gravar resultados:");
           e.printStackTrace();
        }
          ArquivoUteis.escreverArquivoLog(t1, LOG_SB.toString());

    }
}
```

```
// Algoritmo.java
package br.ufsc.ag.alocacao.professores.controle;

import java.util.Random;

import br.ufsc.ag.alocacao.professores.modelo.Alocacao;
import br.ufsc.ag.alocacao.professores.modelo.Individuo;
import br.ufsc.ag.alocacao.professores.modelo.Professor;
import br.ufsc.ag.alocacao.professores.modelo.Turma;
import br.ufsc.ag.alocacao.professores.uteis.Constantes;
import br.ufsc.ag.alocacao.professores.uteis.IndividuoUteis;
import
    br.ufsc.ag.alocacao.professores.uteis.enums.TipoCrossoverEnum;
import
    br.ufsc.ag.alocacao.professores.uteis.enums.TipoSelecaoPaisEnum;

public class Algoritmo {

   /**
    * 0 = "uniforme"
    * 1 = distribudo
    */
   private static int tipoCrossover;
    private static double taxaDeCrossover;
    private static double percentualIndividuosMutados;
    private static int taxaDeMutacao;
    private static int taxaDeTruncamento;
```

```
/**
 * 0 = torneio
 * 1 = truncamento
 */
private static int tipoSelecaoPais = 0;

public static Populacao novaGeracao(Populacao populacao, int
    elitismo) {
    Random r = new Random();
    //nova  populao  do mesmo tamanho da antiga
    Populacao novaPopulacao = new
        Populacao(populacao.getTamPopulacao());

    //se tiver elitismo, mantm o melhor indivduo da  gerao
        atual
    for(int i = 0; i < elitismo; i++) {
       novaPopulacao.setIndividuo(populacao.getIndividuo(i));
    }

    // insere novos indivduos na nova  populao ,  at  atingir
        o tamanho mximo
    while (novaPopulacao.getNumIndividuos() <
        novaPopulacao.getTamPopulacao()) {
     Individuo[] pais;
     switch (tipoSelecaoPais) {
     case 1:
        // seleciona os 2 pais por truncamento
        pais = selecaoTruncamento(populacao);
        break;
     case 0:
     default:
        // seleciona os 2 pais por torneio
        pais = selecaoTorneio(populacao);
        break;
     }

        Individuo[] filhos = new Individuo[2];

        //verifica a taxa de crossover, se sim realiza o
            crossover, se no, mantm os pais selecionados para
            a prxima gerao
        if (r.nextDouble() <= taxaDeCrossover) {
            filhos = crossover(pais[1], pais[0]);
        } else {
            filhos[0] = new Individuo(pais[0].getGenes());
```

```
                filhos[1] = new Individuo(pais[1].getGenes());
            }

            // Aplica  mutao , se for o caso
         mutacao(filhos[0]);
         mutacao(filhos[1]);

            // adiciona os filhos na nova gerao
            novaPopulacao.setIndividuo(filhos[0]);
            novaPopulacao.setIndividuo(filhos[1]);
        }

        //ordena a nova  populao
        novaPopulacao.ordenarPopulacao();
        return novaPopulacao;
    }

    public static void mutacao(Individuo individuo) {
        Random r = new Random();
        //se for mutar, cria um gene aleatrio
        if (r.nextDouble() <=
             Algoritmo.getPercentualIndividuosMutados()) {
         individuo.setAptidao(null);
         boolean[] atributosMutacao = new boolean[4];
         if(IndividuoUteis.HORARIOS_INFORMADOS) {
            atributosMutacao[3] = true;
         } else if(Algoritmo.getTaxaDeMutacao() > 0){
            for (int i = 0; i < Algoritmo.getTaxaDeMutacao();
                 i++) {
               atributosMutacao[r.nextInt(atributosMutacao.length)]
                   = true;
            }
         } else {
            for (int i = 0; i < 4; i++) {
               atributosMutacao[r.nextInt(atributosMutacao.length)]
                   = r.nextBoolean();
            }
         }

         for (Alocacao alocacao : individuo.getGenes()) {
            if(atributosMutacao[0]) {
               alocacao.setDiaSemana(
                   IndividuoUteis.gerarDiaSemanaAleatorio());
            }
            if(atributosMutacao[2]) {
```

```
            alocacao.setTurma(
                IndividuoUteis.gerarTurmaAleatoria());
        }
        if(atributosMutacao[1]) {
            alocacao.setHorario(
            IndividuoUteis.gerarHorarioAleatorio(
                alocacao.getTurma().getDisciplina()
                    .getCurso().getPeriodo()));
        }
        if(atributosMutacao[3]) {
            alocacao.setProfessor(
            IndividuoUteis.gerarProfessorAleatorio(
                alocacao.getTurma()
                    .getDisciplina().getAreaEnsino()));
        }
     }
    }


}

public static Individuo[] crossover(Individuo individuo1,
     Individuo individuo2) {
    Random r = new Random();

    //sorteia os pontos de troca
    boolean[] trocas = new
         boolean[Constantes.GENE.ALOCACAO.QUANT_ATRIBUTOS];
    if(tipoCrossover == 0) {
     if(IndividuoUteis.HORARIOS_INFORMADOS) {
        trocas[3] = r.nextBoolean();
     } else {
        for (int i = 0; i < trocas.length; i++) {
           trocas[i] = r.nextBoolean();
        }
     }
    }

    Individuo[] filhos = new Individuo[2];

    // pega os genes dos pais
    Alocacao[] genesPai1 = individuo1.getGenes();
    Alocacao[] genesPai2 = individuo2.getGenes();
```

```
Alocacao[] genesFilho1 = new Alocacao[genesPai1.length];
Alocacao[] genesFilho2 = new Alocacao[genesPai1.length];

for (int i = 0; i < genesPai1.length; i++) {
 Integer horarioFilho1;
 Integer diaSemanaFilho1;
 Turma turmaFilho1;
 Professor professorFilho1;

 Integer horarioFilho2;
 Integer diaSemanaFilho2;
 Turma turmaFilho2;
 Professor professorFilho2;
 if(tipoCrossover == 0 ? trocas[0] :
     !IndividuoUteis.HORARIOS_INFORMADOS &&
     r.nextBoolean()) {
    // utiliza atributo do pai2 no filho1 e pai1 no filho2
    horarioFilho1 = genesPai2[i].getHorario();
    horarioFilho2 = genesPai1[i].getHorario();
 } else {
    // utiliza atributo do pai1 no filho1 e pai2 no filho2
    horarioFilho1 = genesPai1[i].getHorario();
    horarioFilho2 = genesPai2[i].getHorario();
 }
 if(tipoCrossover == 0 ? trocas[1] :
     !IndividuoUteis.HORARIOS_INFORMADOS &&
     r.nextBoolean()) {
    // utiliza atributo do pai2 no filho1 e pai1 no filho2
    diaSemanaFilho1 = genesPai2[i].getDiaSemana();
    diaSemanaFilho2 = genesPai1[i].getDiaSemana();
 } else {
    // utiliza atributo do pai1 no filho1 e pai2 no filho2
    diaSemanaFilho1 = genesPai1[i].getDiaSemana();
    diaSemanaFilho2 = genesPai2[i].getDiaSemana();
 }
 if(tipoCrossover == 0 ? trocas[2] :
     !IndividuoUteis.HORARIOS_INFORMADOS &&
     r.nextBoolean()) {
    // utiliza atributo do pai2 no filho1 e pai1 no filho2
    turmaFilho1 = genesPai2[i].getTurma();
    turmaFilho2 = genesPai1[i].getTurma();
 } else {
    // utiliza atributo do pai1 no filho1 e pai2 no filho2
    turmaFilho1 = genesPai1[i].getTurma();
    turmaFilho2 = genesPai2[i].getTurma();
```

```
        }
        if(tipoCrossover == 0 ? trocas[3] : r.nextBoolean()) {
           // utiliza atributo do pai2 no filho1 e pai1 no filho2
           professorFilho1 = genesPai2[i].getProfessor();
           professorFilho2 = genesPai1[i].getProfessor();
        } else {
           // utiliza atributo do pai1 no filho1 e pai2 no filho2
           professorFilho1 = genesPai1[i].getProfessor();
           professorFilho2 = genesPai2[i].getProfessor();
        }

        genesFilho1[i] = new Alocacao(diaSemanaFilho1,
            horarioFilho1, turmaFilho1, professorFilho1);
        genesFilho2[i] = new Alocacao(diaSemanaFilho2,
            horarioFilho2, turmaFilho2, professorFilho2);
     }

     // cria o novo indivduo com os genes dos pais
     filhos[0] = new Individuo(genesFilho1);
     filhos[1] = new Individuo(genesFilho2);

     return filhos;
  }

  public static Individuo[] selecaoTorneio(Populacao
       populacao) {
     Random r = new Random();
     Populacao populacaoIntermediaria = new Populacao(3);

     //seleciona 3 indivduos aleatriamente na  populao
     populacaoIntermediaria.setIndividuo(
        populacao.getIndividuo(r.nextInt(
           populacao.getTamPopulacao())));
     populacaoIntermediaria.setIndividuo(
        populacao.getIndividuo(r.nextInt(
           populacao.getTamPopulacao())));
     populacaoIntermediaria.setIndividuo(
        populacao.getIndividuo(r.nextInt(
           populacao.getTamPopulacao())));

     //ordena a  populao
     populacaoIntermediaria.ordenarPopulacao();

     Individuo[] pais = new Individuo[2];
```

```
        //seleciona os 2 melhores deste populao
        pais[0] = populacaoIntermediaria.getIndividuo(0);
        pais[1] = populacaoIntermediaria.getIndividuo(1);

        return pais;
    }

    public static Individuo[] selecaoTruncamento(Populacao
        populacao) {
      populacao.ordenarPopulacao();
      Individuo[] selecaoPais = new Individuo[taxaDeTruncamento];
      for (int i = 0; i < taxaDeTruncamento; i++) {
         selecaoPais[i] = populacao.getIndividuo(i);
      }
      Individuo[] pais = new Individuo[2];
      Random r = new Random();
      pais[0] = selecaoPais[r.nextInt(taxaDeTruncamento)];
      pais[1] = selecaoPais[r.nextInt(taxaDeTruncamento)];
      return pais;
    }

    public static double getTaxaDeCrossover() {
        return taxaDeCrossover;
    }

    public static void setTaxaDeCrossover(double
        taxaDeCrossover) {
        Algoritmo.taxaDeCrossover = taxaDeCrossover;
    }

    public static double getPercentualIndividuosMutados() {
        return percentualIndividuosMutados;
    }

    public static void setPercentualIndividuosMutados(double
        percentualIndividuosMutados) {
        Algoritmo.percentualIndividuosMutados =
            percentualIndividuosMutados;
    }

   public static int getTaxaDeMutacao() {
      return taxaDeMutacao;
   }

   public static void setTaxaDeMutacao(int taxaDeMutacao) {
```

```
   Algoritmo.taxaDeMutacao = taxaDeMutacao;
}

public static void setTaxaDeTruncamento(int
    taxaDeTruncamento) {
   Algoritmo.taxaDeTruncamento = taxaDeTruncamento;
}

public static TipoSelecaoPaisEnum getTipoSelecaoPais() {
   return
       TipoSelecaoPaisEnum.getEnum(Algoritmo.tipoSelecaoPais);
}

/**
 * 0 = torneio
 * 1 = truncamento
 *
 * @param tipoSelecaoPais
 */
private static void setTipoSelecaoPais(TipoSelecaoPaisEnum
    tipoSelecaoPais) {
   Algoritmo.tipoSelecaoPais = tipoSelecaoPais.getValue();
}

public static void setTipoSelecaoTorneio() {
   setTipoSelecaoPais(TipoSelecaoPaisEnum.TORNEIO);
}

public static void setTipoSelecaoTruncamento(int
    taxaDeTruncamento) {
   Algoritmo.taxaDeTruncamento = taxaDeTruncamento;
   setTipoSelecaoPais(TipoSelecaoPaisEnum.TRUNCAMENTO);
}

public static void setTipoCrossover(TipoCrossoverEnum
    tipoCrossover) {
   Algoritmo.tipoCrossover = tipoCrossover.getValue();
}

public static TipoCrossoverEnum getTipoCrossover() {
   return TipoCrossoverEnum.getEnum(Algoritmo.tipoCrossover);
}

public static void setTipoCrossover(int tipoCrossover) {
   Algoritmo.tipoCrossover = tipoCrossover;
```

```
   }

   public static int getTaxaDeTruncamento() {
      return taxaDeTruncamento;
   }
}
```

```
// Populacao.java
package br.ufsc.ag.alocacao.professores.controle;

import java.util.Arrays;
import java.util.Comparator;

import br.ufsc.ag.alocacao.professores.modelo.Individuo;
import br.ufsc.ag.alocacao.professores.uteis.IndividuoUteis;

public class Populacao {

    private Individuo[] individuos;
    private int tamPopulacao;
    private int indicePosDisponivel;

    //cria uma  populao  com indivduos sem valor, ser composto
        posteriormente
    public Populacao(int tamPop) {
      this(tamPop, false);
    }

    public Populacao(int tamPop, boolean
        preencherAleatoriamente) {
       indicePosDisponivel = 0;
      tamPopulacao = tamPop;
       individuos = new Individuo[tamPop];
       for (int i = 0; i < individuos.length; i++) {
           individuos[i] = preencherAleatoriamente ?
                IndividuoUteis.gerarIndividuoAleatorio() : null;
       }
    }

    //coloca um indivduo em uma certa  posio  da  populao
    public void setIndividuo(Individuo individuo, int posicao) {
       individuos[posicao] = individuo;
    }
```

```
    //coloca um individuo na prxima  posio  disponvel da
         populao
    public void setIndividuo(Individuo individuo) {
      if(indicePosDisponivel < tamPopulacao &&
          individuos[indicePosDisponivel] == null) {
         individuos[indicePosDisponivel] = individuo;
         indicePosDisponivel++;
         return;
      }
    }

    //ordena a  populao  pelo valor de aptido de cada individuo,
        do maior valor para o menor, assim se eu quiser obter o
        melhor individuo desta populao, acesso a posio 0 do array
        de individuos
    public void ordenarPopulacao() {
      Arrays.asList(individuos).sort(new Comparator<Individuo>()
          {

         @Override
         public int compare(Individuo o1, Individuo o2) {
            return IndividuoUteis.getAptidao(o2).getAptidao() -
                 IndividuoUteis.getAptidao(o1).getAptidao();
         }

      });
    }

    //nmero  de individuos existentes na  populao
    public int getNumIndividuos() {
        return indicePosDisponivel;
    }

    public int getTamPopulacao() {
        return tamPopulacao;
    }

    public Individuo getIndividuo(int pos) {
        return individuos[pos];
    }
}
```

```
// Alocacao.java
package br.ufsc.ag.alocacao.professores.modelo;
```

```
import
    br.ufsc.ag.alocacao.professores.uteis.Constantes.GENE.ALOCACAO;

public class Alocacao {

   /**
    * Valores entre 0 e 6
    *
    * 0 = Domingo
    * 1 = Segunda-feira
    * ...
    * 6 = Sbado
    */
   private Integer diaSemana;

   /**
    * Valores entre 0 e 13
    *
    * 0 = 7:30-8:20
    * 1 = 8:20-9:10
    * ...
    * 13 = 21:10-22:00
    */
   private Integer horario;
   private Turma turma;
   private Professor professor;

   public Alocacao(Integer diaSemana, Integer horario, Turma
       turma, Professor professor) {
      this.diaSemana = diaSemana;
      this.horario = horario;
      this.turma = turma;
      this.professor = professor;
   }

   public Integer getDiaSemana() {
      return diaSemana;
   }

   public void setDiaSemana(Integer diaSemana) {
      this.diaSemana = diaSemana;
   }

   public Integer getHorario() {
```

```
      return horario;
   }

   public void setHorario(Integer horario) {
      this.horario = horario;
   }

   public Turma getTurma() {
      return turma;
   }

   public void setTurma(Turma turma) {
      this.turma = turma;
   }

   public Professor getProfessor() {
      return professor;
   }

   public void setProfessor(Professor professor) {
      this.professor = professor;
   }

   @Override
   public String toString() {
      return String.join(ALOCACAO.SEPARADOR,
            String.valueOf(this.horario),
            String.valueOf(this.diaSemana),
            this.turma.getCodigo(),
            this.professor.getCodigo());
   }

}
```

```
// Aptidao.java
package br.ufsc.ag.alocacao.professores.modelo;

public class Aptidao {

   private int scoreTurmasAlocadas;
   private int scoreChoqueHorarios;
   private int scoreProfessoresTurma;
   private int scoreAulasAgrupadas;
```

```
   public Aptidao(int scoreTurmasAlocadas, int
       scoreChoqueHorarios, int scoreProfessoresTurma, int
       scoreAulasPares) {
      this.scoreTurmasAlocadas = scoreTurmasAlocadas;
      this.scoreChoqueHorarios = scoreChoqueHorarios;
      this.scoreProfessoresTurma = scoreProfessoresTurma;
      this.scoreAulasAgrupadas = scoreAulasPares;
   }

   public int getScoreTurmasAlocadas() {
      return scoreTurmasAlocadas;
   }

   public int getScoreChoqueHorarios() {
      return scoreChoqueHorarios;
   }

   public int getScoreProfessoresTurma() {
      return scoreProfessoresTurma;
   }

   public int getScoreAulasPares() {
      return scoreAulasAgrupadas;
   }

   public boolean isAptidaoObrigatoriaCumprida(int coeficiente) {
      return (scoreTurmasAlocadas + scoreChoqueHorarios +
          scoreProfessoresTurma) == 3 * coeficiente;
   }

   public int getAptidao() {
      return scoreTurmasAlocadas + scoreChoqueHorarios +
          scoreProfessoresTurma + scoreAulasAgrupadas;
   }

   @Override
   public String toString() {
      return String.valueOf(getAptidao());
   }

}
```

```
// AreaEnsino.java
package br.ufsc.ag.alocacao.professores.modelo;
```

```
public class AreaEnsino extends EntidadeBase {

   private String nome;

   public AreaEnsino(String codigo, String nome) {
      this.codigo = codigo;
      this.nome = nome;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

}
```

```
// Curso.java
package br.ufsc.ag.alocacao.professores.modelo;

import
    br.ufsc.ag.alocacao.professores.uteis.enums.PeriodoCursoEnum;

public class Curso extends EntidadeBase {

   private String nome;
   private PeriodoCursoEnum periodo;
   private int fases;

   public Curso(String codigo, String nome, PeriodoCursoEnum
       periodo) {
      this.codigo = codigo;
      this.nome = nome;
      this.periodo = periodo;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
```

```
      this.nome = nome;
   }

   public PeriodoCursoEnum getPeriodo() {
      return periodo;
   }

   public void setPeriodo(PeriodoCursoEnum periodo) {
      this.periodo = periodo;
   }

   public int getFases() {
      return fases;
   }

   public void setFases(int fases) {
      this.fases = fases;
   }
}
```

```
// Disciplina.java
package br.ufsc.ag.alocacao.professores.modelo;

public class Disciplina extends EntidadeBase {

   private String nome;
   private Curso curso;
   private AreaEnsino areaEnsino;
   private Integer fase;
   private Integer horasAula;

   public Disciplina(String codigo, String nome, Curso curso,
        AreaEnsino areaEnsino, Integer horasAula, Integer fase) {
      this.codigo = codigo;
      this.nome = nome;
      this.curso = curso;
      this.areaEnsino = areaEnsino;
      this.horasAula = horasAula;
      this.fase = fase;
   }

   public Curso getCurso() {
      return curso;
   }
```

```
   public void setCurso(Curso curso) {
      this.curso = curso;
   }

   public AreaEnsino getAreaEnsino() {
      return areaEnsino;
   }

   public void setAreaEnsino(AreaEnsino areaEnsino) {
      this.areaEnsino = areaEnsino;
   }

   public Integer getHorasAula() {
      return horasAula;
   }

   public void setHorasAula(Integer horasAula) {
      this.horasAula = horasAula;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

   public Integer getFase() {
      return fase;
   }

   public void setFase(Integer fase) {
      this.fase = fase;
   }
}
```

```
// EntidadeBase.java
package br.ufsc.ag.alocacao.professores.modelo;

public abstract class EntidadeBase {

   protected String codigo;
```

```
   public String getCodigo() {
      return codigo;
   }

   public void setCodigo(String codigo) {
      this.codigo = codigo;
   }
}
```

```
// Individuo.java
package br.ufsc.ag.alocacao.professores.modelo;

import br.ufsc.ag.alocacao.professores.uteis.Constantes;

public class Individuo {

   private Alocacao[] genes;
   private Aptidao aptidao;

   public Individuo(Alocacao[] genes) {
      this.genes = genes;
   }

   public Alocacao[] getGenes() {
      return genes;
   }

   @Override
   public String toString() {
      String[] genes = new String[this.genes.length];
      for (int i = 0; i < genes.length; i++) {
         genes[i] = this.genes[i].toString();
      }
      return String.join(Constantes.GENE.SEPARADOR, genes);
   }

   public Aptidao getAptidao() {
      return aptidao;
   }

   public void setAptidao(Aptidao aptidao) {
      this.aptidao = aptidao;
   }
```

```
}
```

```
// Professor.java
package br.ufsc.ag.alocacao.professores.modelo;

public class Professor extends EntidadeBase {

   private String nome;
   private AreaEnsino[] areasEnsino;

   public Professor(String codigo, String nome, AreaEnsino[]
       areasEnsino) {
      this.codigo = codigo;
      this.nome = nome;
      this.areasEnsino = areasEnsino;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

   public AreaEnsino[] getAreasEnsino() {
      return areasEnsino;
   }

   public void setAreasEnsino(AreaEnsino[] areasEnsino) {
      this.areasEnsino = areasEnsino;
   }
}
```

```
// Turma.java
package br.ufsc.ag.alocacao.professores.modelo;

public class Turma {

   private String codigo;
   private Disciplina disciplina;
```

```
   public Turma(String codigo, Disciplina disciplina) {
      this.codigo = codigo;
      this.disciplina = disciplina;
   }

   public String getCodigo() {
      return codigo;
   }
   public void setCodigo(String codigo) {
      this.codigo = codigo;
   }
   public Disciplina getDisciplina() {
      return disciplina;
   }
   public void setDisciplina(Disciplina disciplina) {
      this.disciplina = disciplina;
   }
}
```

```
// DiaSemanaEnum.java
package br.ufsc.ag.alocacao.professores.uteis.enums;

public enum DiaSemanaEnum {

   SEGUNDA(1),
   TERCA(2),
   QUARTA(3),
   QUINTA(4),
   SEXTA(5);

   private int value;

   DiaSemanaEnum(int diaSemana) {
      this.value = diaSemana;
   }

   public static DiaSemanaEnum getEnum(int value) {
      for(DiaSemanaEnum diaSemana : values())
            if(diaSemana.getValue() == value) return diaSemana;
        throw new IllegalArgumentException();
   }

   public int getValue() {
      return value;
```

```
   }

}
```

```
// PeriodoCursoEnum.java
package br.ufsc.ag.alocacao.professores.uteis.enums;

public enum PeriodoCursoEnum {

   INTEGRAL("i", new int[]{0,1,2,3,4,5,6,7,8,9}),
   NOTURNO("n", new int[]{10,11,12,13});

   private String value;
   private int[] horarios;

   PeriodoCursoEnum(String value, int[] horarios) {
      this.value = value;
      this.horarios = horarios;
   }

   public String getValue() {
      return value;
   }

   public int[] getHorarios() {
      return horarios;
   }

   public void setHorarios(int[] horarios) {
      this.horarios = horarios;
   }

   public static PeriodoCursoEnum getEnum(String value) {
      for(PeriodoCursoEnum periodo : values())
           if(periodo.getValue().equalsIgnoreCase(value)) return
               periodo;
        throw new IllegalArgumentException();
   }
}
```

```
// TipoCrossoverEnum.java
package br.ufsc.ag.alocacao.professores.uteis.enums;
```

```
public enum TipoCrossoverEnum {

   UNIFORME(0),
   DISTRIBUIDO(1);

   private int value;

   TipoCrossoverEnum(int value) {
      this.value = value;
   }

   public static TipoCrossoverEnum getEnum(int value) {
      for(TipoCrossoverEnum tipoCrossover : values())
           if(tipoCrossover.getValue() == value) return
               tipoCrossover;
        throw new IllegalArgumentException();
   }

   public int getValue() {
      return value;
   }
}
```

```
// TipoSelecaoPaisEnum.java
package br.ufsc.ag.alocacao.professores.uteis.enums;

public enum TipoSelecaoPaisEnum {

   TORNEIO(0),
   TRUNCAMENTO(1);

   private int value;

   TipoSelecaoPaisEnum(int value) {
      this.value = value;
   }

   public static TipoSelecaoPaisEnum getEnum(int value) {
      for(TipoSelecaoPaisEnum tipoSelecao : values())
           if(tipoSelecao.getValue() == value) return
               tipoSelecao;
        throw new IllegalArgumentException();
   }
```

```
   public int getValue() {
      return value;
   }
}
```

```
// AlocacaoComparator.java
package br.ufsc.ag.alocacao.professores.uteis;

import java.util.Comparator;

import br.ufsc.ag.alocacao.professores.modelo.Alocacao;

public class AlocacaoComparator implements Comparator<Alocacao> {
   @Override
   public int compare(Alocacao o1, Alocacao o2) {
      return ((o1.getTurma().getDisciplina().getFase() * 1000)
              + (o1.getDiaSemana() * 100) + o1.getHorario())
            - ((o2.getTurma().getDisciplina().getFase() * 1000)
                + (o2.getDiaSemana() * 100) + o2.getHorario());
   }
}
```

```
// AlocacaoUteis.java
package br.ufsc.ag.alocacao.professores.uteis;

public class AlocacaoUteis {

   public static String getHorario(Integer horario) {
      switch (horario) {
      case 0:
         return "7:30";
      case 1:
         return "8:20";
      case 2:
         return "9:10";
      case 3:
         return "10:10";
      case 4:
         return "11:00";
      case 5:
         return "13:30";
      case 6:
         return "14:20";
```

```
      case 7:
         return "15:10";
      case 8:
         return "16:20";
      case 9:
         return "17:10";
      case 10:
         return "18:30";
      case 11:
         return "19:20";
      case 12:
         return "20:20";
      case 13:
         return "21:10";
      default:
         return "INDEFINIDO";
      }
   }

   public static Integer getHorario(String horario) {
      switch (horario) {
      case "7:30":
         return 0;
      case "8:20":
         return 1;
      case "9:10":
         return 2;
      case "10:10":
         return 3;
      case "11:00":
         return 4;
      case "13:30":
         return 5;
      case "14:20":
         return 6;
      case "15:10":
         return 7;
      case "16:20":
         return 8;
      case "17:10":
         return 9;
      case "18:30":
         return 10;
      case "19:20":
         return 11;
```

```
      case "20:20":
         return 12;
      case "21:10":
         return 13;
      default:
         return null;
      }
   }
}
```

```
// ArquivoUteis.java
package br.ufsc.ag.alocacao.professores.uteis;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import br.ufsc.ag.alocacao.professores.modelo.Alocacao;
import br.ufsc.ag.alocacao.professores.modelo.AreaEnsino;
import br.ufsc.ag.alocacao.professores.modelo.Curso;
import br.ufsc.ag.alocacao.professores.modelo.Disciplina;
import br.ufsc.ag.alocacao.professores.modelo.Individuo;
import br.ufsc.ag.alocacao.professores.modelo.Professor;
import br.ufsc.ag.alocacao.professores.modelo.Turma;
import br.ufsc.ag.alocacao.professores.uteis.enums.DiaSemanaEnum;
import
    br.ufsc.ag.alocacao.professores.uteis.enums.PeriodoCursoEnum;

public class ArquivoUteis {

   public static Map<String, Curso> carregarCursos() {
      Map<String, Curso> mapaCursos = new HashMap<>();
      try (BufferedReader br = carregarArquivoCsv(
             Constantes.ARQUIVOS.CSV.CURSO.CAMINHO_CSV)){
```

```
        String line = "";
        while ((line = br.readLine()) != null) {
           String[] cursoArgs =
               line.split(Constantes.ARQUIVOS.CSV.SEPARADOR);

           PeriodoCursoEnum periodoCurso = null;
           try {
              periodoCurso = PeriodoCursoEnum.getEnum(
                  cursoArgs[Constantes.ARQUIVOS.CSV
                      .CURSO.INDICE_ATRIBUTOS.PERIODO]);
           } catch (IllegalArgumentException e) {
              periodoCurso = PeriodoCursoEnum.INTEGRAL;
           }
           Curso curso = new Curso(
               cursoArgs[Constantes.ARQUIVOS.CSV
                       .CURSO.INDICE_ATRIBUTOS.CODIGO],
                 cursoArgs[Constantes.ARQUIVOS.CSV
                     .CURSO.INDICE_ATRIBUTOS.NOME],
                 periodoCurso);
           mapaCursos.put(curso.getCodigo(), curso);
          }
     } catch (IOException e ) {
        System.out.println("[LOG] Erro ao carregar cursos: " +
            e.getMessage());
     }
     return mapaCursos;
  }


  public static Map<String, AreaEnsino> carregarAreasEnsino() {
     Map<String, AreaEnsino> mapaAreasEnsino = new HashMap<>();
     try (BufferedReader br =
          carregarArquivoCsv(Constantes.ARQUIVOS.CSV
             .AREA_ENSINO.CAMINHO_CSV)){
        String line = "";
        while ((line = br.readLine()) != null) {
           String[] areaEnsinoArgs =
               line.split(Constantes.ARQUIVOS.CSV.SEPARADOR);

           AreaEnsino areaEnsinio = new
               AreaEnsino(areaEnsinoArgs[Constantes.ARQUIVOS.CSV
                  .AREA_ENSINO.INDICE_ATRIBUTOS.CODIGO],
              areaEnsinoArgs[Constantes.ARQUIVOS.CSV
                     .AREA_ENSINO.INDICE_ATRIBUTOS.NOME]);
           mapaAreasEnsino.put(areaEnsinio.getCodigo(),
```

```
            areaEnsinio);
        }
    } catch (IOException e ) {
       System.out.println("[LOG] Erro ao carregar reas de
           ensinio: " + e.getMessage());
    }
    return mapaAreasEnsino;
}


public static Map<String, Professor>
     carregarProfessores(Map<String, AreaEnsino> areasEnsino) {
    Map<String, Professor> mapaProfessores = new HashMap<>();
    try (BufferedReader br =
        carregarArquivoCsv(Constantes.ARQUIVOS.CSV
           .PROFESSOR.CAMINHO_CSV)){
       String line = "";
       while ((line = br.readLine()) != null) {
          String[] professorArgs =
              line.split(Constantes.ARQUIVOS.CSV.SEPARADOR);

          String[] codigosAreasEnsino =
              professorArgs[Constantes.ARQUIVOS.CSV
              .PROFESSOR.INDICE_ATRIBUTOS.AREAS_ENSINO].split(",");
          AreaEnsino[] areasEnsinoAux = new
              AreaEnsino[codigosAreasEnsino.length];
          for (int i = 0; i < areasEnsinoAux.length; i++) {
             areasEnsinoAux[i] =
                 areasEnsino.get(codigosAreasEnsino[i]);
          }

          Professor professor = new
              Professor(professorArgs[Constantes.ARQUIVOS.CSV
                     .PROFESSOR.INDICE_ATRIBUTOS.CODIGO],
               professorArgs[Constantes.ARQUIVOS.CSV
                   .PROFESSOR.INDICE_ATRIBUTOS.NOME],
               areasEnsinoAux);
          mapaProfessores.put(professor.getCodigo(), professor);
         }
    } catch (IOException e ) {
       System.out.println("[LOG] Erro ao carregar professores:
           " + e.getMessage());
    }
    return mapaProfessores;
}
```

```
public static Map<String, Disciplina>
    carregarDisciplinas(Map<String, Curso> cursos,
    Map<String, AreaEnsino> areasEnsino) {
   Map<String, Disciplina> mapaDisciplinas = new HashMap<>();
   try (BufferedReader br =
       carregarArquivoCsv(Constantes.ARQUIVOS.CSV
          .DISCIPLINA.CAMINHO_CSV)){
      String line = "";
      while ((line = br.readLine()) != null) {
         String[] disciplinaArgs =
             line.split(Constantes.ARQUIVOS.CSV.SEPARADOR);

         Disciplina disciplina = new
             Disciplina(disciplinaArgs[Constantes.ARQUIVOS.CSV
             .DISCIPLINA.INDICE_ATRIBUTOS.CODIGO],
              disciplinaArgs[Constantes.ARQUIVOS.CSV
                  .DISCIPLINA.INDICE_ATRIBUTOS.NOME],
              cursos.get(disciplinaArgs[Constantes.ARQUIVOS.CSV
                  .DISCIPLINA.INDICE_ATRIBUTOS.CURSO]),
              areasEnsino.get(disciplinaArgs[Constantes.ARQUIVOS.CSV
                  .DISCIPLINA.INDICE_ATRIBUTOS.AREA_ENSINO]),
              Integer.valueOf(disciplinaArgs[Constantes.ARQUIVOS.CSV
                  .DISCIPLINA.INDICE_ATRIBUTOS.HORAS_AULA]),
              Integer.valueOf(disciplinaArgs[Constantes.ARQUIVOS.CSV
                  .DISCIPLINA.INDICE_ATRIBUTOS.FASE]));

         Curso curso = disciplina.getCurso();

         mapaDisciplinas.put(disciplina.getCodigo(),
             disciplina);
         if(curso == null) {
            System.out.println(String.format("[LOG-WARN]
                Disciplina carregada sem curso: codigo %s",
                disciplina.getCodigo()));
         } else if(curso.getFases() < disciplina.getFase()) {
            curso.setFases(disciplina.getFase());
         }
         if(disciplina.getAreaEnsino() == null) {
            System.out.println(String.format("[LOG-WARN]
                Disciplina carregada sem rea de ensino: codigo
                %s", disciplina.getCodigo()));
         }
        }
   } catch (IOException e ) {
```

```
            System.out.println("[LOG] Erro ao carregar disciplinas:
                " + e.getMessage());
        }
        return mapaDisciplinas;
    }

    public static List<Alocacao> carregarAlocacoes(Map<String,
         Disciplina> disciplinas) {
        List<Alocacao> alocacoes = new ArrayList<>();
        try (BufferedReader br =
            carregarArquivoCsv(Constantes.ARQUIVOS.CSV
               .TURMA.CAMINHO_CSV)){
            String line = "";
            while ((line = br.readLine()) != null) {
                String[] turmaArgs =
                    line.split(Constantes.ARQUIVOS.CSV.SEPARADOR);
                String[] codigoTurma =
                    turmaArgs[0].split(Constantes.ARQUIVOS.CSV
                    .TURMA.SEPARADOR);

                Turma turma = new Turma(turmaArgs[0],
                     disciplinas.get(codigoTurma[Constantes.ARQUIVOS.CSV
                         .TURMA.INDICE_ATRIBUTOS.DISCIPLINA]));

                if(turmaArgs.length == 2) {
                    String[] alocacoesStr =
                        turmaArgs[Constantes.ARQUIVOS.CSV
                        .TURMA.INDICE_ATRIBUTOS.ALOCACOES].split(",");
                    for (int i = 0; i < alocacoesStr.length; i++) {
                        String[] alocacaoArgs =
                            alocacoesStr[i].split("\\.");

                        alocacoes.add(new
                            Alocacao(Integer.parseInt(alocacaoArgs[0])-1,
                            AlocacaoUteis.getHorario(alocacaoArgs[1]),
                            turma, null));
                    }
                } else {
                    for (int i = 0; i <
                        turma.getDisciplina().getHorasAula(); i++) {
                        alocacoes.add(new Alocacao(null, null, turma,
                            null));
                    }
                }
                if(turma.getDisciplina() == null) {
```

```
            System.out.println(String.format("[LOG-WARN] Turma
                carregada sem disciplina: codigo %s",
                turma.getCodigo()));
          }
         }
    } catch (IOException e ) {
       System.out.println("[LOG] Erro ao carregar turmas: " +
           e.getMessage());
    }
    return alocacoes;
}

private static BufferedReader carregarArquivoCsv(String
     caminho) throws FileNotFoundException, IOException {
    return new BufferedReader(new FileReader(caminho));
}

private static DateFormat getDateFormat() {
    return new SimpleDateFormat("yyyyMMdd-HHmms");
}

public static void escreverResultadoCsv(Date dataHoraTermino,
     String caminho, String nomeArquivo, Individuo[]
     individuos) throws FileNotFoundException {
    String dataFormatada =
        getDateFormat().format(dataHoraTermino);
    caminho += "/" + dataFormatada;
    File diretorio = new File(caminho);
    if(!diretorio.isDirectory()) {
       diretorio.mkdirs();
    }
    for (int i = 0; i < individuos.length; i++) {
        try (PrintWriter pw = new PrintWriter(new File(caminho
            + "/" + String.format(nomeArquivo, i+1)))) {
           StringBuilder sb = new StringBuilder();
           Alocacao[] alocacoes = individuos[i].getGenes();
           sb.append(String.join(Constantes.ARQUIVOS.CSV.SEPARADOR,
               "Fase",
               "Disciplina",
               "Turma",
               "Nome da disciplina",
               "Horas/aula",
               "Dia",
               " Horrio ",
               "Professores",
```

```
                "Curso"
                ));
            sb.append("\n");
              for (int j = 0; j < alocacoes.length; j++) {
               Turma turma = alocacoes[j].getTurma();
               Disciplina disciplina = turma.getDisciplina();
               sb.append(String.join(Constantes.ARQUIVOS.CSV.SEPARADOR,
                  String.valueOf(disciplina.getFase()),
                  disciplina.getCodigo(),
                  turma.getCodigo(),
                  disciplina.getNome(),
                  String.valueOf(disciplina.getHorasAula()),
                      DiaSemanaEnum.getEnum(alocacoes[j]
                          .getDiaSemana()).name(),
                      (alocacoes[j].getDiaSemana()+1) + "." +
                           AlocacaoUteis.getHorario(
                              alocacoes[j].getHorario()),
                      alocacoes[j].getProfessor().getNome(),
                      disciplina.getCurso().getCodigo()));
               sb.append("\n");
             }
               pw.write(sb.toString());
            }
      }
   }

   public static void escreverArquivoLog(Date dataHoraTermino,
       String log) {
      File file = new File(String.format("target/%s",
          getDateFormat().format(dataHoraTermino)));
      file.mkdirs();
      try (PrintWriter pw = new PrintWriter(new
          File(String.format("%s/log.txt", file.getPath())))) {
         pw.append(log);
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      }
   }

}
```

// Constantes.java

```
package br.ufsc.ag.alocacao.professores.uteis;
```

```
public interface Constantes {

   public static final int TOTAL_HORARIOS_DISPONIVEIS = 14;

   public interface GENE {
         public static final String SEPARADOR = "|";
      public interface ALOCACAO {
         public static final String SEPARADOR = "+";
         public static final Integer QUANT_ATRIBUTOS =
             INDICE_ATRIBUTOS.PROFESSOR+1;
         public interface INDICE_ATRIBUTOS {
            public static final int HORARIO = 0;
            public static final int DIA_SEMANA = HORARIO+1;
            public static final int TURMA = DIA_SEMANA+1;
            public static final int PROFESSOR = TURMA+1;
         }
      }
   }

   public interface ARQUIVOS {
      public interface CSV {
         public static final String SEPARADOR = ";";
         public static final String CAMINHO_BASE_LEITURA =
             "src/main/resources/csv";
         public static final String CAMINHO_BASE_ESCRITA =
             "target";
         public static final String NOME_ARQUIVO_ESCRITA =
             "alocacao %s.csv";

         public interface CURSO {
            public static final String CAMINHO_CSV =
                CAMINHO_BASE_LEITURA + "/cursos.csv";

            public interface INDICE_ATRIBUTOS {
               public static final int CODIGO = 0;
               public static final int NOME = CODIGO+1;
               public static final int PERIODO = NOME+1;
            }
         }
         public interface AREA_ENSINO {
            public static final String CAMINHO_CSV =
                CAMINHO_BASE_LEITURA + "/areasEnsino.csv";

            public interface INDICE_ATRIBUTOS {
               public static final int CODIGO = 0;
```

```
            public static final int NOME = CODIGO+1;
         }
      }
      public interface PROFESSOR {
         public static final String CAMINHO_CSV =
             CAMINHO_BASE_LEITURA + "/professores.csv";

         public interface INDICE_ATRIBUTOS {
            public static final int CODIGO = 0;
            public static final int NOME = CODIGO+1;
            public static final int AREAS_ENSINO = NOME+1;
         }
      }

      public interface DISCIPLINA {
         public static final String CAMINHO_CSV =
             CAMINHO_BASE_LEITURA + "/disciplinas.csv";

         public interface INDICE_ATRIBUTOS {
            public static final int FASE = 0;
            public static final int CODIGO = FASE+1;
            public static final int NOME = CODIGO+1;
            public static final int CURSO = NOME+1;
            public static final int AREA_ENSINO = CURSO+1;
            public static final int HORAS_AULA = AREA_ENSINO+1;
         }
      }
      public interface TURMA {
         public static final String CAMINHO_CSV =
             CAMINHO_BASE_LEITURA + "/turmas.csv";
         public static final String SEPARADOR = "-";

         public interface INDICE_ATRIBUTOS {
            public static final int DISCIPLINA = 0;
            public static final int ALOCACOES = DISCIPLINA+1;
         }
      }

   }
  }
}
```

// GerenciadorEntidades.java

```
package br.ufsc.ag.alocacao.professores.uteis;
```

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import br.ufsc.ag.alocacao.professores.modelo.Alocacao;
import br.ufsc.ag.alocacao.professores.modelo.AreaEnsino;
import br.ufsc.ag.alocacao.professores.modelo.Curso;
import br.ufsc.ag.alocacao.professores.modelo.Disciplina;
import br.ufsc.ag.alocacao.professores.modelo.Professor;
import br.ufsc.ag.alocacao.professores.modelo.Turma;

public class GerenciadorEntidades {

   public static final Map<String, Curso> CURSOS;
   public static final Map<String, AreaEnsino> AREAS_ENSINO;
   public static final Map<String, Professor> PROFESSORES;
   public static final Map<String, List<Professor>>
       PROFESSORES_POR_AREA_ENSINO;
   public static final Map<String, Disciplina> DISCIPLINAS;
   public static final Map<String, Disciplina>
       DISCIPLINAS_POR_FASE;
   public static final Map<String, Turma> TURMAS;
   public static final List<Alocacao> ALOCACOES;
   public static final int QUANT_TURMAS;
   public static final int TOTAL_FASES;

   static {
      // carregar entidades
      CURSOS = ArquivoUteis.carregarCursos();
      AREAS_ENSINO = ArquivoUteis.carregarAreasEnsino();
      PROFESSORES =
          ArquivoUteis.carregarProfessores(AREAS_ENSINO);
      PROFESSORES_POR_AREA_ENSINO =
          carregarProfessoresPorAreaEnsino(PROFESSORES.values());
      DISCIPLINAS = ArquivoUteis.carregarDisciplinas(CURSOS,
          AREAS_ENSINO);
      DISCIPLINAS_POR_FASE =
          carregarDisciplinasPorFase(DISCIPLINAS.values());
      ALOCACOES = ArquivoUteis.carregarAlocacoes(DISCIPLINAS);
      TURMAS = carregarTurmas(ALOCACOES);
      QUANT_TURMAS = TURMAS.size();
      TOTAL_FASES =
```

```
        GerenciadorEntidades.calcularTotalFases(CURSOS.values());
}

public Curso getCurso(String codigo) {
   return CURSOS.get(codigo);
}

private static Map<String, Turma>
     carregarTurmas(List<Alocacao> alocacoes) {
   Map<String, Turma> mapaTurmas = new HashMap<>();
   for (Alocacao alocacao : alocacoes) {
      mapaTurmas.put(alocacao.getTurma().getCodigo(),
          alocacao.getTurma());
   }
   return mapaTurmas;
}

private static int calcularTotalFases(Collection<Curso>
     cursos) {
   int total = 0;
   for (Curso curso : cursos) {
      total += curso.getFases();
   }
   return total;
}

private static Map<String, Disciplina>
     carregarDisciplinasPorFase(Collection<Disciplina>
     disciplinas) {
   Map<String, Disciplina> disciplinasPorFase = new
       HashMap<>();
   for (Disciplina disciplina : disciplinas) {
      disciplinasPorFase.put(String.valueOf(disciplina.getFase()),
          disciplina);
   }
   return disciplinasPorFase;
}

public static AreaEnsino getAreaEnsino(String codigo) {
   return AREAS_ENSINO.get(codigo);
}

public static Professor getProfessor(String codigo) {
   return PROFESSORES.get(codigo);
}
```

```
   private static Map<String, List<Professor>>
       carregarProfessoresPorAreaEnsino(Collection<Professor>
       professores) {
      Map<String, List<Professor>> professoresPorFase = new
          HashMap<>();
      for (AreaEnsino areaEnsino : AREAS_ENSINO.values()) {
         professoresPorFase.put(areaEnsino.getCodigo(), new
             ArrayList<>());
      }

      for (Professor professor : professores) {
         AreaEnsino[] areasEnsino = professor.getAreasEnsino();
         for (int i = 0; i < areasEnsino.length; i++) {
            professoresPorFase.get(areasEnsino[i].getCodigo())
                .add(professor);
         }
      }
      return professoresPorFase;
   }

   public static Disciplina getDisciplina(String codigo) {
      return DISCIPLINAS.get(codigo);
   }

   public static Turma getTurma(String codigo) {
      return TURMAS.get(codigo);
   }
}
```

```
// IndividuoUteis.java
package br.ufsc.ag.alocacao.professores.uteis;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Set;

import br.ufsc.ag.alocacao.professores.modelo.Alocacao;
```

```
import br.ufsc.ag.alocacao.professores.modelo.Aptidao;
import br.ufsc.ag.alocacao.professores.modelo.AreaEnsino;
import br.ufsc.ag.alocacao.professores.modelo.Disciplina;
import br.ufsc.ag.alocacao.professores.modelo.Individuo;
import br.ufsc.ag.alocacao.professores.modelo.Professor;
import br.ufsc.ag.alocacao.professores.modelo.Turma;
import br.ufsc.ag.alocacao.professores.uteis.enums.DiaSemanaEnum;
import
    br.ufsc.ag.alocacao.professores.uteis.enums.PeriodoCursoEnum;

public class IndividuoUteis {

   private static final int COEFICIENTE_AVALIACAO = 10000;
   private static final int COEFICIENTE_AVALIACAO_PREFERENCIA =
       500;
   public static final boolean HORARIOS_INFORMADOS;
   private static final Integer MAXIMO_HORARIOS_DISPONIVEIS;
   private static final Integer MAXIMO_HORAS_AULA;

   static {
      int horariosDisponiveis = 0;
      for (int i = 0; i < PeriodoCursoEnum.values().length; i++)
          {
         horariosDisponiveis +=
             PeriodoCursoEnum.values()[i].getHorarios().length;
      }
      MAXIMO_HORARIOS_DISPONIVEIS =
          (GerenciadorEntidades.TOTAL_FASES +
          GerenciadorEntidades.PROFESSORES.size()) *
          horariosDisponiveis * DiaSemanaEnum.values().length;

      int horasAulaParaAlocar = 0;
      for (Turma turma : GerenciadorEntidades.TURMAS.values()) {
         horasAulaParaAlocar +=
             turma.getDisciplina().getHorasAula();
      }
      MAXIMO_HORAS_AULA = horasAulaParaAlocar;
      HORARIOS_INFORMADOS =
          GerenciadorEntidades.ALOCACOES.get(0).getDiaSemana()
          != null;
   }


   public static Individuo gerarIndividuoAleatorio() {
      return new Individuo(gerarAlocacoesAleatorias(
```

```
        IndividuoUteis.MAXIMO_HORAS_AULA));
  }

  public static Alocacao[] gerarAlocacoesAleatorias(int length)
      {
     Alocacao[] alocacoes =
         GerenciadorEntidades.ALOCACOES.toArray(new
         Alocacao[length]);
     for (int i = 0; i < alocacoes.length; i++) {
        Disciplina disciplina =
            alocacoes[i].getTurma().getDisciplina();
        alocacoes[i].setProfessor(gerarProfessorAleatorio(
           disciplina.getAreaEnsino()));
        if(!HORARIOS_INFORMADOS || alocacoes[i].getDiaSemana()
            == null) {
           alocacoes[i].setDiaSemana(gerarDiaSemanaAleatorio());
           alocacoes[i].setHorario(
              gerarHorarioAleatorio(disciplina.getCurso().getPeriodo()));
        }
     }
     return alocacoes;
  }

  public static Professor gerarProfessorAleatorio(AreaEnsino
      areaEnsino) {
     List<Professor> professores =
         GerenciadorEntidades.PROFESSORES_POR_AREA_ENSINO.get(
        areaEnsino.getCodigo());
     return professores.get(new
         Random().nextInt(professores.size()));
  }

  public static Turma gerarTurmaAleatoria() {
     List<Turma> turmas = new
         ArrayList<>(GerenciadorEntidades.TURMAS.values());
     return turmas.get(new Random().nextInt(turmas.size()));
  }

  public static int gerarHorarioAleatorio(PeriodoCursoEnum
      periodoCurso) {
     int[] horariosPossiveis = periodoCurso.getHorarios();
     return horariosPossiveis[new
         Random().nextInt(horariosPossiveis.length)];
  }
```

```
public static int gerarDiaSemanaAleatorio() {
   // sorteia apenas dias da semana (segunda a sexta)
   return 1 + new Random().nextInt(5);
}

public static Aptidao getAptidao(Individuo individuo) {
   if(individuo.getAptidao() == null) {
      individuo.setAptidao(avaliar(individuo.getGenes()));
   }
   return individuo.getAptidao();
}

private static int normalizarAvaliacao(int valor, Integer
    valorMaximo) {
   return normalizarAvaliacao(valor, valorMaximo,
       COEFICIENTE_AVALIACAO);
}
private static int normalizarAvaliacao(int valor, Integer
    valorMaximo, int coeficiente) {
   return new Double((valor / valorMaximo.doubleValue()) *
       coeficiente).intValue();
}

private static Aptidao avaliar(Alocacao[] alocacoes) {
   // TURMAS ALOCADAS
   int scoreTurmasAlocadas = MAXIMO_HORAS_AULA;
   Map<String, Integer> turmasAlocadas = new HashMap<>();

   // CHOQUE HORARIOS
   int scoreChoqueHorarios = MAXIMO_HORARIOS_DISPONIVEIS;
   Map<Integer, Map<String, List<Turma>>>
       horariosAlocadosPorFase = new HashMap<>();
   Map<String, Set<String>> horariosAlocadosPorProfessor =
       new HashMap<>();

   // PROFESSORES POR TURMA
   int scoreProfessoresTurma = MAXIMO_HORAS_AULA;
   Map<String, String> mapProfessoresPorTurma = new
       HashMap<>();


   // AULAS AGRUPADAS
   int scoreAlocacoesAgrupadas = MAXIMO_HORAS_AULA;
   Comparator<Alocacao> comparator = new AlocacaoComparator();
   alocacoes = Arrays.stream(alocacoes).sorted(comparator)
```

```
        .toArray(Alocacao[]::new);

    for (int i = 0; i < alocacoes.length; i++) {
       // TURMAS ALOCADAS
       // registra a turma alocada
       String codigoTurma =
           alocacoes[i].getTurma().getCodigo();
       if(turmasAlocadas.containsKey(codigoTurma)) {
          turmasAlocadas.put(codigoTurma,
              turmasAlocadas.get(codigoTurma) + 1);
       } else {
          turmasAlocadas.put(codigoTurma, 1);
       }

       // CHOQUE HORARIOS
       // registra o horrio alocado na fase

       Integer fase =
           alocacoes[i].getTurma().getDisciplina().getFase();
       Map<String, List<Turma>> horariosFase =
           horariosAlocadosPorFase.get(fase);
       if(horariosFase == null) {
          horariosFase = new HashMap<>();
          horariosAlocadosPorFase.put(fase, horariosFase);
       }
       String horarioKey = alocacoes[i].getDiaSemana() + " " +
           alocacoes[i].getHorario();
       List<Turma> turmasPorHorario =
           horariosFase.get(horarioKey);
       Turma turma = alocacoes[i].getTurma();
       if(turmasPorHorario == null) {
          turmasPorHorario = new ArrayList<>();
          turmasPorHorario.add(turma);
          horariosFase.put(horarioKey, turmasPorHorario);
       } else {
          for (Turma turmaAux : turmasPorHorario) {
             if(!turmaAux.getDisciplina()
                 .equals(turma.getDisciplina())
                   || turmaAux.getCodigo().equals(codigoTurma)) {
                // se   outra turma, da mesma disciplina, no
                    decrementa
                scoreChoqueHorarios--;
             }
          }
          if(!turmasPorHorario.contains(turma)) {
```

```
      turmasPorHorario.add(turma);
   }
}

// registra o horrio alocado do professor
String professor =
    alocacoes[i].getProfessor().getCodigo();
Set<String> horariosProfessor =
    horariosAlocadosPorProfessor.get(professor);
if(horariosProfessor == null) {
   horariosProfessor = new HashSet<>();
   horariosAlocadosPorProfessor.put(professor,
       horariosProfessor);
}
if(!horariosProfessor.add(alocacoes[i].getDiaSemana() +
    " " + alocacoes[i].getHorario())) {
   scoreChoqueHorarios--;
}

// PROFESSORES POR TURMA
String professorDaTurma =
    mapProfessoresPorTurma.get(codigoTurma);
if(professorDaTurma == null) {
   mapProfessoresPorTurma.put(codigoTurma, professor);
} else if(!professorDaTurma.equals(professor)) {
   scoreProfessoresTurma--;
}

// AULAS AGRUPADAS
if(alocacoes[i].getTurma().getDisciplina().getHorasAula()
    > 1) {
   if(i > 0) {
      int comparePrevious =
          comparator.compare(alocacoes[i],
          alocacoes[i-1]);
      if(comparePrevious == 0 ||
           (comparePrevious == 1 &&
               codigoTurma.equals(alocacoes[i-1]
              .getTurma().getCodigo()))) {
         continue;
      }
   }
   if(i < (alocacoes.length-1)) {
      int compareNext = comparator.compare(alocacoes[i],
          alocacoes[i+1]);
```

```
            if(compareNext == 0 ||
                  (compareNext == -1 &&
                       codigoTurma.equals(alocacoes[i+1]
                       .getTurma().getCodigo()))) {
               continue;
            }
          }
          scoreAlocacoesAgrupadas--;
       }
    }

    for (Turma turma : GerenciadorEntidades.TURMAS.values()) {
       Integer alocacoesPorTurma =
           turmasAlocadas.get(turma.getCodigo());
       if(alocacoesPorTurma != null) {
          scoreTurmasAlocadas -=
              Math.abs(turma.getDisciplina().getHorasAula() -
              alocacoesPorTurma);
       }
    }

    // Normaliza scores
    scoreTurmasAlocadas =
        normalizarAvaliacao(scoreTurmasAlocadas,
        MAXIMO_HORAS_AULA);
    scoreChoqueHorarios =
        normalizarAvaliacao(scoreChoqueHorarios,
        MAXIMO_HORARIOS_DISPONIVEIS);
    scoreProfessoresTurma =
        normalizarAvaliacao(scoreProfessoresTurma,
        MAXIMO_HORAS_AULA);
    scoreAlocacoesAgrupadas =
        normalizarAvaliacao(scoreAlocacoesAgrupadas,
        MAXIMO_HORAS_AULA, COEFICIENTE_AVALIACAO_PREFERENCIA);

    return new Aptidao(scoreTurmasAlocadas,
        scoreChoqueHorarios, scoreProfessoresTurma,
        scoreAlocacoesAgrupadas);
 }

 public static boolean isAptidaoObrigatoriaCumprida(Individuo
     individuo) {
    return getAptidao(individuo)
        .isAptidaoObrigatoriaCumprida(COEFICIENTE_AVALIACAO);
 }
```

```
   public static int getAptidaoMaxima() {
      return (3 * COEFICIENTE_AVALIACAO) +
          COEFICIENTE_AVALIACAO_PREFERENCIA;
   }

}
```

APÊNDICE C – Solução gerada em teste com os horários pré-definidos

Fase	Disciplina	Turma	Horas / aula	Horário	Professores	Curso
1	INE5601	INE5601-01238A	4	3.18:30	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5601	INE5601-01238A	4	3.19:20	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5601	INE5601-01238A	4	5.20:20	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5601	INE5601-01238A	4	5.21:10	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5602	INE5602-01238A	4	4.20:20	Joao Candido Lima Dovicchi	238
1	INE5602	INE5602-01238A	4	4.21:10	Joao Candido Lima Dovicchi	238
1	INE5602	INE5602-01238A	4	6.20:20	Joao Candido Lima Dovicchi	238
1	INE5602	INE5602-01238A	4	6.21:10	Joao Candido Lima Dovicchi	238
1	INE5603	INE5603-01238A	6	2.18:30	Antonio Carlos Mariani	238
1	INE5603	INE5603-01238A	6	2.19:20	Antonio Carlos Mariani	238
1	INE5603	INE5603-01238A	6	2.20:20	Antonio Carlos Mariani	238
1	INE5603	INE5603-01238A	6	2.21:10	Antonio Carlos Mariani	238
1	INE5603	INE5603-01238A	6	4.18:30	Antonio Carlos Mariani	238
1	INE5603	INE5603-01238A	6	4.19:20	Antonio Carlos Mariani	238
2	INE5605	INE5605-02238A	6	2.18:30	Jean Carlo Rossa Hauck	238
2	INE5605	INE5605-02238A	6	2.19:20	Jean Carlo Rossa Hauck	238
2	INE5605	INE5605-02238A	6	2.20:20	Jean Carlo Rossa Hauck	238
2	INE5605	INE5605-02238A	6	2.21:10	Jean Carlo Rossa Hauck	238
2	INE5605	INE5605-02238A	6	6.20:20	Jean Carlo Rossa Hauck	238
2	INE5605	INE5605-02238A	6	6.21:10	Jean Carlo Rossa Hauck	238
2	INE5606	INE5606-02238	4	3.20:20	Andréa Cristina Konrath	238
2	INE5606	INE5606-02238	4	3.21:10	Andréa Cristina Konrath	238
2	INE5606	INE5606-02238	4	5.20:20	Andréa Cristina Konrath	238
2	INE5606	INE5606-02238	4	5.21:10	Andréa Cristina Konrath	238
2	INE5607	INE5607-02238A	4	4.18:30	Cristian Koliver	238
2	INE5607	INE5607-02238A	4	4.19:20	Cristian Koliver	238
2	INE5607	INE5607-02238A	4	5.18:30	Cristian Koliver	238
2	INE5607	INE5607-02238A	4	5.19:20	Cristian Koliver	238
3	INE5609	INE5609-03238A	6	4.18:30	Jose Eduardo de Lucca	238
3	INE5609	INE5609-03238A	6	4.19:20	Jose Eduardo de Lucca	238
3	INE5609	INE5609-03238A	6	4.20:20	Jose Eduardo de Lucca	238
3	INE5609	INE5609-03238A	6	4.21:10	Jose Eduardo de Lucca	238
3	INE5609	INE5609-03238A	6	6.18:30	Jose Eduardo de Lucca	238
3	INE5609	INE5609-03238A	6	6.19:20	Jose Eduardo de Lucca	238
3	INE5649	INE5649-03238	4	3.20:20	Pedro Alberto Barbetta	238
3	INE5649	INE5649-03238	4	3.21:10	Pedro Alberto Barbetta	238
3	INE5649	INE5649-03238	4	5.20:20	Pedro Alberto Barbetta	238
3	INE5649	INE5649-03238	4	5.21:10	Pedro Alberto Barbetta	238
4	INE5608	INE5608-04238A	4	2.20:20	Christiane Anneliese Gresse Von Wangenhe	238
4	INE5608	INE5608-04238A	4	2.21:10	Christiane Anneliese Gresse Von Wangenhe	238
4	INE5608	INE5608-04238A	4	4.20:20	Christiane Anneliese Gresse Von Wangenhe	238
4	INE5608	INE5608-04238A	4	4.21:10	Christiane Anneliese Gresse Von Wangenhe	238
4	INE5611	INE5611-04238A	4	3.18:30	Cristian Koliver	238
4	INE5611	INE5611-04238A	4	3.19:20	Cristian Koliver	238
4	INE5611	INE5611-04238A	4	6.18:30	Cristian Koliver	238
4	INE5611	INE5611-04238A	4	6.19:20	Cristian Koliver	238
4	INE5612	INE5612-04238A	4	2.18:30	Aldo Von Wangenheim	238
4	INE5612	INE5612-04238A	4	2.19:20	Aldo Von Wangenheim	238

4	INE5612	INE5612-04238A	4	4.18:30	Aldo Von Wangenheim	238
4	INE5612	INE5612-04238A	4	4.19:20	Aldo Von Wangenheim	238
4	INE5613	INE5613-04238A	4	3.20:20	Renato Fileto	238
4	INE5613	INE5613-04238A	4	3.21:10	Renato Fileto	238
4	INE5613	INE5613-04238A	4	5.20:20	Renato Fileto	238
4	INE5613	INE5613-04238A	4	5.21:10	Renato Fileto	238
4	INE5659	INE5659-05238	4	5.18:30	Jose Eduardo de Lucca	238
4	INE5659	INE5659-05238	4	5.19:20	Jose Eduardo de Lucca	238
4	INE5659	INE5659-05238	4	6.20:20	Jose Eduardo de Lucca	238
4	INE5659	INE5659-05238	4	6.21:10	Jose Eduardo de Lucca	238
5	INE5614	INE5614-05238	4	2.18:30	Ricardo Pereira e Silva	238
5	INE5614	INE5614-05238	4	2.19:20	Ricardo Pereira e Silva	238
5	INE5614	INE5614-05238	4	4.18:30	Ricardo Pereira e Silva	238
5	INE5614	INE5614-05238	4	4.19:20	Ricardo Pereira e Silva	238
5	INE5615	INE5615-05238	4	5.20:20	Carlos Becker Westphall	238
5	INE5615	INE5615-05238	4	5.21:10	Carlos Becker Westphall	238
5	INE5615	INE5615-05238	4	6.18:30	Carlos Becker Westphall	238
5	INE5615	INE5615-05238	4	6.19:20	Carlos Becker Westphall	238
5	INE5616	INE5616-05238	4	3.18:30	Ronaldo dos Santos Mello	238
5	INE5616	INE5616-05238	4	3.19:20	Ronaldo dos Santos Mello	238
5	INE5616	INE5616-05238	4	6.20:20	Ronaldo dos Santos Mello	238
5	INE5616	INE5616-05238	4	6.21:10	Ronaldo dos Santos Mello	238
5	INE5645	INE5645-05238A	4	2.20:20	Cristian Koliver	238
5	INE5645	INE5645-05238A	4	2.21:10	Cristian Koliver	238
5	INE5645	INE5645-05238A	4	4.20:20	Cristian Koliver	238
5	INE5645	INE5645-05238A	4	4.21:10	Cristian Koliver	238
5	INE5646	INE5646-05238A	4	3.20:20	Jean Everson Martina	238
5	INE5646	INE5646-05238A	4	3.21:10	Jean Everson Martina	238
5	INE5646	INE5646-05238A	4	5.18:30	Jean Everson Martina	238
5	INE5646	INE5646-05238A	4	5.19:20	Jean Everson Martina	238
6	INE5600	INE5600-06238	2	5.18:30	Ronaldo dos Santos Mello	238
6	INE5600	INE5600-06238	2	5.19:20	Ronaldo dos Santos Mello	238
6	INE5619	INE5619-06238	4	3.18:30	Carla Merkle Westphall	238
6	INE5619	INE5619-06238	4	3.19:20	Carla Merkle Westphall	238
6	INE5619	INE5619-06238	4	6.18:30	Carla Merkle Westphall	238
6	INE5619	INE5619-06238	4	6.19:20	Carla Merkle Westphall	238
6	INE5621	INE5621-06238	2	6.20:20	Roberto Carlos dos Santos Pacheco	238
6	INE5621	INE5621-06238	2	6.21:10	Roberto Carlos dos Santos Pacheco	238
6	INE5622	INE5622-06238A	4	4.18:30	Elder Rizzon Santos	238
6	INE5622	INE5622-06238A	4	4.19:20	Elder Rizzon Santos	238
6	INE5622	INE5622-06238A	4	5.20:20	Elder Rizzon Santos	238
6	INE5622	INE5622-06238A	4	5.21:10	Elder Rizzon Santos	238
6	INE5624	INE5624-06238	4	2.20:20	Fabiane Barreto Vavassori Benitti	238
6	INE5624	INE5624-06238	4	2.21:10	Fabiane Barreto Vavassori Benitti	238
6	INE5624	INE5624-06238	4	4.20:20	Fabiane Barreto Vavassori Benitti	238
6	INE5624	INE5624-06238	4	4.21:10	Fabiane Barreto Vavassori Benitti	238
6	INE5625	INE5625-06238	4	2.18:30	Cristian Koliver	238
6	INE5625	INE5625-06238	4	2.19:20	Cristian Koliver	238
6	INE5625	INE5625-06238	4	3.20:20	Cristian Koliver	238
6	INE5625	INE5625-06238	4	3.21:10	Cristian Koliver	238

7	INE5617	INE5617-07238	4	3.20:20	Jean Carlo Rossa Hauck	238
7	INE5617	INE5617-07238	4	3.21:10	Jean Carlo Rossa Hauck	238
7	INE5617	INE5617-07238	4	6.18:30	Jean Carlo Rossa Hauck	238
7	INE5617	INE5617-07238	4	6.19:20	Jean Carlo Rossa Hauck	238
7	INE5633	INE5633-07238	4	3.18:30	Jerusa Marchi	238
7	INE5633	INE5633-07238	4	3.19:20	Jerusa Marchi	238
7	INE5633	INE5633-07238	4	4.20:20	Jerusa Marchi	238
7	INE5633	INE5633-07238	4	4.21:10	Jerusa Marchi	238
7	INE5643	INE5643-07238	4	2.20:20	Vania Bogorny	238
7	INE5643	INE5643-07238	4	2.21:10	Vania Bogorny	238
7	INE5643	INE5643-07238	4	4.18:30	Vania Bogorny	238
7	INE5643	INE5643-07238	4	4.19:20	Vania Bogorny	238
7	INE5680	INE5680-07238	4	5.18:30	Carla Merkle Westphall	238
7	INE5680	INE5680-07238	4	5.19:20	Carla Merkle Westphall	238
7	INE5680	INE5680-07238	4	6.20:20	Carla Merkle Westphall	238
7	INE5680	INE5680-07238	4	6.21:10	Carla Merkle Westphall	238
8	INE5644	INE5644-08238	4	3.18:30	Jose Leomar Todesco	238
8	INE5644	INE5644-08238	4	3.19:20	Jose Leomar Todesco	238
8	INE5644	INE5644-08238	4	5.18:30	Jose Leomar Todesco	238
8	INE5644	INE5644-08238	4	5.19:20	Jose Leomar Todesco	238

APÊNDICE D – Solução gerada em teste sem os horários pré-definidos

Fase	Disciplina	Turma	Horas / aula	Horário	Professores	Curso
1	INE5601	INE5601-01238A	4	2.18:30	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5601	INE5601-01238A	4	2.19:20	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5601	INE5601-01238B	4	2.19:20	Ricardo Felipe Custódio	238
1	INE5601	INE5601-01238B	4	2.20:20	Ricardo Felipe Custódio	238
1	INE5603	INE5603-01238A	6	3.20:20	Jean Carlo Rossa Hauck	238
1	INE5603	INE5603-01238A	6	3.21:10	Jean Carlo Rossa Hauck	238
1	INE5601	INE5601-01238A	4	4.18:30	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5601	INE5601-01238A	4	4.19:20	Arthur Ronald de Vallauris Buchsbaum	238
1	INE5603	INE5603-01238A	6	4.20:20	Jean Carlo Rossa Hauck	238
1	INE5603	INE5603-01238A	6	4.21:10	Jean Carlo Rossa Hauck	238
1	INE5603	INE5603-01238A	6	5.18:30	Jean Carlo Rossa Hauck	238
1	INE5603	INE5603-01238A	6	5.19:20	Jean Carlo Rossa Hauck	238
1	INE5602	INE5602-01238A	4	5.20:20	Joao Candido Lima Dovicchi	238
1	INE5602	INE5602-01238A	4	5.21:10	Joao Candido Lima Dovicchi	238
1	INE5602	INE5602-01238A	4	6.18:30	Joao Candido Lima Dovicchi	238
1	INE5602	INE5602-01238A	4	6.19:20	Joao Candido Lima Dovicchi	238
1	INE5601	INE5601-01238B	4	6.20:20	Ricardo Felipe Custódio	238
1	INE5601	INE5601-01238B	4	6.21:10	Ricardo Felipe Custódio	238
2	INE5605	INE5605-02238A	6	2.19:20	Frank Augusto Siqueira	238
2	INE5605	INE5605-02238A	6	2.20:20	Frank Augusto Siqueira	238
2	INE5607	INE5607-02238A	4	3.18:30	Laércio Lima Pilla	238
2	INE5607	INE5607-02238A	4	3.19:20	Laércio Lima Pilla	238
2	INE5607	INE5607-02238A	4	3.20:20	Laércio Lima Pilla	238
2	INE5607	INE5607-02238A	4	3.21:10	Laércio Lima Pilla	238
2	INE5606	INE5606-02238	4	4.18:30	Andréa Cristina Konrath	238
2	INE5606	INE5606-02238	4	4.19:20	Andréa Cristina Konrath	238
2	INE5605	INE5605-02238A	6	5.18:30	Frank Augusto Siqueira	238
2	INE5605	INE5605-02238A	6	5.19:20	Frank Augusto Siqueira	238
2	INE5606	INE5606-02238	4	6.18:30	Andréa Cristina Konrath	238
2	INE5606	INE5606-02238	4	6.19:20	Andréa Cristina Konrath	238
2	INE5605	INE5605-02238A	6	6.20:20	Frank Augusto Siqueira	238
2	INE5605	INE5605-02238A	6	6.21:10	Frank Augusto Siqueira	238
3	INE5649	INE5649-03238	4	3.18:30	Andréa Cristina Konrath	238
3	INE5649	INE5649-03238	4	3.19:20	Andréa Cristina Konrath	238
3	INE5609	INE5609-03238A	6	3.20:20	Alexandre Gonçalves Silva	238
3	INE5609	INE5609-03238A	6	3.21:10	Alexandre Gonçalves Silva	238
3	INE5609	INE5609-03238A	6	4.18:30	Alexandre Gonçalves Silva	238
3	INE5609	INE5609-03238A	6	4.19:20	Alexandre Gonçalves Silva	238
3	INE5649	INE5649-03238	4	4.20:20	Andréa Cristina Konrath	238
3	INE5649	INE5649-03238	4	4.21:10	Andréa Cristina Konrath	238
3	INE5609	INE5609-03238A	6	5.18:30	Alexandre Gonçalves Silva	238
3	INE5609	INE5609-03238A	6	5.19:20	Alexandre Gonçalves Silva	238
4	INE5613	INE5613-04238A	4	2.18:30	Vania Bogorny	238
4	INE5613	INE5613-04238A	4	2.19:20	Vania Bogorny	238
4	INE5608	INE5608-04238A	4	2.20:20	Christiane Anneliese Gresse Von Wangenhe	238
4	INE5608	INE5608-04238A	4	2.21:10	Christiane Anneliese Gresse Von Wangenhe	238
4	INE5608	INE5608-04238A	4	3.18:30	Christiane Anneliese Gresse Von Wangenhe	238

4	INE5608	INE5608-04238A	4	3.19:20	Christiane Anneliese Gresse Von Wangenhe	238
4	INE5659	INE5659-05238	4	3.20:20	Roberto Carlos dos Santos Pacheco	238
4	INE5659	INE5659-05238	4	3.21:10	Roberto Carlos dos Santos Pacheco	238
4	INE5611	INE5611-04238A	4	4.18:30	Mario Antonio Ribeiro Dantas	238
4	INE5611	INE5611-04238A	4	4.19:20	Mario Antonio Ribeiro Dantas	238
4	INE5612	INE5612-04238A	4	4.20:20	Jean Everson Martina	238
4	INE5612	INE5612-04238A	4	4.21:10	Jean Everson Martina	238
4	INE5659	INE5659-05238	4	5.18:30	Roberto Carlos dos Santos Pacheco	238
4	INE5659	INE5659-05238	4	5.19:20	Roberto Carlos dos Santos Pacheco	238
4	INE5611	INE5611-04238A	4	5.20:20	Mario Antonio Ribeiro Dantas	238
4	INE5611	INE5611-04238A	4	5.21:10	Mario Antonio Ribeiro Dantas	238
4	INE5612	INE5612-04238A	4	6.18:30	Jean Everson Martina	238
4	INE5612	INE5612-04238A	4	6.19:20	Jean Everson Martina	238
4	INE5613	INE5613-04238A	4	6.20:20	Vania Bogorny	238
4	INE5613	INE5613-04238A	4	6.21:10	Vania Bogorny	238
5	INE5616	INE5616-05238	4	2.18:30	Renato Fileto	238
5	INE5616	INE5616-05238	4	2.19:20	Renato Fileto	238
5	INE5616	INE5616-05238	4	2.20:20	Renato Fileto	238
5	INE5616	INE5616-05238	4	2.21:10	Renato Fileto	238
5	INE5615	INE5615-05238	4	3.18:30	Carlos Becker Westphall	238
5	INE5615	INE5615-05238	4	3.19:20	Carlos Becker Westphall	238
5	INE5615	INE5615-05238	4	3.20:20	Carlos Becker Westphall	238
5	INE5615	INE5615-05238	4	3.21:10	Carlos Becker Westphall	238
5	INE5614	INE5614-05238	4	4.18:30	Fabiane Barreto Vavassori Benitti	238
5	INE5614	INE5614-05238	4	4.19:20	Fabiane Barreto Vavassori Benitti	238
5	INE5646	INE5646-05238A	4	4.20:20	Frank Augusto Siqueira	238
5	INE5646	INE5646-05238A	4	4.21:10	Frank Augusto Siqueira	238
5	INE5645	INE5645-05238A	4	5.18:30	Cristian Koliver	238
5	INE5645	INE5645-05238A	4	5.19:20	Cristian Koliver	238
5	INE5645	INE5645-05238A	4	5.20:20	Cristian Koliver	238
5	INE5645	INE5645-05238A	4	5.21:10	Cristian Koliver	238
5	INE5646	INE5646-05238A	4	6.18:30	Frank Augusto Siqueira	238
5	INE5646	INE5646-05238A	4	6.19:20	Frank Augusto Siqueira	238
5	INE5614	INE5614-05238	4	6.20:20	Fabiane Barreto Vavassori Benitti	238
5	INE5614	INE5614-05238	4	6.21:10	Fabiane Barreto Vavassori Benitti	238
6	INE5621	INE5621-06238	2	2.18:30	Jose Eduardo de Lucca	238
6	INE5621	INE5621-06238	2	2.19:20	Jose Eduardo de Lucca	238
6	INE5622	INE5622-06238A	4	2.20:20	Ricardo Azambuja Silveira	238
6	INE5622	INE5622-06238A	4	2.21:10	Ricardo Azambuja Silveira	238
6	INE5600	INE5600-06238	2	3.18:30	Vania Bogorny	238
6	INE5600	INE5600-06238	2	3.19:20	Vania Bogorny	238
6	INE5624	INE5624-06238	4	3.20:20	Ricardo Pereira e Silva	238
6	INE5624	INE5624-06238	4	3.21:10	Ricardo Pereira e Silva	238
6	INE5625	INE5625-06238	4	4.18:30	Cristian Koliver	238
6	INE5625	INE5625-06238	4	4.19:20	Cristian Koliver	238
6	INE5619	INE5619-06238	4	4.20:20	Carlos Becker Westphall	238
6	INE5619	INE5619-06238	4	4.21:10	Carlos Becker Westphall	238
6	INE5622	INE5622-06238A	4	5.18:30	Ricardo Azambuja Silveira	238
6	INE5622	INE5622-06238A	4	5.19:20	Ricardo Azambuja Silveira	238
6	INE5624	INE5624-06238	4	5.20:20	Ricardo Pereira e Silva	238

6	INE5624	INE5624-06238	4	5.21:10	Ricardo Pereira e Silva	238
6	INE5625	INE5625-06238	4	6.18:30	Cristian Koliver	238
6	INE5625	INE5625-06238	4	6.19:20	Cristian Koliver	238
6	INE5619	INE5619-06238	4	6.20:20	Carlos Becker Westphall	238
6	INE5619	INE5619-06238	4	6.21:10	Carlos Becker Westphall	238
7	INE5680	INE5680-07238	4	2.19:20	Carlos Becker Westphall	238
7	INE5680	INE5680-07238	4	2.20:20	Carlos Becker Westphall	238
7	INE5617	INE5617-07238	4	3.18:30	Fabiane Barreto Vavassori Benitti	238
7	INE5617	INE5617-07238	4	3.19:20	Fabiane Barreto Vavassori Benitti	238
7	INE5643	INE5643-07238	4	3.20:20	Jose Leomar Todesco	238
7	INE5643	INE5643-07238	4	3.21:10	Jose Leomar Todesco	238
7	INE5617	INE5617-07238	4	4.20:20	Fabiane Barreto Vavassori Benitti	238
7	INE5617	INE5617-07238	4	4.21:10	Fabiane Barreto Vavassori Benitti	238
7	INE5633	INE5633-07238	4	5.18:30	Jerusa Marchi	238
7	INE5633	INE5633-07238	4	5.19:20	Jerusa Marchi	238
7	INE5680	INE5680-07238	4	5.20:20	Carlos Becker Westphall	238
7	INE5680	INE5680-07238	4	5.21:10	Carlos Becker Westphall	238
7	INE5633	INE5633-07238	4	6.18:30	Jerusa Marchi	238
7	INE5633	INE5633-07238	4	6.19:20	Jerusa Marchi	238
7	INE5643	INE5643-07238	4	6.20:20	Jose Leomar Todesco	238
7	INE5643	INE5643-07238	4	6.21:10	Jose Leomar Todesco	238
8	INE5644	INE5644-08238	4	5.20:20	Vania Bogorny	238
8	INE5644	INE5644-08238	4	5.21:10	Vania Bogorny	238
8	INE5644	INE5644-08238	4	6.18:30	Vania Bogorny	238
8	INE5644	INE5644-08238	4	6.19:20	Vania Bogorny	238