UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Laura Mundim de Souza Mendes

Programação de horários de cursos universitários com algoritmo genético

Uberlândia, Brasil

2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Laura Mundim de Souza Mendes

Programação de horários de cursos universitários com algoritmo genético

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Alexsandro Santos Soares

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2014

Laura Mundim de Souza Mendes

Programação de horários de cursos universitários com algoritmo genético

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 04 de setembro de 2014:

Prof. Dr. Alexsandro Santos Soares
Orientador

Prof. Dr. Marcelo Rodrigues de Sousa

Profa. Dra. Márcia Aparecida Fernandes

Uberlândia, Brasil
2014

Agradecimentos

Primeiramente a Deus por ter me dado saúde, sabedoria e força para superar todas as dificuldades que não foram poucas. Sou grata pelo apoio e compreensão dos meus pais e familiares que estiveram sempre ao meu lado me incentivando e principalmente ao meu namorado por ter sido paciente e ter me ouvido mesmo sem entender nada sobre o assunto.

À minha querida mãe por ter me ensinado a não desistir, ter acreditado em mim apesar de todo atraso, sempre ter a certeza que um dia eu chegaria até aqui, hoje concretizo o sonho dela que infelizmente não está mais aqui para poder comemorar esta conquista comigo.

Ao meu orientador Alexsandro por ter se dedicado à elaboração deste trabalho e ter se disposto a me ajudar em cada fase desse processo final da minha graduação.

Meus sinceros agradecimentos a todos que de forma direta ou indireta me ajudaram a cumprir mais esta etapa da minha vida.

"Descobrir consiste em olhar para o que todo mundo está vendo e pensar uma coisa diferente." (Roger Von Oech)

Resumo

Mendes, L.M.S. **Programação de horário de cursos universitários com algoritmo genético**. Curso de Sistemas de Informação, Campus Uberlândia, UFU, Uberlândia, Brasil, 2014, 102p.

Este trabalho apresenta o uso de hibridizações da técnica meta-heurística inspirada no processo de seleção natural – Algoritmo Genético – em implementações de algoritmos que otimizam soluções para problemas de elaboração de horários em cursos universitários. São apresentadas também outras técnicas de otimização e ainda os conceitos envolvidos nos mecanismos que o Algoritmo Genético e suas variações abordam.

Será descrita uma implementação, na linguagem de programação Scala, de diversas hibridizações do Algoritmo Genético para obtenção de soluções para instâncias internacionalmente aceitas, e a partir dos resultados analisar-se uma estrutura de memória que armazene os melhores valores para alguns eventos faz com que a técnica Algoritmo Genético obtenha um melhor desempenho resultando em soluções que violem o menor número de restrições possível.

Palavras-chave: Grade horária, Algoritmo Genético, Problema de otimização

Lista de ilustrações

Lista de Algoritmos

Lista de tabelas

Lista de abreviaturas e siglas

PHCU	Programação de Horários em Cursos Universitários
RS	Recozimento Simulado
BT	Busca Tabu
GRASP	Procedimento de Busca Adaptativa Gulosa e Aleatória
AG	Algoritmo Genético
AGEBG	Algoritmo Genético Estendido com Busca Guiada
AGBG	Algoritmo Genético com Busca Guiada
BL	Busca Local
ABH	Algoritmo de Busca Harmônica
ABHM	Algoritmo de Busca Harmônica Modificado
AGMP	Algoritmo de Grau Maior Ponderado
LANP	Lista de aulas não programadas
ATV	Algoritmo Troca de Vários
AGBasico	Algoritmo Genético Básico
AGBasicoMEM	Algoritmo Genético Básico com Memória
AGAlocador	Algoritmo Genético usando Alocação de Salas
AGAlocadorMem	Algoritmo Genético usando Alocação de Salas com Memória
AGAlocacorBL1	Algoritmo Genético usando Alocação de Salas e uma Busca Local
AGALocadoBL2	Algoritmo Genético usando Alocação de Salas e duas Buscas Locais
AGAlocacorMEMBL1	Algoritmo Genético usando Alocação de Salas com Memória e uma Busca Local
AGAlocacorMEMBL2	Algoritmo Genético usando Alocação de Salas com Memória e e duas Buscas Locais
FACOM	Faculdade de Computação
UFU	Universidade Federal de Uberlândia.

Sumário

1 Introdução

Em todo começo de período, uma instituição acadêmica enfrenta a tarefa de projetar a grade horária de um curso, em que se preocupa com a distribuição de salas, laboratórios, turmas, dias e horários para os professores e alunos. Nesse entremeio, organizar horários de disciplinas em universidades perpassa pela determinação de ementas e cargas horárias necessárias para atender o conteúdo proposto, sendo que cada disciplina utiliza uma carga horária diferenciada que, por sua vez, será ministrada por um professor que pode ou não ministrar outras disciplinas. As cargas horárias de cada disciplina, a disponibilidade do horário de cada professor e o período letivo do curso devem se encaixar, formando, assim, a grade horária de um curso.

Esse problema é conhecido na literatura especializada como Programação de Horários em Cursos Universitários (PHCU) [1] e consiste em alocar eventos nos quais um professor leciona uma disciplina a uma turma em algum lugar e num determinado horário. Esses eventos estão sujeitos a algumas condições e, como será visto adiante, isso torna o problema complexo, pois exige um grande esforço computacional para resolvê-lo.

Existem várias maneiras para encontrar uma solução ótima, ou o mais próximo disto, para o PHCU. Neste trabalho serão apresentadas algumas destas formas e ainda serão realizadas combinações entre algumas técnicas de resolver problemas de otimização com a finalidade de verificar qual técnica obtém melhor desempenho.

1.1 Questão de pesquisa

Hibridizar a meta-heurística Algoritmo Genético com uma estrutura de memória onde armazena os melhores valores para alguns eventos faz com que esta técnica resulte em soluções que violem menos restrições?

1.2 Hipótese

Um sistema automático de elaboração de grades horárias para instituições universitárias que combina a meta-heurística Algoritmo Genético com uma estrutura de memória obtém soluções melhores quando se comparadas com as soluções dos algoritmos que não utilizam a estrutura de memória.

[1] Tradução livre da expressão inglesa *University Course Timetabling Problem* (UCTP)

1.3 Objetivo

O objetivo geral deste trabalho é comparar o desempenho das variações da técnica meta-heurística Algoritmo Genético para a resolução aproximada do problema de programação de horários em cursos universitários. Então desmembrou-se esse objetivo em três etapas:

1. Implementar diferentes variações da técnica meta-heurística Algoritmo Genético, com e sem hibridização usando ou não a estrutura de memória.

2. Comparar e analisar as diferentes combinações de valores possíveis para os parâmetros usados para a construção da estrutura de memória (α, β, γ e tau), usando um mesmo conjunto de testes padronizados e internacionalmente aceitos;

3. Usar o melhor conjunto de parâmetros encontrados no item anterior para comparar e analisar o desempenho das variações do Algoritmo Genético, usando um mesmo conjunto de testes padronizados e internacionalmente aceitos;

4. Confrontar os resultados e discutir as vantagens e as desvantagens da utilização da hibridização e da utilização de estruturas de memória.

1.4 Justificativa

Elaborar a grade horária se torna uma tarefa difícil por causa do grande número de possibilidades e restrições existentes. Devido a isso, o processo de construção manual pode ser demorado, podendo durar horas, dias ou até mesmo semanas e, ainda assim, resultar em grades horárias que não satisfaçam plenamente todas as condições iniciais (CISCON, 2006).

Apesar da existência de ferramentas computacionais comerciais feitas para automatizar a elaboração de grades horárias, tais como Horário Fácil [2]; Urânia [3]; DTS Horario 4.03 [4] e Untis Express [5], e ainda de sistemas com código aberto exemplificados por Times'coll [6]; Tablix [7]; Unitime [8] e FET [9], cada instituição tem suas particularidades, tornando inviável o uso de uma ferramenta generalizada, ou a reutilização de uma implementação específica (BARATA et al., 2010). Segundo Yang e Jat (YANG; JAT, 2011) a

[2] Disponível em: <<http://www.horariofacil.com/>>
[3] Disponível em: <<http://www.horario.com.br/>>
[4] Disponível em: <<http://www.superdownloads.com.br/download/30/dts-horario/>>
[5] Disponível em: <<http://www.programahorario.com/index.php?lang=SP>>
[6] Disponível em: <<http://timescool.lotimiza.com/>>
[7] Disponível em: <<http://www.tablix.org/>>
[8] Disponível em: <<http://www.unitime.org/>>
[9] Disponível em: <<http://lalescu.ro/liviu/fet/>>

solução geral de um determinado problema pode não ser adequada para resolver outro problema devido a certas restrições específicas deste.

O PHCU é um problema que tem sido estudado nas últimas cinco décadas. Um dos primeiros trabalhos sobre programação de horários foi o de Csima e Gotlieb (1964), que propõem um método computacional para a construção de horários envolvendo matrizes booleanas. Segundo esses autores, o problema de programação de horários já estava sendo investigado em muitos países. Desde então, surgiram outras pesquisas relacionadas à programação de horários, como pode ser visto nos trabalhos de Schaerf (1999), Lewis (2007), Paim e Greis (2008) e no de MirHassani e Habibi (2011). Além de ser um problema de interesse prático, o PHCU também é um problema teórico aberto, pois pertence à classe NP-difícil como provado inicialmente por Neufeld e Tartar (1974) e depois completado por Cooper e Kingston (1995), reduzindo-o a problemas NP-difíceis da Teoria de Grafos, como a coloração de vértices.

1.5 Organização do Trabalho

Este trabalho está organizado conforme detalhado na sequência. O capítulo 2 apresenta o problema de alocação de horários e suas variações, traz definições importantes para a compreensão do problema e das restrições que o influencia, faz uma análise da complexidade do problema e ainda explica os problemas de otimização e os dois métodos utilizados para resolvê-lo, heurístico e exato. O capítulo 3 apresenta uma revisão de trabalhos existentes na literatura exemplificando os métodos exatos e as meta-heurísticas. O capítulo 4 apresenta o que foi desenvolvido neste trabalho, tais como as instâncias utilizadas, seus respectivos formatos e ainda a forma que estas foram lidas e interpretadas para estruturar o problema. Neste capítulo também são apresentados elementos constituintes do algoritmo genético, tais como o cromossomo e sua aptidão com as devidas restrições, o indivíduo, a população e os operadores genéticos utilizados pelos algoritmos implementados bem como a diferenciação de cada variação do algoritmo genético. O capítulo 5 descreve os resultados obtidos e os discute. O capítulo 6 traz conclusões e propostas para trabalhos futuros. Adiante são apresentadas as referências bibliográficas consultadas seguidas pelos apêndices onde são apresentados os códigos das classes `Problema`, `Tim`, `Ctt` e `MEM`. Por fim tem-se em anexo com o DTD para as instâncias com formato Ctt.

2 Base teórica

Neste capítulo serão apresentados alguns conceitos fundamentais para melhor entendimento deste trabalho.

2.1 Programação de horários e suas variações

De acordo com Schaerf (1999), o problema de programação de horários, que consiste em alocar eventos sujeitos a algumas restrições, pode ser classificado em:

Problema de Programação de Horários Escolares: este problema está associado à programação de horários em escolas de ensino básico. O problema consiste em um conjunto de classes, formadas por grupos disjuntos de alunos que possuem o mesmo currículo; um conjunto de professores especializados em uma ou mais disciplinas; um conjunto de períodos, em que cada período é alocada uma aula ministrada por um professor de uma disciplina a uma classe, respeitando as várias restrições envolvidas.

Problema de Programação de Horários em Universidades: este problema está associado à programação de horários em universidades. O problema consiste em um conjunto de currículos em que cada currículo possui um conjunto de disciplinas, sendo que cada disciplina tem um conjunto de aulas, um conjunto de estudantes, um conjunto de professores, um conjunto de salas e um conjunto de períodos, em que devem ser alocadas todas as aulas de cada disciplina dos currículos, respeitando diversas restrições. Cada aula é dada por um professor de uma disciplina a um grupo de estudantes em uma sala apropriada, e a principal diferença entre horários escolares e horários universitários é que em escolas de ensino secundário os alunos possuem o mesmo currículo, enquanto que em universidades, os alunos podem escolher livremente disciplinas de currículos diferentes.

Problema de Programação de Horários para Exames: este problema consiste em um conjunto de estudantes e um conjunto de disciplinas em que cada disciplina possui um exame que deve ser programado para os estudantes em um conjunto de períodos de tempo reservados para a realização de exames, evitando os exames de disciplinas aos estudantes em comum que estejam no mesmo período, espalhando, assim, os exames no maior número de dias possível. Esse problema é semelhante ao de programação de horários universitários, porém existem diferenças como, por exemplo, haver apenas um exame para cada disciplina e por dia; além disso, a restrição de conflito é mais rigorosa, uma vez que o estudante não pode perder um exame.

2.2 Restrições sobre o problema de programação de horários

O problema de programação de horários está sujeito a variados tipos de restrições que nem sempre podem ser atendidas. Tais restrições são classificadas de acordo com sua obrigatoriedade, a saber:

Invioláveis: são restrições que obrigatoriamente devem ser satisfeitas, se uma delas for violada, a solução não é aceita. Uma solução que atenda todas as restrições invioláveis é chamada de solução viável.

Preferenciais: são restrições não obrigatórias, porém desejáveis. Uma solução que não obedeça a essas restrições ainda é viável, mas não é tão apropriada como as soluções que as satisfazem.

A quantidade e a natureza das restrições torna o problema de programação de horários mais complexo. Essa complexidade pode ser espacial, relacionada à quantidade total de memória necessário, ou temporal, relacionada ao tempo total de execução.

2.3 Análise da complexidade do PHCU

A complexidade de um problema é a quantidade de tempo consumida pelo melhor algoritmo possível para um determinado problema. Almeida e Ziviani (2004) definem os dois tipos de complexidades temporais em:

Polinomial: problemas que podem ser resolvidos em tempo polinomial. Sua complexidade é representada por $O(p(n))$, onde $p(n)$ é um polinômio. Problemas desse tipo são considerados fáceis e, juntos, formam a classe **P**. Groner (2006) assevera que "P é o conjunto de todos os problemas de decisão (ou linguagens) que aceitam o tempo polinomial no pior caso". *Problemas de decisão* são aqueles problemas que possuem resultados do tipo "sim" ou "não".

Exponencial: são problemas que consomem mais tempo para serem resolvidos. Sua complexidade é representada por $O(c^n)$ com $c > 1$. Problemas desse tipo são considerados difíceis e, juntos, formam a classe **NP**. Problemas da classe NP podem ser verificados em tempo polinomial, sendo fato que P está contido em NP. A questão não resolvida é se P é um subconjunto próprio de NP.

A classe NP possui um subconjunto denominado *NP-completo*. Ainda não foi descoberto nenhum algoritmo que resolva, em tempo polinomial, um problema desse subconjunto, mas também não foi provada a inexistência de tal solução. Uma solução em tempo polinomial, para um determinado problema NP-completo, resolve todos os problemas dessa classe pelo fato de que, se um problema A pode se transformar em um problema B – que tem solução em tempo polinomial –, a solução de B também resolve o problema

A. Isso é chamado de *transformação polinomial*.

Groner (2006) e Badue (2011) definem formalmente que um problema de decisão p é considerado NP-completo se:

1. $p \in$ NP,
2. Todos os problemas pertencentes à classe NP podem ser polinomialmente transformados em p.

Se p satisfaz a segunda condição, então ele é considerado NP-difícil. Sobre esse aspecto, Bello (2007) afirma que problemas NP-difíceis são problemas de otimização combinatória que possuem um problema de decisão NP-completo correspondente. Os problemas NP-completos são problemas de decisão, enquanto problemas de otimização são classificados como sendo NP-difíceis. No entanto, um problema de otimização pode ser polinomialmente transformado em um problema de decisão (ALMEIDA; ZIVIANI, 2004).

Para provar que um problema é NP-completo, é necessário demonstrar que ele pertence à classe NP e mostrar que um problema NP-completo conhecido pode ser polinomialmente transformado para tal problema. Entre os exemplos conhecidos de problemas NP-completos estão o caminho em um grafo, a coloração de um grafo, o ciclo de Hamilton, a cobertura de arestas e vértices, o problema de satisfatibilidade (SAT) e outros (GRONER, 2006).

Neufeld e Tartar (1974) reduziram o problema de elaboração de horários a um problema de coloração de grafos, conhecido como NP-difícil, mostrando assim que o PHCU, mesmo sem restrições, é um problema NP-completo. Posteriormente, Cooper e Kingston (1995) provaram que vários problemas de construção de horários são NP-completos, reduzindo-os a problemas NP-difíceis, como a coloração de vértices e o empacotamento.

Bello (2007) apresenta o problema de programação de grade horária escolar como um problema de coloração de grafos baseado no trabalho de Neufeld e Tartar (1974). O autor reformula o problema acrescentando algumas restrições adicionais, o transforma em problema de coloração de grafos e tenta resolvê-lo usando um algoritmo proposto por ele para resolver problemas de coloração de grafos, além de compará-los com resultados obtidos por uma meta-heurística, que será explicada posteriormente, busca tabu. Em seus experimentos, observou que seu algoritmo comparado com busca tabu apresenta bom desempenho para grafos pequenos e médios. Quanto ao tempo de execução, seu algoritmo gasta um tempo consideravelmente menor em todas as instâncias, principalmente nas maiores.

Algoritmos para resolver problemas NP-completos são de complexidade exponencial em relação ao tamanho da entrada, então, como não é conhecido um melhor algoritmo

que o resolva em tempo polinomial, são utilizadas algumas técnicas de otimização na tentativa de encontrar a solução aproximada para o Problema de Programação de Horários em Cursos Universitários.

2.4 Problemas de Otimização

Os problemas de otimização são aqueles que buscam minimizar ou maximizar uma função-objetivo sujeita a um conjunto de restrições, a fim de encontrar a melhor solução dentro de um conjunto de soluções possíveis. Um problema de otimização pode ser modelado matematicamente da seguinte forma:

$$\text{Otimizar } z = f(x_1, x_2, ..., x_n)$$

$$\text{Sujeito a } \begin{cases} g_1(x_1, x_2, ..., x_n) \leq v_1 \\ g_2(x_1, x_2, ..., x_n) = v_2 \\ \qquad\qquad\qquad\qquad ... \\ g_n(x_1, x_2, ..., x_n) \geq v_n \\ x_1, x_2, ..., x_n >= 0 \end{cases},$$

onde $x_1, x_2, ..., x_n$ são as variáveis de decisão, z é a função-objetivo e $g_1, g_2, ..., g_n$ são as funções de restrições.

Para que seja considerado um problema de programação linear, a função-objetivo z e as funções de restrições $g_1, g_2, ..., g_n$ devem ser funções do primeiro grau, ou seja, devem possuir a seguinte estrutura:

$$c_1x_1 + c_2x_2 + ... + c_nx_n,$$

onde $c_1, c_2, ..., c_n$ são constantes que acompanham as variáveis de decisão.

Segundo Goes (2005), para tratar dos problemas de otimização pode-se utilizar dois tipos de métodos:

Método exato: sempre encontra a solução ótima, a que satisfaz todas as restrições do problema, caso exista.

Método heurístico: encontra uma solução, porém não garante ser a ótima. Heurísticos são métodos aproximados que se baseiam nas características das soluções dos problemas – sua complexidade computacional é menor que a de métodos exatos (RIBEIRO, 2007).

Técnicas heurísticas não necessariamente encontram a solução ótima para um determinado problema, mas são capazes de gerar resultados de boa qualidade em tempo razoável (SILVA; SAMPAIO; ALVARENGA, 2005; MARTINS, 2010; PORTO, 2010; GOES, 2005; CISCON, 2006; AL-BETAR; KHADER, 2012). São importantes devido ao bom desempenho quando aplicadas a problemas NP-completos (SUCUPIRA, 2004).

Os métodos heurísticos são muito específicos, fazendo surgir a necessidade de técnicas mais genéricas como as *meta-heurísticas*, que são mais abrangentes e podem ser aplicadas a uma classe maior de problemas. As meta-heurísticas podem incluir várias heurísticas que tentam solucionar o problema de maneira eficiente e podem ser aplicadas a problemas pertencentes à classe dos problemas NP-completos (GOES, 2005).

Ribeiro (2007) apresenta e explica algumas das meta-heurísticas mais conhecidas, tais como: Recozimento Simulado [1], Busca Tabu, Procedimento de Busca Adaptativa Gulosa e Aleatória [2], conhecido como GRASP, Algoritmos Genéticos e Colônias de Formigas. Miagkikh (2012) apresenta e compara algumas meta-heurísticas populares incluindo uma nova, o algoritmo de Busca Harmônica. Já Gomide (2012) apresenta técnicas exatas como o algoritmo simplex e o *branch-and-bound*. Essas técnicas serão explicadas posteriormente.

[1] Tradução livre de Simulated Annealing (SA)
[2] Tradução livre do termo Greedy Randomized Adaptive Search Procedure (GRASP)

3 Revisão Bibliográfica

Há diferentes maneiras de resolver problemas de otimização. Neste capítulo serão descritas algumas técnicas exatas e heurísticas, sendo também relacionados os trabalhos presentes na literatura que as executaram em diferentes problemas.

3.1 Métodos exatos

A partir da modelagem matemática de um problema de programação linear, pode-se utilizar um método exato para a sua resolução. Dos métodos exatos existentes, tem-se o simplex e o *branch-and-bound*.

3.1.1 Método simplex

O método simplex é um procedimento matricial, desenvolvido por George Dantzig em 1947, utilizado para encontrar a melhor solução de problemas de programação linear. O trabalho de Spendley, Hext e Himsworth (1962) foi um dos pioneiros nesse assunto e nele os autores apresentam tal método mostrando as situações nas quais ele deve ser utilizado. Bona et al. (2000) usaram um simplex diferenciado, com iniciação automática, em um aplicativo para otimização e, de acordo com eles, "[...] o simplex é uma figura regular que se desloca sobre uma superfície, de modo a evitar regiões de resposta não satisfatória". Segundo Escarpinati (2012) este método possui as seguintes fases:

Fase 1: Normalização do modelo

Nesta fase serão feitas algumas modificações a fim de que sejam eliminados os sinais de desigualdade em todas as restrições. Para isso, são utilizadas as variáveis de folga (f), quando a restrição possuir o sinal ($\leq$), as variáveis de excesso (e), quando a restrição possuir o sinal ($\geq$), e as variáveis artificiais (a), quando a restrição tiver o sinal ($\geq$) ou o sinal ($=$). Ao acrescentar tais variáveis nas restrições, estas também devem aparecer na função-objetivo da seguinte forma: variáveis de folga são acrescentadas com coeficiente 0; variáveis de excesso são subtraídas com coeficiente 0; e variáveis artificiais são subtraídas (ou acrescentadas, no caso do problema de minimização) com coeficiente M. Considerando este modelo:

$$\text{Maximizar } z = px_1 + qx_2$$

$$\text{Sujeito a } \begin{cases} x_1 + 2x_2 \leq Y \\ 2x_1 + 3x_2 \geq W \\ x_1 + x_2 = K \end{cases}$$

Tem-se o seguinte resultado normalizado:

$$\text{Maximizar } z = px_1 + qx_2 + 0f_1 - 0e_1 - Ma_1 - Ma_2$$

$$\text{Sujeito a } \begin{cases} x_1 + 2x_2 + f_1 = Y \\ 2x_1 + 3x_2 - e_1 + a_1 = W \\ x_1 + x_2 + a_2 = K \end{cases}$$

Fase 2: Montagem do quadro simplex

O quadro simplex é feito a partir do problema normalizado. Suas linhas e colunas são formadas da seguinte forma: a primeira linha é constituída pelas variáveis da função-objetivo, em que cada variável é uma coluna do corpo do quadro simplex e a segunda linha é formada pelos coeficientes das variáveis da primeira linha; na primeira coluna, antes do corpo do quadro, ficam as variáveis de folga e artificiais; na segunda coluna ficam os coeficientes que acompanham as variáveis da primeira coluna; e a última coluna é formada pelos valores que ficam após a igualdade das funções das restrições.

O corpo do quadro é preenchido com os coeficientes que acompanham as variáveis da primeira linha nas funções de restrições. A última linha é calculada da seguinte maneira: cada elemento da segunda coluna é multiplicado pelo elemento da coluna que está calculando em sua linha; os valores dessas multiplicações são somados; e subtrai-se o elemento da segunda linha da coluna que deseja calcular – isso para problemas de maximização. Para problemas de minimização, faz-se o valor do elemento da segunda linha subtraído da soma das multiplicações, como feito anteriormente. Na Tabela 1 é exemplificado o preenchimento do quadro simplex do exemplo anterior.

Tabela 1 – Exemplo quadro simplex

		x1	x2	f1	e1	a1	a2	
		p	q	0	0	-M	-M	
f1	0	1	2	1	0	0	0	Y
a1	-M	2	3	0	-1	1	0	W
a2	-M	1	1	0	0	0	1	K
		-3M-p	-4M-q	0	M	0	0	-WM-KM

A última linha da Tabela 1 é calculada baseada na fórmula: Max $= [(0 \times 1) + (-M \times 2) + (-M \times 1)] - p$; se fosse um problema de minimização a fórmula seria: Min $= p - [(0 \times 1) + (-M \times 2) + (-M \times 1)]$

Fase 3: Resolução do quadro simplex Com o quadro simplex montado, parte-se então para os procedimentos de resolução deste, que no modo mais simples consiste em:

1. Ao desconsiderar a última coluna do quadro, encontrar o número mais negativo da última linha (a coluna desse número é chamada de coluna de trabalho). Se existir mais de um elemento candidato, ou seja, com o mesmo valor, escolher um deles.

2. Ao desconsiderar a última linha, dividir os elementos da última coluna pelos elementos na mesma linha da coluna de trabalho. Dos resultados das divisões, o elemento da coluna de trabalho que resultou no menor número positivo é chamado de pivô. Se existir mais de um candidato, escolher um deles e, caso não haja nenhum resultado positivo, o problema não tem solução.

3. Fazer uma operação em todos os elementos da linha do pivô a fim de torná-lo igual a zero. Em seguida, fazer outras operações nas demais linhas no intuito de tornar os demais elementos da coluna de trabalho iguais a zero..

4. Trocar a variável da primeira coluna na linha do pivô com a variável da primeira linha na coluna do pivô.

5. Se não existir números negativos na última linha, o algoritmo é encerrado. Caso contrário repetir os passos de 1 a 4 até a inexistência de números negativos na última linha.

6. Quando o algoritmo é encerrado, o valor de cada variável da primeira coluna é atribuído ao valor da última coluna em sua respectiva linha. As variáveis que não estiverem na primeira coluna recebem zero.

3.1.2 Método *branch-and-bound*

O método *branch-and-bound*, que pode ser traduzido como "ramifica e limita", consiste em ramificar (*branch*) o conjunto de soluções possíveis de um determinado problema de programação linear, construindo subconjuntos disjuntos e determinando limites (*bound*) para o valor ótimo da solução. Na representação do algoritmo usa-se uma árvore de enumeração na qual são estabelecidos limites para podar os ramos da árvore que não contém a solução ótima - o objetivo é ramificar os nós da árvore até que fiquem amadurecidos e encontrar a solução ótima para o problema. Um nó é amadurecido quando não há solução, chega em um limite ou quando é considerado ótimo (foi proposto por A. H. Land e A. G. Doig em 1960). O procedimento deste método é representado na Figura 1.

3.1.3 Trabalhos que usam métodos exatos

Ferreira et al. (2011) formularam matematicamente o problema de distribuição de encargos didáticos, desenvolvendo um sistema no qual os professores informam suas preferências e disponibilidades e, com esses dados, o sistema gera as matrizes do modelo matemático. Os autores propuseram dois métodos, um usando uma formulação básica matemática e outro que tenta melhorar essa formulação acrescentando uma segunda fase que, após a primeira, que minimiza o número de turmas sem professores e adiciona ao modelo básico uma restrição na qual se diz que o total de turmas sem preferências para

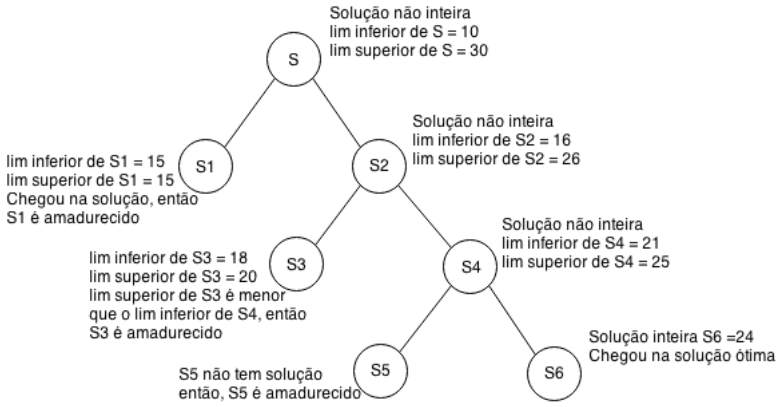

Figura 1 – Exemplo do método branch-and-bound em um problema de maximização

um professor seja N. Segundo os autores, essa segunda fase maximiza a satisfação global dos professores.

Em seus experimentos, os autores supramencionados usaram uma fórmula que expressa a satisfação média dos professores, em que resultados obtidos por meio do método de duas fases tiveram satisfação média um pouco menor do que os da formulação básica. Porém, a formulação básica alocou um número maior de turmas sem preferências para os professores, apesar de gastar menos tempo que o método de duas fases. Diante dos resultados, eles optaram pelo método de duas fases, que minimiza o número de insatisfações dos professores.

Scarpin, Steiner e Andrade (2012) usaram a programação linear binária para gerar a grade horária do curso de Engenharia de Produção da Universidade Federal do Paraná. Os autores pesquisaram a estrutura do curso para saber o que influenciava na grade horária e fizeram um sistema no qual esses dados são cadastrados e, posteriormente, gera-se a grade horária utilizando a modelagem matemática feita pelos autores e atendendo todas as restrições. No modelo proposto, a função-objetivo era de minimização e incluía os pesos das variáveis que representavam a preferência de horários de professores, em que cada peso varia de 1 a 10, sendo os menores valores para os horários de maior preferência. Em seus experimentos, conseguiram uma grade, com tempo de processamento de 42 segundos, que atendeu todas as restrições e, principalmente, respeitou o peso de cada uma delas. Ao final do trabalho, concluiu-se que o uso de tal modelo seria viável para o curso supracitado.

3.2 Meta-heurísticas

Segundo Abensur e Oliveira (2012), heurísticas são "[...] procedimentos de busca intuitivos desenvolvidos para a obtenção de soluções de problemas que não possuam solução matemática exata". Já as meta-heurísticas, para Bello (2007) "[...] são abordagens heurísticas de caráter mais genérico, possibilitando sua aplicação na resolução de dife-

rentes tipos de problemas de otimização, com poucas adaptações". Das meta-heurísticas existentes serão aqui explicadas: Recozimento Simulado, Busca Tabu, GRASP, Algoritmo Genético e Algoritmo de Busca Harmônica.

3.2.1 Recozimento Simulado (RS)

É uma meta-heurística inspirada na subárea da física que estuda causas e efeitos de mudanças de temperatura e de outras grandezas: a termodinâmica. Essa técnica simula um processo térmico no qual acontece um aquecimento, até uma determinada temperatura, de um metal e, em seguida, ele é resfriado lentamente a fim de alterar sua estrutura cristalina e melhorar suas características. Foi inicialmente proposta por Kirkpatrick, Jr. e Vecchi (1983) e, de acordo com Silva, Sampaio e Alvarenga (2005), é um método de busca local que consegue escapar de ótimos locais por meio de movimentos de piora.

De acordo com Poulsen (2012), o algoritmo clássico do RS melhora uma solução inicial existente, sendo que a ideia do algoritmo é perturbar a solução atual, gerando, assim, uma nova solução S durante um número de execuções. A cada iteração, o custo de S é comparado ao custo da solução corrente e, se o custo de S é menor que o custo da solução corrente, S passa a ser a solução corrente. S ainda é comparada com a melhor solução existente; se o custo de S for menor do que o custo da melhor solução, S passa a ser a melhor solução até então e, caso o custo de S seja maior que o custo da solução corrente, o algoritmo sorteia um número entre 0 e 1 e compara o valor de x, sendo $x = e^{\Delta/T}$, onde Δ é a diferença entre o custo de S e o custo da solução corrente, e T é a temperatura atual. Se o valor sorteado for menor que o valor de x, o algoritmo assume S como solução corrente, mesmo sendo mais custosa - isso acontece enquanto a temperatura inicial diminui gradativamente até atingir a temperatura mínima.

O Algoritmo 3.1 representa o pseudocódigo do algoritmo do recozimento simulado retirado do trabalho de Silva, Sampaio e Alvarenga (2005).

Poulsen (2012) tratou o problema de horários escolares com essa meta-heurística e Silva, Sampaio e Alvarenga (2005) apresentaram uma solução para o Problema de Alocação de Salas também utilizando o Recozimento Simulado. Tais autores usaram o algoritmo clássico, e o que os diferencia é a maneira de gerar a solução inicial. Além disso, eles trazem resultados que incentivam o uso de tal meta-heurística: Silva, Sampaio e Alvarenga (2005) conseguiram eliminar os casos em que o número de alunos era maior que a capacidade da sala, e Poulsen (2012) observou que as grades geradas foram de qualidade similar àquela usada pela escola no ano anterior, que é considerada de ótima qualidade.

Algoritmo 3.1: Pseudocódigo do Recozimento Simulado Básico (RS)

Seja s_0 uma solução inicial, T_0 a temperatura inicial, α a taxa de resfriamento e $Iter_{max}$ o número máximo de iterações para atingir o equilíbrio.
$s \leftarrow s_0$ //Solução Corrente
$s' \leftarrow s$; //Melhor solução obtida até então
$T \leftarrow T_0$ //Temperatura Corrente
$IterT \leftarrow 0$ //Número de iterações na temperatura T
enquanto $(T > 0)$ **faça**
  enquanto $(IterT < Iter_{max})$**faça**
    $IterT \leftarrow IterT + 1$;
    Gere um vizinho qualquer $s' \in N(s)$
    $\Delta = f(s') - f(s)$
    se $(\Delta < 0)$ **então**
      $s \leftarrow s'$;
      se $(f(s') < f(s^*)$ **então**
        $s^* \leftarrow s$; //Melhor solução
    fim-se
    senão
      Tome $x \in [0, 1]$;
      se$(x < e^{-\Delta/T})$ **então**
        $s \leftarrow s'$;
    fim-senão
  fim-enquanto
$T \leftarrow \alpha \times T$;
$IterT \leftarrow 0$;
fim-enquanto
Retorne s^*;

3.2.2 Busca Tabu (BT)

Busca Tabu se assemelha ao Recozimento Simulado pelo fato de admitir os movimentos de piora; é uma meta-heurística que orienta (guia) o algoritmo de busca a encontrar a melhor solução global, evitando passar por melhores locais já visitados. Para isso, faz o uso de uma memória de curto prazo, chamada de lista tabu, onde são armazenados os melhores locais encontrados, os quais são considerados proibidos. Essa lista tabu possui um tamanho t, chamado de período tabu; logo, são armazenadas as t mais recentes soluções encontradas. Ao encontrar uma nova solução a ser inserida na lista, a mais antiga é removida e, para que um local pertencente à lista tabu seja aceito, este deve satisfazer o critério de aspiração, ou seja, ser melhor que a melhor solução encontrada até então. Em outras palavras, o algoritmo de Busca Tabu segue supondo que não é interessante se mover em direção a um local ruim, ou seja, busca o melhor movimento possível, exceto quando a intenção é escapar de locais proibidos (GLOVER, 1986).

Tal meta-heurística foi inicialmente apresentada por Glover (1977), que propôs uma heurística de restrição para obter os substitutos próximos - essa é a ideia inicial do

algoritmo de Busca Tabu. No trabalho de Glover (1986), BT é considerado como sinônimo da primeira abordagem.

O algoritmo de Busca Tabu parte de um solução inicial s_0. Com o intuito de melhorá-la, o algoritmo busca a cada iteração em um subconjunto v da vizinhança V da solução corrente $V(s)$; então, ele seleciona um vizinho de s em v, o s', que tenha o menor valor em v, que passa a ser a nova solução corrente mesmo que s' não seja melhor que s. E a melhor solução encontrada até então - $s*$ - é armazenada na lista tabu (isso acontece por um número determinado de iterações).

O Algoritmo 3.2 representa o pseudocódigo do algoritmo de Busca Tabu retirado do trabalho de Subramanian et al. (2011).

Algoritmo 3.2: Pseudocódigo do Busca Tabu Básico (BT)

$s_0 \leftarrow$; Solução inicial
$s* \leftarrow s_0$; //Melhor solução obtida até então
$s \leftarrow s_0$;
$IterT \leftarrow 0$; //Contador do número de iterações
$MelhorIter \leftarrow 0$; //Iteração mais recente que forneceu $s*$
Seja $BTmax$ o número de iterações sem melhora em $s*$;
$T \leftarrow \emptyset$ //Lista Tabu
enquanto $(IterT - MelhorIter \leq BTmaxeIterT < IterMax)$ **faça**
  $IterT \leftarrow IterT + 1$;
  Seja $s' \leftarrow s \oplus m$ o melhor elemento de $v \subseteq V(s)$ tal que o movimento m não seja tabu ($m \notin T$) ou se tabu, atenda ao critério de aspiração, isto é, $f(s') < f(s*)$
  Atualiza lista tabu;
  $s \leftarrow s'$
  se $(f(s') < f(s*))$ **então**
    $s* \leftarrow s'$;
    $MelhorIter \leftarrow IterT$;
  fim-se
fim-enquanto
Retorne $s*$;

Subramanian et al. (2011) aplicaram essa meta-heurística para o problema de alocação de salas em universidades. Os autores usaram o algoritmo de Busca Tabu básico, porém utilizaram uma heurística de construção para gerar a solução inicial a fim de melhorá-la com o algoritmo de Busca Tabu. Segundo os autores, o BT mostrou ser eficiente, pois obteve resultados de alta qualidade com baixo custo computacional.

Burke, Kendall e Soubeiga (2003) aplicaram o algoritmo de Busca Tabu em uma hiper-heurística para o problema de horários escolares e escala de serviço de enfermeiros. Uma hiper-heurística é composta por várias heurísticas de baixo nível, em que uma delas é escolhida para resolver um determinado problema. Nesse diapasão, o algoritmo de Busca Tabu é usado na escolha da heurística de baixo nível, procurando o caminho certo nas

condições certas e, ainda, armazenando as heurísticas que não obtiveram, recentemente, bons resultados na lista tabu. Segundo os autores, essa técnica é reutilizável, exige pouco tempo de desenvolvimento e produz soluções, ao menos, aceitáveis.

3.2.3 Procedimento de Busca Adaptativa Gulosa e Aleatória (GRASP)

É uma meta-heurística interativa em que cada iteração consiste basicamente numa fase de construção: uma solução é construída de maneira gulosa e aleatória; e uma fase de busca local: busca o melhor local na vizinhança da solução construída com o intuito de atualizá-la por uma solução melhor encontrada - após um número determinado de iterações é retornada a melhor solução obtida.

Chvatal (1939) propôs uma heurística gulosa para resolver o problema de cobertura, posteriormente, Feo e Resende (1989) melhoraram seu algoritmo incluindo a randomização, tornando-a semigulosa e surgindo, assim, o GRASP. Nesse último trabalho fizeram uma comparação com a abordagem proposta por Chvatal e constaram que o algoritmo de Feo e Rezende, que usa o algoritmo semiguloso, produz soluções mínimas - o que não aconteceu no primeiro trabalho -, e ainda, a nova abordagem apresentou melhores resultados.

Segundo Nogueira (2008), a cada iteração de uma solução gulosa é selecionado um candidato que melhore a função-objetivo, enquanto que, numa solução semigulosa, o algoritmo guloso cria uma lista de candidatos restritos onde são armazenados os t melhores candidatos - um deles é selecionado aleatoriamente para ser adicionado à solução atual. Ainda segundo. Ainda de acordo com o mesmo autor, uma busca local a partir de uma solução semigulosa resulta em soluções mais variadas do que a busca local a partir de soluções gulosas. O tamanho da lista de candidatos restritos t e o critério de parada, que pode ser o número de iterações ou tempo, são parâmetros de ajuste do algoritmo que devem ser pré-estabelecidos.

O foco dessa técnica está na fase de construção da solução inicial que, de acordo com Moura, Silveira e Santos (2004), ao aplicá-la ao problema de PHCU, pode ser similar à construção de grades horárias manual, em que se agendam primeiramente as aulas mais difíceis de serem alocadas nos períodos mais críticos, violando o mínimo possível de restrições. A dificuldade de uma aula está associada a uma função de penalidades que considera as restrições e a criticidade de um período, sendo devido ao número de indisponibilidades para tal fator.

O Algoritmo 3.3 representa o pseudocódigo do algoritmo GRASP apresentado por Resende (2012).

Moura e Scaraficci (2007) aplicam o algoritmo GRASP clássico seguido de uma melhoria chamada de religação de caminho, em que são exploradas as soluções em uma

Algoritmo 3.3: Pseudocódigo do GRASP Básico

```
    f(s*) ← *∞
    para (i = 1, ..., N) faça
3:      Construir uma solução s usando um algoritmo guloso aleatorizado
        Aplicar um procedimento de busca local a partir de s, obtendo a solução s'
        se (f(s') < f(s*)) então
6:          s* ← s';
    fim-para
    Retorne s*;
```

trajetória entre duas boas soluções de acordo com um critério específico, em um problema de programação de horários escolares. Os autores mostram que o GRASP melhorado com outra técnica de busca local – religação de caminho – tem probabilidade maior de encontrar uma boa solução.

Layeb e Chenche (2012) abordaram uma nova meta-heurística baseada no GRASP para o problema de empacotamento. Na fase de construção do algoritmo usaram um algoritmo híbrido ganancioso que escolhe aleatoriamente, a cada construção de uma solução, se vai usar o método primeiro da lista, o qual será empacotado, ou o melhor da lista, em que procura o melhor em uma lista para o empacotar – isso oferece grande diversidade ao GRASP. Na fase de busca local usaram o algoritmo de Busca Tabu, sendo que a combinação das duas técnicas chegou a ótimos resultados, porém com um tempo maior. Os autores explicam que a eficiência da abordagem é devida à diversificação na construção e à intensificação na busca local.

3.2.4 Algoritmo Genético (AG)

Esta meta-heurística é baseada no processo de seleção natural proposto por Charles Darwin, no qual os mais aptos sobrevivem. Cada solução é chamada de indivíduo de uma população e a função-objetivo, de função de aptidão. Para Martins (2010), essa técnica aborda dois mecanismos:

Seleção: mecanismo em que os mais aptos em relação ao hábitat têm maiores chances de sobreviver e deixar descendentes.

Variação: mecanismo no qual são gerados novos indivíduos por meio da reprodução, em que há mistura de material genético ou mutações imprevisíveis e são criadas novas informações genéticas.

O algoritmo genético mais comum parte de uma população inicial de indivíduos gerada aleatoriamente. Na evolução da população, no processo de seleção, cada indivíduo é avaliado pela função de aptidão com o intuito de selecionar os mais aptos; assim tem-se uma nova população com os melhores indivíduos que vão passar pelo processo de variação,

em que os indivíduos se reproduzem e sofrem mutações para gerar novos indivíduos com outras características. Essa nova população com mais indivíduos passa novamente pelo processo de seleção e variação até atingir um critério de parada, que pode ser um número máximo de iterações ou um limite de tempo.

Lucas (2000) discute em seu trabalho cada etapa do algoritmo genético e as possibilidades de operações em cada uma delas assim como se segue:

Inicialização: etapa na qual é gerada a população inicial que passará pelas demais etapas. Geralmente é construída de maneira simples e aleatória, mas, para melhores desempenhos do algoritmo, pode-se usar outras heurísticas.

Avaliação: gerada a população inicial, começa o processo de seleção. Para selecionar os melhores indivíduos é necessário avaliá-los para reconhecê-los; então, para cada indivíduo é calculado seu grau de aptidão segundo a função de aptidão, também conhecida como função-objetivo. Normalmente essa função é baseada em penalidades, isto é, quanto maior o grau de aptidão, maior é o número de penalidades existentes nessa solução; logo, os indivíduos com menor grau de aptidão são os melhores.

Seleção: etapa na qual serão selecionados os indivíduos que irão se reproduzir. Ocorre um sorteio em que os indivíduos com o menor grau de aptidão têm maior chance de serem escolhidos. Existem diferentes formas de selecionar os indivíduos, como, por exemplo, a seleção por posição, em que os indivíduos são ordenados pelo grau de aptidão e os p indivíduos que estiverem em melhores posições, conforme o tamanho da população, são escolhidos; a seleção por giro de roleta, na qual cada indivíduo ocupa uma porção da roleta proporcional ao seu grau de aptidão, sendo que a roleta é girada algumas vezes conforme o tamanho da população e em cada giro um indivíduo é selecionado; a seleção por torneio, em que são escolhidos grupos de indivíduos e os mais aptos de cada grupo são selecionados; e a seleção uniforme, em que todos os indivíduos têm a mesma chance de serem selecionados.

Reprodução: os indivíduos selecionados terão uma probabilidade pré-definida de passar pelo processo de cruzamento, no qual partes dos genes de dois indivíduos (pai e mãe) são combinadas para gerar novos indivíduos (filhos). O cruzamento entre dois indivíduos pode ser: cruzamento de um ponto, em que são divididos os cromossomos dos pais em um ponto aleatório (a primeira parte do cromossomo do pai e a segunda parte do cromossomo da mãe são copiadas, formando um filho) e, em seguida copia-se a primeira parte do cromossomo da mãe e a segunda parte do cromossomo do pai formando um segundo filho; cruzamento por dois pontos, que é semelhante ao cruzamento de um ponto, porém são escolhidos aleatoriamente dois pontos de corte e são gerados dois filhos copiando as partes dos cromossomos dos pais e trocando somente a parte entre os dois pontos selecionados; cruzamento multiponto, no qual são sorteados n pontos de corte, sendo que n é um número fixo e não pode ser maior que o tamanho do cromossomo

em que ocorrem n mudanças em sua sequência; cruzamento segmentado, semelhante ao cruzamento multiponto mas que, nesse caso, n não é fixo; cruzamento uniforme, no qual os genes escolhidos para serem copiados para os filhos são sorteados entre os genes do pai ou da mãe; e cruzamento por combinação parcial, em que são sorteados dois pontos de corte e os filhos recebem todos os genes do pai ou da mãe, sendo que a parte entre os pontos de corte é preenchida com valores considerados adequados para cada filho.

Depois do processo de cruzamento os novos indivíduos gerados passam, com uma probabilidade pré-definida, pelo processo de mutação, em que seus cromossomos serão modificados. Das opções de mutação tem-se, conforme Lucas (2000), a mutação aleatória, na qual cada gene que será modificado recebe um valor qualquer dentre os possíveis; a mutação por troca, em que são sorteados n pares de genes que serão modificados, e cada par de genes troca de valor entre si; e a mutação de deformação, na qual o valor do gene é deformado, aumentando ou diminuindo um valor dele.

Atualização: após a etapa de reprodução, os novos indivíduos precisam ser agregados à população, porém, no AG clássico, a população se mantém com um tamanho fixo, como a cada dois indivíduos pais são gerados dois indivíduos filhos – normalmente os filhos substituem os pais, mantendo o tamanho da população. Mas como apresentado por Lucas (2000), existem alternativas como: gerar menos indivíduos, variar o tamanho da população, o critério de inserção pode ser outro (como os filhos que só substituem os pais, caso sejam mais aptos ou simplesmente mantém os n indivíduos mais aptos).

Finalização: etapa em que ocorre a verificação do critério de parada que pode ser, por exemplo, o número de iterações ou até mesmo o tempo. Caso o critério de parada seja alcançado, a execução é terminada e devolve-se o melhor indivíduo encontrado.

O Algoritmo 3.4 representa o pseudocódigo do algoritmo genético.

Algoritmo 3.4: Pseudocódigo do Algoritmo Genético Básico (AG)

```
Seleção
  Gerar população inicial com n indivíduos;
  Avaliar aptidão de cada indivíduo;
enquanto A condição de parada não for satisfeita
  Repita até que a nova população esteja completa
    Selecionar os pais da geração;
  Variação
    Aplicar o cruzamento para gerar os filhos;
    Aplicar mutação nos filhos;
    Avaliar aptidão dos filhos;
    Inserir os novos descendentes na nova população;
    geração atual = nova população;
```

Hamawaki et al. (2005) e Skrabe (2007) modelaram o AG clássico para o problema de geração de horários em universidades. No primeiro, o AG foi executado e testado, sendo

que os resultados obtidos indicam o bom desempenho do algoritmo. Ciscon (2006) também usou o AG clássico, porém, para resolver o problema de alocação de horários escolares com foco em eliminação de janelas, notou a qualidade do algoritmo, relatando que a diretoria e os professores da escola gostaram dos resultados alcançados.

Já Porto (2010) realizou uma análise comparativa entre Algoritmos Genéticos com outra meta-heurística, o Algoritmo de Seleção Clonal, para definir a técnica que trata o problema de alocação de quadros de horários com melhor desempenho. Segundo o autor, o AG obteve melhores soluções em menor tempo, enquanto o algoritmo de seleção clonal tende a melhores resultados em tempo maior.

Lucas (2000) implementou um aplicativo experimental para a resolução de um problema de grade horária para medir alguns índices de desempenho entre várias técnicas, e concluiu que Algoritmos Genéticos não apresentam um bom desempenho. Segundo o autor, para resolver tal problema, o uso de heurísticas nos operadores genéticos tem sido a solução mais usada.

Em 2009, Jat e Yang (2009) propuseram um Algoritmo Genético com Busca Guiada (AGBG) que usa uma técnica de busca local para o problema de programação de horários em cursos universitários. A estratégia de busca guiada é usada para gerar filhos com base em uma estrutura de dados denominada MEM, que armazena informações de bons indivíduos, e a técnica de busca local é usada nas melhorias dos indivíduos. Alguns anos depois, Yang e Jat (2011) procuraram melhorar essa abordagem e a estenderam incluindo outra estratégia de busca local, o algoritmo passou a ser chamado de Algoritmo Genético Estendido com Busca Guiada (AGEBG). Esse algoritmo será apresentado com mais detalhes adiante, pois é de interesse neste projeto implementá-lo.

Algoritmo Genético Estendido com Busca Guiada para PHCU

Para compreender o Algoritmo Genético Estendido com Busca Guiada é necessário saber antes como funciona o Algoritmo Genético com Busca Guiada. O AGBG, de forma geral, parte de uma solução inicial de indivíduos gerados aleatoriamente, ou seja, os eventos são alocados em quaisquer horários, enquanto as salas são atribuídas com o uso de um algoritmo de correspondência. Depois de construída a população inicial, cada indivíduo desta passa por um método de busca local (BL), sendo que, nas próximas gerações, os filhos podem ser gerados utilizando-se uma estrutura de dados MEM ou por um cruzamento que usa o método de torneio. Nesse algoritmo, na fase de reprodução é gerado apenas um filho que passa por um processo de mutação e que depois é melhorado pelo BL1; finalmente, o pior indivíduo da população é substituído pelo filho gerado. Esses procedimentos acontecem até que um critério de parada seja alcançado (JAT; YANG, 2009).

O algoritmo de busca local mencionado acima usa três estruturas de vizinhanças

(V) que realizam operações nos eventos mudando seus horários. Segundo Yang e Jat (2011), são elas:

- V1: é defina por uma operação em que se move um evento para um horário diferente;
- V2: é definida por uma operação em que ocorre a troca de horários entre dois eventos;
- V3: é definida por uma operação em que ocorre a troca de horários entre três eventos.

O pseudocódigo do BL1 é apresentado no Algoritmo 3.5. Ele é baseado no algoritmo apresentado por Yang e Jat (2011), assim como todos os algoritmos referentes ao AGEBG. Segundo os autores, BL1 funciona em duas etapas, a saber:

1. Verificam-se as violações de restrições invioláveis em cada evento alocado; caso ocorra alguma, BL1 tenta resolvê-las por meio das operações em V1, V2 e V3 até que alcance algum critério de parada. Depois de cada operação realizada, é usado um algoritmo de correspondência nos horários afetados pela operação, no intuito de resolver complicações geradas, sendo que, em seguida, avalia-se o resultado. Se todos os movimentos de vizinhança foram aplicados em todos os eventos – e ainda exista pelo menos uma violação de restrição inviolável –, o BL1 vai parar; se não, BL1 executará a segunda etapa;
2. O primeiro passo retorna uma solução viável, que passa pela segunda etapa, na qual se lida com as restrições preferenciais de forma parecida com a etapa 1. Para cada evento, BL1 realiza as operações V1, V2 e V3 na tentativa de diminuir o número de violações de restrições preferenciais, isso sem violar restrições invioláveis. Para cada operação realizada, aplica-se o algoritmo de correspondência para resolver complicações geradas e o resultado é avaliado. Após as etapas de BL1, tem-se uma solução possivelmente melhor.

No AGBG, depois de inicializada a população, constrói-se uma estrutura de dados MEM a cada geração τ, que é usada para guiar a geração de filhos. MEM é uma lista de dois níveis, em que o primeiro nível é uma lista de eventos sem violações de restrições e o segundo, uma lista de pares de sala e horário (s, h) de cada evento do primeiro nível. O uso dessa estrutura tem por objetivo melhorar a solução, reintroduzindo melhores indivíduos de outras gerações. A estrutura de MEM é ilustrada na Figura 2 retirada do trabalho de Yang e Jat (2011).

Para a construção da estrutura MEM, é selecionado um conjunto Q dos $\alpha \times N$ melhores indivíduos da população, onde α é um percentual e N é o tamanho da população. Em seguida, para cada indivíduo $q \in Q$ é verificado o valor de penalidade de cada evento; se essa penalidade é zero, então, a informação deste é armazenada na MEM e, se ainda

Algoritmo 3.5: Pseudocódigo do Algoritmo de Busca Local (BL1)

Entrada: Indivíduo I da população
para cada evento $e \in E$ **faça**
  se e_i é inviável **então**
    se existe um movimento deixado **então**
      aplicar os movimentos: primeiro V1, depois V2 se V1 falhar e finalmente V3 se V1 e V2 falharem
      aplicar o algoritmo de correspondência para os horários afetados pelas operações e para alocar salas para os eventos
      avalia o resultado do movimento
      se as operações reduziram as restrições invioláveis **então**
        fazer os movimentos e ir para linha 4
      fim-se
    fim-se
  fim-se
fim-para
se nenhuma restrição inviolável **então**
  para cada evento $e \in E$ **faça**
    se e_i possui violações de restrições preferenciais **então**
      se existe um movimento deixado **então**
        aplicar os movimentos: primeiro V1, depois V2 se V1 falhar e finalmente V3 se V1 e V2 falharem
        aplicar o algoritmo de correspondência para os horários afetados pelas operações e para alocar salas para os eventos
        avalia o resultado do movimento
        se as operações reduziram as restrições preferenciais **então**
          fazer os movimentos e ir para linha 17
        fim-se
      fim-se
    fim-se
  fim-para
fim-se
Saída: Indivíduo I possivelmente melhor

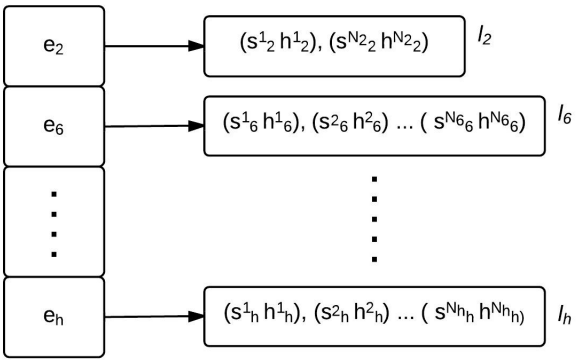

Figura 2 – Estrutura MEM

Fonte: Adaptada de Yang e Jat (2011)

não existe uma lista de pares para esse evento na MEM, então é criada uma lista para ele. O Algoritmo 3.6 mostra o procedimento do método para a construção da estrutura MEM.

Algoritmo 3.6: Pseudocódigo do ConstruirMEM()

```
Entrada: toda população P
Classificar a população P de acordo com a aptidão dos indivíduos
Q ← seleção dos α × N melhores indivíduos de P
para cada indivíduo q_i ∈ Q faça
  para cada evento e_i ∈ q_i faça
    calcular penalidade de e_i
    se e_i é viável então
      adicionar o par de sala e horário (s,h) atribuído em e_i na lista l_i
    fim-se
  fim-para
fim-para
Saída: Estrutura de dados MEM
```

Para exemplificar numericamente a MEM, considere uma população p de 10 indivíduos constituídos por 20 eventos que podem ser alocados em 15 horários possíveis e 2 salas. Se $\alpha = 0.3$, então Q será um conjunto com os três melhores indivíduos de p, ou seja, $Q = \{i_1, i_2, i_3\}$. Suponha-se que ao avaliar os eventos dos indivíduos em Q obteve-se os resultados apresentados na Tabela 2 em que os eventos listados alocados nas salas e horários mencionados entre parênteses não violam nenhuma restrição.

Tabela 2 – Salas e horários dos eventos avaliados cuja penalidade é zero

Indivíduo	Evento (Sala e Horário)			
i_1	$e_2(1, 5)$	$e_5(0, 4)$	$e_{11}(1, 7)$	$e_{15}(0, 11)$
i_2	$e_5(0, 6)$	$e_{10}(1, 3)$	$e_{15}(0, 10)$	
i_3	$e_1(1, 3)$	$e_2(0, 4)$	$e_5(1, 4)$	

Com os dados mencionados neste exemplo constrói-se uma MEM conforme estrutura ilustrada na Figura 3

Em cada geração do algoritmo, um filho pode ser gerado usando a estrutura MEM, usando o método *ConstruçãoGuiada()* mostrado no Algoritmo 3.7, ou pelo cruzamento comum, usando o método *Cruzamento()* mostrado no Algoritmo 3.8 (isso depende da probabilidade γ). Antes da criação de um filho, um número aleatório entre 0 e 1 é gerado, sendo que, se $\rho < \gamma$, o filho será gerado usando a estrutura MEM, se não, será gerado com um operador de cruzamento. Para gerar um filho usando MEM, primeiro é selecionado um conjunto E de $\beta \times n$ eventos, onde β é um percentual e n é o número total de eventos. Em seguida, para cada evento $e \in E$, é escolhido um par de salas e horários (s, h) na lista

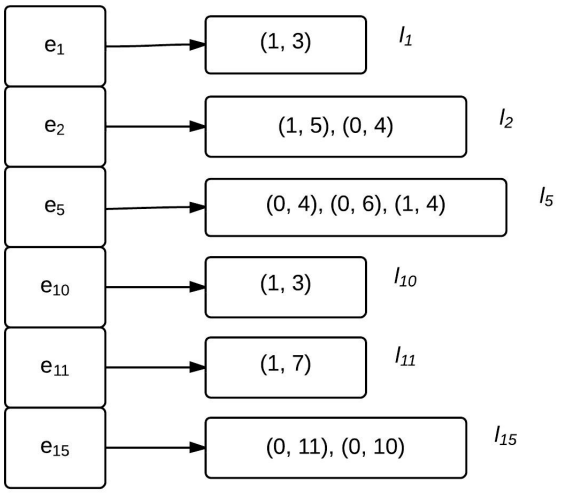

Figura 3 – Exemplo numérico da MEM

de pares de e na MEM; caso e não tenha uma lista de pares (s, h) na MEM, então são atribuídos qualquer sala e qualquer horário para esse evento.

Para os outros eventos que não estão em E, as salas e os horários são atribuídos aleatoriamente e, para gerar um filho por cruzamento, primeiro são selecionados, pelo método de torneio, dois indivíduos pais, em que um deles é selecionado aleatoriamente e o horário do filho é atribuído ao horário do pai escolhido; em seguida aplica-se o algoritmo de correspondência para alocar as salas. Depois que o filho foi gerado, este passa pela mutação com uma probabilidade de P_m e pelas operações de BL1; logo, o pior indivíduo é substituído pelo filho gerado - isso ocorre até que um critério de parada seja atingido.

Algoritmo 3.7: Pseudocódigo do Construção Guiada()

Entrada: estrutura de dados MEM
$E \leftarrow$ seleção aleatória dos $\beta \times n$ eventos
para cada evento $e_i \in E$ **faça**
  Selecionar aleatoriamente um par (s, h) da lista l_i
  Atribuir o par (s, h) do evento e_i para o filho
fim-para
para cada evento restante $e_i \notin E$ **faça**
  atribuir qualquer horário e qualquer sala para evento e_i
fim-para
Saída: Novo filho gerado usando a estrutura de dados MEM

O algoritmo apresentado até então e representado no Algoritmo 3.9 é o Algoritmo Genético de Busca Guiada, estudado por Yang e Jat (2011) no intuito de estendê-lo, acrescentando um novo método de busca local denominado BL2. BL2 é aplicado após BL1 na solução aleatória da população inicial, bem como após a geração do filho - o objetivo do BL2 é melhorar ainda mais o indivíduo que passa por BL1.

Algoritmo 3.8: Pseudocódigo do Cruzamento()

Entrada: população corrente
Selecionar os pais $P1$ e $P2$ pelo método de torneio
para cada evento e_i do filho F **faça**
  se $(rand(0,1) < 0.5)$ **faça**
    e_i de $F \leftarrow$ o horário alocado para $P1$
  senão
    e_i de $F \leftarrow$ o horário alocado para $P2$
  fim-se
fim-para
alocar salas para todos os horários alocados usando o algoritmo de correspondência
Saída: Novo filho gerado usando cruzamento

Algoritmo 3.9: Pseudocódigo do Algoritmo Genético com Busca Guiada (AGBG)

Entrada: uma instancia do problema I
Gerar população inicial aleatoriamente
Avaliar os indivíduos da população
Aplicar busca local BS1 para cada indivíduo da população
Definir contador de geração $g \leftarrow 0$
enquanto A condição de parada não for satisfeita **faça**
  se $(g \bmod \tau == 0)$ **então**
    aplicar ConstruirMEM() para construir MEM
  fim-se
  gerar filho usando MEM ou cruzamento com probabilidade γ
  Aplicar mutação no filho gerado com probabilidade P_m
  Aplicar busca local BS1 sobre o filho
  Substituir o pior indivíduo da população pelo filho gerado
  $g \leftarrow g + 1$
fim-enquanto
Saída: Melhor solução para a instancia I

O algoritmo BL2 seleciona um percentual pré-determinado de horários a partir do número total de horários T, formando um conjunto S de horários. Em seguida, é calculada a penalidade de cada horário selecionado e é escolhido o horário com maior penalidade. Depois, BL2 aplica a operação de V1 nos eventos com maiores penalidades e verifica se tal penalidade diminui com a operação - se sim, aplica-se a operação nos demais; caso contrário, não se aplica a operação. É esperado que, com o uso de BL2, a solução fique ainda melhor. O Algoritmo 3.10 mostra os procedimentos de BL2.

Yang e Jat (2011) compararam o desempenho do AGEBG com outros algoritmos. Eles postulam que os parâmetros α, que é a porcentagem dos melhores indivíduos selecionados a partir da população atual para a criação da MEM; β, que é o valor de porcentagem do número total de eventos que são usados para gerar o filho por meio da MEM; γ, que é a probabilidade que indica se o filho será gerado usando a MEM ou pelo cruzamento; e τ,

Algoritmo 3.10: Pseudocódigo do Algoritmo de Busca Local (BL2)

Entrada: indivíduo I que passou por BL1
$S \leftarrow$ seleção aleatória de uma porcentagem do total dos horários T.
para cada horário $t_i \in S$ **faça**
  para cada evento j no horário t_i **faça**
    calcular penalidade do evento j
  fim-para
  soma todas as penalidades dos eventos no horário t_i
fim-para
seleciona os horários de S com maiores penalidades w
para cada evento $i \in w$ **faça**
  aplicar em i a operação de N1
  aplicar o algoritmo de correspondência nos eventos com horários afetados pela operação
fim-para
se a operação reduziu o número de violações de restrições **então**
  Aplica as operações
senão
  Não aplica as operações
fim-se
Saída: Indivíduo I possivelmente melhor

que decide a frequência da atualização da MEM, afetam diretamente o desempenho dos algoritmos. Por causa disto, os autores testaram os algoritmos com diferentes valores dos parâmetros e concluíram que o AGEBG produziu boas soluções devido ao uso da MEM, visto que os piores resultados encontrado pelo AGEBG são melhores que os resultados obtidos por outros algoritmos comparados.

3.2.5 Algoritmo de Busca Harmônica (ABH)

Esta meta-heurística é baseada na improvisação musical, em que músicos (variáveis de decisão) tocam notas em seus instrumentos (valor da variável) buscando a harmonia perfeita de acordo com uma estimativa estética, ou seja, buscando a melhor solução em consonância com a função-objetivo. Prática após prática, os músicos ficam mais experientes, pois armazenam em suas memórias as notas das melhores harmonias; assim, aumenta-se a possibilidade de melhorar a harmonia nas próximas práticas – ela foi inicialmente proposta por Geem, Kim e Loganathan (2001).

Na improvisação musical, os músicos podem optar entre três maneiras de improvisar a nota: improvisar qualquer nota armazenada em sua memória, ajustar (modificar) uma nota armazenada em sua memória ou improvisar qualquer nota dentro das possíveis.

Ao considerar um grupo de músicos composto por um violinista, um saxofonista e um tecladista, cada um possui inicialmente em sua memória (Ré, Mi, Dó), (Fá, Sol

Lá) e (Lá, Lá, Mi) respectivamente. As notas são armazenadas na memória de maneira ordenada, em que a melhor nota fica na primeira posição (essa estrutura da memória harmônica é ilustrada na Figura 4).

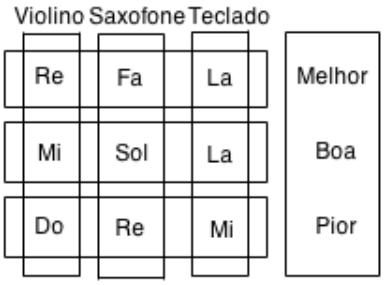

Figura 4 – Estrutura da memória harmônica.

Fonte: Adaptada de Geem, Kim e Loganathan (2001)

Na improvisação musical cada músico toca uma nota, por exemplo, o violinista toca (Dó) escolhido a partir de sua memória (Ré, Mi, Dó), o saxofonista ajusta (Ré) para chegar a (Mi) e o tecladista toca (Sol) dentro nas notas possíveis (Dó, Ré, Mi, Fá, Sol, Si), formando a nova harmonia (Dó, Mi, Sol). Essa nova harmonia passa pela avaliação do padrão de estética: como a nova harmonia (Dó, Mi, Sol) é melhor que a boa harmonia da memória harmônica (Mi, Sol, Lá), então, a pior harmonia é excluída da memória harmônica, a boa harmonia passa ser a pior e a nova harmonia é inserida como boa. Esta atualização da memória harmônica é ilustrada na Figura 5.

O funcionamento do algoritmo de busca harmônica representado no Algoritmo 3.11 adaptado de Al-Betar e Khader (2012), consiste em basicamente quatro fases:

1. **Inicializar parâmetros e memória harmônica**

   O ABH usa alguns parâmetros, tais como: Memória Harmônica (MH), em que são armazenadas as TMHs melhores harmonias (TMH é o tamanho da MH); Taxa de Consideração de Memória (TCM), usada para decidir a maneira de improvisar a

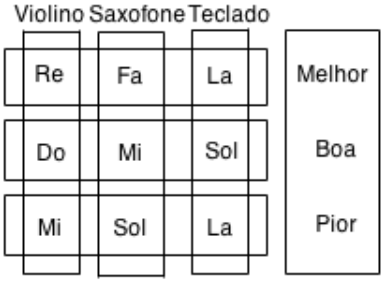

Figura 5 – Estrutura da memória harmônica atualizada.

Fonte: Adaptada de Geem, Kim e Loganathan (2001)

nota; Taxa de Ajuste de Nota (TAN), usada para modificar as notas na improvisação; e Número de Iterações (NI). A MH é um vetor de soluções que pode ser representado como:

$$MH = [S_1, S_2, ..., S_i, ..., S_{TMH}]$$

Nessa fase são definidos os valores dos parâmetros TMH, TCM, TAN, NI e é inicializada a MH com TMHs harmonias geradas aleatoriamente..

2. **Improvisar nova harmonia**

   Nesta fase, os músicos improvisam uma nova harmonia usando os três operadores: consideração de memória, ajuste de nota e aleatório.

   Com uma probabilidade de TCM, onde $0 \leq TCM \leq 1$, será utilizado o operador de consideração de memória, em que o músico escolhe um valor qualquer armazenado em sua memória. Com uma probabilidade de $1 - TCM$, será utilizado o operador aleatório, no qual o músico escolhe um valor qualquer dentro do conjunto de notas possíveis. Cada nota da nova harmonia que foi obtida por meio do operador de consideração de memória deve ser avaliada para decidir se deve modificá-la ou não. Com uma probabilidade de TAN, onde $0 \leq TAN \leq 1$ a nota será modificada, com probabilidade $1-TAN$ a nota não será modificada e com probabilidade $TAN \times PAR$ a nota é modificada para o seu vizinho.

   Resumindo tem-se:

$$\text{Operador} = \begin{cases} 0 \leq TCM \leq 1, & \text{consideração de memória} \\ 1 - TCM, & \text{aleatório} \\ 0 \leq TAN \leq 1, & \text{nota será modificada} \\ 1 - TAN, & \text{nota não será modificada} \\ TAN \times TCM, & \text{nota será modificada para seu vizinho} \end{cases}$$

3. **Atualização da memória harmônica**

   Improvisada uma nova harmonia, esta é comparada com as harmonias existentes na MH. Se a nova harmonia é melhor que uma das harmonias na MH, então, a pior harmonia em HM é excluída e a nova harmonia é inserida na MH de forma ordenada e de acordo com a harmonia.

4. **Verifica critério de parada**

   Depois da atualização da MH, verifica-se se o critério de parada foi alcançado, termina-se execução e retorna-se à melhor harmonia na MH, caso contrário, os passos 3 e 4 são repetidos até que o critério de parada seja satisfeito.

Na primeira implementação do ABH feita por Geem, Kim e Loganathan (2001), aplicaram o algoritmo com sucesso em três aplicações diferentes: caixeiro viajante, um

Algoritmo 3.11: Pseudocódigo do Algoritmo de Busca Harmônica (ABH)

Fase 1: Inicializar parâmetros e memória harmônica
TMH ← valor1
TCM ← valor2
TAN ← valor3
NI← valor4
MH ← $s^1, s^2, ..., s^{TMH}$

Fase 2: Improvisar nova harmonia $s_{nova} = [x^1, x^2, ..., x^n]$
$x' = \emptyset$ // vetor nova harmonia
para $(i = 1, ...n)$
se (rand ≤ TCM)
$x'_i \in \{x^i, x^i, ..., x_i^{TMH}\}$ //Escolhe um valor aleatório da MH para o componente x'_i
se (rand ≤ TAN)
$x'_i = v_{i,k\pm m}$ // $x'_i = v_{i,k}$ ajustar o valor do componente x_i
fim-se
senão
$x'_i \in X_i$ Escolhe um valor aleatório para o componente x_i
fim-para
retorna s_{nova}

Fase 3: Atualização da MH
se $(f(s_{nova}) < f(s_{TMH}))$
Inclui s_{nova} em MH
Exclui s_{TMH} de MH

Fase 4: Verifica critério de parada
enquanto(NI não for alcançado)
Repete as fases 2 e 3

problema de minimização simples com restrições e uma rede de gasodutos de abastecimento de água, garantindo, assim, que o algoritmo de busca harmônica é capaz de resolver problemas de variáveis contínuas, bem como problemas combinatórios. Os autores compararam ABH com AG e RS em um problema de rede, sendo que ABH resolveu o problema com um número consideravelmente menor de iterações em relação ao AG e ao RS.

Leea e Geemb (2005) usaram o ABH clássico para vários problemas de otimização em engenharia, em que obteve melhores resultados do que algoritmos matemáticos e abordagens baseadas em AG.

Segundo Das et al. (2011), o ABH tem sido usado com sucesso para vários problemas de otimização. Os autores fizeram uma análise matemática sobre o mecanismo de busca exploratória do ABH e modelaram uma pequena modificação no passo de improvisação: ao invés de improvisar uma nova harmonia, eles improvisam uma população de novas harmonias usando os mesmos operadores do ABH básico; tal abordagem produz

resultados mais precisos em quase todas as instâncias testadas em menor tempo computacional, comparando com outras variações do ABH da literatura.

Al-Betar e Khader (2012) aplicaram um algoritmo de busca harmônica modificado, que será explicado a seguir, para o problema e programação de horários em cursos universitários.

Algoritmo de Busca Harmônica Modificado (ABHM)

Nesta seção será explicada detalhadamente a abordagem de Al-Betar e Khader (2012) que aplicaram o algoritmo de busca harmônica comum e um Algoritmo de Busca Harmônica Modificado (ABHM) ao problema de programação de horários em cursos universitários. Todos os pseudocódigos apresentados foram adaptados a partir do trabalho desses autores, os quais utilizaram as seguintes equivalências de termos de otimização com o contexto musical:

Improvisação	$\leftrightarrow$	Geração ou construção
Harmonia	$\leftrightarrow$	Vetor solução
Músico	$\leftrightarrow$	Decisão
Nota	$\leftrightarrow$	Valor
Tessitura	$\leftrightarrow$	Faixa de valor
Padrão áudio-estético	$\leftrightarrow$	Função objetivo
Prática	$\leftrightarrow$	Iteração
Harmonia agradável	$\leftrightarrow$	Solução quase ideal

Na fase de inicialização, os autores aplicaram um algoritmo de retrocesso e o Algoritmo Troca de Vários (ATV) [1] para gerar soluções aleatórias, além de alocarem as aulas usando o Algoritmo de Grau Maior Ponderado (AGMP) [2] que começa tal função com o maior número de restrições. As aulas que não forem possíveis de serem alocadas com GMP são introduzidas em uma ista denominada Lista de Aulas não Programadas (LANP), que passa pelo algoritmo de retrocesso que a cada aula $a_i \in$ LANP seleciona todas as demais aulas que têm alunos em comum com a_i e as retira da solução; em seguida, as insere na LANP e tenta alocar o maior número possível de aulas pertencentes a LANP - isso acontece até alocar todas as aulas da lista e, se não for possível alocá-las, usa-se o ATV em todos os horários, embaralhando as aulas de um mesmo horário em salas diferentes (se mesmo assim não conseguir gerar uma solução viável, o processo é reiniciado). O Algoritmo 3.12 mostra o funcionamento do processo de construção da MH.

Na improvisação musical, cujo procedimento é representado no Algoritmo 3.13, modificou-se a funcionalidade do operador de consideração de memória, escolhendo sempre

[1] Tradução livre de MultiSwap
[2] Tradução livre de Weighted Largest Degree (WLD)

Algoritmo 3.12: Pseudocódigo do Algoritmo de Inicialização da MH

```
para (i = 1, ...TMH) faça
  x' = ∅
  AGMP(x')
  enquanto (x' não está completa || alcance o número máximo de iterações) faça
    Retrocesso(x')
    Troca de Vários (x')
  fim-enquanto
  insere x' na MH
  calcular f(x')
fim-para
```

a nota da melhor solução na MH. Foi modificada também, a funcionalidade do operador de ajuste de nota, em que somente as aulas programadas com o operador de consideração de memória são examinadas pelo operador de ajuste de nota. Dividiu-se a TAN em três parâmetros (TAN1, TAN2, TAN3), em que cada um controla o processo de ajustamento da seguinte forma:

$$\text{Operação} = \begin{cases} 0 \leq TAN \leq 1, & \text{a nota será modificada} \\ 0 \leq TAN1 & \text{, a aula é movida para qualquer local livre possível} \\ TAN1 \leq TAN2, & \text{troca de local com outra aula agendada} \\ TAN2 \leq TAN3, & \text{selecionam-se todas as aulas desse horário e as trocam} \\ & \text{com todas as aulas de outro horário sem alterar as salas} \\ TAN3 \leq 1, & \text{não faz nada.} \end{cases}$$

Os ajustes mencionados anteriormente são aceitos com a condição de que o valor da função-objetivo da nova solução improvisada não seja afetado de forma negativa.

Os autores incluíram um processo de reparação para agendar as aulas ainda não programadas que estão em uma lista. Esse processo é interativo e funciona da seguinte forma: seleciona-se uma aula ainda não programada a da lista; em seguida, procuram-se todos os locais possíveis para a; dentre esses locais que já estão ocupados por outras aulas, escolhe-se a melhor localização em termos de função-objetivo para a; exclui-se a aula que está nesse local e a inclui na lista de aulas não programadas, aloca-se a nesse local e exclui a da LANP. Tal processo tenta encontrar bons locais para todas as aulas da lista de aulas não programadas; caso esse processo falhe, reinicia-se o processo de improvisação.

Em seus experimentos, os autores concluíram que quanto maior o TMH e a TAN, maior será o tempo de execução do algoritmo. Na comparação entre o ABH comum e o ABH modificado. eles notaram que o ABH comum consegue encontrar soluções viáveis para instâncias de pequeno e médio porte, mas não para as de grande porte. Já o ABH modificado conseguiu encontrar soluções viáveis para todas as instâncias testadas; todavia, o tempo de execução do ABH comum é menor do que o tempo de execução do ABH modificado, pois este usa a função-objetivo em cada iteração para aceitar os ajustes.

Algoritmo 3.13: Pseudocódigo do Algoritmo de Improvisação da Nova Solução

$x' = \emptyset$
para $(i = 1, ...n)$ **faça**
  $x'_i = menorposição$
  se $(rand \leq TCM)$
    $x'_i = x_i^j$, onde $(x_i^j \in B^{melhor})$
    se $(rand \leq TAN1)$
      Move aula(x'_i)
    senão se$(rand \leq TAN2)$
      Move sala(x'_i)
    senão se$(rand \leq TAN3)$
      Move horário(x'_i)
  fim-se
13: **senão**
    $x'_i \in Q_i$, onde $Q_i = \{l | l$é livre para$x'_i, l \in [0, P \times K - 1]\}$
fim-para
processo de reparação

3.2.6 Outras meta-heurísticas

Das técnicas possíveis na resolução de problemas de otimização, existem trabalhos que as modelam, executam e comparam, como as pesquisas de Goes (2005) e de Abensur e Oliveira (2012).

Goes (2005) fez uma comparação entre algoritmos exato, heurístico e misto para o problema de horários escolares. A técnica heurística usada pelo autor foi o Algoritmo Genético, sendo que o método misto utiliza um método exato para a construção da solução inicial e uma heurística para o melhoramento desta. O autor relata os seguintes resultados: o método exato foi gerado em 1 minuto 43 segundos, obtendo a solução ótima; o método heurístico foi gerado em 53 segundos, com boas soluções; e o método misto foi gerado em 19 segundos, com solução quase ótima. É notável que o tempo para obter a solução ótima é aceitável, mas nem sempre é assim, uma vez que há a dependência da quantidade de variáveis e restrições do problema. Dados esses resultados, é possível afirmar que, para obter resultados mais rápidos, é necessário perder um pouco da qualidade.

Abensur e Oliveira (2012) aplicaram uma heurística construtiva para elaborar a grade horária dos cursos de engenharia de uma universidade, em que foi feito um modelo matemático tanto para programação linear quanto para heurísticas para o problema – eles tentaram resolvê-lo usando apenas a heurística construtiva, tentando melhorá-la com a heurística de melhoria e também com o Algoritmo Genético. Em seus experimentos, observaram que, ao usar somente a heurística construtiva, obteve-se um resultado médio da função-objetivo similar comparado ao resultado em que foi utilizada a meta-heurística construtiva e de melhoria, enquanto que, ao utilizar a heurística construtiva e o Algo-

ritmo Genético, o resultado médio superou os demais. Em termos de tempo de execução, a heurística de construção, sozinha, gastou um tempo consideravelmente menor que as demais.

Outras meta-heurísticas não citadas também são usadas em problemas de horários, Faleiros (2007), por exemplo, usou a meta-heurística inspirada no sistema imunológico dos animais vertebrados e o algoritmo de seleção clonal, que parte de uma população inicial de anticorpos gerada aleatoriamente respeitando as restrições invioláveis. Assim como no AG, cada anticorpo é avaliado de acordo com o seu grau de afinidade, que nesse caso é a distância entre o antígeno e o anticorpo. Nesse entremeio, o anticorpo selecionado é clonado e o número de clones é proporcional à sua afinidade, sendo que todos os clones passarão pelo processo de mutação desde que a taxa de mutação, que é proporcional à da afinidade, seja alta - afinidade de um clone não pode ser inferior À afinidade do anticorpo clonado, e os clones que obtiverem melhorias são selecionados. Dessa forma, os anticorpos antigos com as piores afinidades são substituídos pelos clones selecionados, e isso se repete até que os critérios de paradas sejam alcançados. Segundo o autor, tal meta-heurística encontrou valores próximos a valores obtidos em outros trabalhos, porém com tempo maior.

Na tentativa de obter resultados melhores e em menos tempo, pesquisadores combinaram meta-heurísticas num processo chamado hibridização, como o que foi feito no trabalho de Abdullah et al. (2012). Nele, foram combinados um mecanismo eletromagnético, que se simula atração e repulsão de pontos de amostra em movimento, e o algoritmo de grande dilúvio, um algoritmo de busca local que aceita soluções piores, desde que estejam dentro de um nível, para resolver problemas de horários universitários. A força dinâmica calculada a partir do mecanismo eletromagnético é usado na diminuição do nível do grande dilúvio, sendo que a hibridização produziu resultados melhores do que os verificados em algumas abordagens publicadas.

Wang et al. (2011) e Nguyen, Nguyen e Tran (2012) hibridizaram o algoritmo de busca harmônica. No primeiro trabalho, o ABH foi hibridizado com o sistema de formigas, enquanto que, no segundo, o ABH foi combinado com o algoritmo de abelhas. Nguyen et al. aplicaram o ABH híbrido para a solução do problema de programação de horários universitários com o uso do algoritmo de abelhas para a construção das soluções inciais para a inicialização da memória harmônica, aumentando a probabilidade de ABH retornar uma solução ainda melhor. Os resultados apresentados mostram que tal hibridização é capaz de resolver o problema de PHCU, obtendo bons resultados.

Além disso há a possibilidade de dividir o problema em problemas menores, assim como Martins (2010) fez. Em sua pesquisa, ele dividiu o problema de programação de horários universitários em dois subproblemas menores: “Professor x Disciplina” e “Professor x Disciplina x Horário” para facilitar a resolução dos problemas individualmente.

Diante disso, Moreira (2011) explica algumas técnicas possíveis para a resolução do problema de horários, assim como faz uma tabela sintetizando cada uma delas com a enumeração das vantagens e as desvantagens.

4 Desenvolvimento

Neste capítulo serão apresentados os procedimentos realizados para o desenvolvimento deste trabalho, começando pelo detalhamento do problema de programação de grades horárias em cursos universitários e suas restrições. Em seguida, será feita uma breve explicação das diferentes técnicas utilizadas para tentar encontrar a melhor solução para instâncias distintas de PHCU. A implementação desse trabalho foi disponibilizada em: <https://github.com/lauramundim/PHCU>.

Toda implementação foi feita usando a linguagem de programação *Scala* [1], pois essa linguagem incorpora recursos de linguagens orientadas a objetos e funcionais, além disso, é plenamente interoperável com *Java*.

A linguagem Scala permite que o código fique mais limpo e ao mesmo tempo mais claro, além do mais, é uma linguagem fortemente tipada, o que pode dificultar a implementação, no entanto, isso impede a ocorrência de erros relacionados a tipo e facilita o rastreamento dos erros que eventualmente acontecem. Como Scala é uma linguagem nova – desenvolvida em 2001 por Martin Odersk – muito discutida e apontada como sendo a substituta direta da linguagem Java (SOUZA, 2013), despertou-se o interesse de conhecer e aprender essa linguagem.

4.1 Problema

Como já explicado anteriormente o problema Programação de Horários em Cursos Universitários (PHCU) consiste em alocar eventos nos quais um professor ministra uma disciplina a um grupo de alunos em uma sala em um determinado horário e ainda respeitando algumas restrições. Das restrições existentes para problemas de PHCU foram selecionadas algumas dos trabalhos encontrados na literatura.

Baseado no trabalho de Yang e Jat (2011) foram implementadas as seguintes restrições:

Invioláveis

- *Conflito de alunos*: eventos diferentes com alunos em comum não podem ser alocados em um mesmo horário.
- *Conflito de eventos*: apenas um evento pode acontecer em uma sala e em um horário.

[1] Para mais informações acesse: <http://www.scala-lang.org/>

- *Sala Adequada*: a sala deve possuir os recursos necessários para atender um determinado evento e ainda ter tamanho suficiente para comportar o número de alunos que participam deste evento.

Preferenciais

- *Último horário*: é preferível que os alunos não tenham aula no último horário do dia.
- *Aulas consecutivas*: não é desejável que alunos tenham mais de duas aulas consecutivas sejam estas de uma mesma disciplina ou não.
- *Aula única*: ideal seria se os alunos não tivessem apenas uma aula no dia.

Foram implementadas também algumas restrições sugeridas no trabalho de Bonutti et al. (2010):

Invioláveis

- *Aulas*: todas as aulas de todas as disciplinas devem ser agendadas.
- *Conflitos*: os eventos que pertencem ao mesmo conjunto de disciplinas correlacionadas – currículo – ou que tem os mesmos professores não podem acontecer no mesmo horário.
- *Salas*: não pode acontecer mais de uma aula no mesmo dia, horário e sala.
- *Disponibilidade*: aulas não devem ser agendadas em horários que o professor não esteja disponível.

Preferenciais

- *Capacidade da sala*: o número de alunos que participam de um evento deve ser compatível com a capacidade da sala.
- *Mínimo de dias*: as aulas de cada disciplina devem ser alocadas em um determinado número mínimo de dias.
- *Aulas isoladas*: uma aula é isolada quando não existe uma aula antes ou depois do seu horário, sendo assim, aulas pertencentes ao mesmo currículo não devem ficar isoladas.
- *Estabilidade da sala*: aulas de uma disciplina devem acontecer em uma mesma sala.

Esses dois grupos de restrições foram implementados separadamente de acordo com os conjuntos de instâncias usados nos testes e serão explicados adiante.

4.1.1 Instâncias

Foram utilizados dois formatos de instâncias: *tim* e *ctt*. O primeiro foi usado no conjunto de instâncias PATAT2002 [2], apresentado por Yang e Jat (2011) em seus experimentos. O segundo para as instâncias ITC2007 [3] formuladas no padrão proposto por Bonutti et al. (2010).

Formato tim

Esse formato é um arquivo texto cuja extensão é *.tim*. O arquivo é preenchido apenas com números inteiros e o espaço é o separador entre os dados de uma linha. (ROSSI-DORIA et al., 2002a)

Os arquivos de instância *tim* têm o seguinte formato:

- *Primeira linha*: é uma sequencia de quatro valores que representam respectivamente o número de eventos, o número de salas, o número de recursos e o número de alunos.
- *Uma linha para cada sala*: a partir da segunda linha, cada linha representa o tamanho de uma sala. Isso é claro, até chegar o número de salas existentes.
- *Uma linha para cada aluno por evento*: após o tamanho da última sala, é preenchida uma linha para cada aluno e para cada evento com valor 1 ou 0 que significa respectivamente se o aluno participa ou não do evento. Por exemplo, se existem 2 alunos, 3 eventos e uma sequencia de linhas da seguinte forma:

  0
  1
  0
  0
  1
  1

  Monta-se uma matriz bidimensional onde uma das suas dimensões é o número de alunos e a outra é o número de eventos, ficando assim:

```
        nEventos
nAlunos 0  1  0
        0  1  1
```

[2] Para mais informações acesse: <<http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html>>
[3] Para mais informações acesse: <<http://satt.diegm.uniud.it/ctt/index.php?page=instances>>

A partir dessa matriz pode-se entender que o primeiro aluno frequenta apenas o segundo evento e o segundo aluno participa do segundo e do terceiro evento.

- *Uma linha para cada sala por recurso*: ao finalizar as posições da matriz anterior, é preenchida uma linha para cada sala e para cada recurso com valor 1 ou 0 que significa respectivamente se a sala possui ou não o recurso. Por exemplo, se existem 2 salas, 3 recursos e uma sequencia de linhas da seguinte forma:

  0
  1
  0
  0
  1
  0

  Monta-se uma matriz bidimensional onde uma das suas dimensões é o número de salas e a outra é o número de recursos, ficando assim:

```
       nRecursos
nSalas 0  1  0
       0  1  0
```

  A partir dessa matriz pode-se entender que a primeira e a segunda sala é portadora apenas do segundo recurso.

- *Uma linha para cada evento por recurso*: ao finalizar as posições da matriz anterior, é preenchida uma linha para cada evento e para cada recurso com valor 1 ou 0 que significa respectivamente se o evento necessita ou não do recurso. Por exemplo, se existem 2 eventos, 3 recursos e uma sequencia de linhas da seguinte forma:

  0
  1
  0
  1
  1
  0

  Monta-se uma matriz bidimensional onde uma das suas dimensões é o número de eventos e a outra é o número de recursos, ficando assim:

```
         nRecursos
nEventos 0  1  0
         1  1  0
```

A partir dessa matriz pode-se entender que o primeiro evento precisa apenas do segundo recurso e o segundo evento necessita do primeiro e do segundo recurso.

Formato ctt

Esse formato consiste descrever uma instância no padrão XML com as características do problema distribuídas de maneira hierárquica. Bonutti, Gaspero e Schaerf (2014) criaram um site que contém informações mais detalhadas sobre as instâncias neste formato.

O Anexo A traz o DTD (linguagem para definir a estrutura de um documento XML) das instâncias que usam o formato ctt e faz uso das seguintes *tags*:

<descriptor> traz informações gerais sobre a instância, dentro dessa *tag* tem-se:

- <days> informa o número de dias letivos da instância.
- <periods_per_day> informa o número de horários por dia que podem ser alocados os eventos.
- <daily_lectures> informa o número mínimo e máximo de dias que as disciplinas podem ser agendadas.

<courses> traz a descrição das disciplinas pertencentes a instância. Dentro dessa *tag* tem-se a *tag* <course> que se refere as disciplinas com seus atributos, que são:

- id: refere-se ao código da disciplina.
- teacher: refere-se ao código do professor que ministra a disciplina.
- lectures: número de aulas que a disciplina tem.
- min_days: refere-se ao número mínimo de dias em que a disciplina deve ser agendada.
- students: refere-se ao número de alunos que fazem a disciplina
- double_lectures: indica se a disciplina permite que aconteça aulas duplas ou consecutivas.

<rooms> traz a descrição das salas pertencentes a instância. Dentro dessa *tag* tem-se a *tag* <room> que se refere as salas com seus atributos, são eles:

- id: refere-se ao código da sala.
- size: refere-se a capacidade máxima da sala.
- building: refere-se ao prédio onde a sala está localizada.

<curricula> traz a descrição dos currículos escolares pertencentes a instância. Dentro dessa *tag* tem-se a *tag* <curriculum> que tem o código do currículo e dentro de <curriculum> tem as disciplinas pertencentes ao currículo referenciadas por <course>.

<constraints> traz a descrição das restrições em que a instância está sujeita. Dentro dessa *tag* tem-se a *tag* <constraint> com os atributos das restrições:

`type`: refere-se ao tipo de restrição, pode ser por exemplo referente a período ou sala.

`course`: refere-se ao código da disciplina que está envolvida na restrição.

Ainda dentro da *tag* <constraints> tem-se outro sub-nível que depende do tipo da restrição, se for restrição do tipo `period`, a *tag* será <timeslot> com os atributos `day` e `period`, informando o ou os dias e horários em que a disciplina não poderá ser agendada. Se a restrição for do tipo `room`, a *tag* será <room> com a referência de qual ou quais salas a disciplina não pode ser alocada

A Seguir veja um pequeno exemplo de como são estruturadas as instâncias.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE instance SYSTEM "cb_ctt.dtd">
<instance name="Fis0506-1">
  <descriptor>
    <days value="5"/>
    <periods_per_day value="6"/>
    <daily_lectures min="2" max="5"/>
  </descriptor>
  <courses>
    <course id="c0001" teacher="t000" lectures="6"
    min_days="4" students="130" double_lectures="yes"/>
    <course id="c0002" teacher="t001" lectures="6"
    min_days="4" students="75" double_lectures="yes"/>
    <course id="c0004" teacher="t002" lectures="7"
    min_days="3" students="117" double_lectures="yes"/>
  </courses>
  <rooms>
    <room id="rB" size="200" building="0"/>
    <room id="rC" size="100" building="2"/>
  </rooms>
  <curricula>
    <curriculum id="q000">
      <course ref="c0001"/>
      <course ref="c0002"/>
    </curriculum>
    <curriculum id="q001">
      <course ref="c0001"/>
      <course ref="c0004"/>
    </curriculum>
  </curricula>
  <constraints>
    <constraint type="period" course="c0002">
      <timeslot day="0" period="4"/>
```

```
      <timeslot day="0" period="5"/>
    </constraint>
    <constraint type="period" course="c0004">
      <timeslot day="3" period="0"/>
    </constraint>
    <constraint type="room" course="c0002">
      <room ref="rC"/>
    </constraint>
    <constraint type="room" course="c0001">
      <room ref="rB"/>
    </constraint>
  </constraints>
</instance>
```

Nesse padrão de instância, os eventos são cada uma das aulas de cada disciplina descrita.

Como o objetivo deste trabalho é realizar experimentos com os dois tipos de formatos mencionados, fez-se necessário implementar duas maneiras de ler esses arquivos e montar as estruturas do problema a ser resolvido.

4.1.2 Leitura das instâncias de um PHCU

A estrutura de dados que representa o problema obtém os dados das instâncias através de um arquivo que são estruturados de acordo com o formato da instância. Foram implementadas duas formulações para o problema: uma que lê e obtém dados de um arquivo no formato *tim*; e outra que lê e extrai informações de um arquivo XML no formato *ctt*.

Cada formulação foi entendida como um problema, então, foi implementada uma classe abstrata `Problema`, que possui como argumento de entrada um nome de arquivo, se esse arquivo tiver extensão *.xml* será instanciada a classe `Ctt`, caso a extensão do arquivo seja *.tim* será instanciada a classe `Tim`. Essa representação é ilustrada na Figura 6 cada uma dessas classes serão explicadas adiante.

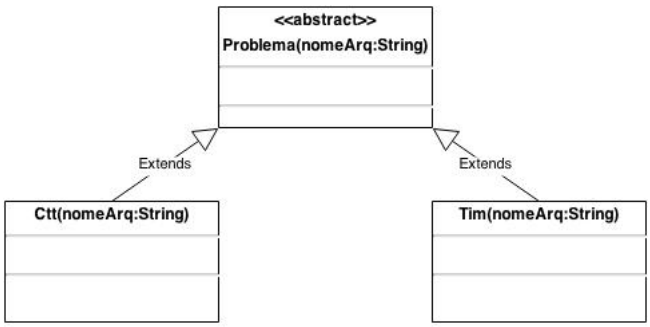

Figura 6 – Representação do módulo problema, onde tem-se uma abstração `Problema` com duas extensões que representam os formatos tim e ctt.

Classe Problema

`Problema` é uma classe abstrata que define os atributos que serão implementados nas classes filhas, alguns destes atributos serão efetivamente usados em apenas uma das classes. O Apêndice A traz a codificação dessa classe.

Foi definida essa classe abstrata para que o restante do sistema conheça apenas um problema, independente se este provém de arquivo com extensão *.tim* ou *.xml*.

Para se fazer possível a leitura dos arquivos de instâncias foram declaradas algumas variáveis e estruturas, são elas:

As variáveis `nEventos`, `nSalas`, `nHorários`, `nDisciplinas`, `nCurriculos`, `nRestrições`, `nEstudantes`, `periodosPorDia` e `diasPorSemana` são valores inteiros que contabilizam o número de eventos, salas, horários, etc. de acordo com seus respectivos nomes.

`estudanteNum` é um vetor em que cada posição representa um evento, o valor a preencher é o número de estudantes que participam do evento.

`tamSalas` é um vetor onde cada posição representa uma sala e esta é preenchida com o tamanho da sala.

`estudanteEventos` é uma matriz bidimensional onde uma de suas dimensões é o número de estudantes e a outra é o número de eventos, quando um estudante e participa de um evento ev, a posição (e, ev) da matriz recebe valor 1.

`diasHorários` é uma estrutura que mapeia para cada dia da semana os horários que pertencem a ele. Exemplo: em uma semana com 5 dias letivos e 4 horários por dia, tem-se um total de 20 horários possíveis, no dia d_0 tem-se os horários h_0, h_1, h_2 e h_3, no dia d_1 tem-se os horários h_4, h_5, h_6 e h_7 e assim por diante, até chegar no dia d_4 e no horário h_{19}.

`siglasSalas` é um vetor onde cada posição representa uma sala e esta é preenchida com a descrição (sigla) da sala.

`possiveisSalas` é uma matriz bidimensional onde uma das dimensões é o número eventos e a outra é o número de salas, quando uma sala s comporta um evento e, a posição (e, s) da matriz recebe valor `true`, visto que é uma matriz de valores *booleanos*.

`listaDisciplinas` é uma sequencia de `Curso` que é um objeto que representa uma determinada disciplina que possui um id (sigla), um professor, a quantidade de aulas, o número de alunos que estão matriculados na disciplina e ainda se essa disciplina pode ou não acontecer em duas aulas consecutivas.

`listaCurriculos` é uma sequencia de `Curriculum` que é um objeto que representa um conjunto de disciplinas que pertencem a um mesmo currículo, ou seja são disciplinas correlacionadas.

`listaSalas` é uma sequencia de `Sala` que é um objeto que representa uma sala que contém um id (sigla), sua capacidade (tamnaho) e o prédio onde está localizada.

`listaRestricoes` é uma sequencia de `Restricao` que representa uma restrição de uma disciplina que pode ser referente a sala ou a horário. Se for uma restrição de sala, possui uma lista das salas em que a disciplina não pode ser alocada e se for restrição de horário possui uma lista de horários em que a disciplina não pode ser agendada.

`eventos` é um vetor onde cada posição representa um evento e esta é preenchida com a descrição (sigla) da disciplina.

`eventoCorrelacoes` é uma matriz bidimensional onde as duas dimensões é o número de eventos, quando um evento e_1 é correlacionado com evento e_2, a posição (e_1, e_2) da matriz recebe valor `true`, visto que é uma matriz de valores *booleanos*.

Alguns desses atributos são específicos para a classe Ctt e outros para a classe Tim, a seguir serão apresentadas as diferenças entre essas classes.

Classe Tim

A classe `Tim` é utilizada para implementar problemas cujo arquivo de entrada possua extensão *.tim* e que seja estruturado de acordo com o padrão *tim*. O Apêndice B traz a codificação desta classe.

Das diferenças entre as classes `Tim` e `Ctt` tem-se que na primeira as variáveis `diasPorSemana` e `periodosPorDia` possuem valores fixos, que são respectivamente 5 e 9. Além do mais no padrão *tim* não usa os termos currículo e restrição. Para usar o termo disciplina usado no formato *ctt*, cada evento foi considerado como uma disciplina com apenas uma aula. Entretanto, no formato *tim*, tem-se o termo recursos, logo, fez-se necessário a criação de estruturas para ler estas informações, são elas:

`salaRecursos`: é uma matriz bidimensional onde uma de suas dimensões é o número de salas e a outra é o número de recursos disponíveis.

`eventoRecursos`: é uma matriz bidimensional onde uma de suas dimensões é o número de eventos e a outra é o número de recursos disponíveis.

As demais estruturas são comuns entre as classes `Tim` e `Ctt`, porém algumas delas se diferem na forma como são preenchidas. A matriz `eventoCorrelacoes`, por exemplo, é

preenchida a medida em que se lê as informações contida no arquivo, ou seja, as correlações dos eventos são dadas diretamente no arquivo de entrada não sendo necessário nenhuma verificação em outra estrutura.

Já a matriz `possiveisSalas` é montada a partir do vetor `tamSalas`, para verificar se a sala comporta o evento e ainda verificando nas matrizes `salaRecursos` e `eventoRecursos` quais salas possuem os recursos necessários para o evento.

Classe Ctt

A classe `Ctt` é utilizada para implementar problemas cujo o arquivo de entrada é um XML e segue padrão proposto por Bonutti et al. (2010), ou seja, o formato *ctt*. O Apêndice C traz a codificação desta classe.

Nessa formulação para o problema não existe o termo estudante separadamente assim como é usado no formato *tim*. Além do mais, foram acrescentadas algumas variáveis para ser possível a leitura do arquivo no formato *ctt*, são elas:

As variáveis `nomeInstancia`, `minAulas` e `maxAulas` são para guardar informações lidas na descrição da instância, tais como, nome, o número mínimo e máximo de aulas por dia.

`impossiveisSalas` é uma matriz auxiliar bidimensional onde uma das dimensões é o número eventos e a outra é o número de salas. Esta matriz é preenchida de acordo com as restrições de salas, quando um evemto e não puder ser alocado na sala s, a posição (e, s) da matriz recebe valor `true`, visto que é uma matriz de valores *booleanos*. A partir dessa matriz e do vetor de `tamSalas` é construída a matriz `possiveisSalas`.

Diferentemente do formato *tim*, no padrão *ctt* a matriz `eventoCorrelacoes` é montada a partir do vetor `listaCurriculos`. Quando um evento e_1 pertence a uma disciplina d_1, um evento e_2 pertence a uma disciplina d_2 e essas disciplinas fazem parte de um mesmo currículo c, a posição (e_1, e_2) da matriz recebe valor `true`, visto que é uma matriz de valores *booleanos*.

Nesse formato cada aula de cada disciplina foi considerada como um evento. Por exemplo, tem-se duas disciplinas d_1 e d_2 com 4 aulas cada uma, então as posições $0, 1, 2$ e 3 do vetor `eventos` são equivalentes a eventos relacionados à d_1, e as posições $4, 5, 6$ e 7 à d_2.

Diante destes problemas foram investigadas diferentes formas para resolvê-los. Na próxima sessão serão apresentados os algoritmos implementados para obter uma melhor solução para instâncias de problemas nos formatos citados.

4.2 Algoritmos implementados

Na tentativa de encontrar uma melhor solução para o PHCU, baseando-se nos trabalhos de Yang e Jat (2011) e de Rossi-Doria et al. (2002b), foram implementados oito diferentes programas combinando a meta-heurística Algoritmo Genético com busca local e uma estrutura de memória denominada MEM. Os algoritmos implementados foram:

1. Algoritmo Genético Básico (AGBasico)
2. Algoritmo Genético Básico com Memória (AGBasicoMEM)
3. Algoritmo Genético usando Alocação de Salas (AGAlocador)
4. Algoritmo Genético usando Alocação de Salas com Memória (AGAlocadorMem)
5. Algoritmo Genético usando Alocação de Salas e uma Busca Local (AGAlocacorBL1)
6. Algoritmo Genético usando Alocação de Salas e duas Buscas Locais (AGALocadoBL2)
7. Algoritmo Genético usando Alocação de Salas com Memória e uma Busca Local (AGAlocacorMEMBL1)
8. Algoritmo Genético usando Alocação de Salas com Memória e e duas Buscas Locais (AGAlocacorMEMBL2)

Os detalhes sobre como foi estruturado e implementado cada parte das hibridizações dos Algoritmos Genéticos serão apresentados nas próximas sessões.

4.2.1 Implementação do Algoritmo Genético

Para o desenvolvimento da técnica meta-heurística Algoritmo Genético e suas variações foram implementados alguns módulos interdependentes para o funcionamento dos algoritmos sobre um PHCU sujeito às suas restrições.

O módulo AG e suas extensões dependem da População e dos Operadores – que são constituídos por seleção, mutação, buscas locais e atualização – que por sua vez dependem do Indivíduo que é composto por um Cromossomo – que é um conjunto de Genes – e sua Aptidão que necessita do Problema com suas Restrições para ser calculada. Na Figura 7 é ilustrada a dependência entre esses módulos.

Para facilitar o entendimento dos algoritmos implementados esses módulos serão apresentados a seguir. Primeiramente, será explicado como foram estruturados o cromossomo e sua aptidão, o indivíduo, a população e os operadores que atuam sobre ela.

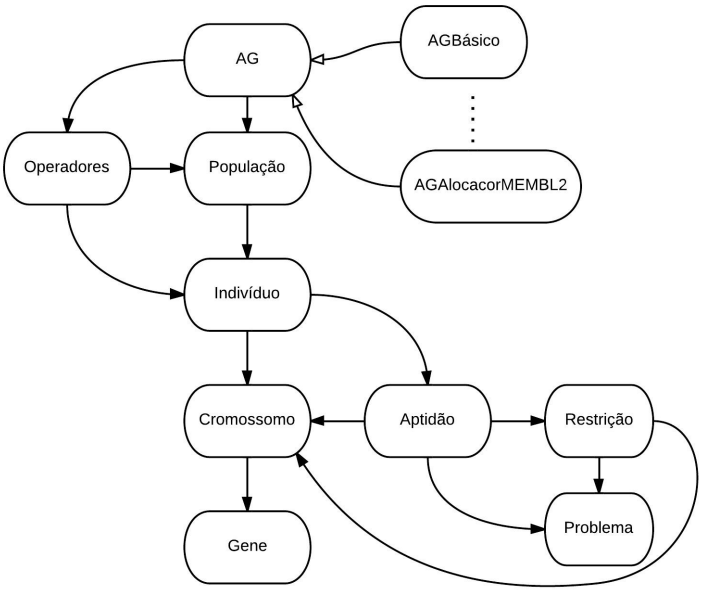

Figura 7 – Representação da dependência entre os módulos onde as extensões do AG dependem da População e dos Operadores que por sua vez dependem do Indivíduo que é composto por um Cromossomo – que é constituído por Genes – e sua Aptidão que necessita das Restrições e do Problema para ser calculada.

Cromossomo

A estrutura do Cromossomo foi definida como sendo um vetor de genes, onde `Gene` é uma classe com dois atributos inteiros, um representando a sala e outro o horário onde o evento que este gene representa será alocado. O tamanho do cromossomo é definido pelo número total de eventos onde cada posição do vetor representa um evento.

Na Figura 8 é ilustrada a estrutura do cromossomo na qual pode-se notar que a primeira posição do vetor armazena o $Gene_1$ que representa o primeiro evento e_1. O $Gene_1$ é formado por uma sala s_1 e um horário h_1 em que o evento e_1 está alocado no momento atual. Essa interpretação é equivalente para os demais eventos e_i na posição i do vetor.

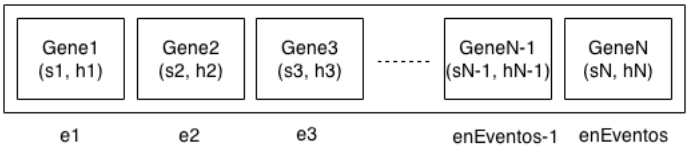

Figura 8 – Estrutura do cromossomo

Assim que um cromossomo é criado, este passa por um processo de avaliação onde é calculado sua aptidão.

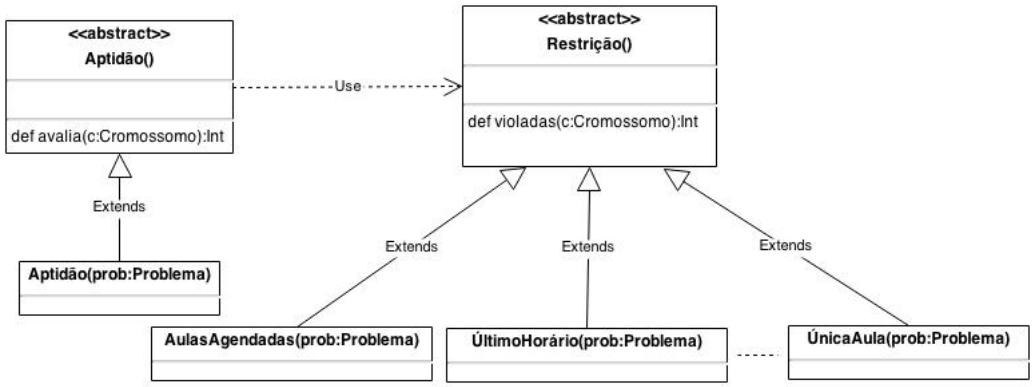

Figura 9 – Estrutura da avaliação do cromossomo na qual a classe `Aptidão` usa todas as extensões da classe `Restrição` implementadas. Aqui é ilustrado apenas 3 das 12 classes de restrições

Aptidão

Um cromossomo é avaliado de acordo com a quantidade de restrições violadas, cada violação tem sua penalidade e agrega o valor da aptidão, ou seja, quanto maior o valor da aptidão de um cromossomo, maior o número de violações de restrições.

A implementação da avaliação do cromossomo foi dividida em duas abstrações: `Aptidão` e `Restrição` das quais a primeira depende da segunda conforme estrutura ilustrada na Figura 9.

A classe `Aptidão` define métodos que efetivamente avalia o cromossomo, ou seja, somam as penalidades tanto do cromossomo inteiro quanto as violações por evento. A classe `Restrição` declara os métodos para toda e qualquer restrição, a partir dela são implementadas as restrições baseando-as em um problema específico que é passado como argumento de entrada.

Essa estruturação foi feita para desacoplar as restrições de um problema da técnica que busca a solução, pois a medida que surgem novas restrições, estas não afetam o restante do código, da mesma forma que remover uma restrição também não afeta visto que cada restrição é tratada como uma classe independente.

Para cada restrição mencionada na sessão 4.1 tem-se uma classe concreta que implementa a classe abstrata `Restrição`, exceto as restrições invioláveis do problema no formato *tim*, pois estas são apenas verificações que podem ser realizadas dentro de um mesmo laço otimizando assim o tempo de execução.

As classes que representam as restrições nas quais os problemas no formato *tim* estão submetidos são:

`ViolacõesInv`: calcula violações das restrições “Conflito de alunos”, “Conflito de eventos”e “Sala adequada”. Cada evento que viole uma dessas restrições é contabilizado como uma violação inviolável.

`ÚltimoHorário`: calcula o número de alunos que participam de eventos alocados no último horário do dia. Cada um desses alunos representa uma violação inviolável.

`AulasConsecutivas`: contabiliza quantos alunos participam de mais de dois eventos seguidos. Cada um desses alunos também representa uma violação inviolável.

`ÚnicaAula`: conta quantos alunos participam de apenas um evento no dia. Cada um desses alunos equivale a uma violação inviolável.

Enquanto que os problemas no formato *ctt* estão sujeitos às restrições implementas nas seguintes classes:

`AulasAgendadas`: calcula o número total de aulas não agendadas para todas as disciplinas, sendo que as aulas de uma mesma disciplina devem ser programadas em horários diferentes. Assim sendo, cada aula de uma disciplina agendada a mais para um mesmo horário conta como uma violação inviolável.

`MembrosConflitantes`: contabiliza o número total de conflitos entre eventos que possuam alunos ou professores em comum e que ocorram no mesmo horário. Cada conflito de eventos equivale uma violação inviolável.

`SalasConflitantes`: calcula o número total de conflitos de salas, ou seja, salas em que aconteçam mais de um evento em um mesmo horário. Cada evento a mais em uma salas é equivalente a uma violação inviolável.

`IndisponibilidadesProf`: conta o número de eventos alocados em horários em que o professor do evento não está disponível. Cada aula agendada em um horário que o professor esteja indisponível equivale a uma violação inviolável.

`CapacidadeSalas`: calcula o número de alunos em excesso em cada sala para cada evento. Cada aluno conta como uma violação preferencial.

`MínimoDias`: calcula o número de dias a baixo do mínimo em que as aulas de uma disciplina foram alocadas. Cada aula abaixo do mínimo de aulas equivale a cinco violações preferenciais.

`AulasIsoladas`: contabiliza o número de aulas isoladas do mesmo currículo. Cada evento isolado conta duas violações, mesmo que represente aula de uma mesma disciplina.

`EstabilidadeSala`: conta o número salas diferentes em que eventos de uma mesma disciplina foram alocados. Cada sala diferente que a aula de uma disciplina esteja alocada equivale a uma violação preferencial.

Todas estas classes implementam o método `violadas` da classe `Restrição` que retorna a penalidade de uma determinada restrição. Implementam também o método `penalizaEvento` que calcula quantas restrições um determinado evento violou em um cromossomo e ainda tem o método `eventosAfetados` que conta quantas restrições foram violadas ao mudar um evento de horário e/ou sala.

A classe `Aptidão` é onde reúne todas as restrições para aplicar a função de avaliação. Nesta classe tem-se os atributos `rPreferenciais` e `rInvioláveis` que são listas de restrições. Como não tem nada na restrição que indique se ela é do tipo preferencial ou inviolável, a lista onde ela é inserida através do método `addRestriçãoInviolável` ou `addRestriçãoPreferencial` é que define seu tipo.

Dado a soma das violações invioláveis e a soma das violações preferenciais pode-se avaliar o cromossomo através do método `avalia` cuja função de aptidão, assim como no trabalho de Yang e Jat (2011), é dada por:

$$1000000 \times \text{invioláveis} + \text{preferenciais}$$

onde 1000000 é um valor constante que penaliza as violações de restrições invioláveis. Foi usado esse valor para melhor visualização da quantidade de violações de restrições invioláveis e de restrições preferenciais, por exemplo, uma aptidão cujo o valor é 1000010, pode ser interpretada como 1 violação de restrição inviolável e 10 violações de restrições preferenciais.

Ao criar um cromossomo a partir de um problema, avaliá-lo segundo suas restrições e ainda armazenar seu valor de aptidão, tem-se o indivíduo.

Indivíduo

Integrando o problema ao cromossomo e calculando sua aptidão forma-se o indivíduo que representa uma solução para um problema. A classe `Indivíduo` tem com parâmetros de entrada o problema e o cromossomo, e nela é calculada e armazenada a aptidão do cromossomo.

População

A população é um conjunto de indivíduos que são ordenados de acordo com sua aptidão de forma crescente, ou seja, começando pelo melhor indivíduo. A classe `População` tem métodos que retornam a melhor, a pior, a média e o desvio padrão entre as aptidões de seus indivíduos, bem como tem um método que retorna o melhor indivíduo da população. O tamanho da população é definido por parâmetros.

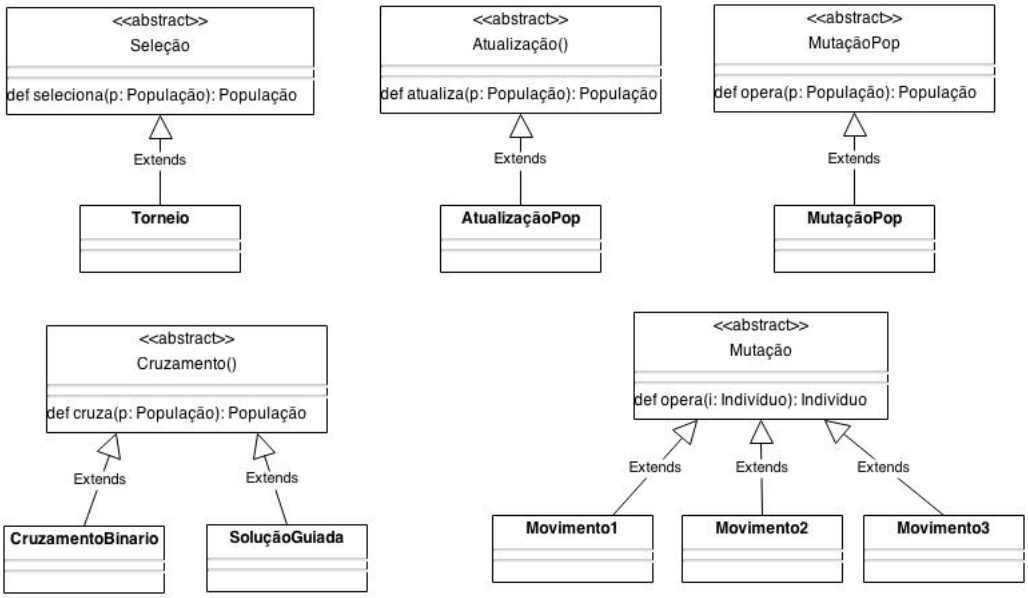

Figura 10 – Estrutura dos operadores genéticos onde a classe `Torneio` é uma extensão da classe abstrata `Seleção`, a classe `AtualizaçãoPop` é uma extensão da classe abstrata `Atualização`, a classe `MutaçãoPop` é uma extensão da classe abstrata `Seleção`, as classes `CruzamentoBinário` e `SoluçãoGuiada` são extensões da classe abstrata `Cruzamento`, e as classes `Movimento1`, `Movimento2` e `Movimento3` são extensões da classe abstrata `Mutação`.

Operadores Genéticos

Os operadores também foram estruturados de forma a generalizar o código, tendo uma classe abstrata definindo um operador e uma ou mais classes concretas que implementam a classe mãe facilitando a inclusão de novos operadores.

Na Figura 10 é ilustrada a estrutura dos operados genéticos implementados na qual classe `Torneio` é uma extensão da classe abstrata `Seleção`, a classe `AtualizaçãoPop` é uma extensão da classe abstrata `Atualização`, a classe `MutaçãoPop` é uma extensão da classe abstrata `Seleção`, as classes `CruzamentoBinário` e `SoluçãoGuiada` são extensões da classe abstrata `Cruzamento`, e as classes `Movimento1`, `Movimento2` e `Movimento3` são extensões da classe abstrata `Mutação`.

Usando para exemplificar o operador seleção, tem-se uma classe abstrata `Seleção` que declara um método `seleciona`, o tipo de seleção usado foi o torneio, então, tem-se uma classe `Torneio` que implementa a classe `Seleção`. Caso seja necessário adicionar um outro operador de seleção, roleta por exemplo, basta criar uma nova classe que estende a classe `Seleção` e criar um novo cruzamento que use tipo de seleção usando a mesma ideia, estendendo a classe `Cruzamento`.

A seleção por torneio foi implementada de forma a selecionar n indivíduos, o melhor de cada t grupos aleatórios de indivíduos, onde n e t são parâmetros de entrada. Neste trabalho usou-se os valores $n = 1$ e $t = 2$ assim como no trabalho de Yang e Jat (2011) para facilitar a comparação dos resultados.

Após selecionados os indivíduos, eles passam pelo operador de cruzamento com

intuito de gerar um novo indivíduo. Foram implementados dois tipos de cruzamento, um simples e outro que utiliza uma estrutura de memória. A forma como esses cruzamentos funcionam é mostrada respectivamente nos Algoritmos 3.8 e 3.7 que foram apresentados na sessão 3.2.4 e são baseados no trabalho de Yang e Jat (2011).

Uma vez que gerou um novo indivíduo, este pode passar ou não por um processo de mutação. Neste trabalho a mutação pode ser um dos três movimentos de vizinhança apresentados na sessão 3.2.4, onde a escolha do movimento é aleatória.

E por fim o novo indivíduo deve ser inserido na população. A atualização da população é dada por substituir pior indivíduo da população pelo novo indivíduo gerado.

O Algoritmo Genético nada mais é do que a execução dos operadores sobre uma população de indivíduos repetidas vezes até que encontre uma solução aceitável para um problema. No entanto o AG sozinho pode não ser muito útil, por isso foram feitas algumas variações neste algoritmo com intenção de obter melhores resultados.

4.2.2 Variações do Algoritmo Genético

Tendo como base os algoritmos apresentados no trabalho de Yang e Jat (2011) expostos na sessão 3.2.4 foram elaborados oito diferentes algoritmos, desde o mais simples algoritmo genético até o mais robusto no intuito de comparar o efeito que cada modificação exerce sobre o AG.

Para facilitar a inclusão e/ou remoção de uma variação do AG foi implementada uma classe abstrata `AG`, nela é declarado apenas um método `run` que deve ser implementado nas demais extensões dessa classe. Este método não tem parâmetro de entrada, apenas chama as funções necessárias e retorna uma população de possíveis soluções.

Cada variação do Algoritmo Genético é considerado como uma classe concreta que implementa `AG`, a seguir veja as semelhanças e diferenças de cada uma.

Algoritmo Genético Básico (AGBasico)

A base de todos os algoritmos implementados neste trabalho é o Algoritmo Genético simples explicado na sessão 3.2.4 e representado no Algoritmo 3.4. A população neste AG é gerada totalmente aleatória e os operados genéticos utilizados foram os apresentados na página 61, sendo usado apenas o cruzamento simples representado no Algoritmo 3.8.

Algoritmo Genético Básico com Memória (AGBasicoMEM)

Para saber o real efeito de utilizar a estrutura MEM explicada na página 31, foi acrescentado ao AGBasico o cruzamento que faz uso da estrutura MEM apresentado no

Algoritmo 3.7. A construção da MEM é dada pelo Algoritmo 3.6. O código que implementa a MEM pode ser visualizado no Apêndice D.

Algoritmo Genético usando Alocação de Salas (AGAlocador)

Para que a população inicial do AGBasico não ficasse totalmente aleatória causando impacto negativo nas soluções, foi usado na alocação das salas um algoritmo de emparelhamento, no entanto a alocação dos horários permanece aleatória assim como no trabalho de Yang e Jat (2011).

O algoritmo de emparelhamento é baseado no método de Hopcroft e Karp e aloca todos os eventos em possíveis salas. Para isso monta-se um grafo bipartido, onde os nós de uma partição são os eventos e os nós da outra partição são as salas. As arestas deste grafo liga um evento a uma sala que ele possa ser alocado. No caso do emparelhamento não ser perfeito, cada evento não emparelhado é alocado na sala menos ocupada na qual ele possa ser realizado.

Algoritmo Genético usando Alocação de Salas com Memória (AGAlocadorMem)

Essa variação consiste apenas em acrescentar o cruzamento que faz uso da estrutura MEM apresentado no Algoritmo 3.7 ao AGAlocador.

Algoritmo Genético usando Alocação de Salas e uma Busca Local (AGAlocacorBL1)

Nesta modificação do AG acrescentou-se a busca local apresentada no Algoritmo 3.5 ao AGALocador em dois pontos do fluxo normal do AG, após criar a população inicial, melhorando todos os indivíduos e após a criação do novo indivíduo em cada geração.

Algoritmo Genético usando Alocação de Salas e duas Buscas Locais (AGALocadoBL2)

Acrescentou-se a segunda busca local apresentada no Algoritmo 3.10 ao AGAlocadorBL1 no intuito de melhorar ainda mais, se possível, o indivíduo que passou pela primeira busca local.

Algoritmo Genético usando Alocação de Salas com Memória e uma Busca Local (AGAlocacorMEMBL1)

Esse algoritmo consiste apenas em acrescentar o cruzamento que faz uso da estrutura MEM apresentado no Algoritmo 3.7 ao AGAlocadorBL1 formando assim o Algoritmo 3.9 que foi denominado por AGAlocacorMEMBL1.

Algoritmo Genético usando Alocação de Salas com Memória e e duas Buscas Locais (AGAlocacorMEMBL2)

Essa implementação é a mais robusta deste trabalho, não necessariamente a melhor. Acrescentou-se o cruzamento que faz uso da estrutura MEM apresentado no Algoritmo 3.7 ao AGAlocadorBL2.

Todas as variações do AG aqui apresentadas são combinações dos algoritmos apresentados no trabalho de Yang e Jat (2011) visto que o objetivo deste trabalho não é expor um novo algoritmo e sim verificar o real efeito da estrutura de memória MEM sobre o AG.

4.2.3 Parâmetros

Quase todos os algoritmos implementados precisam de parâmetros fixos de entrada para sua execução. Com objetivo de facilitar o armazenamento desses parâmetros para a execução dos experimentos criou-se uma classe denominada `LerParâmetros` que obtém os dados a partir de um XML e cria uma lista de objetos do tipo `Parâmetros` que é passado para todos os métodos que careçam de algum parâmetro.

A seguir tem-se um exemplo de arquivo de parâmetro em formato XML, onde o cometário explica a usabilidade do parâmetro.

```
<parametros
  tamPop="50" <!-- Número de indivíduos na população -->
  tamSel="2" <!-- Número de indivíduos que serão selecionados para o cruzamento -->
  pc="0.9" <!-- Probabilidade de cruzamento simples -->
  pm="0.1" <!-- Probabilidade de mutação -->
  semente="1234485" <!-- Para gerar números aleatórios -->
  nMaxGeração="500" <!-- Número máximo de gerações do AG -->
  tamTorneio="2" <!-- Tamanho do torneio  -->
  nFilhos="1" <!-- Número de filhos a gerar no cruzamento -->
  divisor="20" <!-- Divide as gerações do AG, apenas para imprimir na tela -->
  prob1="1.0" <!-- Probabilidade de usar o movimento 1 na BL1 -->
  prob2="1.0" <!-- Probabilidade de usar o movimento 2 na BL1 -->
  prob3="1.0" <!-- Probabilidade de usar o movimento 3 na BL1 -->
  maxSteps="300" <!-- Número máximo de passos da BL1 -->
  limiteBL="300" <!-- Limite máximo de tempo da BL1 -->
  maxStepsLS2="300" <!-- Número máximo de passos da BL2 -->
  percentualLS2="0.2" <!-- Percentagem de horários para BL2 -->
  alfa="0.2" <!-- Perc. dos melhores indivíduos para construir MEM -->
  beta="0.3"  <!-- Perc. dos eventos para cruzamento com MEM -->
  gama="0.8" <!-- Probabilidade de usar cruzamento com MEM -->
  tal="0.2" <!-- Frequência de atualização da MEM -->
  limiteExec="1000" <!-- Limite máximo de tempo do AG -->
  limiteBL2="60" <!-- Limite máximo de tempo da BL2 -->
  nExecuções="1"/> <!-- Número de vezes que o AG será executado -->
```

Todos esses parâmetros ficam em um mesmo objeto ainda que alguns deles não seja usado em determinado momento. Como a primeira etapa dos experimentos consiste

em usar um conjunto de diferentes parametrizações isso facilitou o processo de execução dos testes.

No próximo capítulo será explicado como foram elaborados os experimentos seguidos de seus resultados com suas respectivas análises.

5 Experimentos e Discussões

Neste capítulo serão apresentados os experimentos realizados a fim de investigar qual são os melhores valores para os parâmetros α, β, γ e τ usados pelos algoritmos que usam a MEM e ainda analisar o desempenho de cada variação do AG implementada sobre as instâncias testadas e verificar o real efeito de se usar a estrutura MEM.

Toda a codificação foi feita em *Scala* sob a versão 2.11.0. A execução dos testes foi realizada em uma máquina cujo processador é Intel® Core™ i3-3217U CPU @ 1.80GHz x 4, com 4 Gb de memória e usando o sistema operacional Ubuntu 14.04.

Foram repetidos neste trabalho os experimentos realizados por Yang e Jat (2011), ou seja, os testes foram separados em duas etapas, onde a primeira consiste em analisar a sensibilidade dos parâmetros necessários para usar a estrutura MEM e a segunda etapa resume-se em comparar o desempenho de todas as variações do AG. De modo a facilitar a comparação dos resultados foi definida a mesma parametrização utilizada pelos autores, sendo esta com os seguintes valores: 50 para o tamanho da população, 50% a probabilidade de acontecer mutação e 80% a probabilidade de usar o cruzamento simples. Assim como no trabalho deles, foi utilizado o conjunto de instâncias PATAT2002 para execução dos experimentos.

A seguir será explicado como foram executados os testes em cada etapa e os respectivos resultados seguidos pela sua análise.

5.1 Primeira etapa: análise de sensibilidade dos parâmetros

O bom desempenho do método de estratégia guiada — que faz uso da estrutura de memória MEM — depende fortemente dos parâmetros e dos operadores genéticos utilizados (YANG; JAT, 2011).

Pode-se dizer então que os parâmetros α, β, γ e τ são de fundamental importância e afetam diretamente o desempenho dos AGs que usam a estrutura MEM visto que α é a porcentagem dos melhores indivíduos que serão selecionados para a construção da MEM, β é a porcentagem dos eventos que serão usados no cruzamento com a MEM, γ é a probabilidade de usar o cruzamento com a MEM e por último τ que é a frequência que a MEM será atualizada.

Para esta primeira etapa dos testes foi usado o conjunto de dados PATAT2002, pelo fato de estas instâncias serem classificadas em fáceis, médias e difíceis de acordo com a sua dificuldade. Deste conjunto de instâncias foram usadas cinco fáceis, três médias e uma difícil. Na Tabela 3 são apresentados os dados das instâncias PATAT2002 retirados

do sítio elaborado por Rossi-Doria et al. (2002a).

Tabela 3 – Dados das instâncias PATAT2002

Fonte: Dados obtidos no sítio de Rossi-Doria et al. (2002a)

Dados da Instância	Fácil	Média	Difícil
Número de eventos	100	400	400
Número de salas	5	10	10
Núemro de recursos	5	5	10
Recursos aproximados por sala	3	3	5
Percertagem de recursos usados	70	80	90
Número de alunos	80	200	400
Máximo de eventos por aluno	20	20	20
Máximo de aluno por evento	20	50	100

Assim como no trabalho de Yang e Jat (2011), executou-se o AGAlocacorMEMBL1 com diferentes configurações de parâmetros que são apresentadas na Tabela 4. Diferentemente do trabalho deles, não foram usados os valores 60, 80 e 100 para o parâmetro τ visto que foram executadas apenas 40 gerações do AGAlocacorMEMBL1 devido a limitação de tempo disponível para a execução dos testes. Vale ressaltar que no trabalho desses autores eles apresentaram 20 como sendo o melhor valor para τ.

Combinando os valores da Tabela 4 obteve-se um total de 240 combinações de parâmetros, além do mais, como foram utilizadas três sementes diferentes para geração dos números aleatórios resultou-se então 720 parametrizações que foram gravadas em um arquivo XML conforme explicado na sessão 4.2.3.

Tabela 4 – Configuração dos parâmetros para o AGAlocacorMEMBL1

Fonte: Adapta de Yang e Jat (2011)

Parâmetro	Valores					
α	0.2	0.4	0.6	0.8		
β	0.1	0.3	0.5	0.7	0.9	
γ	0.0	0.2	0.4	0.6	0.8	1.0
τ	20	40				

O arquivo de parâmetros é generalizado, ou seja, contém valores de todos os parâmetros para todas as variações do AG. Este arquivo foi gerado de forma automática, combinando os valores da Tabela 4 com os demais parâmetros nos quais podemos destacar como sendo os necessários para o AGAlocadorMEMBL1: o tamanho da população, a probabilidade de cruzamento e de mutação, o número máximo de gerações, a probabilidades de usar os movimentos de vizinhança na primeira busca local, número máximo de passos e de tempo para a primeira local e o limite de tempo de execução do AG além dos parâmetros α, β, γ e τ.

Veja a seguir um exemplo de parametrização no formato XML com os valores usados nos testes desta etapa, vale lembrar que desses parâmetros, os únicos que variam são: α, β, γ, τ e a semente.

```
<parametros tamPop="10" tamSel="2" pc="0.8" pm="0.5" semente="-239965617960281284"
nMaxGeração="40" tamTorneio="2" nFilhos="1" divisor="2" prob1="1.0" prob2="1.0"
prob3="1.0" maxSteps="100" limiteBL="60" maxStepsLS2="300" percentualLS2="0.2" alfa="0.2"
beta="0.1" gama="0.0" tal="20.0" limiteExec="60" limiteBL2="60" nExecuções="1"/>
```

Foram realizadas 720 execuções do AGAlocadorMEMBL1, cada uma delas usando uma parametrização armazenada em um arquivo de parâmetros semelhante ao exemplo acima. Dos resultados obtidos dessas execuções, foi feita uma análise para escolher a semente que obteve melhores soluções para serem apresentadas neste trabalho.

Para cada semente, s_1, s_2 e s_3, foi elaborado um arquivo com os melhores resultados da execução do AGAlocadorMEMBL1 com as 240 combinações dos parâmetros para as 9 instâncias. A partir desses arquivos foi calculada a pior, a melhor e a média das soluções encontradas entre as combinações de parâmetros para cada instância a fim de selecionar a semente que em média obteve as melhores resultados. O resultado destes cálculos podem ser visualizados nas Tabelas 5 e 6 onde são apresentados os valores gerados a partir dos resultados obtidos da primeira etapa dos testes.

Tabela 5 – Piores, melhores, e médias das soluções encontradas para as instâncias fáceis (F) do conjunto PATAT2002 com três diferentes sementes (s).

		F1	F2	F3	F4	F5
s_1	Pior	191	22000190	15000199	164	169
	Melhor	60	56	76	61	56
	Média	98,646	91791,11	62612,4	103,025	104,208
s_2	Pior	183	24000202	10000205	5000247	14000227
	Melhor	58	70	47	71	42
	Média	109,167	191791,57	41769,95	20953,21	58443,662
s_3	Pior	135	172	177	160	162
	Melhor	61	74	58	67	57
	Média	97,892	124,121	113,3	114,95	106,037

Tabela 6 – Piores, melhores e médias das soluções encontradas para as instâncias médias (M) e difícil (D) do conjunto PATAT2002 com três diferentes sementes (s).

		M1	M2	M3	D1
s_1	Pior	376000437	357000461	366000427	463001041
	Melhor	165000444	168000455	201000483	400000993
	Média	251275446,6	251512954,28	275242111,22	422888516,84
s_2	Pior	363000503	387000529	376000449	498000850
	Melhor	173000479	177000444	213000487	381000965
	Média	255183808,91	258487947,18	290908812,44	429375946,23
s_3	Pior	317000472	341000514	335000489	485000940
	Melhor	171000448	176000469	213000458	397000968
	Média	244583810,84	248879643,05	274946319,37	452213447,24

Diante destes dados, é possível observar que s_1 obteve os melhores resultados para duas instâncias pequenas e para as três médias. Já s_2 teve melhores resultados para três instâncias pequenas e para a única grande. No entanto, s_1 registrou médias melhores que s_2. Como s_3 não chegou em melhor resultado em nenhum das instâncias, optou-se por apresentar neste trabalho os resultados gerados a partir de s_1.

Tabela 7 – Melhores resultados obtidos através do AGAlocadorMEMBL1 com diferentes combinações de parâmetros para as instâncias fáceis (F), médias (M) e dificeis(D) do conjunto PATAT2002.

α	β	γ	τ	F1	F2	F3	F4	F5	M1	M2	M3	D1
0.2	0.1	0.0	20	76	87	104	61	75	217000422	172000479	206000420	400000993
0.2	0.1	0.0	40	76	87	104	61	75	214000485	172000479	206000420	400000993
0.2	0.1	0.2	20	71	122	103	90	87	211000438	194000497	260000481	410001107
0.2	0.1	0.2	40	79	122	119	75	87	223000451	184000462	265000446	410001107
0.2	0.1	0.8	20	120	144	99	112	81	277000525	321000458	330000433	463001041
0.2	0.1	0.8	40	112	152	101	142	102	279000498	273000450	295000502	416001031
0.2	0.3	0.0	20	76	87	104	61	75	214000485	172000479	206000420	400000993
0.2	0.3	0.0	40	76	87	104	61	75	214000485	172000479	206000420	400000993
0.2	0.3	0.2	20	83	104	76	95	78	181000463	189000489	260000418	410001107
0.2	0.3	0.2	40	79	122	119	75	87	223000451	184000462	265000446	410001107
0.2	0.3	0.8	20	107	153	101	142	116	288000432	273000450	312000500	422000884
0.2	0.3	0.8	40	112	152	101	142	102	279000498	273000450	295000502	422000884
0.2	0.5	0.0	20	76	87	104	61	75	217000454	183000465	206000420	400000993
0.2	0.5	0.0	40	76	87	104	61	75	217000454	183000465	206000420	400000993
0.2	0.5	0.2	20	67	107	94	97	92	213000463	186000453	260000461	410001107
0.2	0.5	0.2	40	79	122	119	75	87	224000457	200000468	265000446	410001107
0.2	0.5	0.8	20	105	136	101	142	116	270000435	282000446	307000434	422000884
0.2	0.5	0.8	40	112	152	101	142	102	279000498	273000450	295000502	422000884
0.6	0.1	0.0	20	76	87	104	61	75	217000454	183000465	206000420	400000993
0.6	0.1	0.0	40	76	87	104	61	75	217000454	183000465	206000420	400000993
0.6	0.1	0.2	20	70	99	110	71	68	178000463	181000446	234000456	410001107
0.6	0.1	0.2	40	79	122	119	75	87	224000457	200000468	265000446	410001107
0.6	0.1	0.8	20	120	148	101	142	108	288000432	302000417	318000442	422000884
0.6	0.1	0.8	40	112	152	101	142	102	279000498	273000450	295000502	422000884
0.6	0.3	0.0	20	76	87	104	61	75	217000454	183000465	206000420	400000993
0.6	0.3	0.0	40	76	87	104	61	75	217000454	183000465	206000420	400000993
0.6	0.3	0.2	20	82	115	104	86	66	176000500	182000441	256000443	410001107
0.6	0.3	0.2	40	79	122	119	75	87	224000457	200000468	265000446	410001107
0.6	0.3	0.8	20	107	112	101	142	116	283000509	302000417	294000479	422000884
0.6	0.3	0.8	40	112	152	101	142	102	279000498	273000450	295000502	422000884
0.6	0.5	0.0	20	76	87	104	61	75	217000454	183000465	206000420	400000993
0.6	0.5	0.0	40	76	87	104	61	75	217000454	183000465	206000420	400000993
0.6	0.5	0.2	20	73	113	92	99	94	203000468	173000465	273000445	410001107
0.6	0.5	0.2	40	79	122	119	75	87	224000457	200000468	265000446	410001107
0.6	0.5	0.8	20	120	130	92	142	109	264000479	295000440	304000455	422000884
0.6	0.5	0.8	40	112	152	101	142	102	279000498	273000450	295000502	422000884

Mesmo escolhendo os valores de apenas uma semente, ainda sim seriam muitas informações a serem apresentadas, então, dos diferentes valores utilizados para os parâmetros escolheu-se para apresentar apenas os resultados com α sendo 0.2 e 0.6; β sendo 0.1, 0.3 e 0.5; γ sendo 0.0, 0.2 e 0.8; e τ sendo 20 e 40. O critério utilizado para escolha desses valores para os parâmetros foram os melhores resultados.

Na Tabela 7 são apresentas as melhores aptidões encontradas pelo AGAlocadorMEMBL1 para as instâncias PATATA2002, onde as inicias F, M e D significam respectivamente fácil, média e difícil. Como na função de aptidão violações de restrições invioláveis são penalizadas multiplicando-as por 1000000, os valores maiores que isto significa que a solução violou restrições invioláveis, por exemplo, o valor 217000422, interpreta-se como 217 violações invioláveis e 422 violações preferenciais.

Analisando os resultados encontrados, selecionou-se como sendo a melhor parametrização $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.0$ e $\tau = 20$, em contraste com o trabalho de Yang e Jat (2011) que afirmaram que quanto maior o valor para γ, 0.8 por exemplo, melhores soluções são encontradas.

Na Figura 11 são ilustrados os gráficos em que são apresentados os efeitos de cada parâmetro no desempenho do AGAlocadorMEMBL1 sobre uma instância fácil. A fim de verificar

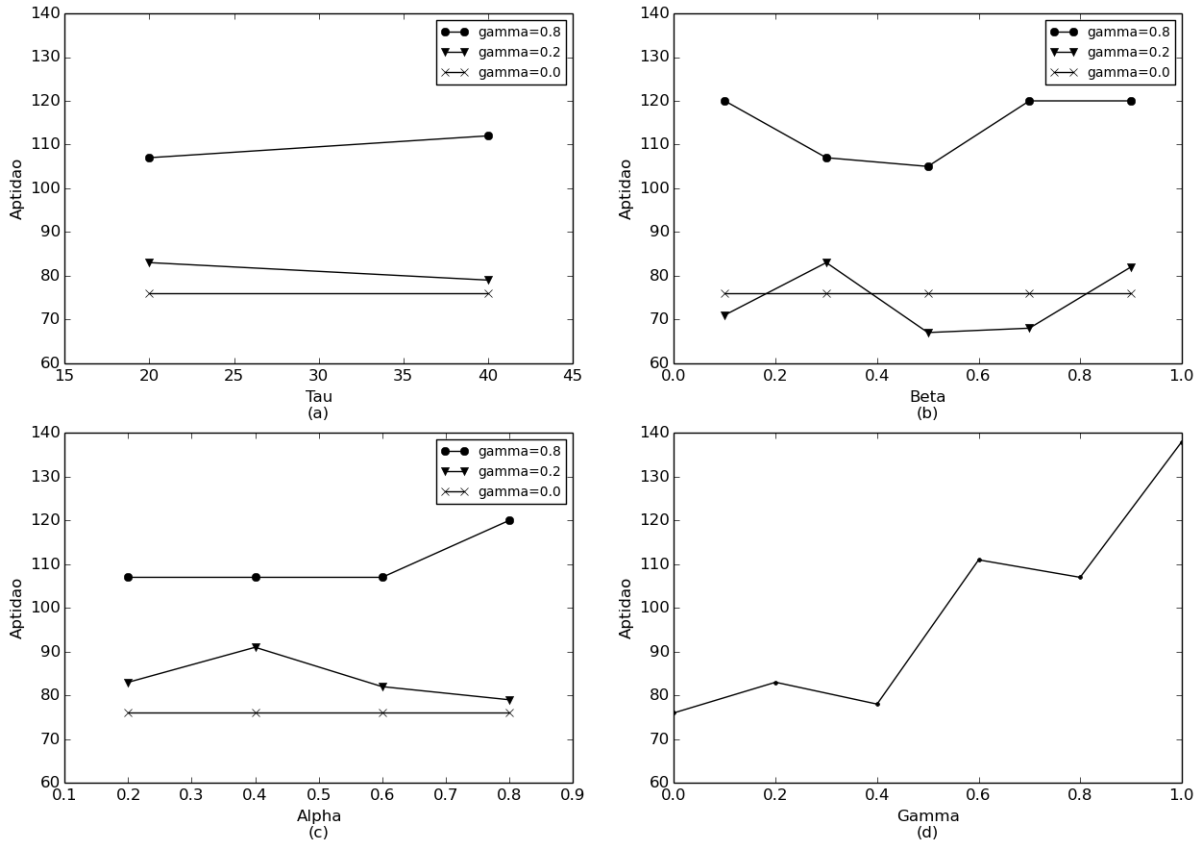

Figura 11 – Comparação dos efeitos de cada parâmetro no desempenho do AGAlocadorMEMBl1 sobre uma instância fácil. (a) variação de τ com $\alpha = 0.2$, $\beta = 0.3$ e $\gamma = \{0.0, 0.2, 0.8\}$. (b) variação de β com $\alpha = 0.2$, $\tau = 20$ e $\gamma = \{0.0, 0.2, 0.8\}$.(c) variação de α com $\beta = 0.3$, $\tau = 20$ e $\gamma = \{0.0, 0.2, 0.8\}$. (d) variação de γ com $\alpha = 0.2$, $\beta = 0.3$ e $\tau = 20$.

o efeito de γ com mais precisão, a variação dele foi ilustrada nas quatro situações da Figura 11. Na situação (a) da Figura 11 é ilustrado a variação de τ com $\alpha = 0.2$, $\beta = 0.3$ e $\gamma = \{0.0, 0.2, 0.8\}$. Neste caso, pode-se notar que quanto menor o valor de γ e de τ melhor o valor da aptidão.

Na Figura 11(b) é mostrado o efeito da variação de β com $\alpha = 0.2$, $\tau = 20$ e $\gamma = \{0.0, 0.2, 0.8\}$, novamente percebe-se que quanto menor o valor de γ melhor o valor da aptidão. É possível notar também que a aptidão é melhor quando o valor de β está entre 0.3 e 0.7. Já na situação (c) da Figura 11 é apresentado o efeito da variação de α com $\beta = 0.3$, $\tau = 20$ e $\gamma = \{0.0, 0.2, 0.8\}$. Mais uma vez é possível notar que com valores menores para γ resulta em aptidão com valor menor. Em relação ao parâmetro α existe uma piora na aptidão ao aumentar o valor desse parâmetro. O efeito da variação de γ é ilustrado na Figura 11(d) com $\alpha = 0.2$, $\beta = 0.3$ e $\tau = 20$, onde pode ser verificado que quanto maior o valor de γ maior o valor da aptidão.

Analisando a Figura 11 verificou-se que não usar o cruzamento que utiliza a MEM resulta em melhores soluções, porém, esse resultado pode ser devido ao pequeno tamanho da população que faz com que tenha poucos eventos armazenados na MEM. Diante disto optou-se por usar $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.2$ e $\tau = 20$ na segunda etapa dos testes a fim de analisar o efeito de usar a estrutura MEM, mesmo que com probabilidade baixa.

Na próxima sessão será apresentado como foi executada a segunda etapa dos testes seguido de seus respectivos resultados e análises.

5.2 Segunda etapa: comparação desempenho dos AGs

A partir da parametrização definida na primeira etapa dos experimentos, executou-se a segunda etapa que consiste em comparar o desempenho de todas as variações do AG implementadas e analisar qual o efeito da estrutura MEM sobre o desempenho do AG.

Nesta segunda etapa foram introduzidas aos testes algumas instâncias do conjunto ITC2007 para verificar como os algoritmos implementados se comportam com instâncias mais recentes e com diferentes restrições. Por restrição de tempo disponível para os testes, foram usadas apenas 5 das 21 instâncias deste conjunto. Na Tabela 8 são apresentados os das instâncias utilizadas obtidos através do sítio de Bonutti, Gaspero e Schaerf (2014).

Tabela 8 – Dados das instâncias ITC2007 onde as siglas significam: D - número de disciplinas, S - número de salas, H - quantidade de horários por dia, DS - quantidade de dias por semana, C - número de currículos, DensConf - densidade dos conflitos (por curso/por aula), DispProf - disponibilidade dos professores (por curso/por aula), AdSalas - adequação das salas (por curso/por aula), OcpSalas - ocupação das salas.

Fonte: Dados obtidos no sítio de Bonutti, Gaspero e Schaerf (2014)

Inst.	D	S	H	DS	C	DensConf curso/aula %	DispProf curso/aula %	AdSalas curso/aula %	OcpSalas %
comp01	30	6	6	5	14	10.35/13.23	94.12/93.09	58.34/60.01	88.89
comp06	108	18	5	5	70	5.16/5.24	76.6/78.32	57.1/58.18	80.23
comp11	30	5	9	5	13	11.45/13.82	93.04/94.19	52.01/52.72	72.01
comp16	108	20	5	5	71	4.95/5.13	80.82/81.54	63.2/63.08	73.21
comp21	94	18	5	5	78	6.18/6.09	80.3/82.38	66.2/66.38	

Para todas as instâncias testadas foram usados os mesmos valores de parâmetros e com três diferentes sementes a fim de ter uma comparação válida sobre o desempenho dos algoritmos. Com exceção do limite de tempo máximo de execução onde para as instâncias fáceis do conjunto PATAT2002 foi dado um tempo de 300s e para as demais instâncias, seja do conjunto PATAT2002 ou do conjunto ITC2007, foi dado um tempo máximo 900s. Os demais parâmetros fixos podem ser vistos a seguir:

```
<parametros tamPop="50" tamSel="2" pc="0.8" pm="0.5" semente="-239965617960281284"
nMaxGeração="50000" tamTorneio="2" nFilhos="1" divisor="2" prob1="1.0" prob2="1.0"
prob3="1.0" maxSteps="1000" limiteBL="300" maxStepsLS2="300" percentualLS2="0.2"
alfa="0.2" beta="0.3" gama="0.2" tal="20" limiteExec="900" limiteBL2="300"
nExecuções="1"/>
```

As oito variações do AG foram executadas com as mesmas três sementes s_1, s_2, e s_3 da primeira etapa dos experimentos. Como na etapa anterior a melhor semente foi definida de acordo com as instâncias do conjunto PATAT2002, resolveu-se que nesta segunda etapa as sementes seriam analisadas com base nas cinco instâncias do conjunto ITC2007.

Calculou-se então, a pior, a melhor e a média das melhores e piores soluções encontradas durante as execuções dos oito algoritmos para as cinco instâncias. O resultado destes cálculos podem ser visualizados na Tabela 9 onde são apresentados os valores gerados a partir dos resultados obtidos da segunda etapa dos testes.

Tabela 9 – Piores, melhores e médias das melhores e piores soluções encontradas pelas oito variações do AG para as instâncias ITC2007 com três diferentes sementes(s).

			Melhor solução	Pior solução
s1	comp01	Pior	4000190	17000175
		Média	3500663,75	6750687,5
		Melhor	2002273	2002273
	comp06	Pior	161001184	170001198
		Média	107877525,875	135752549,75
		Melhor	89001062	93006465
	comp11	Pior	1693	1693
		Média	466,625	495,125
		Melhor	20	20
	comp16	Pior	127001152	137001184
		Média	81877239,625	112752311,125
		Melhor	61005602	63005605
	comp21	Pior	116001121	134001132
		Média	71877222,625	98377229
		Melhor	43005491	43005492
s2	comp01	Pior	8002025	14000196
		Média	4500613,5	6125636,125
		Melhor	2002226	2002226
	comp06	Pior	161001218	179001199
		Média	104752512,375	136502594,125
		Melhor	79001021	93006320
	comp11	Pior	2078	3001850
		Média	536,375	625588,75
		Melhor	28	28
	comp16	Pior	140001160	144001178
		Média	84627320,5	121127314,25
		Melhor	59005699	61005703
	comp21	Pior	126001091	294006432
		Média	72752233,5	130377295,75
		Melhor	47005564	47005577
s3	comp01	Pior	6002041	131002270
		Média	4125628,5	23500683,5
		Melhor	3002264	3002264
	comp06	Pior	154001197	168001207
		Média	103877610,5	132752656,25
		Melhor	83007331	87007323
	comp11	Pior	154001197	168001207
		Média	57288805,1428571	84610282,9642857
		Melhor	3002264	3002264
	comp16	Pior	154001197	168001207
		Média	70288287,1428572	83734735,9642857
		Melhor	3002264	3002264
	comp21	Pior	28	135
		Média	51984939,8942308	82369665,9086538
		Melhor	154001197	287006755

Analisado os resultados apresentados na Tabela 9 pode-se observar que apesar de s_2 ter gerado as melhores soluções para *comp01* e *comp06*, as médias das melhores soluções encontradas paras as cinco instâncias foram as piores se comparadas com as melhores soluções encontradas a partir de s_1 e s_3, dessa forma, descartou-se s_2.

Confrontando os resultados de s_1 e s_3, nota-se que além de s_1 ter gerado as melhores soluções para as instâncias *comp11* e *comp21*, das melhores aptidões encontradas pelos algoritmos implementados, s_1 obteve as menos piores soluções para as instâncias *comp01*, *comp11* e *comp16* enquanto que s_3 encontrou a melhor solução apenas para a instância *comp16*. Sendo assim, selecionou-se s_1 como sendo a melhor semente novamente, ou seja, os resultados que serão apresentados a seguir são os valores obtidos a partir de s_1.

Serão apresentados primeiramente os resultados e as análises dos experimentos usando as instâncias do conjunto PATAT2002 e, em seguida, serão apresentados os resultados e as análises

dos testes sobre as instâncias do conjunto ITC2007.

5.2.1 Resultados e análises para as instâncias PATAT2002

Nas Tabelas 10 e 11 são apresentados os dados da população resultante da execução de cada variação do algoritmo genético para as instâncias fáceis (F), médias (M) e difíceis (D) do conjunto PATAT2002. Os dados da Tabelas 10 consistem na melhor aptidão encontrada ao final da execução do AG, a média e o desvio padrão entre as aptidões dos indivíduos pertencentes à população da última geração do AG, e por fim a pior solução da população final. Na Tabela 11 não foi possível ser apresentado a média e o desvio padrão, pois o cálculo foi realizado com números muito grandes e o tipo da variável não estava preparado para receber o valor então o resultado do cálculo ficou errado.

Analisando os resultados apresentados nas Tabelas 10 e 11 nota-se que o AGAlocador obteve a melhor desempenho para a maioria destas instâncias. O AGAlocadorMEM também resultou em boas soluções em comparação com os demais algoritmos.

Tabela 10 – Melhor, média, desvio padrão e pior aptidão da população resultante da execução de cada hibridização do AG para as instâncias fáceis (F) do conjunto PATAT2002

Instâncias	Algoritmo	Melhor	Média	Desvio Padrão	Pior
F1	AGBasico	40000021	4.000002126E7	1.82	40000034
	AGBasicoMEM	44000017	4.486002074E7	6020026.18	87000204
	AGAlocador	12	12.0	0.0	12
	AGAlocadorMEM	17	40018.24	280002.25	2000034
	AGAlocadorBL1	40	62.22	11.05	91
	AGAlocadorBL2	22	52.74	11.98	70
	AGAlocadorMEMBL1	11	120061.92	839999.44	6000058
	AGAlocadorMEMBL2	22	54.28	12.69	71
F2	AGBasico	47000014	4.704001422E7	280001.54	49000025
	AGBasicoMEM	50000021	5.0000021E7	0.0	50000021
	AGAlocador	20	20.0	0.0	20
	AGAlocadorMEM	17	17.0	0.0	17
	AGAlocadorBL1	53	77.54	11.73	93
	AGAlocadorBL2	51	77.06	12.25	95
	AGAlocadorMEMBL1	57	81.62	11.14	96
	AGAlocadorMEMBL2	48	72.96	11.41	91
F3	AGBasico	45000027	4.504002804E7	280000.423	47000031
	AGBasicoMEM	37000011	3.7000011E	0.0	37000011
	AGAlocador	14	14.44	3.08	36
	AGAlocadorMEM	22	22.0	0.0	22
	AGAlocadorBL1	41	60.08	8.23	73
	AGAlocadorBL2	20	54.36	13.82	96
	AGAlocadorMEMBL1	37	20066.44	140011.65	1000148
	AGAlocadorMEMBL2	30	61.22	10.39	75
F4	AGBasico	20000023	2.00200231E7	140000.7	21000028
	AGBasicoMEM	23000017	2.404002146E7	7280024.363	75000192
	AGAlocador	15	20015.28	140001.96	1000029
	AGAlocadorMEM	23	23.0	0.0	23
	AGAlocadorBL1	39	62.74	10.0	107
	AGAlocadorBL2	31	57.08	11.51	72
	AGAlocadorMEMBL1	35	63.38	10.56	78
	AGAlocadorMEMBL2	34	20059.36	140001.95	1000073
F5	AGBasico	32000005	3.206000532E7	420002.24	35000021
	AGBasicoMEM	31000011	3.108001102E7	560000.14	35000012
	AGAlocador	14	14.0	0.0	14
	AGAlocadorMEM	11	11.0	0.0	11
	AGAlocadorBL1	29	49.06	9.23	62
	AGAlocadorBL2	24	46.02	10.53	68
	AGAlocadorMEMBL1	32	56.76	10.99	98
	AGAlocadorMEMBL2	20	45.64	10.95	63

Tabela 11 – A melhor e a pior aptidão da população resultante da execução de cada hibridização do AG para as instâncias médias (M) e difíceis (D) do conjunto PATAT2002

Instâncias	Algoritmo	Melhor	Pior
M1	AGBasico	240000338	241000338
	AGBasicoMEM	264000358	271000363
	AGAlocador	49000365	58000364
	AGAlocadorMEM	114000411	118000410
	AGAlocadorBL1	79000426	186000476
	AGAlocadorBL2	137000387	196000404
	AGAlocadorMEMBL1	89000454	186000438
	AGAlocadorMEMBL2	48000469	195000449
M2	AGBasico	276000359	276000359
	AGBasicoMEM	295000380	297000380
	AGAlocador	46000350	49000349
	AGAlocadorMEM	119000380	126000373
	AGAlocadorBL1	68000422	190000486
	AGAlocadorBL2	115000348	206000411
	AGAlocadorMEMBL1	95000437	198000506
	AGAlocadorMEMBL2	107000410	208000511
M3	AGBasico	273000329	275000329
	AGBasicoMEM	342000377	344000378
	AGAlocador	50000380	53000381
	AGAlocadorMEM	119000364	757000518
	AGAlocadorBL1	123000443	224000474
	AGAlocadorBL2	131000470	239000446
	AGAlocadorMEMBL1	135000439	224000481
	AGAlocadorMEMBL2	152000405	238000425
M4	AGBasico	272000317	277000322
	AGBasicoMEM	317000357	318000357
	AGAlocador	52000363	57000356
	AGAlocadorMEM	119000407	127000402
	AGAlocadorBL1	84000436	182000437
	AGAlocadorBL2	78000373	196000458
	AGAlocadorMEMBL1	83000448	199000418
	AGAlocadorMEMBL2	119000431	197000398
M5	AGBasico	290000352	290000352
	AGBasicoMEM	316000371	320000372
	AGAlocador	40000370	40000381
	AGAlocadorMEM	67000345	68000346
	AGAlocadorBL1	88000490	153000450
	AGAlocadorBL2	102000368	158000455
	AGAlocadorMEMBL1	100000493	154000490
	AGAlocadorMEMBL2	92000435	162000369
D1	AGBasico	438000729	438000732
	AGBasicoMEM	487000924	488000927
	AGAlocador	105000721	109000701
	AGAlocadorMEM	154000809	164000805
	AGAlocadorBL1	293000954	419001019
	AGAlocadorBL2	291000840	419000933
	AGAlocadorMEMBL1	293000954	419001019
	AGAlocadorMEMBL2	291000840	419000933
D2	AGBasico	426000822	430000829
	AGBasicoMEM	466000798	903000908
	AGAlocador	101000662	102000668
	AGAlocadorMEM	163000882	169000858
	AGAlocadorBL1	288000930	401001032
	AGAlocadorBL2	288000848	388000821
	AGAlocadorMEMBL1	288000930	401001032
	AGAlocadorMEMBL2	288000848	388000821

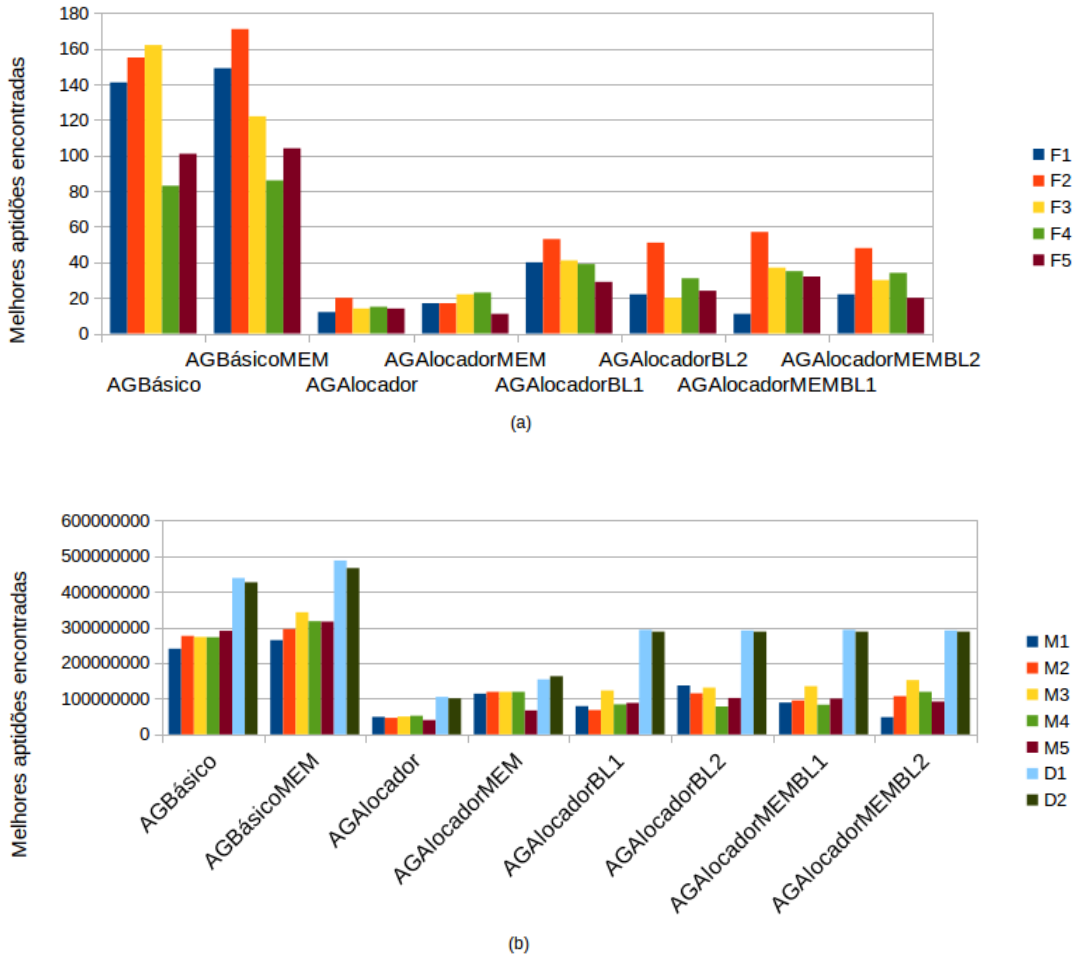

Figura 12 – Comparação das melhores aptidões encontradas pelas hibridizações do AG sobre as instâncias fáceis (F), médias (M), e difíceis (D) do conjunto PATAT2002

Na Figura 12 são ilustrados os gráficos onde são comparadas as melhores aptidões encontradas pelas hibridizações do AG para as instâncias PATAT2002. É importante ressaltar que na Figura 12(a), para melhor visualização no gráfico, a função aptidão foi alterada penalizando apenas em 3 as violações invioláveis em vez de 1000000, por exemplo, a aptidão final de F1 foi 40000021, porém, no gráfico foi representada com $40 \times 3 + 21 = 141$.

Analisando os gráficos da Figura 12 percebe-se quão melhores foram as aptidões resultantes do AGAlocador e do AGALocadorMEM em relação aos demais algoritmos. O AGBasico e o AGBasicoMEM foram os algoritmos que obtiveram os piores resultados.

Em testes anteriores verificou-se que as buscas locais são altamente eficientes, veja por exemplo, para uma instância fácil o AGAlocador encontrou na população inicial um melhor indivíduo avaliado em 24000229, um pior indivíduo com aptidão igual a 53000160 cuja média entre as aptidões dos indivíduos da população inicial foi 36980210.52. Usando a mesma semente, o AGAlocadorBL1 resultou na população inicial a melhor solução avaliada em 44, o pior indivíduo com aptidão igual a 153 cuja a média entre as aptidões foi 98.66. No entanto, os AGs hibridizados tanto com a primeira busca local quanto com as duas não obteve soluções melhores que as

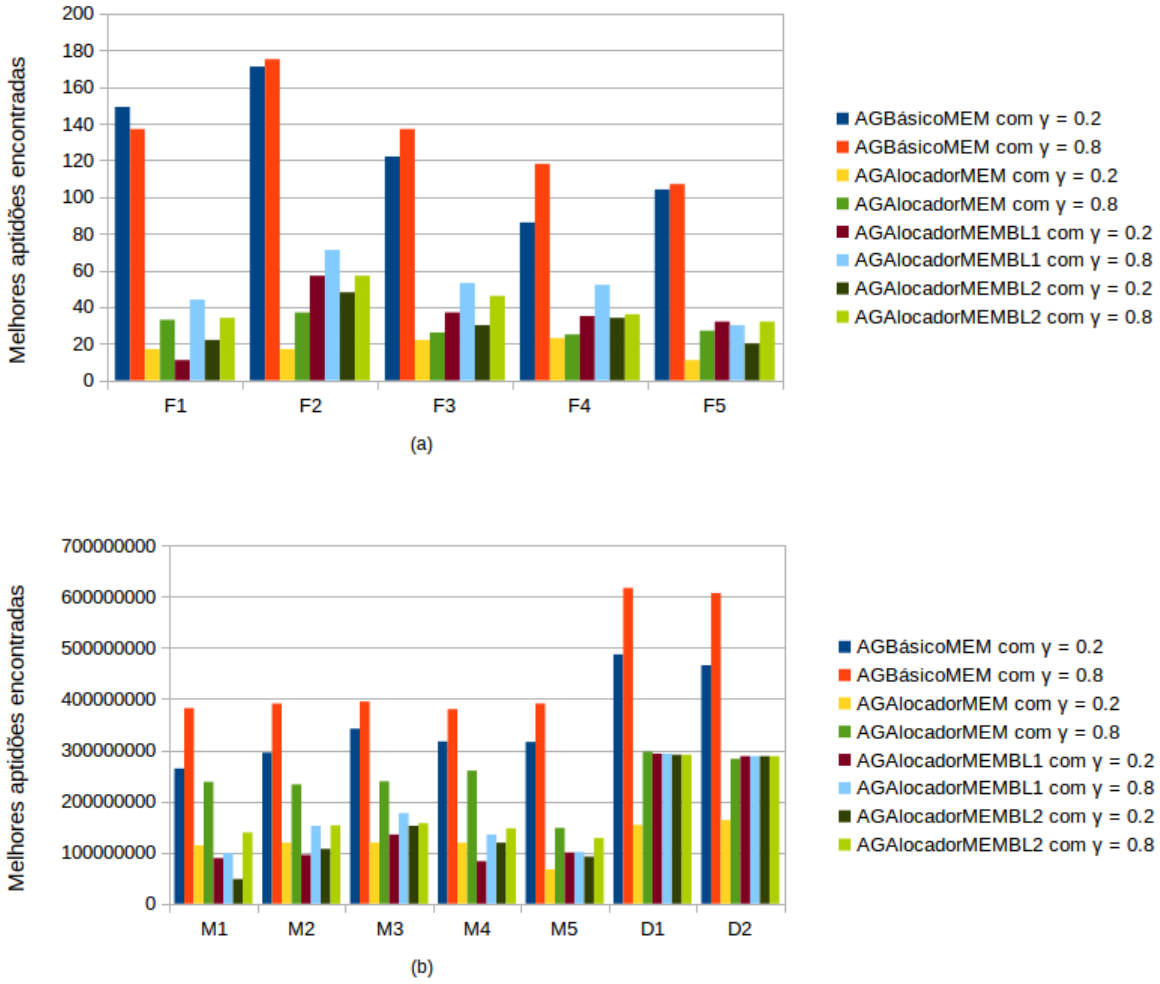

Figura 13 – Comparação entre as hibridizações do AG com a MEM sobre as instâncias fáceis (F), médias (M), e difíceis (D) do conjunto PATAT2002 com $\gamma = 0.2$ e $\gamma = 0.8$

soluções dos AGs que não usam BL1 ou BL2 como era esperado.

Esses resultados inesperados podem ter ocorrido pelo fato de ter deixado um limite de tempo alto para as buscas locais quando se comparado com o limite de tempo máximo de execução do AG, fazendo com que na maior parte, senão todo o tempo de execução, ficasse executando apenas as buscas locais não sobrando tempo para futuras gerações do AG.

Comparando-se o AGBasico com o AGBasicoMEM, AGAlocador com o AGAlocadorMEM, o AGAlocadorBL1 com o AGALocadorMEMBL1 e o AGALocadorBL2 com o AGALocadorMEML2 tem-se 48 comparações das quais 12 a MEM melhorou o desempenho do AG, 31 a MEM piorou a performance do AG e 5 a MEM não fez diferença. Essas últimas 5 são relacionadas aos AGs que usam pelo menos a BL1, provavelmente a MEM não fez diferença pelo fato a busca local ter excedido o limite de tempo de execução.

Como os resultados dos algoritmos que usam a estrutura MEM não foram os melhores, como era esperado, resolveu-se executar esses algoritmos usando o parâmetro $\gamma = 0.8$, assim como no trabalho de Yang e Jat (2011), com intuito de verificar se ao usar com mais frequência

o cruzamento com a MEM a performance das hibridizações do AG é melhorada.

Na Figura 13 são ilustrados os gráficos onde são comparadas as melhores aptidões encontradas pelas variações do AG que usam a MEM tanto com $\gamma = 0.2$ quanto com $\gamma = 0.8$. De forma análoga a Figura 12(a), a função aptidão foi alterada penalizando apenas em 3 as violações invioláveis para melhor visualização do gráfico na Figura 13(a).

A partir Figura 13 pode-se afirmar que ao aumentar a probabilidade de usar o cruzamento com a MEM piorou a maioria das soluções encontradas pelos algoritmos. Em alguns casos que a mudança de parametrização não fez diferença, são os casos em que a primeira busca local ou as duas excederam o limite máximo de tempo de execução.

Diante desses resultados, conclui-se que para as instâncias PATAT2002 a estrutura MEM piorou o desempenho dos algoritmos, o que não era esperado, de acordo com os resultados apresentado por Yang e Jat (2011). A seguir serão apresentados os resultados e as análises dos experimentos realizados sobre o conjunto de instâncias ITC2007.

5.2.2 Resultados e análises para as instâncias ITC2007

Todas as variações do AG foram executadas com o mesmos parâmetros para cinco instâncias do conjunto ITC2007. Na Tabela 12 são apresentados os dados da população resultante da execução de cada hibridização do AG. Estes dados, assim como os das Tabelas 10 e 11 consistem na melhor solução encontrada pela variação do AG, na média e no desvio padrão entre as aptidões da população resultante e por fim, aptidão do pior indivíduo dessa população. Onde aparecer (–) é a mesma situação da Tabela 11, como o cálculo foi realizado com valores muito alto, a variável não armazenou o valor correto.

Comparando as aptidões apresentadas na Tabela 12 tem-se uma oscilação entre as variações do AG que obteve a melhor solução, no entanto, verificou-se que o AGBasico encontrou o melhor resultado para as instâncias *comp16* e *comp21*. Este algoritmo encontrou também a segunda melhor solução para as instâncias *comp01* e *comp06*, o que faz concluir que o AGBasico obteve o melhor desempenho sobre as instâncias ITC2007.

Pode-se observar que o AGBasico obteve o melhor desempenho também através da Figura 14 que ilustra os gráficos onde são comparadas as melhores soluções encontradas pelas variações do AG. Uma observação a ser feita sobre a Figura 14 é que a barra que representa a instância *comp01* não aparece no gráfico pois sua aptidão é menor que 100 para todos os algoritmos.

As hibridizações do AG com a primeira busca local ou as duas obtiveram as soluções mais próximas das soluções obtidas através do AGBasico, isso apenas na população inicial pelo fato de o tempo das buscas locais terem excedido o tempo de limite máximo de execução. Diante disto acredita-se que se disponibilizar um tempo maior de modo a permitir novas gerações os algoritmos hibridizados com buscas locais teriam melhor desempenho.

Assim como para as instâncias PATAT2002, também foram executadas as hibridizações do AG que utilizam a estrutura MEM usando o parâmetro $\gamma = 0.8$ para as instâncias ITC2007

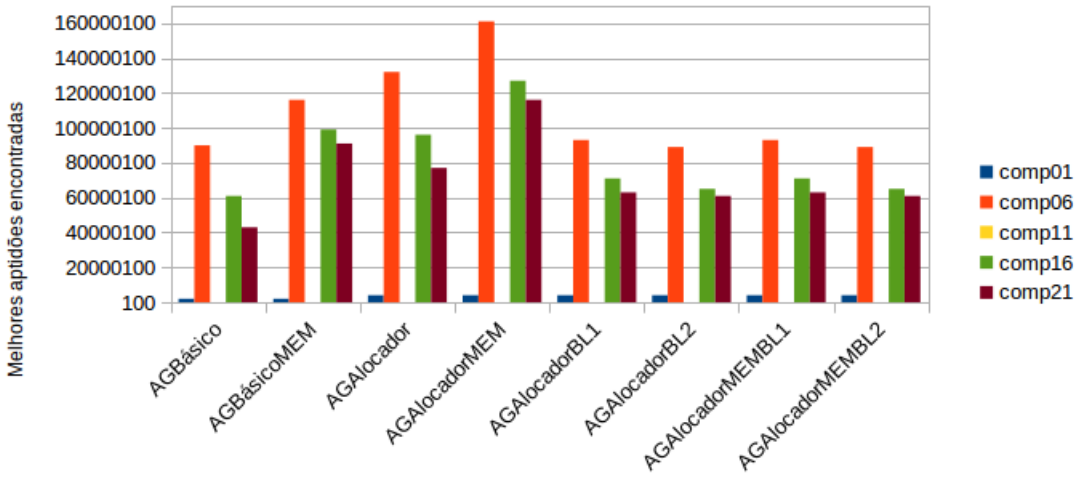

Figura 14 – Comparação das melhores aptidões encontradas pelas hibridizações do AG sobre as instâncias do conjunto ITC2007

Tabela 12 – Melhor, média, desvio padrão e pior aptidão da população resultante da execução de cada hibridização do AG para as instâncias do conjunto ITC2007

Instância	Algoritmo	Melhor	Média	Desvio Padrão	Pior
comp01	AGBasico	2002338	2042338.12	280000.84	4002344
	AGBasicoMEM	2002273	2002273.0	0.0	2002273
	AGAlocador	4000037	4000037.0	0.0	4000037
	AGAlocadorMEM	4000067	4020067.84	140001.45	5000078
	AGAlocadorBL1	4000167	4480196.26	499589.06	5000199
	AGAlocadorBL2	4000120	4660164.32	681475.32	8000180
	AGAlocadorMEMBL1	4000190	4800199.12	916511.19	9000214
	AGAlocadorMEMBL2	4000118	5060157.68	1826582.2	17000175
comp06	AGBasico	90006462	6247119.98	300001.3934	93006465
	AGBasicoMEM	116007127	1.1732713622E8	733209.969	119007143
	AGAlocador	132001146	1.348411115E8	3413852.252	144001142
	AGAlocadorMEM	161001184	1.6304118896E8	3621935.361	170001198
	AGAlocadorBL1	93001082	–	–	144001162
	AGAlocadorBL2	89001062	–	–	136001063
	AGAlocadorMEMBL1	93001082	–	–	144001162
	AGAlocadorMEMBL2	89001062	–	–	136001063
comp11	AGBasico	1653	1653.0	0.0	1653
	AGBasicoMEM	1693	1693.0	0.0	1693
	AGAlocador	20	20.0	0.0	20
	AGAlocadorMEM	33	33.0	0.0	33
	AGAlocadorBL1	85	109.32	12.91	131
	AGAlocadorBL2	72	108.92	12.56	127
	AGAlocadorMEMBL1	90	122.6	16.06	143
	AGAlocadorMEMBL2	87	119.3	15.12	161
comp16	AGBasico	61005602	6.480560706E7	447211.520	63005605
	AGBasicoMEM	99005859	1.030658595E8	903546.512	103005857
	AGAlocador	96001096	1.018011287E8	1697057.559	115001109
	AGAlocadorMEM	127001152	1.3114117992E8	10220005.912	137001184
	AGAlocadorBL1	71001099	–	–	123001227
	AGAlocadorBL2	65001005	–	–	119001140
	AGAlocadorMEMBL1	71001099	–	–	123001227
	AGAlocadorMEMBL2	65001005	–	–	119001140
comp21	AGBasico	43005491	4.400549612E7	1.336223447079	43005492
	AGBasicoMEM	91005712	5926370.48	315595.874	92005717
	AGAlocador	77001059	8.130106828E7	754984.763	80001063
	AGAlocadorMEM	116001121	1.2196124384E8	1.16062957953	134001132
	AGAlocadorBL1	63001139	3021783.42	–	111001096
	AGAlocadorBL2	61001060	221703.04	–	108001118
	AGAlocadorMEMBL1	63001139	3021783.42	–	111001096
	AGAlocadorMEMBL2	61001060	221703.04	–	108001118

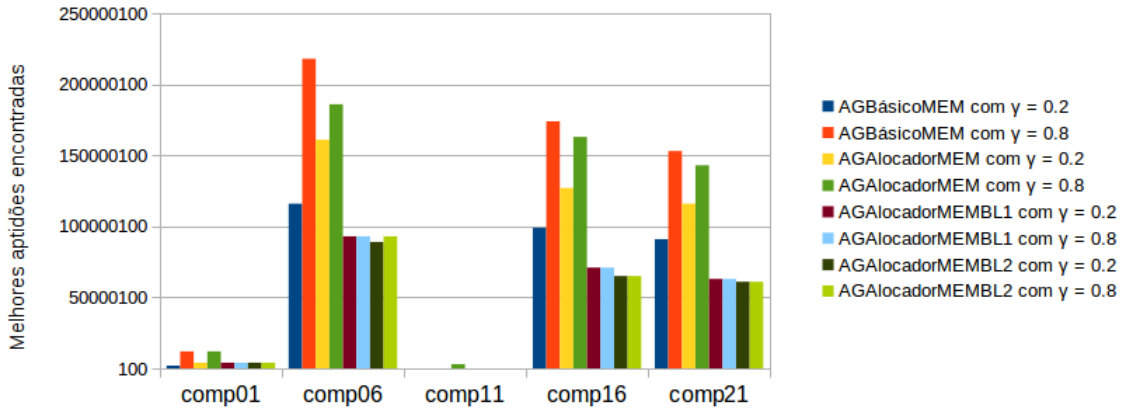

Figura 15 – Comparação entre as hibridizações do AG com a MEM sobre as instâncias ITC2007 com $\gamma = 0.2$ e $\gamma = 0.8$

a fim de verificar se essa alteração melhorara o desempenho desses algoritmos.

Na Figura 15 é ilustrado o gráfico onde são comparados os resultados das variações do AG que utilizam a MEM tanto com $\gamma = 0.2$ quanto com $\gamma = 0.8$. Como a instância *comp11* é mais fácil que as demais, a maioria dos algoritmos encontraram soluções menores que 100, por isso quase todas as barras que representam estas instâncias não aparecem no gráfico ilustrado na Figura 15.

É possível perceber através do gráfico da Figura 15 que ao aumentar a probabilidade de cruzamento com a MEM as soluções encontradas pelos os AGs violaram mais restrições ainda, logo, pode-se afirmar que hibridizar o AG com a MEM não melhora a performance do algoritmo genético também para as instâncias ITC2007.

A seguir serão apresentados os desempenhos de cada variação do AG ao longo de suas gerações sobre uma instância fácil do conjunto PATAT2002.

5.2.3 Performances dos algoritmos

Durante a execução dos algoritmos foram armazenados a cada geração a melhor, a pior, a média e o desvio padrão das soluções. Com esses dados foram elaborados gráficos onde são ilustradas as performances das hibridizações do AG sobre a instância fácil F1 do conjunto PATAT2002. Estes gráficos são ilustrados nas Figuras 16 17, 18 e 19, neles são registradas as melhores, as piores e as médias das aptidões dos indivíduos pertencentes às populações ao longo das gerações do algoritmo.

Na Figura 16 é ilustrado o gráfico onde é mostrado o desempenho das 1000 primeiras gerações do AGBasico e do AGBasicoMEM. É possível observar que as piores soluções do segundo algoritmo são bem mais diversificadas do que as do primeiro. Como durante o processo de atualização os filhos substituem os piores indivíduos a cada geração, isso indica que cruzamento com a MEM fez com que gerasse soluções piores do que o pior indivíduo da população na maioria das gerações. Ao contrário do AGBasico, onde mesmo as piores soluções incluídas

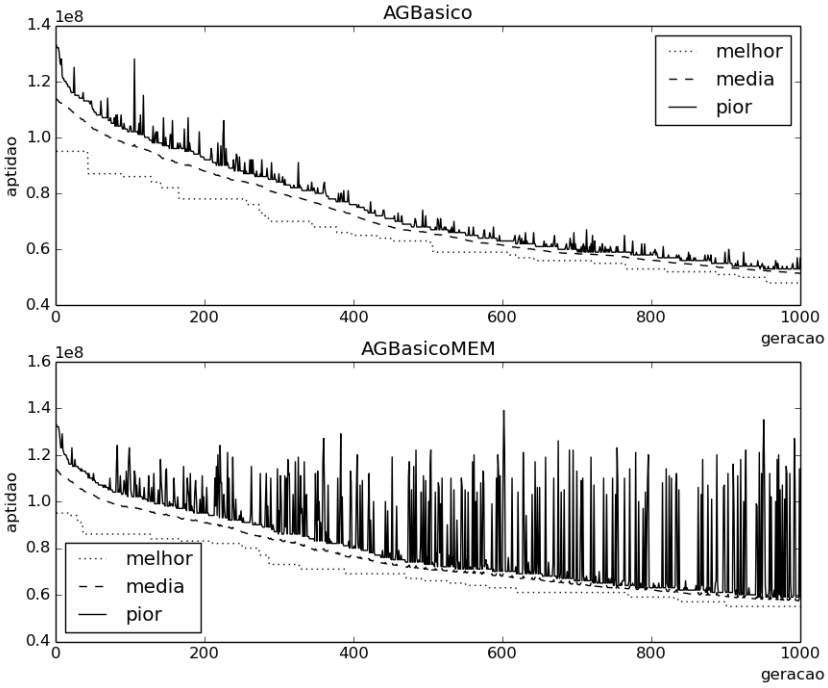

Figura 16 – Desempenho do AGBasico e do AGBasicoMEM sobre a instância fácil F1 do conjunto PATAT2002

foram diminuindo as penalidades ao longo das gerações tendendo a igualar com a média.

Já na Figura 17 é ilustrado o desempenho das 1000 primeiras gerações do AGAlocador e do AGAlocadorMEM. Semelhante aos algoritmos da Figura 16 percebe-se que o cruzamento com a MEM gerou soluções bem piores que o pior indivíduo da população na maioria das gerações. O AGAlocador durante as primeiras gerações, também diverge bastante as piores soluções, no

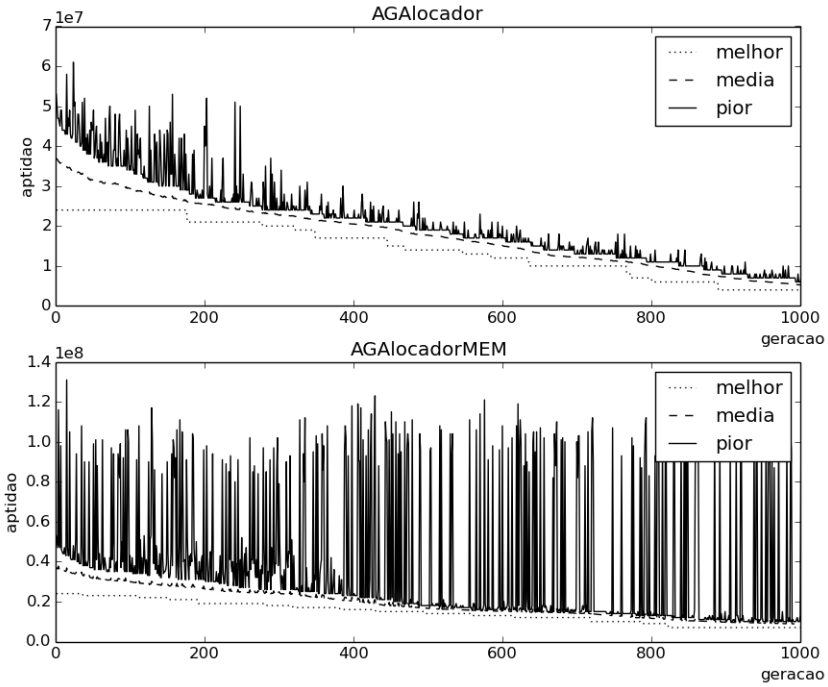

Figura 17 – Desempenho do AGAlocador e do AGAlocadorMEM sobre a instância fácil F1 do conjunto PATAT2002

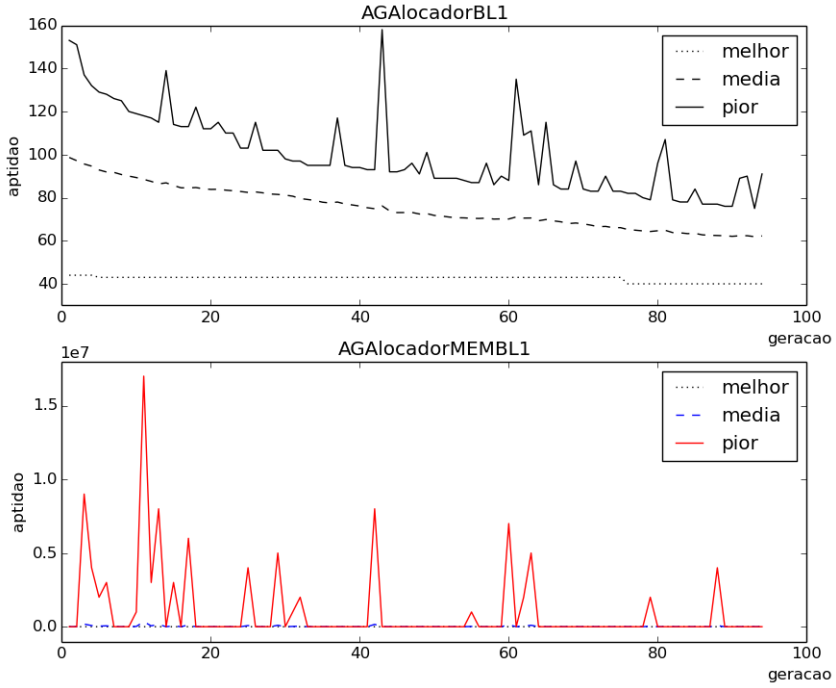

Figura 18 – Desempenho do AGAlocadorBL1 e do AGALocadorMEMBL1 sobre a instância fácil F1 do conjunto PATAT2002

entanto, a partir de um certo ponto ele começa a convergir e as piores soluções não estão tão acima da média. Pode-se observar, comparando essas duas figuras, que o método de alocação de salas fez com que as aptidões caíssem mais rápido.

Na Figura 18 é ilustrado o desempenho das gerações que AGAlocadorBL1 e o AGAloca-

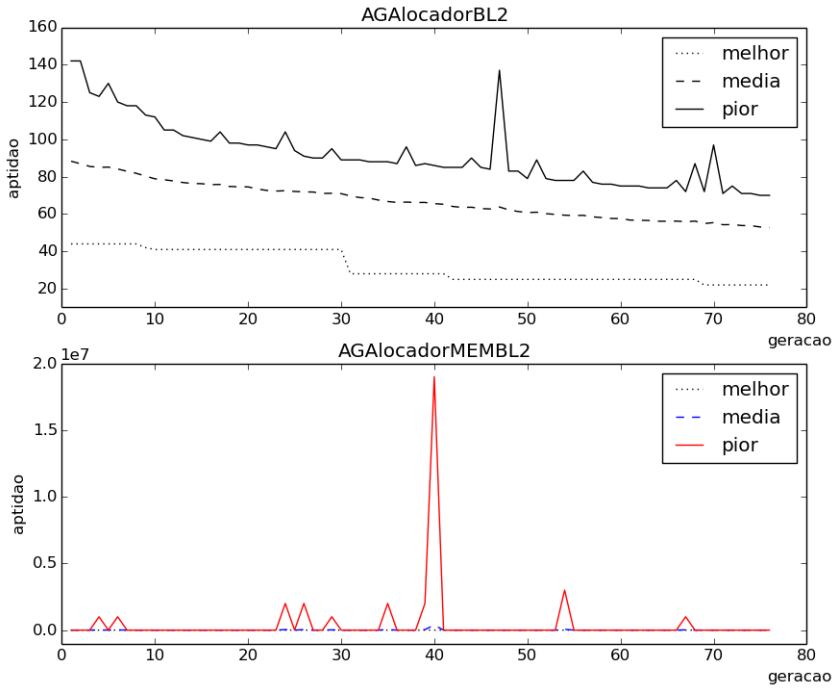

Figura 19 – Desempenho do AGAlocadorBL2 e do AGAlocadorMEMBL2 sobre a instância fácil F1 do conjunto PATAT2002

dorMEMBL1 conseguiram alcançar no tempo disponibilizado. Observe que por causa do tempo gasto com a busca local, o AG não alcançou nem 100 gerações, porém, conseguem eliminar as violações invioláveis já na população inicial. Durante as gerações do AGAlocadorBL1MEM, foram geradas soluções que violaram restrições invioláveis, enquanto que a melhor solução já estava apenas com restrições preferencias, por isso a dispersão dos resultados.

Na Figura 19 é ilustrado o desempenho das gerações que AGAlocadorBL2 e o AGAlocadorMEMBL2 conseguiram alcançar no tempo disponibilizado. Usando as duas buscas locais o número de gerações foi ainda menor, entretanto, ao comparar os gráficos das figuras 18 e 19 observa-se uma pequena melhora na performance do AG ao incluir a segunda busca local.

Através destes gráficos foi possível visualizar a performance dos algoritmos durante a execução e não apenas baseando na solução final. No próximo capítulo serão apresentadas as conclusões deste trabalho bem como sugestões para trabalhos futuros.

6 Conclusão

Neste capítulo serão apresentadas as conclusões que foram alcançadas através deste trabalho e ainda algumas propostas de trabalhos futuros com intuito de dar continuidade a esta pesquisa.

6.1 Conclusões

Após realizada a primeira etapa dos experimentos verificou-se que os algoritmos se desenvolvem melhor quando não se utiliza o cruzamento com a MEM, ou seja, $\gamma = 0.0$. Sendo assim, os demais parâmetros não seriam necessários pois a MEM não seria usada. Como os parâmetros utilizados nesta etapa foram pequenos, tais como uma população de apenas 10 indivíduos, o que pode ter influenciado nos resultados, definiu-se como sendo a melhor parametrização para os algoritmos que usam a estrutura de memória MEM: $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.2$ e $\tau = 20$ para serem utilizados na segunda etapa dos experimentos.

Como os algoritmos implementados são estocásticos por natureza, o ideal seria executá-los com outras sementes a fim de analisar possíveis soluções diferentes. Além do mais, poderia também disponibilizar um limite de tempo de execução maior para as instâncias mais difíceis, principalmente para os algoritmos hibridizados com as buscas locais, a fim te possibilitar a execução das buscas locais juntamente com algumas gerações do AG. Neste trabalho isso não foi possível devido a limitação de tempo disponível para a execução dos testes.

Mas diante dos resultados obtidos nos experimentos, pode-se concluir que o AGBasico, apesar de possuir um grau de aleatoriedade bem maior que os demais algoritmos e de ser o mais simples deles, alcançou os melhores resultados para as instâncias do conjunto ITC2007. Quanto a MEM, conclui-se que acrescentar a estrutura MEM ao AGBasico piorou a performance deste algoritmo.

O AGAlocador foi o algoritmo que conquistou as melhores soluções para as instâncias PATAT2002. Em outros testes, percebeu-se que ao usar o método para a alocação de salas, de forma que a construção do cromossomo não fosse totalmente aleatória, diminuía bastante as penalidades da população inicial, o que contribuiu para este resultado. No que diz respeito a MEM, acrescentá-la ao AGAlocador também fez com que piorasse o desempenho deste algoritmo.

As buscas locais mostraram ser altamente eficiente, diminuindo bastante a penalidade do indivíduo quando nele aplicada. No entanto, como é um método de exaustão, pois tenta todas as possibilidades de movimento enquanto acontece alguma melhora na aptidão, tem um custo de tempo razoavelmente grande, principalmente quando se comparado ao limite máximo de tempo de execução aplicado aos algoritmos. Acredita-se que por falta de tempo as hibridizações do AG com as buscas locais não obtiveram melhores resultados.

Em relação a estrutura de dados MEM, acrescentá-la a alguma variação do algoritmo

genético melhorou o desempenho deste em alguns casos, porém, em sua maioria a hibridização com a MEM piorou o desempenho do AG, mesmo o AGBasico.

A menos que tenha algum erro na interpretação e/ou na implementação desta técnica de construção guiada, ao contrário de Yang e Jat (2011), chegou-se a conclusão que utilizar a estrutura de memória MEM não fez com que resultasse em soluções com menos violações de restrições para o PHCU combinando-a com o Algoritmo Genético.

6.2 Trabalhos futuros

Apresenta-se como propostas de trabalhos futuros:

- Executar novos experimentos usando outras sementes e ainda disponibilizando um limite máximo de tempo de execução maior para as hibridizações do AG com as buscas locais.
- Comparar e analisar os desempenhos das técnicas meta-heurísticas – algoritmo genético estendido com busca guiada e algoritmo de busca harmônica – usando um mesmo conjunto de testes padronizados e internacionalmente aceitos.
- Modelar e aplicar a meta-heurística que obtiver melhores resultados no caso da Faculdade de Computação (FACOM) da Universidade Federal de Uberlândia (UFU), e analisar seu desempenho.
- Modelar e analisar o desempenho da programação linear inteira na resolução do problema real apresentado pela FACOM.
- Confrontar os resultados da meta-heurística e do método exato e discutir as vantagens e as desvantagens da utilização de métodos exatos ou meta-heurísticos no PHCU, tendo em conta o caso real da FACOM.
- Usar programação paralela, já que as meta-heurísticas exigem muito tempo de execução e os algoritmos podem ser divididos em sub-funções que podem ser paralelizadas e distribuídas por múltiplos núcleos de processamento, isso diminuiria o custo de tempo de execução.

Referências

ABDULLAH, S. et al. A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, v. 18, n. 1, p. 1–23, 2012. Disponível em: <http://www.springerlink.com/content/0m7126113kuu1526/>. Acesso em: 17 jun. 2012. Citado na página 44.

ABENSUR, E. O.; OLIVEIRA, R. C. de. Um método heurístico construtivo para o problema da grade horária escolar. *Revista Eletrônica Pesquisa Operacional para o Desenvolvimento*, v. 4, n. 2, p. 230–248, 2012. Disponível em: <http://www.podesenvolvimento.org.br/inicio/index.php?journal=podesenvolvimento&page=article&op=view&path%5B%5D=158&path%5B%5D=173>. Acesso em: 22 ago. 2012. Citado 2 vezes nas páginas 23 e 43.

AL-BETAR, M. A.; KHADER, A. T. A harmony search algorithm for university course timetabling. *Annals of Operations Research*, v. 194, n. 1, p. 3–31, 2012. Disponível em: <http://www.springerlink.com/content/q3331g33344153w0/>. Acesso em: 17 jun. 2012. Citado 3 vezes nas páginas 18, 38 e 41.

ALMEIDA, C. O.; ZIVIANI, N. Problemas NP-completos e algoritmos aproximados. In: _____. *Projeto de Algoritmos com Implementações em Pascal e C*. 2. ed. [s.n.], 2004. cap. 9. Disponível em: <http://www.dcc.ufmg.br/algoritmos/cap9/slides/c/completo4/cap9.pdf>. Citado 2 vezes nas páginas 16 e 17.

BADUE, C. *Problemas NP-completos*. 2011. Transparências, color. Disponível em: <http://www.inf.ufes.br/~claudine/courses/paa11/transparencias/cormen02/cap34.pdf>. Acesso em: 18 jul. 2012. Citado na página 17.

BARATA, B. M. P. et al. Problema de alocação de horários: um estudo de caso utilizando o software livre FET. *Revista Eletrônica TECCEN*, v. 3, p. 13–22, 2010. Disponível em: <http://www.uss.br/revistateccen/v3n22010/pdf/002Problema_Alocacao_Horarios.pdf>. Acesso em: 15 mai. 2012. Citado na página 13.

BELLO, G. S. *Abordagem do problema de programação de grade horária sujeito a restrições utilizando coloração de grafos*. Dissertação (Mestrado em Informática) — Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, 2007. Disponível em: <http://www.inf.ufes.br/~crangel/dissertacao-geraldobello.pdf>. Acesso em: 23 ago. 2012. Citado 2 vezes nas páginas 17 e 23.

BONA, E. et al. Aplicativo para otimização empregando o método simplex sequêncial. *Acta Scientiarum*, v. 22, n. 5, p. 1201–1206, 2000. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/159>. Acesso em: 26 jul. 2012. Citado na página 20.

BONUTTI, A. et al. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, v. 194, n. 1, p. 59–70, 2010. Disponível em: <http://www.springerlink.com/content/932k651m03330t38/>. Acesso em: 11 jun. 2012. Citado 3 vezes nas páginas 47, 48 e 55.

BONUTTI, A.; GASPERO, L. D.; SCHAERF, A. *Curriculum-Based Course TimeTabling*. 2014. Disponível em: <http://satt.diegm.uniud.it/ctt/index.php>. Acesso em: 04 ago. 2014. Citado 2 vezes nas páginas 50 e 71.

BURKE, E.; KENDALL, G.; SOUBEIGA, E. A tabu-search hyperheuristic for timetabling and rostering. *Jornal of Heuristics*, v. 9, p. 451–470, 2003. Disponível em: <http://dl.acm.org/citation.cfm?id=965864>. Acesso em: 23 jul. 2012. Citado na página 26.

CHVATAL, V. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, v. 4, n. 3, p. 233–235, 1939. Disponível em: <http://www.jstor.org/discover/10.2307/3689577?uid=3737664&uid=2129&uid=2&uid=70&uid=4&sid=21100939553813>. Acesso em: 23 jul. 2012. Citado na página 27.

CISCON, L. A. *O problema de geração de horários*: um foco na eliminação de janelas e aulas isoladas. 54 p. Monografia (Trabalho de Conclusão de Curso) — Departamento de Ciência da Computação, Universidade Federal de Lavras, Minas Gerais, 2006. Disponível em: <http://www.bcc.ufla.br/monografias/2005/O_problema_de_geracao_de_horarios_um_foco_na_eliminacao_de_janelas_e_aulas_isoladas.pdf>. Acesso em: 11 jun. 2012. Citado 3 vezes nas páginas 13, 18 e 31.

COOPER, T. B.; KINGSTON, J. H. *The complexity of timetable construction problems*. [S.l.], 1995. Disponível em: <http://sydney.edu.au/engineering/it/research/tr/tr495.pdf>. Acesso em: 11 jun. 2012. Citado 2 vezes nas páginas 14 e 17.

CSIMA, J.; GOTLIEB, C. C. Tests on a computer method for constructing school timetables. *Communications of the ACM*, v. 7, p. 160–163, 1964. Disponível em: <http://dl.acm.org/citation.cfm?id=363986>. Acesso em: 12 jul. 2012. Citado na página 14.

DAS, S. et al. Exploratory power of the harmony search algorithm: analysis and improvements for global numerical optimization. *IEEE Systems, Man, and Cybernetics Society*, v. 41, n. 1, p. 89–106, 2011. Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5454370&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5454370>. Acesso em: 27 ago. 2012. Citado na página 40.

ESCARPINATI, M. C. *Soluções algébricas para problemas de programação linear matemática*. 2012. Disponível em: <http://www.facom.ufu.br/~mauricio/GSI027%20-%20Otimizacao/Apostila%20Programa%e7%e3o%20Linear%20(M%e9todo%20Simplex).pdf>. Acesso em: 26 jul. 2012. Citado na página 20.

FALEIROS, T. de P. *Abordagem imunológica para a elaboração automática de quadros horários*. 71 p. Monografia (Trabalho de Conclusão de Curso) — Universidade Federal de Goiás, Catalão, 2007. Disponível em: <http://www.catalao.ufg.br/cc/disc/pfc/2007/1/propostaThiago2.pdf>. Acesso em: 12 mai. 2012. Citado na página 44.

FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, v. 8, p. 67–71, 1989. Disponível em: <http://www.sciencedirect.com/science/article/B6V8M-48MPRWR-G/2/68e1718a1a87737c279bfcbfe6b15968>. Acesso em: 20 jul. 2012. Citado na página 27.

FERREIRA, P. S. et al. Aplicação de programação inteira na distribuição de encargos didáticos em instituições de ensino. *Tendências em Matemática Aplicada e Computacional*, v. 12, n. 2, p. 135–144, 2011. Disponível em: <http://sbmac.org.br/tema/seer/index.php/tema/article/view/84/32>. Acesso em: 22 ago. 2012. Citado na página 22.

GEEM, Z. W.; KIM, J. H.; LOGANATHAN, G. V. A new heuristic optimization algorithm: harmony search. *Simulation*, v. 76, p. 60–68, 2001. Disponível em: <http://sim.sagepub.com/content/76/2/60.short>. Acesso em: 18 jul. 2012. Citado 3 vezes nas páginas 37, 38 e 39.

GLOVER, F. Heuristic for integer programming using surrogate constraints. *Decision Sciences*, v. 8, n. 5, p. 156–166, 1977. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1111/j.1540-5915.1977.tb01074.x/abstract>. Acesso em: 19 jul. 2012. Citado na página 25.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers and Operations*, v. 13, p. 533–549, 1986. Disponível em: <http://leeds-faculty.colorado.edu/glover/TS%20-%20Future%20Paths%20for%20Integer%20Programming.pdf>. Acesso em: 19 jul. 2012. Citado 2 vezes nas páginas 25 e 26.

GOES, A. R. T. *Otimização na distribuição da carga horária de professores*: método exato, método heurístico, método misto e interface. Dissertação (Mestrado em Ciências) — Setores de Tecnologia e de Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2005. Disponível em: <http://www.ppgmne.ufpr.br/arquivos/diss/117.pdf>. Acesso em: 12 mai. 2012. Citado 3 vezes nas páginas 18, 19 e 43.

GOMIDE, F. *2-Programação linear*. 2012. Disponível em: <http://www.dca.fee.unicamp.br/~gomide/courses/IA810/transp/IA810ProgramacaoLinear_2.pdf>. Acesso em: 24 jul. 2012. Citado na página 19.

GRONER, L. *NP-completude*. 40 p. Monografia (Trabalho de Conclusão de Disciplina) — Fundação de Assistência e Educação, Faculdades Integradas Espírito-Santenses, Vitória, 2006. Disponível em: <http://www.slideshare.net/loianeg/np-completude-loiane>. Acesso em: 16 jul. 2012. Citado 2 vezes nas páginas 16 e 17.

HAMAWAKI, C. D. L. et al. Alocação de grade horária em instituições de ensino superior utilizando algoritmos genéticos. In: SIR/RS, 2006, Santa Maria. *IV Simpósio de Informática da Região Centro do RS*. Santa Maria, 2005. p. 1–8. Disponível em: <http://www.sirc.unifra.br/artigos2005/artigo23.pdf%>. Acesso em: 12 mai. 2012. Citado na página 30.

JAT, S. N.; YANG, S. A guided search genetic algorithm for the university course timetabling problem. *Multidisciplinary International Conference on Scheduling: Theory and Applications*, p. 180–191, 2009. Disponível em: <http://bura.brunel.ac.uk/bitstream/2438/5880/3/Fulltext.pdf>. Acesso em: 03 Ago. 2012. Citado na página 31.

KIRKPATRICK, S.; JR., C. D. G.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983. Disponível em: <http://www.fisica.uniud.it/~ercolessi/MC/kgv1983.pdf>. Acesso em: 19 jul. 2012. Citado na página 24.

LAYEB, A.; CHENCHE, S. A novel GRASP algorithm for solving the bin packing problem. *International Journal of Information Engineering and Electronic Business*, v. 4, n. 2, p. 8–14, 2012. Disponível em: <http://www.mecs-press.org/ijieeb/ijieeb-v4-n2/v4n2-2.html>. Acesso em: 11 jun. 2012. Citado na página 28.

LEEA, K. S.; GEEMB, Z. W. A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, v. 194, p. 3902–3933, 2005. Disponível em: <http://folk.uib.no/ssu029/Pdf_file/Lee05.pdf>. Acesso em: 17 jul. 2012. Citado na página 40.

LEWIS, R. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, v. 30, n. 1, p. 167–190, 2007. Disponível em: <http://orca.cf.ac.uk/13966/>. Acesso em: 12 mai. 2012. Citado na página 14.

LUCAS, D. C. *Algoritmos genéticos*: um estudo de seus conceitos fundamentais e aplicação no problema de grade horária. 66 p. Monografia (Trabalho de Conclusão de Curso) — Instituto de Física e Matemática, Univerdade Federal de Pelotas, Pelotas, dezembro 2000. Disponível em:

<http://www.ufpel.tche.br/prg/sisbi/bibct/acervo/info/2000/Mono-Diogo.pdf>. Acesso em: 12 mai. 2012. Citado 3 vezes nas páginas 29, 30 e 31.

MARTINS, J. P. *O problema do agendamento semanal de aulas*. Dissertação (Mestrado em Computação) — Institudo de Informática, Universidade Federal de Goiás, Goiânia, 2010. Disponível em: <http://www.inf.ufg.br/mestrado/sites/www.inf.ufg.br.mestrado/files/uploads/Dissertacoes/JeanPaulo.pdf>. Acesso em: 12 mai. 2012. Citado 3 vezes nas páginas 18, 28 e 44.

MIAGKIKH, V. A survey of reinforcement learning and agent-based approaches to combinatorial optimization. v. 1, p. 1–32, 2012. Disponível em: <http://www2.hawaii.edu/~miagkikh/RLSurvey.pdf>. Acesso em: 04 jul. 2012. Citado na página 19.

MIRHASSANI, S.; HABIBI, F. Solution approaches to the course timetabling problem. *Artificial Inteligence Review*, v. 1, p. 1–17, 2011. Disponível em: <http://www.springerlink.com/content/g9r690841214w510/>. Acesso em: 12 mai. 2012. Citado na página 14.

MOREIRA, C. M. da R. *Aplicação da OCL à especificação do problema de elaboração de horários*. Dissertação (Mestrado em Gestão de Sistemas de Informação) — Departamento de Ciências e Tecnologia da Informação, Instituto Universitário de Lisboa, 2011. Disponível em: <http://repositorio-iul.iscte.pt/handle/10071/2800>. Acesso em: 10 jun. 2012. Citado na página 45.

MOURA, A.; SILVEIRA, R. S. R.; SANTOS, V. dos. Técnicas metaheuristicas aplicadas à construção de grades horárias escolares. In: XXXVI - SBPO, 2004, São Jão del-Rei. *O impacto da Pesquisa Operacional nas Novas Tendências Multidisciplinares*. São Jão del-Rei, 2004. p. 1–12. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/Timetabling.pdf>. Acesso em: 12 mai. 2012. Citado na página 27.

MOURA, A. V.; SCARAFICCI, R. A. *A grasp strategy for a more constrained school timetabling problem*. [S.l.], 2007. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.67.4015>. Acesso em: 23 jul. 2012. Citado na página 27.

NEUFELD, G.; TARTAR, J. Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM*, v. 17, p. 450–453, 1974. Disponível em: <http://dl.acm.org/citation.cfm?id=361092>. Acesso em: 11 jun. 2012. Citado 2 vezes nas páginas 14 e 17.

NGUYEN, K.; NGUYEN, P.; TRAN, N. A hybrid algorithm of harmony search and bees algorithm for a university course timetabling problem. *International Journal of Computer Science Issues*, v. 9, p. 12–17, 2012. Disponível em: <http://www.ijcsi.org/papers/IJCSI-9-1-1-12-17.pdf>. Acesso em: 25 jun. 2012. Citado na página 44.

NOGUEIRA, M. *GRASP*. FEUP, 2008. 24 transparências, color. Disponível em: <http://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0CFEQFjAC&url=http%3A%2F%2Fpaginas.fe.up.pt%2F~mac%2Fensino%2Fdocs%2FODST20072008%2FGRASP_Marcelo_Nogueira.pptx&ei=vloNUMfGBIik8AS53ZHoCg&usg=AFQjCNHKGv70o-XeNg8n3D-9cABPTOULgw&sig2=NJ3FtRH_ZqaBN9F_kd790A%>. Acesso em: 23 jul. 2012. Citado na página 27.

PAIM, A. da S.; GREIS, I. C. Abordagens para elaboração automatizada de tabela de horários acadêmicos. *XI Seminário Intermunicipal de Pesquisa*, 2008. Disponível em: <http://guaiba.ulbra.tche.br/pesquisa/2008/artigos/administracao/376.pdf>. Acesso em: 22 ago. 2012. Citado na página 14.

PORTO, F. R. *Abordagem comparativa de técnicas metaheurísticas na elaboração automática de quadros horários, com enfoque nas técnicas*: algoritmos genéticos e algoritmo de seleção clonal. 64 p. Monografia (Trabalho de Conclusão de Curso) — Departamento de Ciência da Computação, Universidade Federal de Goiás, Campus Catalão, Catalão, 2010. Citado 2 vezes nas páginas 18 e 31.

POULSEN, C. J. B. *Desenvolvimento de um modelo para school timetabling problem baseado na meta-heurística simulated annealing*. Dissertação (Mestrado em Administração) — Escola de Administração, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2012. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/39522/000825681.pdf?sequence=1>. Acesso em: 22 mai. 2012. Citado na página 24.

RESENDE, M. *GRASP - Greedy randomized adaptative procedure*. 2012. 15 transparências, color. Disponível em: <http://www-usr.inf.ufsm.br/~andrezc/ia/metaheuristicas_grasp_transparencias.pdf>. Acesso em: 27 jul. 2012. Citado na página 27.

RIBEIRO, C. C. *Metaheurísticas e aplicações*. 2007. 223 transparências, color. Disponível em: <http://www.ic.uff.br/~celso/disciplinas/metah1.pdf>. Acesso em: 12 jul. 2012. Citado 2 vezes nas páginas 18 e 19.

ROSSI-DORIA, O. et al. *A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem*. 2002. Disponível em: <http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html#instances>. Acesso em: 04 ago. 2014. Citado 2 vezes nas páginas 48 e 67.

ROSSI-DORIA, O. et al. A comparison of the performance of different metaheuristics on timetabling problem. *IN: PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON PRACTICE AND THEORY OF AUTOMATED TIMETABLING (PATAT 2002*, v. 2740, p. 329–351, 2002. Disponível em: <http://w3.ualg.pt/~lpaquete/papers/patat2002a.pdf>. Acesso em: 11 jun. 2012. Citado na página 56.

SCARPIN, C. T.; STEINER, M. T. A.; ANDRADE, P. R. de L. Programação linear binária na geração da grade horária do curso de engenharia de produção da ufpr. *Mathematical Programming II*, 2012. Disponível em: <https://www.casnav.mar.mil.br/spolm/pdf/101793.pdf>. Acesso em: 22 ago. 2012. Citado na página 23.

SCHAERF, A. A survey of automated timetabling. *Artificial Inteligence Review*, v. 13, p. 87–127, 1999. Disponível em: <http://www.teiser.gr/arximidis/pdf/kazarlis/PE1/1d.pdf>. Acesso em: 23 jun. 2012. Citado 2 vezes nas páginas 14 e 15.

SILVA, A. S. N. e; SAMPAIO, R. M.; ALVARENGA, G. B. Uma aplicação de simulated annealing para o problema de alocação de salas. *INFOCOMP Journal of Computer Science*, v. 4, p. 59–66, 2005. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v4.3/art08.pdf>. Acesso em: 12 mai. 2012. Citado 2 vezes nas páginas 18 e 24.

SKRABE, M. S. *Proposta de algoritmo genético para aplicação no problema da grade horária*. 92 p. Monografia (Trabalho de Conclusão de Curso) — Instituto de Ciências Exatas e Tecnológicas, Centro Univerisitário Feevale, Novo Hamburgo, 2007. Citado na página 30.

SOUZA, D. *Introdução a Linguagem Scala*. 2013. Disponível em: <http://www.devmedia.com.br/introducao-a-linguagem-scala/29800>. Acesso em: 01 set. 2014. Citado na página 46.

SPENDLEY, W.; HEXT, G. R.; HIMSWORTH, F. R. Sequential application of simplex designs in optimisation and evolutionary operation. *American Spciety Quality*, v. 4, p. 441–461, 1962. Disponível em: <http://www.jstor.org/stable/pdfplus/1266283.pdf?acceptTC=true>. Acesso em: 26 jul 2012. Citado na página 20.

SUBRAMANIAN, A. et al. Aplicação da metaheurística busca tabu ao problema de alocação de aulas a salas em uma instituição universitária. *Revista Produção Online*, v. 11, n. 1, p. 1676–1901, 2011. Disponível em: <http://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CFMQFjAA&url=http%3A%2F%2Fproducaoonline.org.br%2Frpo%2Farticle%2Fdownload%2F419%2F762&ei=UmocUIyPAYLG6wHiioHoCw&usg=AFQjCNHSMtjoKEhGeg_4E2SfQ9_GUVmS_w&sig2=GDkMFWZ6eneVe-rXJT-8yg>. Acesso em: 27 jul. 2012. Citado na página 26.

SUCUPIRA, I. R. *Métodos heurísticos genéricos*: meta-heurísticas e hiper-heurísticas. 40 p. Monografia (Trabalho de Conclusão de Curso) — Departamento de Ciência da Computação, Universidade de São Paulo, São Paulo, 2004. Disponível em: <http://www.ime.usp.br/~igorrs/monografias/metahiper.pdf>. Acesso em: 11 mai. 2012. Citado na página 18.

WANG, L. et al. A hybrid binary harmony search algorithm inspired by ant system. *Cybernetics and Intelligent Systems*, v. 5, p. 153–158, 2011. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6070319&contentType=Conference+Publications>. Acesso em: 03 jul. 2012. Citado na página 44.

YANG, S.; JAT, S. N. Genetic algorithms with guided and local search strategies for university course timetabling. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 41, p. 93–106, 2011. Disponível em: <http://bura.brunel.ac.uk/handle/2438/5818>. Acesso em: 11 jun. 2012. Citado 20 vezes nas páginas 13, 31, 32, 33, 35, 36, 46, 48, 56, 60, 61, 62, 63, 64, 66, 67, 69, 76, 77 e 84.

Apêndices

APÊNDICE A – Classe Problema

Segue o código da implementação da classe abstrata Problema.

```
conteúdo...abstract class Problema extends ag.Problema {
  val nome = "Grade Horária Universitária"
  val nEventos:Int
  val nSalas:Int
  val nHorários:Int
  val nDisciplinas:Int
  val nCurriculos:Int
  val nRestricoes:Int
  val nEstudantes: Int
  val periodosPorDia:Int
  val diasPorSemana: Int
  val estudanteNum: Array[Int]
  val tamSalas: Array[Int]
  val estudanteEventos: Array[Array[Int]]
  var diasHorários: HashMap[Int, Array[Int]]
  val siglasSalas: Array[String]
  val possiveisSalas: Array[Array[Boolean]]
  val listaDisciplinas: Seq[Curso]
  val listaCurriculos: Seq[Curriculum]
  val listaSalas: Seq[Sala]
  val listaRestricoes: Seq[Restricao]
  val eventos: Array[String]
  val eventoCorrelacoes: Array[Array[Boolean]]
}
object Problema {
case class ExtensaoEx(message: String) extends Exception(message)
  def apply(nome_arq: String):Problema = {
    nome_arq.split('.').drop(1).lastOption match {
      case Some("tim") => new Tim(nome_arq)
      case Some("xml") => new Ctt(nome_arq)
      case _ =>
        throw ExtensaoEx("Arquivo sem extensão ou com extensão não conhecida")
    }
  }
}
```

APÊNDICE B – Classe Tim

Segue o código da implementação da classe `Tim` que estende a classe Problema.

```
class Tim(arquivo: String) extends Problema {
  val diasPorSemana = 5
  val periodosPorDia = 9
  val nHorários = diasPorSemana * periodosPorDia
  private val linhas = Source.fromFile(arquivo).getLines

  // Variáveis da classe Problema que não serão efetivamente
  //   usadas neste formato de instância.
  val siglasSalas = Array[String]()
  val listaCurriculos = List[Curriculum]()
  val nCurriculos = listaCurriculos.length
  val listaRestricoes = List[Restricao]()
  val nRestricoes = listaRestricoes.length

  // Le uma instancia de problema de um arquivo .tim
  // Os primeiros quatro inteiros são
  //    nEventos: numero total de eventos
  //    nSalas: numero total de salas
  //    nRecursos: numero de recursos disponíveis
  //    nEstudantes: numero de estudantes
  val Array(nEventos,nSalas,nRecursos,nEstudantes) =
    linhas.next.split(" ") map(_.toInt)

  // Vetor com as capacidades de cada sala
  val tamSalas = new Array[Int](nSalas)
  for (i <- 0 until nSalas)
    tamSalas(i) = linhas.next.split(" ")(0).toInt

  // Matriz estudante e suas participações em eventos
  val estudanteEventos = Array.ofDim[Int](nEstudantes, nEventos)
  for (i <- 0 until nEstudantes)
    for (j <- 0 until nEventos)
      estudanteEventos(i)(j) = linhas.next.split(" ")(0).toInt

  // Vetor que guarda o numero de estudantes em cada evento
  val estudanteNum = new Array[Int](nEventos)
  for (i <- 0 until nEventos) {
    var soma = 0
    for (j <- 0 until nEstudantes)
      soma += estudanteEventos(j)(i)
    estudanteNum(i) = soma
  }
  val listaSalas =
    (for (i <- 0 until nSalas) yield {
      val id = "r%03d".format(i)
      val tamanho = tamSalas(i)
      val predio = "0"
      Sala(id, tamanho, predio)
    }).toList
  val listaDisciplinas =
    (for (i <- 0 until nEventos) yield {
      val id = "c%04d".format(i)
```

```
    val professor = "t%04d".format(i)
    val aulas = 1
    val min_dias = 1
    val estudantes = estudanteNum(i)
    val aulas_duplas = "yes"
    Curso(id, professor, aulas, min_dias, estudantes, aulas_duplas)
  }).toList

val nDisciplinas = listaDisciplinas.length

// O vetor eventos é um vetor que mapeia cada evento (o índice)
//  com o id da disciplina que é por ele representado.
val eventos = new Array[String](nEventos)

// Matriz que guarda informação sobre eventos que tenham
//   estudantes em comum.
val eventoCorrelacoes = Array.ofDim[Boolean](nEventos,nEventos)

for (i <- 0 until nEventos)
  for (j <- 0 until nEventos)
    breakable { for (k <- 0 until nEstudantes)
      if ( estudanteEventos(k)(i) == 1 &&
        estudanteEventos(k)(j) == 1 ) {
        eventoCorrelacoes(i)(j) = true
        break
      }
    }

// Matriz que guarda as informações sobre os recursos disponíveis
//   em cada sala
val salaRecursos = Array.ofDim[Int](nSalas,nRecursos)
for (i <- 0 until nSalas)
  for (j <- 0 until nRecursos)
    salaRecursos(i)(j) = linhas.next.split(" ")(0).toInt

// Matriz que armazena a informação sobre os recursos necessários
//   para cada evento
val eventoRecursos = Array.ofDim[Int](nEventos,nRecursos)
for (i <- 0 until nEventos)
  for (j <- 0 until nRecursos)
    eventoRecursos(i)(j) = linhas.next.split(" ")(0).toInt

// Matriz que mantêm a informação sobre qual sala e adequada
// para cada evento
val possiveisSalas = Array.ofDim[Boolean](nEventos,nSalas)
for (i <- 0 until nEventos)
  for (j <- 0 until nSalas)
    if (tamSalas(j) >= estudanteNum(i)) {
      var k = 0
      var eAdequada = true
      while ( k < nRecursos && eAdequada) {
        if (eventoRecursos(i)(k) == 1 && salaRecursos(j)(k) == 0)
          eAdequada = false
        k += 1
      }
      possiveisSalas(i)(j) = eAdequada
    }

// diasHorários mapeia os horários que estão em um dia
var diasHorários = new HashMap[Int, Array[Int]]
```

```
  for(i <- 0 until diasPorSemana) diasHorários += i -> Array()

  var k = 0
  for(i <- 0 until diasPorSemana){
    for(j <- 0 until periodosPorDia){
      diasHorários(i) = diasHorários(i) :+ k
      k = k+1
    }
  }
}
```

APÊNDICE C – Classe Ctt

Segue o código da implementação da classe `Ctt` que também estende a classe Problema.

```
class Ctt(nome_arquivo: String) extends Problema {

   // Variáveis da classe Problema que não serão efetivamente
   //   usadas neste formato de instância.
   val estudanteEventos = Array(Array(0))
   val nEstudantes = 0

  // Leitura do arquivo XML
  private val arq = scala.xml.XML.loadFile(nome_arquivo)

  //Pegando os dados e convertendo os tipos
  //--------------DESCRITOR-------------------
  val nomeInstancia =
   ((arq \\ "instance").map{_ \ "@name"}).head.text
  val diasPorSemana =
    ((arq \ "descriptor" \"days").map{_ \ "@value"}).head.text.toInt
  val periodosPorDia =
    ((arq \ "descriptor" \"periods_per_day").map{_ \ "@value"}).head.text.toInt
  val nHorários = diasPorSemana * periodosPorDia
  val minAulas =
    ((arq \ "descriptor" \"daily_lectures").map{_ \ "@min"} ).head.text.toInt
  val maxAulas =
    ((arq \ "descriptor" \"daily_lectures").map{_ \ "@max"}).head.text.toInt
  //---------------- LISTAS--------------------
  // Criando  lista de disciplinas
  val listaDisciplinas = (arq \ "courses" \ "course").map {
   course =>
    Curso(
      (course \ "@id").text,
      (course \ "@teacher").text,
      (course \ "@lectures").text.toInt,
      (course \ "@min_days").text.toInt,
      (course \ "@students").text.toInt,
      (course \ "@double_lectures").text
    )
  }
  // Criando lista de salas
  val listaSalas = (arq \ "rooms" \ "room").map { room =>
    Sala(
      (room \ "@id").text,
      (room \ "@size").text.toInt,
      (room \ "@building").text
    )
  }

  // Criando lista de currículos
  val listaCurriculos = (arq \ "curricula" \ "curriculum").map{ c =>
    val id = (c \ "@id").text
    val cursos = (c \ "course").map { cur =>
      CursoRef((cur \ "@ref").text)
    }
      Curriculum(id, cursos)
```

```
}

//Criando uma lista de restrições
val listaRestricoes = (arq \ "constraints" \ "constraint").map{
 c =>
  val tipo = (c \ "@type").text
  val curso = (c \"@course").text
  val timeslots = (c \ "timeslot").map { time =>
    Timeslot((time \ "@day").text.toInt, (time \ "@period").text.toInt)
  }
  val salas = (c \ "room").map { sala =>
    SalaRef((sala \ "@ref").text)
  }
    Restricao(tipo, curso, timeslots, salas)
}
//---------contagem dos elementos das listas----
// O número de salas, restrições, currículos
val nSalas = listaSalas.length
val nRestricoes = listaRestricoes.length
val nCurriculos = listaCurriculos.length
val nDisciplinas = listaDisciplinas.length

// Cada uma das aulas de uma disciplina é um evento
val nEventos = listaDisciplinas.foldLeft(0)( (total,d) =>
  total + d.aulas )
val disciplinas =
  listaDisciplinas.foldRight(Map[String,(Int,Int)]())((d,m) =>
    m + (d.id -> (d.aulas, d.aulas) ))


//-----------vetor de siglas de evento e salas-------
// O vetor eventos é um vetor que mapeia cada evento (o índice)
//  com o id da disciplina que é por ele representado.
val eventos = new Array[String](nEventos)

val siglasSalas = new Array[String](nSalas)
for(i <- 0 until nSalas){
  siglasSalas(i) = listaSalas(i).id
}

//-------- Vetores com número de alunos e tamanho de salas ------

// Vetor que guarda o numero de estudantes em cada evento
val estudanteNum = new Array[Int](nEventos)
for (i <- 0 until nEventos) {
  var aux = listaDisciplinas.find(a => a.id == eventos(i)).toArray
    estudanteNum(i) = aux(0).estudantes
}

// vetor com as capacidades das salas
val tamSalas = new Array[Int](nSalas)
for(i <- 0 until nSalas){
  tamSalas(i) = listaSalas(i).tamanho
}
//----------------- Possíveis salas ------------------
// Salas que não podem ser alocadas para determinado evento de acordo
//  com as restrições de salas.
val impossiveisSalas = Array.ofDim[Boolean](nEventos,nSalas)
for (i <- 0 until nEventos){
  for (j <- 0 until nSalas){
```

```
    for(k <- 0 until nRestricoes){
      if(listaRestricoes(k).tipo == "room"){
        for(l <- 0 until listaRestricoes(k).salas.length){
          if(listaRestricoes(k).salas(l).refs == listaSalas(j).id &&
             eventos(i) == listaRestricoes(k).curso)
            impossiveisSalas(i)(j) = true
        }
      }
    }
  }
}
//Salas que possuem capacidade suficiente para as disciplinas
val possiveisSalas = Array.ofDim[Boolean](nEventos,nSalas)
for (i <- 0 until nEventos){
  for (j <- 0 until nSalas) {
    //procurar qual disciplina tem o id do evento,
    // retorna array de uma posicao
    var aux =
      listaDisciplinas.find(a => a.id == eventos(i)).toArray
    if (tamSalas(j) >= aux(0).estudantes) {
      possiveisSalas(i)(j) = true
    }
  }
}
// Tirando as impossíveis salas das possíveis
for (i <- 0 until nEventos){
  for (j <- 0 until nSalas) {
    if(impossiveisSalas(i)(j)){
      possiveisSalas(i)(j) = false
    }
  }
}
//------------- Correlacionando os eventos -----------
// Eventos de disciplinas pertencentes a um mesmo currículo
//  são correlacionados
val eventoCorrelacoes = Array.ofDim[Boolean](nEventos,nEventos)
  //todos os evento -> todos os horarios de cada materia
for ( i <-0 until nEventos){
  for (j <-0 until nEventos){
    var disciplinai = listaDisciplinas.find(a =>
                                      a.id == eventos(i)).toArray
    var di = disciplinai(0)
    var disciplinaj = listaDisciplinas.find(a =>
                                      a.id == eventos(j)).toArray
    var dj = disciplinaj(0)
    if (di.professor == dj.professor ||//possui o mesmo prof. ou
        eventos(i) == eventos(j)) {     // é a mesma disciplina
       eventoCorrelacoes(i)(j) = true
    }
    else{
      //se evento i está no mesmo currículo do evento j
      for(k <-0 until nCurriculos){
        if(listaCurriculos(k).refs.contains(CursoRef(eventos(i))) &&
           listaCurriculos(k).refs.contains(CursoRef(eventos(j)))){
          eventoCorrelacoes(i)(j) = true
        }
      }
    }
  }
}
```

```
  // diasHorários mapeia os horários que estão em um dia
  var diasHorários = new HashMap[Int, Array[Int]]
  for(i <- 0 until diasPorSemana) diasHorários += i -> Array()

  var k = 0
  for(i <- 0 until diasPorSemana){
    for(j <- 0 until periodosPorDia){
      diasHorários(i) = diasHorários(i) :+ k
                        k = k+1
    }
  }
}
```

APÊNDICE D – Classe MEM

Segue o código da implementação da classe `MEM` onde é implementado o método de construção da MEM.

```
class MEM[Ind <: Indivíduo[Cromossomo, Ind],
          P <: População[Cromossomo,Ind,P]](alfa:Double) {

  def empty:TMEM = Map[Int, List[Gene]]()

  def constroi(pop:P):TMEM = {
    val mmem = MMap[Int, List[Gene]]()
    val inds =  pop.indivíduos
    // Seleciona os melhores alfa * |pop| indivíduos da população
    val q = inds.take((alfa * inds.size).toInt)
      q foreach { ind => // para cada indivíduo
        ind.cromossomo.indices foreach { e => // para cada evento e
          // calcula a penalidade para o evento e de ind
          // se e é factível, ou seja, e possui penalidade 0,
          if (ind.fapt.avaliaEvento(e, ind.cromossomo) == 0) {
            // então adicione o par sala-horário (r_i, t_i) atribuído a
            // e na lista l_i
            if (mmem contains e) {
              if (!(mmem(e) contains ind.cromossomo(e)))
                mmem(e) = ind.cromossomo(e) :: mmem(e)
            } else
              mmem(e) = List(ind.cromossomo(e))
          }
        }
      }
      val entradas = mmem.values map (_.size)
      val media = entradas.sum.toDouble / entradas.size
      mmem.toMap
  }
}
```

Anexos

ANEXO A – DTD para as instâncias com formato Ctt

O DTD define um conjunto de regras que estrutura um documento. Segue o DTD que valida o padrão XML das instâncias no formato ctt.

```
<!ELEMENT instance (descriptor, courses, rooms, curricula, constraints)>
<!ATTLIST instance name CDATA #REQUIRED>
  <!ELEMENT descriptor (days, periods_per_day, daily_lectures)>
    <!ELEMENT days EMPTY>
      <!ATTLIST days value CDATA #REQUIRED>
    <!ELEMENT periods_per_day EMPTY>
      <!ATTLIST periods_per_day value CDATA #REQUIRED>
    <!ELEMENT daily_lectures EMPTY>
      <!ATTLIST daily_lectures min CDATA #REQUIRED
                max CDATA #REQUIRED>
  <!ELEMENT courses (course+)>
    <!ELEMENT course EMPTY>
      <!ATTLIST course id ID #IMPLIED
                teacher CDATA #IMPLIED
                lectures CDATA #IMPLIED
                min_days CDATA #IMPLIED
                students CDATA #IMPLIED
                double_lectures (yes | no) #IMPLIED
                ref IDREF #IMPLIED>
  <!ELEMENT rooms (room+)>
    <!ELEMENT room EMPTY>
      <!ATTLIST room id ID #IMPLIED
                size CDATA #IMPLIED
                building CDATA #IMPLIED
                ref IDREF #IMPLIED>
  <!ELEMENT curricula (curriculum+)>
    <!ELEMENT curriculum (course+)>
      <!ATTLIST curriculum id ID #REQUIRED>
  <!ELEMENT constraints (constraint+)>
    <!ELEMENT constraint (timeslot* | room*)>
      <!ATTLIST constraint type (period | room) #REQUIRED
                course IDREF #REQUIRED>
      <!ELEMENT timeslot EMPTY>
        <!ATTLIST timeslot period CDATA #REQUIRED
                  day CDATA #REQUIRED>
```