UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE GUARATINGUETÁ
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Andressa Fernanda Saemi Matsubara Ohata

ATRIBUIÇÃO DE TURMAS PARA PROFESSORES VIA BEAM SEARCH EM JAVA

Guaratinguetá
2012

Andressa Fernanda Saemi Matsubara Ohata

ATRIBUIÇÃO DE TURMAS PARA PROFESSORES VIA BEAM SEARCH EM JAVA

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como requisito para obtenção do diploma de Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Anibal Tavares de Azevedo
Coorientador: Prof. Dr. José Celso Freire Júnior

Guaratinguetá

2012

Ohata, Andressa Fernanda Saemi Matsubara
O369a Atribuição de Turmas para Professores via Beam Search em Java / Andressa Fernanda Saemi Matsubara Ohata – Guaratinguetá : [s.n], 2012.
112 f : il.
Bibliografia: f. 85-86

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2012.
Orientador: Prof. Dr. Anibal Tavares de Azevedo
Coorientador: Prof. Dr. José Celso Freire Junior

1. Algoritmos de computador 2. Java (Linguagem de programação de computador). I. Título

CDU 519.712

unesp

UNIVERSIDADE ESTADUAL PAULISTA
Faculdade de Engenharia do Campus de Guaratinguetá

ATRIBUIÇÃO DE TURMAS PARA PROFESSORES
VIA BEAM SEARCH EM JAVA

ANDRESSA FERNANDA SAEMI MATSUBARA OHATA

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO
COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO
DIPLOMA DE **GRADUADO EM ENGENHARIA ELÉTRICA**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Prof. Dr. Samuel Euzédice de Lucena
Coordenador

BANCA EXAMINADORA:

Prof. Dr. ANÍBAL AZEVEDO TAVARES
Orientador / UNICAMP

Prof. Dr. JOSÉ CELSO FREIRE JUNIOR
UNESP-FEG

Profa. Dra. PALOMA MARIA SILVA ROCHA RIZOL
UNESP-FEG

AGOSTO de 2012

Dedico este trabalho ao meu avô que, apesar de não estar mais presente, me deu forças para concluí-lo por meio de seus sábios ensinamentos.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus por ter me guiado, mesmo por caminhos duvidosos, e permitido que eu concluísse este Trabalho de Graduação. Acredito que aprendi muitas lições de vida que vão muito além do significado deste trabalho.

Agradeço à minha mãe *Luzia* e ao meu pai *Wagner* pela enorme paciência e apoio, sem os quais a conclusão desta etapa da minha vida não seria possível.

Ao meu orientador, *Prof. Dr. Anibal Tavares de Azevedo*, por ter me incentivado sempre apesar das minhas idas e vindas esporádicas e por contribuir com este trabalho independentemente da situação.

Ao coorientador, *Prof. Dr. José Celso Freire Junior*, por ter aceitado participar da banca examinadora e ter um papel fundamental no despertar do meu interesse por programação computacional.

À *Profª. Dr. Paloma Maria Silva Rocha Rizol*, por participar da banca examinadora e estar sempre disposta a colaborar.

Aos familiares e aos amigos que estiveram sempre presentes e me apoiaram nos momentos mais críticos da minha vida, em especial à *Priscila Satie* e *Patrícia Saito* que conheceu bem a minha caminhada durante a confecção do trabalho, ajudando a me manter firme e não desistir.

Às irmãs de república que estiveram sempre comigo, em especial *Cintia* e *Flávia* que dividiram muitos momentos e histórias.

Aos colegas de classe que passavam horas e, às vezes, madrugadas estudando, desenvolvendo projetos, esquentando as cadeiras da biblioteca e eventualmente nos divertindo.

Aos funcionários e professores e aos cidadãos brasileiros que fizeram possível a minha formação nesta instituição. E a todos que auxiliaram de alguma maneira a conclusão deste projeto.

OHATA, A. F. S. M. **Atribuição de turmas a professores via beam search em Java**. 2012. 112f. Trabalho de Graduação (Graduação em Engenharia Elétrica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2012.

RESUMO

Nas universidades, antes do começo de cada ano letivo, é realizada a distribuição das turmas entre os professores disponíveis. Para tanto, é preciso respeitar a carga horária máxima semanal de cada professor e suas preferências por cada disciplina, impedir que um professor dê aula em dois locais distintos ao mesmo tempo e evitar que alguns professores fiquem sobrecarregados enquanto outros com grande folga. Este processo, realizado manualmente, consome muito tempo e não possibilita a visualização de outras combinações de atribuição das turmas aos professores, além de ser passível de erro. Assim, este trabalho tem como propósito o desenvolvimento de uma ferramenta de apoio à decisão para o problema de atribuição de professores para turmas de faculdade. O projeto engloba o desenvolvimento de um programa de computador utilizando os conceitos de orientação a objeto e um algoritmo de busca em árvore de natureza combinatória denominado Beam Search. A linguagem de programação empregada é Java e o programa possui uma interface gráfica para inserção e manipulação dos dados do problema. Uma vez obtidos os dados relativos aos horários das turmas, bem como os dados dos professores é possível, por meio da ferramenta, realizar diversas simulações de atribuições e ajustes manuais para atingir o resultado final. Trata-se de um método eficiente de distribuição de turmas, considerando a rapidez na execução da tarefa e o fato de gerar somente resultados factíveis.

PALAVRAS-CHAVE: Problema de Atribuição. Beam Search. *Timetabling*.

OHATA, A. F. S. M. **Assigning teachers to classes with beam search in Java**. 2012. 112p. Graduation work (Graduation in Electrical Engineering) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2012.

ABSTRACT

In the universities, before the start of each school year, is held the distribution of classes among available teachers. Therefore, it is necessary to consider the maximum workweek for each teacher and their preferences for each discipline, to prevent a teacher to give lessons in two separate locations at the same time and to avoid some teachers to become overloaded while others with large clearance. This process, manually performed, is time consuming and does not allow the visualization of other combinations of assignment of teachers to classes, besides being liable to error. This work aims to develop a decision support tool for the problem of assigning teachers to classes in college. The project encompasses the development of a computer program using the concepts of object orientation and a tree search algorithm of a combinatorial nature called Beam Search. The programming language used is Java and the program has a graphical interface for entering and manipulating data of the problem. Once obtained the schedule data of classes and teachers is possible, by means of the tool, perform various simulations and manual adjustments to achieve the final result. It is an efficient method of class scheduling, considering the speed of task execution and the fact that it generates only feasible results.

PALAVRAS-CHAVE: Assignment Problem. Beam Search. Timetabling.

LISTA DE FIGURAS

Figura 2.1: (a) Relação das variáveis de uma *timetabling*. **(b)** Representação das variáveis para uma *course timetabling*. Os diagramas só mostram a relação das variáveis com o evento, não entre variáveis. 16
Figura 2.2: Árvore Ilustrativa. 20
Figura 2.3: Árvore de Decisão. 21
Figura 3.1: Ocupação dos *slots* (1,2), (2,2), (1,4) e (2,4) na grade horária pela turma de Cálculo I. 33
Figura 3.2: Ocupação dos *slots* (1,2), (2,2), (3,2), (1,4), (2,4), (3,4) e (4,4) na grade horária pelas matérias de Cálculo I e Física I. 34
Figura 4.1: Representação do problema de atribuição através de um grafo bipartido. 36
Figura 4.2: Possíveis soluções para as raízes P1 (árvore à esquerda) e P2 (árvore á direita). 38
Figura 4.3: Desenvolvimento dos níveis 1 e 2 do exemplo, correspondentes aos passos **ii** e **iii** do algoritmo. 42
Figura 4.4: Arborescência dos nós do nível 2, corresponde ao algoritmo guloso do passo **iv**. 43
Figura 4.5: Arborescência dos nós do nível 2, corresponde ao algoritmo guloso do passo **iv** até o subitem **d**. 43
Figura 4.6: Seleção dos nós do nível 2 (passo **iv** subitem **e**) e desenvolvimento dos nós do nível 3 (passos **ii** e **iii**). 44
Figura 4.7: Arborescência do nível 3 (passo **iv** até subitem **d**). 44
Figura 4.8: Seleção dos nós do nível 3 (passo **iv** subitem **e**) e desenvolvimento dos nós do nível 4 (passos **ii** e **iii**). 45
Figura 4.9: Solução do problema, com todas as atribuições realizadas. 45
Figura 5.1: Diagrama de Classes. 48
Figura 5.2: Barra de menu - Arquivo e itens do menu – Novo, Abrir e Salvar. 55
Figura 5.3: Abas e caixa de seleção de abertura de arquivo. 56
Figura 5.4: Arquivo carregado na aba “Dados das Turmas” e botões de manipulação das informações. 57
Figura 5.5: Arquivo carregado na aba “Dados dos Professores”. 57
Figura 5.6: Arquivo carregado na aba “Dados de Preferência” e caixa de diálogo para fornecimento da sigla 58
Figura 5.7: Arquivo carregado na aba “Dados de Similaridade”. 59

Figura 5.8: Aba “Atribuição de Turmas”, caixa de diálogo para inserção da largura de busca e botão de atribuição. 60
Figura 5.9: Lista com atribuições realizadas, botão de ajuste e caixa de alerta. 61
Figura 5.10: Botões de manipulação de informações habilitados, caixa de texto para filtro, botões de filtro e resumo de grade horária. 61
Figura 6.1: Dados das turmas. 62
Figura 6.2: Dados dos professores. 63
Figura 6.3: Valores de preferência. 63
Figura 6.4: Valores de similaridade. 64
Figura 6.5: Inserção da largura de busca igual a 2. 64
Figura 6.6: Resultado da simulação. 65
Figura 6.7: Teste com dados reais – turmas. 66
Figura 6.8: Teste com dados reais – professores. 67
Figura 6.9: Teste com dados reais – valores de preferência. 67
Figura 6.10: Teste com dados reais – valores de similaridade. 68
Figura 6.11: Teste com dados reais – resultado das atribuições. 68
Figura 6.12: Teste com dados reais – filtro. 69
Figura 6.13: Comparação da distribuição das disciplinas pelo método computacional e manual. 71

LISTA DE TABELAS

Tabela 3.1: Tabela exemplo com custos de preferência de cada professor ao ministrar uma dada disciplina. 30
Tabela 3.2: Tabela exemplo com custos de similaridade entre disciplinas. 31
Tabela 4.1: Demanda de carga horária de cada turma. 37
Tabela 4.2: Carga horária de cada professor. 37
Tabela 4.3: Custo de preferência por disciplina. 37
Tabela 4.4: Custo de similaridade entre disciplinas. 37

SUMÁRIO

1 INTRODUÇÃO 11

1.1 CONTEXTO 11
1.2 MOTIVAÇÃO 11
1.3 OBJETIVO 13
1.4 ORGANIZAÇÃO DO TRABALHO 13

2 FUNDAMENTAÇÃO TEÓRICA 15

2.1 EDUCATIONAL TIMETABLING 15
2.1.1 Course Timetabling 16
2.2 PROBLEMA DE OTIMIZAÇÃO 17
2.2.1 Programação Linear e Otimização Combinatória 18
2.3 ESTRUTURAS DE ÁRVORES 19
2.3.1 Árvores de Decisão 20
2.4 EFICIÊNCIA DO ALGORITMO 21
2.4.1 Complexidade do Algoritmo 22
2.4.2 Problemas NP-Completo 22
2.4.3 Soluções Heurísticas 23
2.5 BEAM SEARCH 23
2.6 ALGORITMO GULOSO 25
2.7 PROGRAMAÇÃO ORIENTADA A OBJETO 26
2.7.1 Linguagem de Programação Java 27

3 O PROBLEMA DE ATRIBUIÇÃO DE TURMAS A PROFESSORES 29

3.1 CONSIDERAÇÕES PARA O DESENVOLVIMENTO DO MODELO MATEMÁTICO 29
3.1.1 Condições desejáveis de serem cumpridas 30
3.2 MODELO MATEMÁTICO 31

4 ALGORITMO PROPOSTO 36

4.1 EXEMPLO SIMPLIFICADO PARA O PROBLEMA PROPOSTO 36
4.1.1 Árvore de soluções para o exemplo 37
4.2 ALGORITMO BEAM SEARCH 39

4.2.1 Resolução do exemplo utilizando o algoritmo proposto 41

5 O PROGRAMA COMPUTACIONAL 47

5.1 REQUISITOS 47

5.2 DESENVOLVIMENTO DAS CLASSES 47

5.2.1 Descrição de atributos e métodos das classes 48

5.3 ELABARORAÇÃO DA INTERFACE GRÁFICA 54

6 TESTE E VALIDAÇÃO DOS RESULTADOS 62

6.1 SIMULAÇÃO COM DADOS REAIS 65

7 CONCLUSÕES E TRABALHOS FUTUROS 72

REFERÊNCIAS BIBLIOGRÁFICAS 73

APÊNDICE – Código-fonte desenvolvido em Java 75

ANEXO – Distribuição de disciplinas no Departamento de Matemática em 2011 111

1 INTRODUÇÃO

1.1 CONTEXTO

Ao construir a grade horária (*timetable*) de uma universidade, há uma grande dificuldade para relacionar as diversas variáveis como estudantes, professores e salas de aula, principalmente considerando-se pré-requisitos já estabelecidos pela universidade, preferências individuais de professores/alunos por determinadas disciplinas a serem ministradas/cursadas e evitando tempo ocioso entre aulas. Somado a isso, existem riscos de erros no momento de definição das grades, que provavelmente serão detectados somente quando as aulas já tiverem começado.

Exemplos de erros são turmas distintas alocadas em uma mesma sala, um único professor encarregado por turmas diferentes ao mesmo tempo, professores sobrecarregados enquanto outros com folga, alunos alocados em duas turmas com choque de horário, e muitas outras possibilidades.

Por se tratar de uma tarefa que deve conciliar um grande número de variáveis e restrições, e que pode gerar uma quantidade significativa de combinações de alocação de recursos como resultado, realizada manualmente, esta atividade pode levar horas e até dias para que seja concluída.

Diversas pesquisas têm sido realizadas na área de gestão de recursos educacionais (*educational timetabling)*, e muitas escolas e universidades têm adotado recursos computacionais para solucionar problemas de *timetabling* de forma rápida e eficaz.

1.2 MOTIVAÇÃO

Tendo em vista que a gestão eficiente de recursos educacionais é um problema complexo e envolve uma série de variáveis e restrições de recursos, decidiu-se pelo planejamento de recursos disponíveis de forma modularizada, ou seja, por meio da resolução de seis problemas tais que os resultados fornecidos por um problema são aplicados como dados de entrada para outro.

i. MAPO: Mapeamento dos processos atualmente em uso, destacando o fluxo de informações e como cada processo é realizado (manual, semi-automático ou automático, ou seja, se utiliza algum recurso computacional);

ii. PREV: Desenvolvimento de uma ferramenta que realize previsões do número de alunos de cada curso para cada matéria, sendo capaz de simular os efeitos de mudança de pré-requisitos, destacar matérias que necessitem de maior apoio, e outras análises;

iii. NUMT: Desenvolvimento, a partir das informações obtidas por meio da ferramenta do PREV, de uma nova ferramenta capaz de estimar o número de turmas necessário, tendo em vista as restrições de número máximo de alunos por turma para aulas teóricas e práticas;

iv. GRAD: Alocação das turmas, determinadas em NUMT, em uma grade horária de modo a minimizar número de alunos que possuem pré-requisitos para realizar uma matéria o possam fazer sem existir incompatibilidade de horários entre turmas de diferentes matérias;

v. ALOP: Atribuição de cada turma a um professor, respeitando limites de carga horária máxima semanal, preferências de cada professor por cada matéria e impedindo que um professor dê aula em dois locais distintos ao mesmo tempo;

vi. ALOS: Alocação das turmas nas salas de aula de acordo com a disciplina (por exemplo, teoria ou laboratório) e número de alunos.

Neste trabalho, optou-se pelo desenvolvimento da iniciativa ALOP, denominado aqui como problema de Atribuição de Turmas a Professores (PATP). O PATP é um problema de otimização combinatória pertencente à classe NP-completo e exige uma técnica de solução enumerativa, cuja execução do algoritmo pode não ser viável em termos de tempo computacional. Nesses casos, é comum utilizar-se de métodos heurísticos para obter soluções factíveis em tempo razoável. Assim, decidiu-se por empregar o beam search, um método heurístico baseado em uma técnica de enumeração, para obter um algoritmo eficaz na atribuição de turmas a professores.

Dispor-se de uma ferramenta computacional que realize este trabalho de forma a gerar somente soluções factíveis ao respeitar todas as restrições do problema e, ainda, possibilitando a simulação e visualização dos diferentes resultados, é de grande relevância para a gestão eficiente dos recursos educacionais. Portanto, para a programação da ferramenta computacional foi escolhida a linguagem de programação Java, pela facilidade na implementação de uma interface gráfica capaz de realizar tais atividades.

1.3 OBJETIVO

Este trabalho irá tratar do problema de Atribuição de Turmas a Professores dentro de uma universidade. Para isso, primeiramente, será realizado o mapeamento das informações necessárias e variáveis do problema: dados característicos de professores e alunos, situações desejáveis de serem cumpridas, restrições pertinentes ao problema.

Com as informações relevantes ao problema em mãos, será feita a modelagem matemática: definição das variáveis, formalização das restrições do modelo e elaboração da função objetivo e critério de otimização.

Após a modelagem matemática será desenvolvido o algoritmo baseado no método heurístico de solução beam search e o modelo de classes de acordo com os conceitos de programação orientada a objeto.

Finalmente, a elaboração da interface gráfica e a criação do protótipo serão feitos por meio do uso da linguagem de programação Java, com posterior validação dos resultados.

1.4 ORGANIZAÇÃO DO TRABALHO

O Capítulo 1 descreveu o contexto em que este trabalho está inserido e a motivação que levou ao desenvolvimento do mesmo. Além disso, foram colocados os objetivos de forma a definir o escopo do trabalho e, finalmente, a organização da monografia.

No Capítulo 2 são apresentados conceitos relacionados ao *Educational Timetabling* (agendamento de aulas de uma instituição de ensino) focados na área de *course timetabling*, definição de problemas de otimização e conceitos de otimização combinatória, detalhamento da estrutura de árvores e funcionamento da árvore de decisões, a eficiência dos algoritmos incluindo complexidade dos algoritmos, problemas NP-completo e soluções heurísticas. Além disso, são explicados os funcionamentos do algoritmo Beam Search e do algoritmo guloso e as vantagens e desvantagens obtidas com o uso desses algoritmos. Finalmente é introduzido o conceito de linguagem orientada a objeto e a linguagem Java em específico utilizada para o desenvolvimento deste trabalho.

No Capítulo 3 o problema de Atribuição de Turmas a Professores é elaborado de forma a deixar claro seus requisitos, restrições e condições desejáveis de serem cumpridas. Com isso, é desenvolvido o modelo matemático para o problema.

Utilizando o problema matemático desenvolvido no Capítulo 3 e os conceitos do Capítulo 2, desenvolve-se o algoritmo que será usado para a programação computacional do problema. O Capítulo 4 mostra o desenvolvimento do algoritmo, baseado nas equações restritivas e na função objetivo obtida no modelo matemático e na metodologia beam search.

O Capítulo 5 fala sobre os requisitos computacionais, explica a forma como foi desenvolvido o programa computacional e também mostra a interface gráfica e seus recursos.

Para verificar o funcionamento do programa e realizar a validação dos resultados obtidos, no Capítulo 6, são mostradas algumas imagens de algumas simulações realizadas durante o teste do protótipo.

Por fim, no Capítulo 7 está o resumo das conclusões obtidas durante a elaboração deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 EDUCATIONAL TIMETABLING

Educational Timetabling é uma subclasse de *Timetabling* na qual os eventos ocorrem em uma instituição de ensino. *Timetabling* engloba todas as atividades relacionadas com a construção de uma *Timetable* (calendário/agenda/grade horária).

Timetable é uma tabela de eventos que especifica quem participará, onde será realizado e quando tal evento ocorrerá. Assim, uma *timetable* deve satisfazer a todos os envolvidos simultaneamente e não deve haver choques de horário.

Diversos problemas de *educational timetabling* têm sido propostos com diferentes tipos de soluções baseados no tipo de instituição envolvida (escola ou universidade) e nos tipos de restrições. As *Educational Timetabling* podem ser divididas em três principais classes:

a) *School Timetabling:* agendamento semanal de todas as aulas de toda a escola;

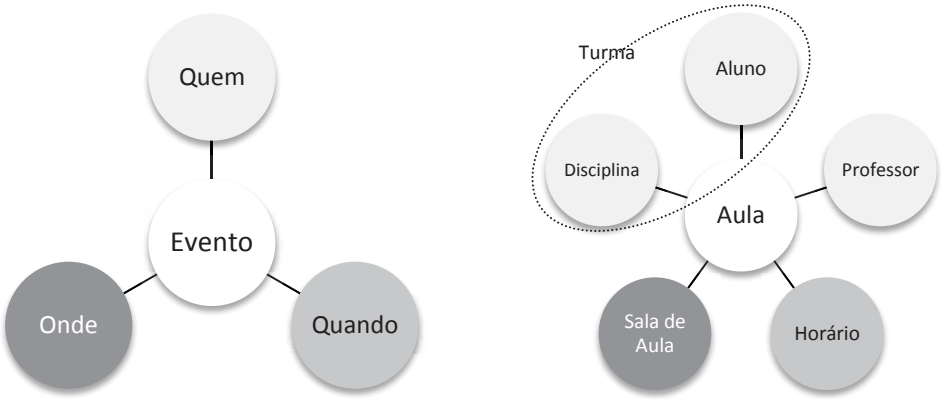
b) *Course Timetabling:* agendamento semanal de todas as aulas de um conjunto de cursos de uma universidade;

c) *Examination Timetabling:* agendamento de provas de um conjunto de cursos de uma universidade.

A principal diferença entre uma *school timetabling* e uma *course timetabling* está no fato de que numa universidade (*course timetabling*) os alunos têm a possibilidade de escolher o conjunto de matérias que desejam cursar durante o ano letivo, possuem matérias optativas ou ainda disciplinas pendentes.

Baseado nesta classificação é possível desenvolver diferentes tipos de resolução para cada um desses três tipos de problema. Contudo, essa classificação não é estrita, já que cada instituição possui características e regras próprias, podendo existir problemas que não se enquadram exatamente em uma das classificações. Portanto, existem variados tipos de problemas para as diversas instituições, de forma que dificilmente existam dois problemas exatamente iguais (WILLEMEN, 2002).

Neste caso, estamos lidando com a grade horária de uma universidade. Trata-se, portanto, de uma *course timetabling*, em que os eventos são aulas de uma dada disciplina ministradas por um dado professor, a um grupo de alunos (turma), em uma sala de aula, num dado horário. De uma forma simplificada podemos esquematizar as variáveis de uma *timetable* e uma *course timetable* da seguinte forma:



(a) **(b)**

Figura 2.1: (a) Relação das variáveis de uma *timetabling*. **(b)** Representação das variáveis para uma *course timetabling*. Os diagramas só mostram a relação das variáveis com o evento, não entre variáveis.

2.1.1 **Course Timetabling**

A construção de uma *course timetable* envolve basicamente problemas de alocação de recursos: atribuição de professor, atribuição de sala de aula e atribuição de horário para cada aula. Além disso, temos a atribuição de alunos para a formação das diferentes turmas.

O fato dos alunos possuírem um conjunto de matéria bem diversificada e dependente de pré-requisitos na grade horária dificulta o agendamento das aulas, pois no momento de alocar os alunos nas turmas é necessário minimizar a sobreposição de duas ou mais aulas em suas grades. Neste trabalho, estamos lidando somente com a atribuição de turmas a professores, ou seja, determinando qual professor será responsável por cada turma, logo, no momento da atribuição, pode-se considerar que as turmas já estão formadas com o mínimo de sobreposição de aulas.

Como no caso da atribuição de alunos para turmas, toda atribuição está atrelada ou depende de pré-requisitos e/ou preferências. Portanto, é necessário estudar os diversos requisitos e situações desejáveis de serem cumpridas para que a *timetable* resultante da atribuição de turmas a professores seja aceitável.

Exemplos de requisitos exigidos para a atribuição de turmas a professores:

- um professor não pode estar em duas aulas ao mesmo tempo;
- existem horários em que o professor não está disponível ou não pode dar aula, sendo impossível atribuir uma aula nesse horário;
- a carga horária máxima semanal de cada professor deve ser respeitada.

Exemplos de condições desejáveis de serem cumpridas:

- a distribuição das turmas deve ser feita de forma que não fiquem professores sobrecarregados;
- se possível, as preferências de cada professor pelas diferentes matérias devem ser respeitadas;
- professores devem ficar o máximo possível, responsáveis por matérias similares entre si, para facilitar o preparo das aulas e provas.

Algumas dessas condições serão utilizadas posteriormente para a modelagem matemática do PATP.

2.2 PROBLEMA DE OTIMIZAÇÃO

O problema de atribuição de turmas a professores consiste basicamente na busca de uma solução que satisfaça todas as condições (requisitos) exigidas por ele. Se formulado desta forma, poderia ser classificado como um problema de atribuição (cada professor fica responsável por um conjunto de turmas/disciplinas), no qual o resultado seria uma configuração qualquer. Em muitos casos, isto é, se for aplicado um método heurístico, realizado manualmente e apenas com o objetivo de atender às restrições, a primeira configuração encontrada pelo algoritmo pode ser adotada. No entanto, se forem estabelecidos critérios que permitam a avaliação de diferentes soluções através de uma função objetivo, então, é possível comparar as várias possibilidades de configuração que atendam às restrições de modo a selecionar a "melhor" configuração (SCHAERF, 1999).

Da mesma forma, para o PATP, exige-se ainda que a solução seja tal que o custo resultante das atribuições seja mínimo, ou seja, como efeito geral, professores devem ministrar as aulas com o mínimo de esforço possível. Isso significa que as condições desejáveis de serem cumpridas devem ser executadas de modo a agradar os professores ao máximo. Neste caso, o problema passa a ser um problema de otimização, onde todos os requisitos devem ser atendidos e o custo dado pela função objetivo deve ser minimizado.

Podemos definir os problemas de otimização da seguinte forma:

Definição: Uma instância de um problema de otimização, em particular um problema de minimização, é um par (F, c), onde F é um conjunto, o domínio de pontos factíveis; e c é a função custo, um mapeamento

$$c : F \rightarrow R$$

O problema é encontrar um $f \in F$ para o qual

$$c(f) \leq c(y) \quad \text{para todo} \quad y \in F$$

Este ponto *f* é a solução ótima para a dada instância (PAPADIMITRIOU *et al.*, 1998).

2.2.1 **Programação Linear e Otimização Combinatória**

Os problemas de otimização podem ser de dois tipos:

a) problemas cujas variáveis, restrições e função objetivo são reais e contínuas ou de programação linear (PL);

b) problemas com variáveis discretas ou de programação inteira (PI), também conhecidos como problemas de otimização combinatória.

Nos problemas contínuos, busca-se um conjunto de números reais que satisfaçam a definição ou mesmo uma função; já nos problemas combinatórios, busca-se um objeto dentro de um conjunto finito, tipicamente de números inteiros (PAPADIMITRIOU *et al.*, 1998).

O PATP faz parte do conjunto de problemas de otimização combinatória, em que a preocupação está em tentar obter a solução ótima ou muito próxima da ótima, da atribuição de um conjunto de aulas adotando um conjunto de recursos (professores) sujeito a certas restrições.

A necessidade de formular e resolver problemas de programação inteira veio inicialmente do fato que, em algumas aplicações, é inconveniente obter soluções fracionadas. Contudo, frequentemente, uma PI é o resultado do uso de restrições inteiras para a modelagem de restrições combinatórias ou não linearidades de diferentes tipos.

Tomando o PATP como exemplo, tem-se como restrição inteira a carga horária máxima semanal dos professores, onde não é possível atribuir 8,5 aulas para um professor cuja carga horária máxima é de 8 horas (considerando-se que cada aula tem duração de 1 hora). Além disso, 8,5 aulas significaria dizer que uma dada turma teria metade da aula ministrada por um professor e outra metade por outro. Assim, a carga horária da turma também passa a ser uma restrição inteira. Arredondado a carga horária para 8, metade de uma aula deverá ser ministrada por outro professor. Seja outro professor cuja carga horária máxima é de 9 aulas semanais, e que obteve 9,5 horas semanais ao final da atribuição que por arredondamento se tornaram 9. Novamente metade da aula deverá ser ministrada por outro professor. Somando-se 0,5 aulas do primeiro professor e 0,5 aulas do segundo, o resultado é que existe 1 aula que deverá ser atribuída a um outro professor. Diga-se ainda que não exista nenhum outro

professor disponível durante 1 hora (existem professores com 0,3 horas, 0,5 horas, 0,7 horas de sobra e assim por diante), então fatalmente uma aula não será atribuída a nenhum professor. Portanto, a inconveniência do arredondamento em variáveis inteiras, que nem sempre produzirá uma solução factível.

Outro problema é a não linearidade, que pode ser exemplificada no PATP pelo custo de preferência dos professores por ministrar cada disciplina. A preferência de cada professor não é necessariamente linear, ou seja, o custo de dar aulas apenas de cálculo pode ser 1, mas se outra disciplina diferente tiver que ser ministrada em conjunto, por exemplo, programação de computadores, então, o custo pode passa para 3.

Existem diversas técnicas para a busca da solução em PIs, como técnicas de enumeração, corte, heurísticas e meta-heurísticas.

A melhor forma de encontrar uma solução exata para o problema de PATP é por meio de uma busca exaustiva, utilizando-se uma técnica de enumeração ou árvore de decisão, examinando todo o espaço de soluções possíveis. Esse método pode se tornar computacionalmente inviável dependendo do número de soluções existentes. O número de soluções existentes para este problema cresce de forma exponencial com o número de entradas, limitando o método à solução de problemas com entradas muito pequenas. Isso será ilustrado na Seção 2.3.1.

2.3 ESTRUTURAS DE ÁRVORES

Para entender a estrutura em árvore é necessário, primeiramente, saber algumas definições:

- Grafo: estrutura representada por um conjunto de pontos (vértices) ligados por retas (arestas), utilizada para estudar relações entre objetos de um determinado conjunto. Dependendo da aplicação, as arestas podem ou não ter direção, podem ligar ou não um vértice a ele próprio e vértices e arestas podem ter um peso numérico associado. Grafos são úteis na representação de problemas reais como mapa de estradas para definição de caminhos mais econômicos, redes de computadores, planejamento de tarefas (definição da ordem e custo de execução), etc.
- Caminho: é uma sequência de vértices tal que de cada um de seus vértices há uma aresta para o próximo vértice da sequência.

- Grafo conexo: um grafo é dito conexo se para quaisquer dois vértices distintos existe um caminho que os une. Quando tal não acontece o grafo diz-se desconexo.

Uma árvore é um grafo conexo acíclico com um nó especial, denominado **raiz** da árvore. A Figura 2.2 ilustra um exemplo de árvore.

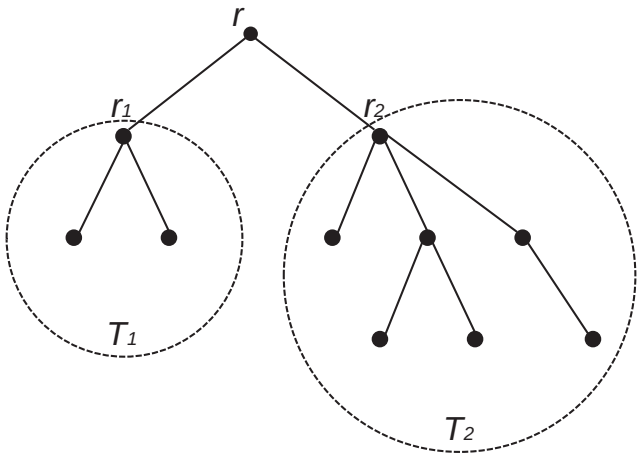


Figura 2.2: Árvore Ilustrativa.

Na Figura 2.2, r é a raiz da árvore. Os nós r_1 e r_2 são os **filhos** de r e r é **pai** de r_1 e r_2.

Como a árvore é um grafo conexo, existe um caminho da raiz para qualquer outro nó da árvore; e como a árvore é acíclica, cada caminho é único. A profundidade de um nó em uma árvore é o comprimento do caminho da raiz ao nó. A profundidade da árvore é o comprimento do caminho mais comprido da raiz até um dos nós. Nó sem filho é chamado de **folha** da árvore, e todos os nós que não são folhas são nós **internos**. A ligação entre dois nós é denominada **arco**. Se T_1 e T_2 fossem árvores disjuntas, T_1 e T_2 formariam uma **floresta**.

2.3.1 **Árvores de Decisão**

Uma das aplicações das árvores é representar as diversas possibilidades de combinação das entradas de um problema, isto é, representar de forma ordenada as diferentes soluções do problema. Por exemplo, se jogássemos uma moeda três vezes, quais poderiam ser as combinações obtidas. Uma árvore de decisão poderia representar as possíveis combinações de resultados para este problema.

Uma árvore de decisão é uma árvore na qual os nós internos representam ações (jogar moeda), os arcos representam os resultados de uma ação (cara (C) ou coroa (K)) e as folhas representam resultados finais. A Figura 2.3 mostra a árvore de decisão para o exemplo da moeda.

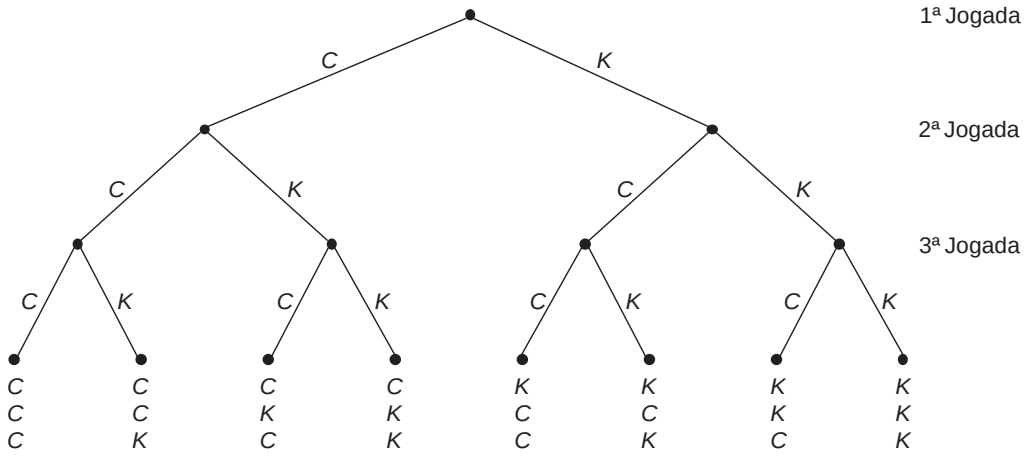


Figura 2.3: Árvore de Decisão.

Da mesma forma, é possível representar as atividades de um algoritmo por meio da árvore de decisão, onde as ações executadas pelos algoritmos são representadas pelos nós internos, os filhos dos nós internos representa a próxima ação a ser executada baseada na resposta obtida do nó pai e as folhas representarão alguma informação que será obtida ao final do algoritmo (GERSTING, 2008).

Assim, pode-se desenvolver um algoritmo enumerativo para o PATP, no qual os nós internos representarão a ação de atribuição de uma turma a um professor, os arcos os professores e as folhas a combinação de professores. Considerando-se *n* o número de turmas, ou seja, *n* atribuições, e *m* o número de professores, então para cada atribuição (nó interno) existem *m* possibilidades de resposta. Logo, a cada atribuição o número de combinações resultantes aumenta exponencialmente, obedecendo a relação m^n.

Nota-se facilmente que para um grande número de entradas (turmas e professores), a quantidade de combinações ou folhas obtidas ao final do algoritmo será tão grande que a busca pela melhor solução (solução exata ou de menor custo) se tornará inviável, já que será necessário mapear todas as possibilidades para obter a combinação de menor custo.

2.4 EFICIÊNCIA DO ALGORITMO

Atualmente, a principal ferramenta utilizada na aplicação dos diversos algoritmos para problemas que tratam de um grande número de variáveis e restrições são os computadores digitais. Apesar dos extraordinários recursos e agilidade dos computadores modernos, em algumas situações, mesmo ele pode ser inadequado à representação dos modelos, impondo severas restrições às conclusões teóricas. Isto é, a princípio tem-se um algoritmo que

resolveria qualquer instância do problema, no entanto, isso nem sempre é satisfatório devido ao tempo excessivo requerido para chegar a uma solução, o que torna o algoritmo completamente inútil.

Essa dificuldade de executar um modelo lógico num recurso computacional implica num número significativo de situações em que o modelo acaba se afastando do ideal pretendido para a otimização (GOLDBERG *at al.*, 2005). Ser capaz de estabelecer um conjunto de instruções que solucionam certo problema não significa que seja uma estratégia razoável. Assim, o objetivo de um algoritmo é solucionar o problema de uma forma rápida e econômica.

2.4.1 **Complexidade do Algoritmo**

Considera-se um algoritmo satisfatório quando este não consome muito tempo. De uma forma mais correta, a taxa de crescimento do tempo computacional gasto pelo melhor algoritmo encontrado para o problema em questão determina se o mesmo pode ser aplicado na prática. Neste sentido, a complexidade de um algoritmo pode ser entendida como a taxa de crescimento do algoritmo em relação ao tamanho sua entrada.

A taxa de crescimento pode ser considerada de uso prático se sua complexidade cresce polinomialmente em relação ao tamanho da entrada, por exemplo, algoritmos de complexidade $O(n^k)$ ou ainda n log n, onde n é o tamanho da entrada e $k \in \Re$. Já os algoritmos cuja instância é muito grande, são considerados algoritmos exponenciais, que podem ser do tipo k^n, $n!$, 2^{n^2}, n^n, $n^{\log n}$. Quando o tamanho da entrada aumenta, consequentemente algoritmos polinomiais apresentam-se cada vez mais eficientes do que os exponenciais. Outra característica dos algoritmos polinomiais é que podem tirar maior vantagem sobre os avanços tecnológicos (PAPADIMITRIOU *et al.*, 1998). Cada vez que uma nova tecnologia surge com processadores mais rápidos, a quantidade de instâncias que ele pode resolver num determinado período de tempo aumenta consideravelmente para os algoritmos polinomiais enquanto para algoritmos exponenciais o número não é tão expressivo.

2.4.2 **Problemas NP-Completo**

Entende-se que um algoritmo eficiente é um algoritmo cujo número de passos cresce

polinomialmente em relação ao tamanho da entrada. No entanto, sabe-se que pouco sucesso foi obtido nas pesquisas com algoritmos de otimização combinatória, exceto por alguns poucos exemplos. Assim, surgiu um novo conjunto de problemas, denominado problemas do tipo NP-completo.

Os problemas NP-completo são problemas que não podem ser resolvidos por nenhum algoritmo polinomial conhecido. Na prática isso significa que são problemas intratáveis do ponto de vista computacional, não suscetíveis a soluções algorítmicas eficientes, e que qualquer algoritmo que resolva corretamente um problema NP-completo irá exigir uma quantia de tempo exponencial, sendo, consequentemente, impraticável exceto para problemas de pequenas instâncias (PAPADIMITRIOU *et al.*, 1998).

Nesses casos, os modelos podem ser solucionados por técnicas não exatas, ou seja, o resultado não será necessariamente ótimo. É possível utilizar-se de técnicas heurísticas, onde se sacrifica o ideal de se obter com certeza a melhor solução em troca de melhorar a eficiência de um processo de busca e obter uma solução próxima à ótima.

2.4.3 **Soluções Heurísticas**

O termo heurística é derivado do grego *heuriskein*, que significa descobrir ou achar. No entanto, o significado aqui vai além disso, pode-se dizer que uma heurística refere-se a um método de busca de soluções em que não existe nenhuma garantia de sucesso.

Define-se a heurística como uma técnica que busca alcançar uma boa solução utilizando um esforço computacional considerado razoável, sendo capaz de garantir a viabilidade ou a otimalidade da solução encontrada, ou ainda, em muitos casos, ambas, especialmente nas ocasiões em que essa busca partir de uma solução viável próxima ao ótimo (GOLDBERG *et al.*, 2005).

Sabe-se que o PATP é um problema que exige um algoritmo exponencial, conforme a Seção 2.3.1, sendo necessário utilizar um método heurístico para sua solução. Decidiu-se trabalhar com um algoritmo denominado beam search que será detalhado na Seção 2.5.

2.5 BEAM SEARCH

Beam search é um método heurístico para resolver problemas de otimização através de

uma árvore de decisão, onde somente os nós mais promissores são mantidos a cada nível enquanto os demais são descartados permanentemente. Como uma grande parte da árvore é agressivamente eliminada, seu tempo de execução é polinomial (SABUNCUOGLU *et al.*, 1998). Essa técnica desenvolve um baixo número de soluções em paralelo de forma sistemática numa tentativa de encontrar uma boa solução com um esforço computacional mínimo (BERGHMAN *et al.*, 2005).

Na abordagem clássica do beam search, somente os β nós mais promissores factíveis de cada nível da árvore de decisão são mantidos como nós geradores das ramificações, onde β é denominado largura de busca. Determinados os β nós, os demais nós são descartados e não há como retroceder, já que o intuito desta técnica é uma busca rápida (VALENTE *et al.*, 2004). O algoritmo segue independentemente para cada nó gerando uma árvore parcial para cada. A árvore parcial é chamada *beam* que significa ‘viga’ ou ‘feixe’. Para cada um desses feixes somente um nó descendente de cada nó pai é selecionado e os demais descartados. Dessa forma, β nós são obtidos a cada nível da árvore de busca até que chegue o último nível.

O beam search conta com uma função de avaliação ou estimativa para determinar quais nós serão explorados. Dois tipos de função de avaliação têm sido utilizados: função de avaliação de prioridade e função de avaliação de custo total. A função de avaliação de prioridade simplesmente calcula uma taxa de prioridade entre os nós por meio de uma regra rápida. A função de avaliação do custo total estima o custo total mínimo da melhor solução que pode ser obtida pela atribuição parcial representada pelo nó. Isso pode ser feito através de uma regra que complete a atribuição parcial existente até o momento, ou seja, desenvolvendo a árvore até o último nível. Diz-se que a função de prioridade possui uma visão local do problema, pois considera apenas a atribuição no nível em questão; já a função de custo total possui uma visão global, pois projeta da solução parcial atual até a solução completa para estimar o custo total.

Funções de prioridade são de baixo custo computacional, mas são muito imprecisas e podem descartar nós importantes. As funções de custo total, por outro lado, são mais precisas, mas requerem um esforço computacional maior (VALENTE *et al.*, 2004). Para este trabalho será utilizado uma função de custo total baseada num algoritmo guloso, resultando numa solução mais precisa com um esforço computacional não muito grande devido às características deste tipo de algoritmo.

A maior desvantagem do método beam search é a não garantia de obter uma solução ótima e a inabilidade de restauração após decisões ‘erradas’. Assim, ao descartar um nó que

iria levar à solução ótima ou muito próxima à ótima, não há mais como chegar a uma solução ótima.

O beam search foi inicialmente utilizado em inteligência artificial para tratar de problemas de reconhecimento de fala (LOWERRE, 1976). Desde então, surgiram diversas aplicações do método para diferentes problemas de agendamento (*scheduling*) e roteamento.

2.6 ALGORITMO GULOSO

Juntamente com o método beam search será utilizado um algoritmo guloso para a função de avaliação de custo total. O algoritmo guloso é também uma estratégia para resolver problemas de otimização, o qual escolhe a opção que parece ser a melhor ou mais “apetitosa” no momento, e espera que assim consiga-se chegar a uma solução ótima global.

Um problema de otimização combinatória consiste em encontrar um elemento de valor ótimo num conjunto finito. Esse valor ótimo pode ser máximo ou mínimo. Assim, em um caso de minimização, por exemplo, a estratégia do algoritmo guloso é calcular um mínimo “local” na esperança de encontrar um mínimo “global”.

No entanto, a estratégia gulosa nem sempre chega num resultado ótimo, pois se trata de uma técnica enumerativa míope, ou seja, tomam decisões com base apenas nas informações disponíveis no momento, sem se preocupar com suas consequências. Assim, não é possível voltar atrás: as decisões feitas a cada passo do algoritmo guloso são definitivas. Devido a tais características, de forma geral é um algoritmo de fácil implementação, e são muito eficientes, principalmente pela agilidade na resolução do problema.

> Um algoritmo guloso começa com uma solução de um subproblema muito pequeno e multiplica sucessivamente a uma solução para o grande problema. O aumento é feito de uma forma "gulosa", ou seja, prestando atenção para o ganho de curto prazo ou local, sem levar em conta se ele vai levar a uma boa solução a longo prazo ou global. Como na vida real, algoritmos gulosos, por vezes, leva à melhor solução, outras a soluções muito boas, e outras a soluções ruins. O truque é determinar quando deverá ser guloso. (PARBERRY; GASARCH, 2002, p. 101).

Pode-se dizer que um algoritmo guloso trabalha da seguinte forma: a cada nível da árvore desenvolvida dentro da função de custo total, determina-se qual o melhor nó candidato para cada nó pai do nível anterior e os demais nós são descartados. Os filhos dos nós selecionados passam pela mesma análise, restando novamente um único nó para cada nó pai.

Esse procedimento é feito até que o algoritmo chegue ao último nível da árvore. Os β nós que gerarem os melhores custos totais, ou seja, os nós mais promissores continuarão fazendo parte da árvore no beam search.

2.7 PROGRAMAÇÃO ORIENTADA A OBJETO

A Programação Orientada a Objetos (POO) surgiu da necessidade de representar o mundo real, no qual o ser humano pensa em termos de objetos (pessoas, animais, transportes, computadores, etc.), através da linguagem computacional.

Os seres humanos entendem os objetos por meio de suas características e comportamento. Da mesma forma, num projeto de software, os objetos permitem representar conceitos do mundo real através de suas características – os **atributos,** e de suas ações – as **operações** (DEITEL *et al.*, 2006)**.**

Um objeto tem identidade própria que o distingue dos demais e seus atributos podem mudar. Além disso, objetos podem comunicar entre si ou com um usuário por meio de mensagens e interfaces bem definidas e só existem durante o tempo de execução do programa.

Para que seja possível a criação de um objeto é necessário que haja um padrão, uma espécie de planta arquitetônica. Assim como construímos uma casa a partir de uma planta arquitetônica, construímos objetos a partir de um padrão denominado **classe**. Uma classe é a descrição de um conjunto de objetos que compartilham os mesmos atributos, operações e relacionamento com outros objetos, classes ou usuários. Exemplificando, poderíamos ter uma classe casa, da qual são criados objetos como casa amarela e casa branca. Ambas são casas, com o mesmo atributo “cor”, porém são objetos diferentes criados a partir de uma mesma classe “casa”.

Na POO, atributos e operações sofrem **encapsulamento**, isto é, são empacotados nos objetos de forma a ocultar e proteger suas informações, assim, os detalhes de sua implementação e sua estrutura interna não são visíveis e o usuário somente poderá modificar o valor dos atributos por meio de uma interface.

Outras características importantes da POO são os mecanismos de **herança** e **polimosfismo**. A herança é um mecanismo de reutilização de um código para a criação de uma classe (subclasse ou classe filha) a partir de outra classe (superclasse, classe mãe ou classe base) já existente. Trata-se de um mecanismo de generalização/especialização, onde a nova classe irá utilizar das características comuns à classe base e irá implementar suas

particularidades, ou seja, pontos onde necessita de uma maior especialização. Um exemplo de herança é: superclasse – automóvel e subclasse – conversível (teto removível).

O polimorfismo é caracterizado quando duas ou mais classes distintas tem métodos de mesmo nome, porém com implementações diferentes, de forma que ao serem utilizadas em uma função, esta possa utilizar um objeto de qualquer uma das classes polimórficas sem necessidade de tratar de forma diferenciada conforme a classe do objeto.

Os mecanismos de herança e polimorfismo e a forma como é estruturada em classes e objetos trazem algumas vantagens na utilização da POO, como a capacidade de reutilização do código, facilidade na manutenção do código, modularidade e clareza na compreensão do código.

2.7.1 **Linguagem de Programação Java**

Java é o resultado de um projeto de pesquisa corporativa interna financiado pela Sun Microsystems, em 1990, que tinha como objetivo a criação de uma plataforma e uma linguagem de programação para produtos eletrônicos de consumo popular, o qual deveria proporcionar portabilidade, ou seja, independência de uma plataforma de *hardware*.

No entanto, o projeto passou por algumas dificuldades, pois o crescimento do mercado de dispositivos eletrônicos portáteis não estava se desenvolvendo tão rápido. Em 1993, com a chegada da Web, o Java começou a ser utilizado para adicionar conteúdos dinâmicos como interatividade e animação às páginas da Web. Assim, em 1995, o Java foi oficialmente anunciado na *SunWorld Conference*.

Atualmente o Java é largamente utilizado no desenvolvimento de aplicativos corporativos de grande porte e aplicativos para dispositivos portáteis de consumo popular, e no aprimoramento de funcionalidades de servidores da Web (DEITEL *et al*., 2006).

A plataforma Java pode ser encontrada em quatro versões destinadas para cada tipo de desenvolvimento:

- Java SE (Standard Edition): versão básica, destinada ao desenvolvimento da maior parte das aplicações para computadores desktop e estações de trabalho;
- Java EE (Enterprise Edition): versão destinada ao desenvolvimento de aplicações do tipo cliente/servidor de grande porte para empresas;
- Java ME (Micro Edition): destinada ao desenvolvimento de programas para periféricos móveis ou de capacidade de armazenamento limitada;

- Java Card: tecnologia que permite pequenas aplicações baseadas em Java serem executadas em dispositivos como smart cards e chip cards.

Um ambiente típico de desenvolvimento Java consiste de um programa editor, onde é desenvolvido o código-fonte e salvo com a extensão **.java**, um compilador Java que converte o código para bytecodes que é posteriormente convertido em linguagem de máquina durante execução do programa através do *Java Runtime Environment* (JRE) e o *Java Virtual Machine* (JVM) responsável pela execução do programa. Ao contrário da linguagem de máquina que depende do hardware do computador, os bytecodes não dependem de uma plataforma de hardware particular, assim, são portáteis podendo ser executados em qualquer plataforma contendo uma JVM.

O desenvolvimento dos programas em Java também pode ser realizado em ambientes de desenvolvimento integrado (*Integrated Development Environments* - IDEs) disponíveis a partir de muitos fornecedores de *software*. IDEs fornecem ferramentas que suportam o processo de desenvolvimento de *software*, incluindo editores e depuradores. Neste trabalho o IDE utilizado foi o JCreator.

A linguagem de programação Java é baseada na programação orientada a objeto onde as classes são formadas por **campos** que implementam os atributos e por **métodos** que implementam as operações. Além das classes construídas pelo próprio programador, existe também a biblioteca de classe Java, conhecida como Java APIs (*Application Programming Interface*), constituída por diversas classes prontas para serem utilizadas pelo programador.

3 O PROBLEMA DE ATRIBUIÇÃO DE TURMAS A PROFESSORES

Entende-se por atribuição de turmas a professores, a determinação de qual professor será responsável por ministrar a aula para cada turma. Para tanto, é necessário um sistema que tenha a capacidade de manipular dados de cadastros de professores e das turmas da universidade. Além disso, deverá ser desenvolvida uma interface para que os funcionários possam atuar, inserindo, alterando ou excluindo dados.

Na faculdade, as turmas são caracterizadas pelos seguintes dados: número da turma, curso, código da matéria, disciplina por extenso, T/L (teoria ou laboratório), carga horária, A/S (anual ou semestral), número de alunos, horário (2 grades – 1º e 2º semestre). Já os professores são caracterizados pelo: número de registro, nome completo, disciplinas ministradas, carga horária, horários disponíveis. Para o PATP serão utilizados somente alguns dados relevantes como o número de identificação e carga horária de turmas e professores.

A atribuição da turma ao professor se dá através da verificação dos professores que ministram ou podem ministrar a disciplina correspondente à aula da turma. A turma não poderá ser atribuída a professores cuja carga horária já tenha chegado ao máximo. Obviamente, nenhuma turma poderá ficar sem professor e cada professor só poderá dar aula para uma única turma em cada horário da grade.

3.1 CONSIDERAÇÕES PARA O DESENVOLVIMENTO DO MODELO MATEMÁTICO

Algumas considerações iniciais são necessárias para o desenvolvimento do modelo matemático do problema:

- as turmas já estarão formadas com o número de alunos correto;
- a distância física entre aulas é insignificante, ou seja, o tempo necessário para que o professor se locomova de uma sala de aula para outra não será considerado.

Para simplificar, como foi mencionado anteriormente, no desenvolvimento deste trabalho serão utilizados somente alguns dados dos professores e das turmas, sem que haja perda na confiabilidade dos resultados. Assim, será necessário que os seguintes dados estejam disponíveis:

- todas as turmas deverão ter o número de identificação da turma, curso ao qual pertence, sigla e nome da disciplina aplicada, carga horária e horário definido na grade;

- todos os professores deverão ter número de registro, nome completo e carga horária máxima definida;

O problema de atribuição de *m* turmas para *n* professores numa universidade é um problema que poderá gerar inúmeras combinações diferentes como resultado. Não há um método exato capaz de encontrar uma solução ótima para o problema em tempo razoável. A única maneira de se garantir uma solução ótima é através de uma busca exaustiva, sendo necessário examinar todo o espaço de soluções possíveis, o que é inviável devido à grande quantidade de soluções. Por exemplo, para uma universidade de 50 turmas e 10 professores, o número de soluções possíveis é (10^{50}). Sabendo-se ainda que uma universidade é composta por uma quantidade bem maior que os valores mencionados, teremos uma infinidade de soluções para o problema que deverão ser examinadas afim de encontrar a melhor delas. Se fosse utilizado um computador capaz de examinar uma solução a cada 1ns, seriam $3{,}17 \times 10^{33}$ anos para examinar todos os resultados para os valores mencionados. Logo, para este problema é conveniente o uso de métodos heurísticos para encontrar a solução.

3.1.1 **Condições desejáveis de serem cumpridas**

Para desenvolver o modelo utilizando o método heurístico será preciso definir algumas condições desejáveis de serem cumpridas para possibilitar o desenvolvimento da função de avaliação das atribuições ou, no caso, a função objetivo. A função objetivo deverá gerar como resultado um "custo" ao satisfazer as condições. Quanto menor o custo, melhor a solução encontrada.

A primeira condição que deverá ser satisfeita no problema é a preferência de determinados professores por ministrar determinadas disciplinas. Quanto maior a satisfação do professor em lecionar uma dada matéria, menor deverá ser o custo relacionado a essa disciplina. A Tabela 3.1 mostra uma situação hipotética para tal condição, com os valores de custo relativos a cada disciplina.

Custo por Preferência de Disciplina			
Professor/Turma	Turma 1 (Disciplina 1)	Turma 2 (Disciplina 2)	Turma 3 (Disciplina 3)
Professor A	5	3	1
Professor B	3	4	2
Professor C	2	1	5

Tabela 3.1: Tabela exemplo com custos de preferência de cada professor ao ministrar uma dada disciplina.

A segunda condição trata da similaridade entre as matérias ministradas pelo professor. Assim, quanto mais próximos os conteúdos das disciplinas, menor será o custo para ministrar tais matérias. A Tabela 3.2 mostra uma relação hipotética entre turmas de diferentes disciplinas, com os respectivos valores de satisfação.

Custo por Similaridade entre Disciplinas Ministradas			
Turma/Turma	Turma 1 (Disciplina 1)	Turma 2 (Disciplina 2)	Turma 3 (Disciplina 3)
Turma 1 (Disciplina 1)	0	1	3
Turma 2 (Disciplina 2)	1	0	5
Turma 3 (Disciplina 3)	3	5	0

Tabela 3.2: Tabela exemplo com custos de similaridade entre disciplinas.

Cumprindo as duas condições listadas anteriormente, chega-se a uma das possíveis soluções ideais para o problema de atribuição de turmas a professores.

3.2 MODELO MATEMÁTICO

O problema de atribuição de turmas a professores pode ser formulado da seguinte maneira:

Sejam m turmas que devem ser atribuídas a n professores. O custo de preferências de professores k por determinadas turmas de disciplinas i ($i = 1,...,m$; $k = 1,...,n$) é dado pela variável P_{ik} e o custo de similaridade entre as disciplinas das turmas i e turmas j ($i, j = 1,...,m$) é dado pela variável S_{ij}. A demanda de horas de cada turma i por semana ($i = 1,...,m$) é CT_i e a carga horária disponível de cada professor k por semana ($k = 1,...,n$) é CP_k.

Seja x_{ik} a variável definida como:

$$x_{ik} = \begin{cases} 1 \text{ se a turma } i \text{ for atribuída ao professor } k \\ 0 \text{ caso contrário} \end{cases}$$

E a variável auxiliar y_{ij} na qual:

$$y_{ij} = \begin{cases} 1 \text{ para turmas } i \text{ e } j \text{ pertencentes ao mesmo professor } k \\ 0 \text{ para turmas de professores diferentes} \end{cases}$$

Dessa forma, pode-se dizer que:

$$y_{ij} = \sum_{k=1}^{n} x_{ik} x_{jk} \qquad \text{para } i, j = 1,...,m \qquad \textbf{(3.1)}$$

O objetivo deste problema consiste em atribuir cada turma a um professor de modo a minimizar o custo total das atribuições definido pelas condições desejáveis: preferência por disciplina e similaridade entre disciplinas. Então:

a) O custo total por preferência de disciplina será:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} P_{ik}\, x_{ik} \tag{3.2}$$

b) O custo total de similaridade entre disciplinas será:

$$\sum_{i=1}^{m}\sum_{j=1}^{m} S_{ij}\, y_{ij} \tag{3.3}$$

O custo total é dado pela soma dos custos de preferência por disciplinas com os custos de similaridade entre disciplinas, e pode ser escrito como:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} P_{ik}\, x_{ik} + \sum_{i=1}^{m}\sum_{j=1}^{m} S_{ij}\, y_{ij} \tag{3.4}$$

A otimização do problema deve ser feita de modo a seguir algumas restrições que serão representadas pelas equações que seguem:

a) Cada turma deve ser atribuída a um único professor:

$$\sum_{k=1}^{n} x_{ik} = 1 \quad \text{para } i = 1,\ldots,m \tag{3.5}$$

b) A carga horária máxima semanal dos professores deve ser respeitada, ou seja, a somatória da carga horária das turmas atribuídas ao professor deve ser menor ou igual à carga horária semanal do professor:

$$\sum_{i=1}^{m} CT_i\, x_{ik} \le CP_k \quad \text{para } k = 1,\ldots,n \tag{3.6}$$

Além destas restrições relacionadas aos professores é necessário verificar a compatibilidade da alocação das turmas a um dado professor *k* em termos do horário que estas ocupam em sua grade horária. Assim, para representar que um determinado *slot* (*r*,*c*) da grade horária é ocupado por uma turma *i* alocada para um professor *k* usa-se uma variável $h_i(r,c)$, onde:

$$h_i(r,c) = \begin{cases} 1 \text{ para } slot \text{ preenchido} \\ 0 \text{ para } slot \text{ vazio} \end{cases}$$

Esta nova variável deverá obedecer às restrições:

c) Atendimento da carga horária semanal da turma, representada pela equação:

$$\sum_{r=1}^{R}\sum_{c=1}^{C} h_i(r,c) = CT_i \qquad \text{para } i = 1,...,m \tag{3.7}$$

Para ilustrar numericamente o emprego da Equação (3.7) supõe-se que exista apenas a turma de "Cálculo I" (i = 1) com carga horária CT_1 = 4 e que seus horários sejam como os dados na Figura 3.1.

Grade Horária (Exemplo)

Horário	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
07:30~08:20		Cálculo I		Cálculo I		
08:20~09:10		Cálculo I		Cálculo I		
09:30~10:20						
10:20~12:10						

Figura 3.1: Ocupação dos *slots* (1,2), (2,2), (1,4) e (2,4) na grade horária pela turma de Cálculo I.

Neste caso, a variável $h_i(r,c)$ assume os valores como dado abaixo:

$$h_i(r,c) = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.8}$$

Aplicando-se a Equação (3.7) para a variável $h_i(r,c)$, dada anteriormente, obtém-se:

$$\sum_{r=1}^{R}\sum_{c=1}^{C} h_i(r,c) = 1+1+1+1 = 4 \tag{3.9}$$

Este valor é igual à carga horária da matéria de "Cálculo I" (CT_1 = 4), confirmando a consistência da Equação (3.7). Em princípio, o problema de decisão de atribuição de turmas para professores já considera que todas as turmas possuem uma variável $h_i(r,c)$ consistente para esta restrição. A definição desta variável, porém, é importante para assegurar que soluções que violem a restrição de atribuição de duas turmas com conflitos de horário para o mesmo professor não ocorram. Este tipo de restrição será descrito a seguir.

d) Choque de horário entre turmas para um mesmo professor, ou seja, um slot (r,c) ocupado por uma turma i não pode ser compartilhado por outra turma atribuída ao mesmo professor k, caso contrário, haverá o choque de horário entre turmas. Na alocação de turmas para um professor é necessário optar pela alocação de apenas

uma delas. Em termos matemáticos esta restrição é representada pela Equação (3.10).

$$\sum_{r=1}^{R}\sum_{c=1}^{C}\sum_{i=1}^{I} h_i(r,c)x_{ik} \leq 1 \quad \text{para } k = 1,...,n \qquad \textbf{(3.10)}$$

Supondo que um dado professor *k* tenha sido atribuído para ministrar duas matérias cujos *slots* ocupados na grade horária são dados tal como ilustrado na Figura 3.2.

Grade Horária (Exemplo)						
Horário	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
07:30~08:20		Cálculo I		Cálculo I		
08:20~09:10		Cálculo I Física I		Cálculo I		
09:30~10:20		Física I		Física I		
10:20~12:10				Física I		

Figura 3.2: Ocupação dos *slots* (1,2), (2,2), (3,2), (1,4), (2,4), (3,4) e (4,4) na grade horária pelas matérias de Cálculo I e Física I.

Na situação descrita na Figura 3.2 existe uma incompatibilidade de horários entre as duas matérias de modo que a alocação de turmas deverá optar por uma delas. Se a restrição de evitar a incompatibilidade não for atendida, então a Equação (3.8) não será cumprida tal como calculado abaixo, onde: $i = 2$ (número de matérias), $k = 1$ (número de professores), e é feita apenas a análise do *slot* ($r = 2$, $c = 2$). Tem-se:

$$\sum_{i=1}^{2} h_{ik}(2,2)x_{ik} = h_{11}(2,2)x_{11} + h_{21}(2,2)x_{21} = 1\times1 + 1\times1 = 2 \qquad \textbf{(3.11)}$$

Para todos os *slots* de horário (1,2), (3,2), (1,4), (2,4), (3,4) e (4,4) o resultado a ser obtido é igual a 1, como pode ser verificado para o *slot* (1,2):

$$\sum_{i=1}^{2} h_{ik}(1,2)x_{ik} = h_{11}(1,2)x_{11} + h_{21}(1,2)x_{21} = 1\times1 + 0\times1 = 1 \qquad \textbf{(3.12)}$$

Os demais *slots* de horário produzem valor zero como para o *slot* (1,1):

$$\sum_{i=1}^{2} h_{ik}(1,1)x_{ik} = h_{11}(1,1)x_{11} + h_{21}(1,1)x_{21} = 0\times1 + 0\times1 = 0 \qquad \textbf{(3.13)}$$

Assim, o problema a ser resolvido consiste em minimizar a função objetivo dada pela Equação (3.4), sujeita às restrições correspondentes às Equações (3.5), (3.6), (3.7) e (3.10).

Neste trabalho é proposto um algoritmo baseado no método Beam Search para resolver o problema de atribuição. Para tanto, é necessário modificar a formulação do problema de modo a eliminar a variável y_{ij} da função objetivo substituindo a Equação (3.1) na Equação (3.4). Dessa forma, obtém-se o problema:

Minimizar:
$$\sum_{i=1}^{m}\sum_{k=1}^{n} P_{ik}\,x_{ik} + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n} S_{ij}\,x_{ik}\,x_{jk} \qquad \textbf{(3.14)}$$

Sujeito à:
$$\sum_{k=1}^{n} x_{ik} = 1 \quad \text{para } i = 1,...,m \qquad \textbf{(3.15)}$$

$$\sum_{i=1}^{m} CT_i\,x_{ik} \le CP_k \quad \text{para } k = 1,...,n \qquad \textbf{(3.16)}$$

$$\sum_{r=1}^{R}\sum_{c=1}^{C} h_i(r,c) = CT_i \quad \text{para } i = 1,...,m \qquad \textbf{(3.17)}$$

$$\sum_{r=1}^{R}\sum_{c=1}^{C}\sum_{i=1}^{m} h_i(r,c)\,x_{ik} \le 1 \quad \text{para } k = 1,...,n \qquad \textbf{(3.18)}$$

Onde:
$$x_{ik} = \begin{cases} 0 \\ 1 \end{cases} \quad \text{para } i = 1,...,m \text{ e } k = 1,...,n \qquad \textbf{(3.19)}$$

$$h_i(r,c) = \begin{cases} 0 \\ 1 \end{cases} \quad \text{para } i = 1,...,m \qquad \textbf{(3.20)}$$

4 ALGORITMO PROPOSTO

Para resolver o PATP será utilizada o Beam Search que é uma heurística baseada na enumeração completa a partir de uma estrutura em árvore tal como descrito na Seção 2.3.

Antes de apresentar o algoritmo proposto, será desenvolvido um exemplo simplificado do problema a fim de tornar mais fácil a visualização da árvore de decisões (atribuições), caso fossem enumeradas todas as soluções possíveis. Dessa forma, pode-se identificar a melhor solução e a forma de ação do algoritmo para gerá-la.

4.1 EXEMPLO SIMPLIFICADO PARA O PROBLEMA PROPOSTO

Seja $m = 4$ o número de turmas que devem ser atribuídas para $n = 2$ professores. Cada turma T_i demanda uma carga horária CT_i. Assim, o número de horas que a turma T_1 demanda é CT_1 e CT_2 é o número de horas demandada pela turma T_2, e assim por diante. Essas horas (períodos) serão ocupadas por um professor P_k, se as aulas da turma T_i forem ministradas pelo professor P_k. Cada professor tem uma carga horária disponível CP_k para ocupar com aulas. Então, CP_1 é a carga horária disponível do professor P_1, CP_2 é a carga horária disponível do professor P_2 e assim sucessivamente. Cada turma só pode ser ministrada por um único professor e não pode haver duas ou mais turmas alocadas num mesmo *slot* da grade horária de um professor. A Figura 4.1 representa o problema em grafo bipartido.

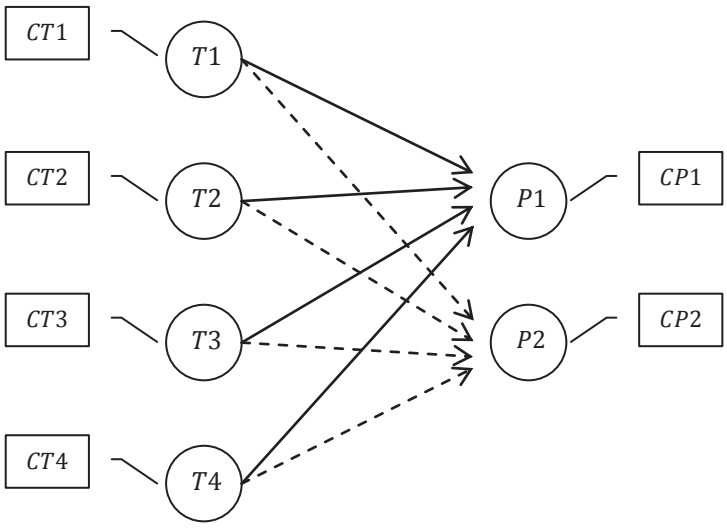


Figura 4.1: Representação do problema de atribuição através de um grafo bipartido.

Para que o programa consiga chegar numa solução ideal é necessário buscar informações de valores das cargas horárias (carga demandada pelas turmas e carga horária disponível de cada professor) e de custos de cada atribuição (custos de preferência por disciplina e custos de similaridade entre disciplinas). As Tabelas 4.1, 4.2, 4.3 e 4.4 mostram tais valores, respectivamente.

Turma	Carga Horária (h)
Turma 1	4
Turma 2	2
Turma 3	3
Turma 4	3

Tabela 4.1: Demanda de carga horária de cada turma.

Professor	Carga Horária (h)
Professor 1	8
Professor 2	7

Tabela 4.2: Carga horária de cada professor.

Professor/Turma	Turma 1 (Disciplina 1)	Turma 2 (Disciplina 2)	Turma 3 (Disciplina 3)	Turma 4 (Disciplina 3)
Professor 1	1	2	6	6
Professor 2	5	3	1	1

Tabela 4.3: Custo de preferência por disciplina.

Turma/Turma	Turma 1 (Disciplina 1)	Turma 2 (Disciplina 2)	Turma 3 (Disciplina3)	Turma 4 (Disciplina 3)
Turma 1 (Disciplina 1)	0	5	12	12
Turma 2 (Disciplina 2)	5	0	3	3
Turma 3 (Disciplina 3)	12	3	0	0
Turma 4 (Disciplina 3)	12	3	0	0

Tabela 4.4: Custo de similaridade entre disciplinas.

4.1.1 **Árvore de soluções para o exemplo**

A árvore de soluções enumera todas as possíveis soluções para o problema. Para realizar essa enumeração é necessário estabelecer algumas definições:

- a árvore de soluções possui níveis e no nível *h* é atribuída a *h*-ésima turma a um professor;
- uma solução completa só é obtida ao definir todas as atribuições de turmas aos professores até o nível *m*;

- a cada nova tentativa de atribuição é necessário verificar se a carga horária do professor permite a atribuição da carga horária da *h*-ésima turma, considerando todas as atribuições *h*-1 atribuições anteriores;
- a cada nível *h*, ao realizar a atribuição da *h*-ésima turma a um professor, são contabilizados os custos de preferência de disciplina e similaridade entre disciplinas, considerando todas as *h*-1 atribuições anteriores.

A Figura 4.2 mostra a árvore com todas as possíveis soluções para o problema. Nesta figura, os números dos professores estão representados dentro dos nós e os números das turmas estão representados ao lado pela letra *T*, por exemplo, T1 representa a turma 1. Cada nó da árvore representa uma atribuição, por exemplo, o primeiro nó do nível N1 simboliza a atribuição da turma T_1 ao professor P_1. Após a atribuição dos nós raiz, no caso, P_1 e P_2, são realizadas as demais atribuições para os nós seguintes, sendo que para cada nó deve ser dado o valor de custo considerando as atribuições anteriores.

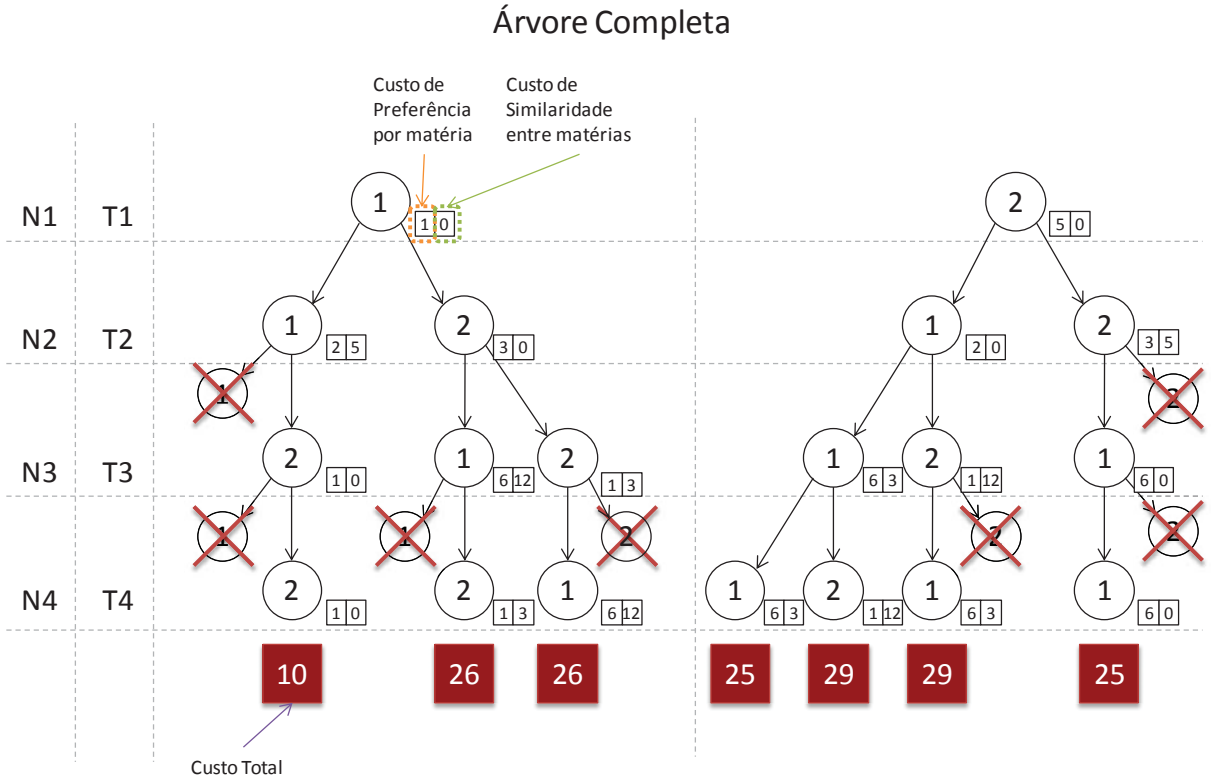


Figura 4.2: Possíveis soluções para as raízes P1 (árvore à esquerda) e P2 (árvore á direita).

Na Figura 4.2, em cada nó estão representados os valores dos custos referentes à preferência e similaridade de disciplinas relativas à atribuição efetuada ao nó. Os nós marcados com uma cruz (X) fornecem uma solução infactível e, portanto, são eliminados do processo. Isso ocorre porque após algumas atribuições, alguns professores atingem o limite

máximo de sua carga horária. Dessa forma, entende-se que apesar do número total de soluções ser igual a $n^m = 2^4 = 16$, somente 7 dessas soluções são factíveis.

É importante também entender que a atribuição de uma turma a um professor realizada a cada nível fornece apenas um custo de similaridade entre disciplinas **parcial**. O custo **total** de similaridade só será obtido quando todas as turmas forem atribuídas a um professor. Por exemplo, no nível 1 é feita a atribuição da turma T_1 a cada um dos professores (raízes), porém ainda não é possível contabilizar o custo de similaridade entre disciplinas, pois não se sabe a atribuição das demais turmas.

No nível 2 é feita a atribuição da turma T_2 a cada um dos professores, e com isto já pode-se contabilizar o custo de similaridade entre as disciplinas de acordo com as atribuições realizadas para cada professor. Assim, no caso de ambas as turmas T_1 e T_2 serem atribuídas para o mesmo professor, se as disciplinas forem diferentes, o custo será algum valor diferente de zero, conforme tabela de custos de similaridade entre disciplinas; caso a disciplina ministrada seja a mesma ou as turmas T_1 e T_2 sejam atribuídas a professores diferentes, o custo adicional de similaridade será zero.

Da mesma forma, no nível 3, o custo de similaridade será dado entre as turmas T_3 e T_1 e entre as turmas T_3 e T_2. Se as três turmas forem de disciplinas diferentes, os custos de similaridade entre as turmas deverão ser somados.

Assim, uma solução só será completa quando chegar ao último nível da árvore (último nó), com todas as atribuições e contabilizações de custos concluídas.

4.2 ALGORITMO BEAM SEARCH

O algoritmo proposto não gera todos os nós da árvore como mostrado no exemplo anterior e, portanto, não é feita a enumeração de todas as soluções factíveis, evitando o crescimento exponencial da árvore. Dessa forma, foram criadas algumas regras para a geração dos nós. Essas regras visam também impedir a criação de soluções infactíveis.

Primeiramente, deve-se ter em mente que para cada nó criado deverá ser armazenadas as seguintes informações:

- identificação do professor;
- identificação da turma;
- soma dos custos de similaridade entre disciplinas desde o nó raiz;

- custo parcial do nó, ou seja, soma do custo de preferência pela disciplina e do custo de similaridade entre disciplinas considerando atribuições feitas até aquele nó.

Além disso, para criar um nó, deve-se verificar a carga horária remanescente do professor. Um nó só será criado no nível *h* se a carga horária CT_i da turma T_i for menor ou igual à carga horária $CP_k^{(h)}$ remanescente do professor P_k no nível *h*. Então, se $CP_k^{(h)} < CT_i$, todos os ramos que se originam a partir deste nó não irão existir, pois a turma T_i não poderá ser atribuída ao professor P_k.

Criado o nó, ele só permanecerá na árvore se passar pela avaliação global, o algoritmo guloso. Somente os β nós com a menores soluções calculadas a partir do algoritmo guloso farão parte da árvore. A partir dos β nós serão criados os feixes, e a cada nível de desenvolvimento da árvore, só continuará fazendo parte de cada feixe o nó que gerar a solução de menor custo e, portanto, restarão β nós por nível após o processo de avaliação global. Assim, ao final da árvore só existirão β soluções.

De uma forma resumida o algoritmo deverá seguir os passos listados a seguir.

i. Inicialmente o algoritmo se encontra no nível de solução zero, pois nenhuma atribuição foi feita até o momento, então **nível = 0**.

ii. A seguir faz-se **nível = nível + 1** e são criados os nós deste nível, um para cada professor, levando em consideração sua carga horária. Para cada nó são calculados os custos de preferência por matéria e de similaridade entre matérias, e o custo parcial dado pela soma dos outros dois mais o custo parcial calculado até o nível anterior.

Obs.: Se o número de nós criados no passo **ii** for menor ou igual à largura de busca β volta-se para o passo **ii**, e todos os nós serão expandidos no nível seguinte.

iii. Para cada nó criado no passo **ii**, faz-se:

- Armazenar a identificação da turma e do professor;
- Armazenar o custo parcial do nó;
- Armazenar a carga horária remanescente de cada professor, ou seja, a partir do momento que uma turma é atribuída a um professor, reduz-se deste professor a carga horária correspondente à da turma alocada.

iv. Se o nível definido no passo **ii** for menor que o número de turmas, aplica-se o algoritmo guloso, considerando os custos parciais e as cargas horárias remanescentes até então. O detalhamento do algoritmo guloso é feito a seguir:

a) Para os nós resultantes do nível (nós pai) faz-se uma arborescência, ou seja, criam-se os nós filhos cujas atribuições sejam factíveis considerando as cargas

horárias remanescentes de cada professor e calculam-se os custos de preferência e de similaridade;

b) Para cada nó pai, ordenam-se os nós filhos conforme somatória de custos e seleciona-se o nó de menor custo;

c) Armazenam-se, para os nós resultantes em **b**, o custo parcial e a carga horária remanescente dos professores;

d) Se o nível em questão for menor que o número de turmas, volta-se ao passo **a**, se não, selecionam-se as β soluções que geraram os menores custos. Caso o desenvolvimento da árvore esteja em feixes, deve-se selecionar um nó por feixe;

e) Entre os nós criados no passo **ii**, escolhem-se os β nós que geraram as soluções gulosas de menor custo obtidas no passo **c**, e descartam-se os demais.

v. Se o nível definido no passo **ii** for igual ao número de turmas o algoritmo termina, já que a árvore já chegou ao último nível. Se não, volta-se ao passo **ii**.

No algoritmo proposto, o valor do custo total de um ramo da árvore obtido na solução gulosa, funciona como um limitante superior (valor de corte) para gerar ou não os demais nós da árvore no passo **ii**. Este limitante superior deve ser atualizado à medida que for encontrado um ramo com valor de custo menor. Dessa forma, têm-se dois critérios de corte para se gerar ou não um nó da árvore: a carga horária remanescente dos professores e o menor custo total obtido pela solução gulosa.

4.2.1 **Resolução do exemplo utilizando o algoritmo proposto**

Desenvolveu-se uma resolução detalhada para o exemplo da Seção 4.1, para melhor entendimento do algoritmo mostrado na Seção 4.2.

Lembrando-se que $m = 4$ é o número de turmas que devem ser atribuídas para $n = 2$ professores. Além disso, deve-se considerar $\beta = 2$, e as Tabelas 4.1, 4.2, 4.3 e 4.4 para consulta de carga horária das turmas, carga horária dos professores, custos preferenciais dos professores pelas disciplinas e custos de similaridade entre as disciplinas, respectivamente.

Na Figura 4.3 criaram-se os nós do primeiro nível, conforme as cargas horárias das tabelas 1 e 2. Assim, por exemplo, para o nó 1 no nível N1 temos: carga horária da turma $T_1 = 4$ e carga horária do professor $P_1 = 8$, como $CP_1 > CT_1$ é possível realizar a atribuição. Os custos de preferência (valor do quadrado à esquerda) e de similaridade (valor do quadrado à direita) foram obtidos para os nós conforme consulta nas tabelas 3 e 4: o custo de preferência

do professor P_1 pela disciplina da turma T_1 é 1 e o custo de similaridade é zero pois é a primeira atribuição. Como o número de nós obtidos é igual à largura de busca β = 2, não é necessário aplicar o algoritmo guloso. Desenvolvem-se, então, os nós factíveis do nível N2 e calculam-se seus custos. Até este ponto foram realizados somente os passos **ii** e **iii**.

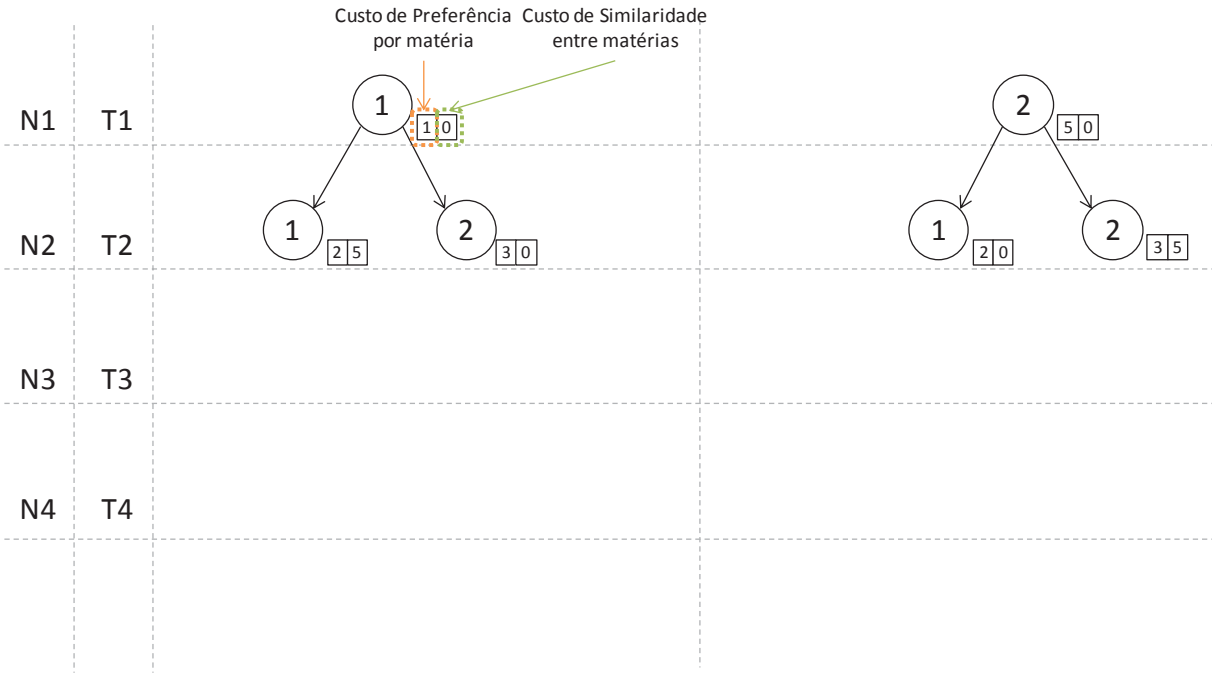


Figura 4.3: Desenvolvimento dos níveis 1 e 2 do exemplo, correspondentes aos passos **ii** e **iii** do algoritmo.

Na Figura 4.4 é feita uma arborescência a partir dos nós pai criados no nível N2. Como se pode verificar na figura, dois nós foram eliminados devido à carga horária da turma que excedeu a carga horária remanescente dos professores. Esses nós foram marcados com um **X** e desenhados um pouco acima dos demais. Os custos preferenciais e de similaridade foram calculados, obtendo-se, assim, os custos parciais de cada nó. Como o algoritmo guloso permite somente um nó filho para cada nó pai, apenas um nó filho foi mantido para cada nó pai e os demais foram eliminados.

Da mesma forma, na Figura 4.5 foi realizada a arborescência a partir dos nós que restaram no nível anterior, calcularam-se os custos parciais e eliminaram-se os nós desnecessários. Como chegou ao último nível da árvore, ou seja, nível = número de turmas = 4, somente as β = 2 soluções com menor custo total continuarão fazendo parte da árvore. As duas soluções estão marcadas com uma seta abaixo do custo total.

As Figuras 4.4 e 4.5 correspondem ao passo **iv** até o subitem **d** do algoritmo guloso.

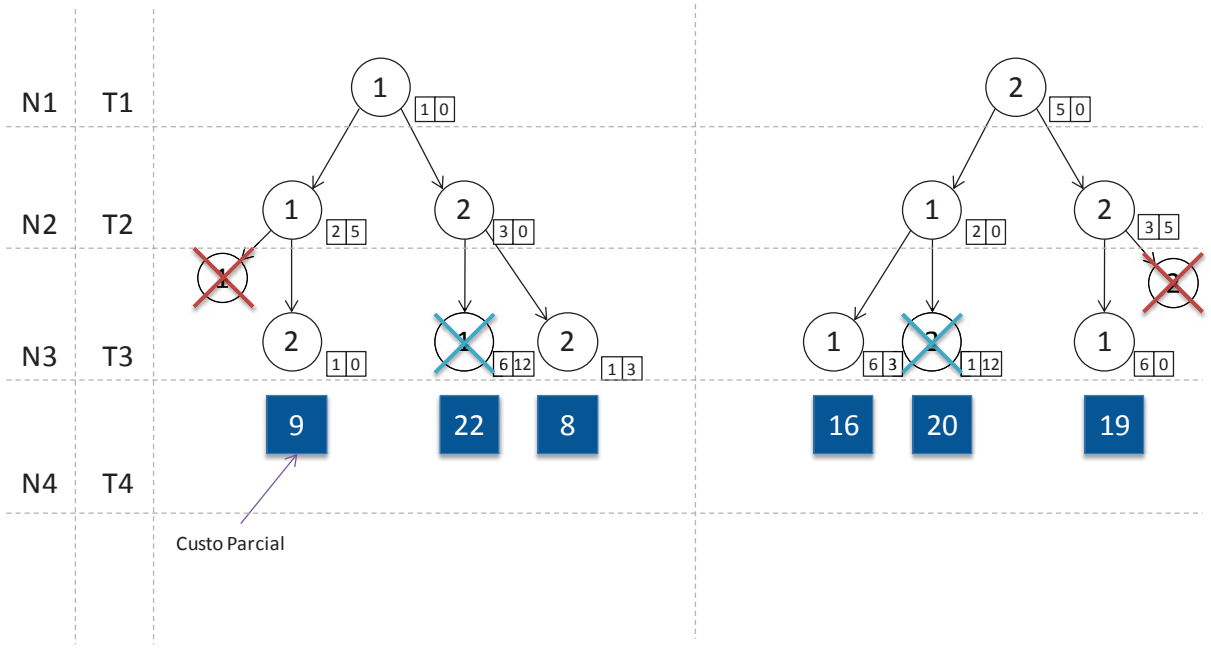


Figura 4.4: Arborescência dos nós do nível 2, corresponde ao algoritmo guloso do passo **iv**.

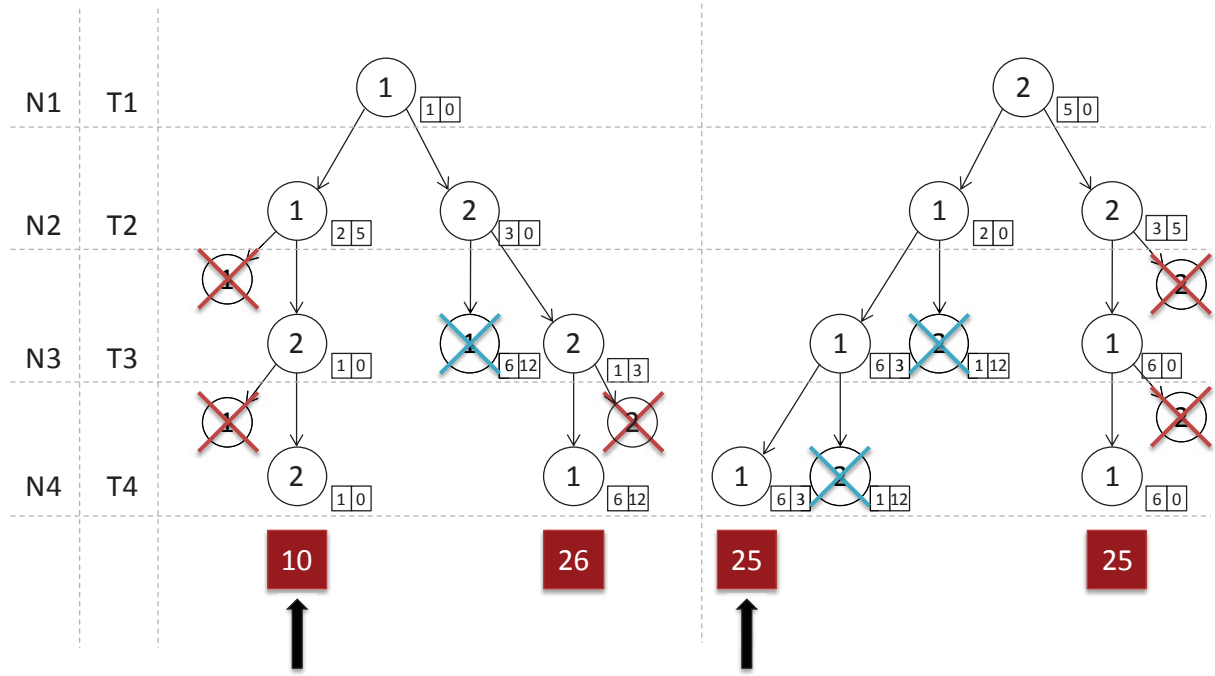


Figura 4.5: Arborescência dos nós do nível 2, corresponde ao algoritmo guloso do passo **iv** até o subitem **d**.

Na Figura 4.6 são mostrados os nós selecionados no nível N2 pelas β = 2 melhores soluções gulosas obtidas na Figura 4.5, correspondente ao passo **iv** subitem **e**. Além disso, foram desenvolvidos os nós do nível N3 e calculados seus custos (passos **ii** e **iii**).

Na Figura 4.7 é aplicado o algoritmo guloso novamente, agora para os nós do nível N3, e selecionadas as β = 2 melhores soluções (passo **iv** até subitem **d**).

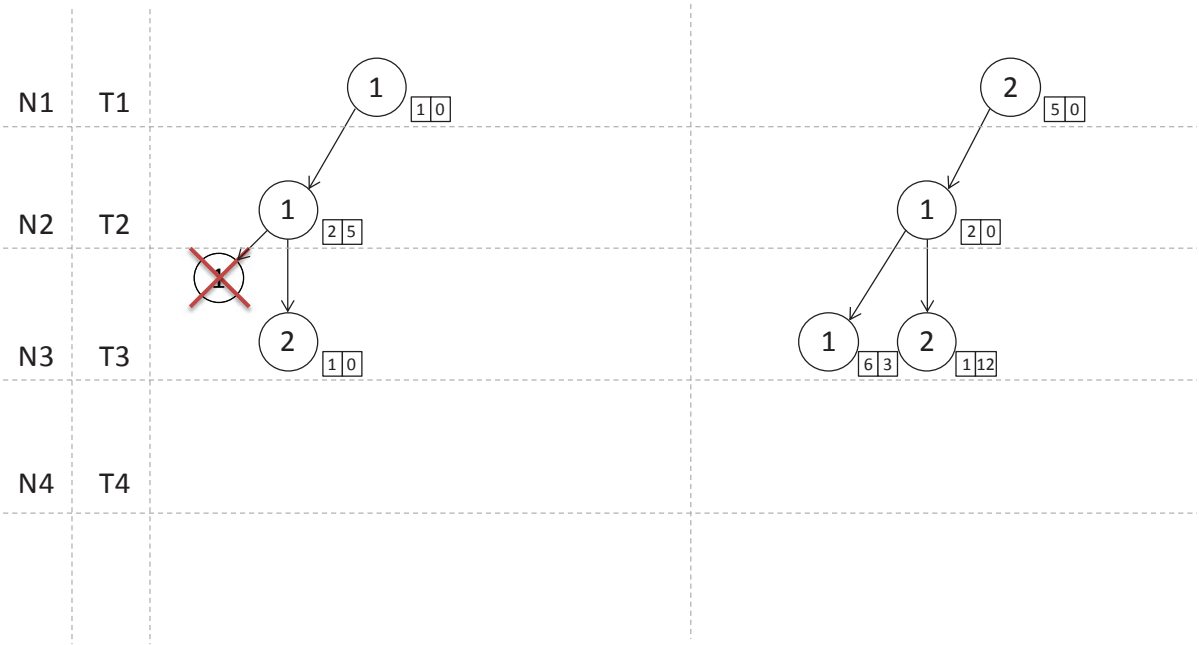


Figura 4.6: Seleção dos nós do nível 2 (passo **iv** subitem **e**) e desenvolvimento dos nós do nível 3 (passos **ii** e **iii**).

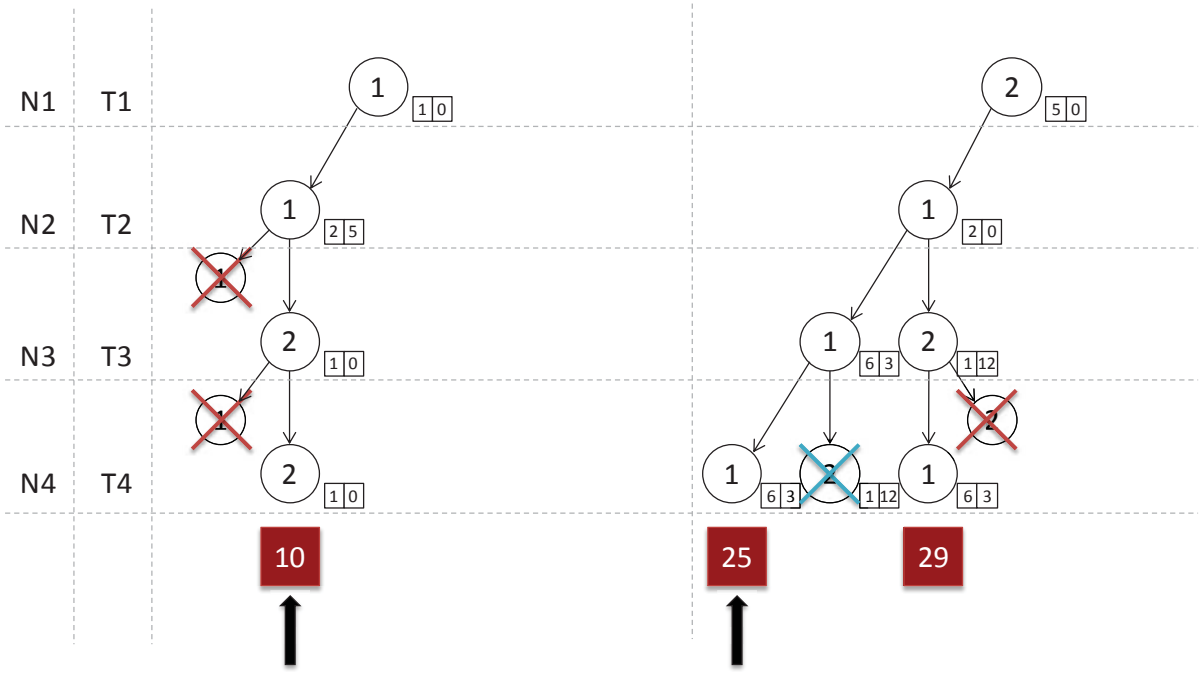


Figura 4.7: Arborescência do nível 3 (passo **iv** até subitem **d**).

A Figura 4.8 mostra os nós selecionados para continuar no nível N3, escolhidos pelas duas melhores soluções gulosas (passo **iv** subitem **e**). Desenvolvem-se, também, os nós do nível N4 e seus custos (passos **ii** e **iii**). Nota-se que não será necessário utilizar o algoritmo guloso, pois a árvore chegou ao seu último nível. Portanto, os custos calculados serão os custos totais das soluções, e dentre elas serão selecionadas somente as $\beta = 2$ melhores soluções, uma para cada feixe.

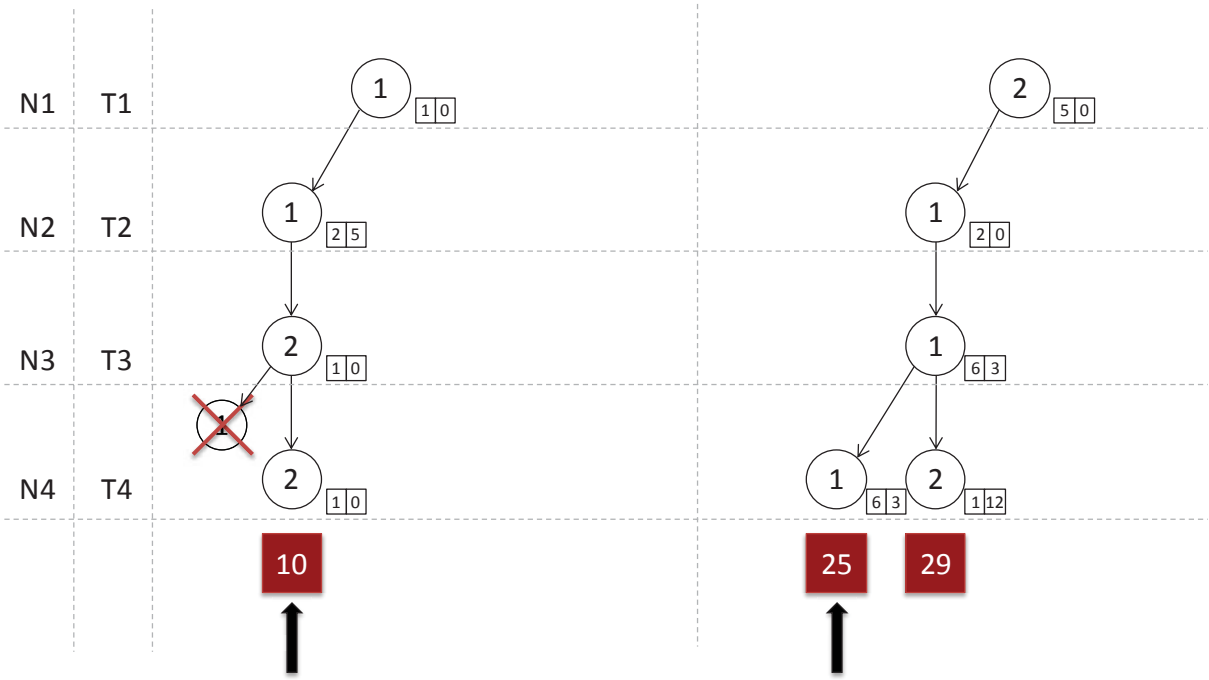


Figura 4.8: Seleção dos nós do nível 3 (passo **iv** subitem **e**) e desenvolvimento dos nós do nível 4 (passos **ii** e **iii**).

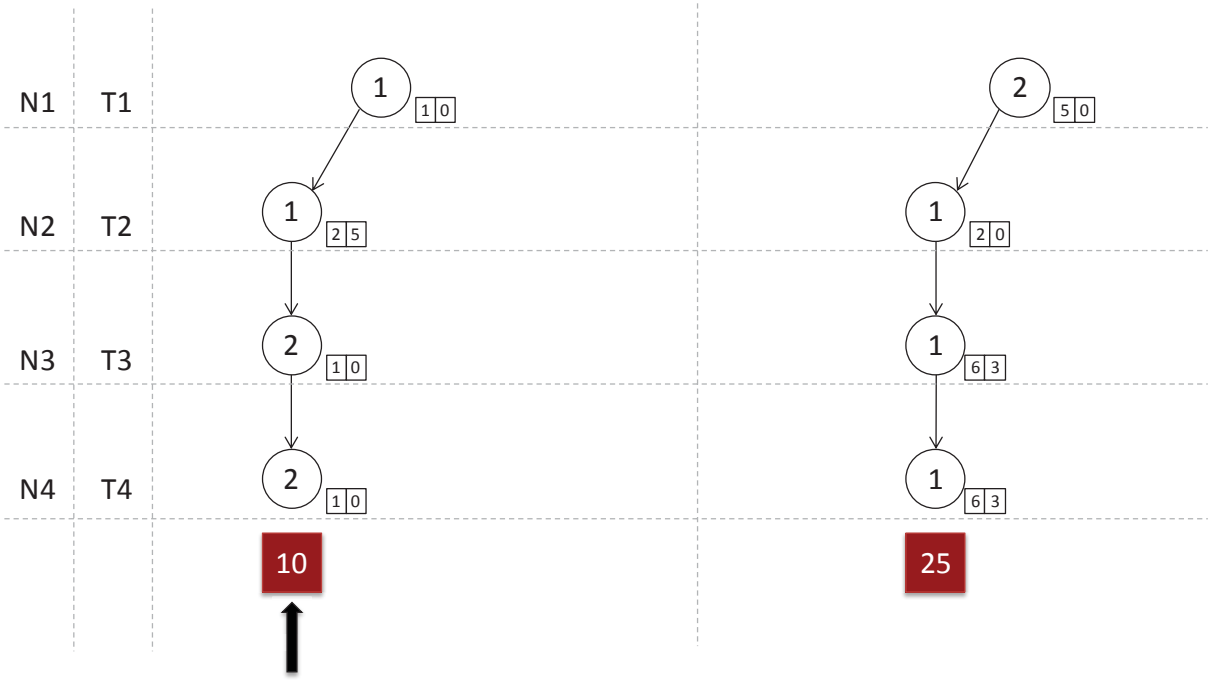


Figura 4.9: Solução do problema, com todas as atribuições realizadas.

Finalmente, a Figura 4.9 mostra a solução do problema de atribuição completo, pois o algoritmo chegou ao final já que N4 é o último nível da árvore. Assim, dentre as $\beta = 2$ melhores soluções, somente uma será considerada a solução final, a que possuir menor custo total. Então, a solução final será: turmas T1 e T2 atribuídas ao professor P1 e turmas T3 e T4 atribuídas ao professor P2.

Todo esse desenvolvimento foi realizado para mostrar como funciona o algoritmo, inclusive o algoritmo guloso. No entanto, se considerado o critério de corte de menor custo total obtido na solução gulosa, ao obter um custo total igual a 10 na Figura 4.6, toda a árvore da direita seria descartada. Isso ocorre porque os valores de custo parciais obtidos no nível N3 da árvore da direita excedem 10. Dessa forma, o limitante superior elimina boa parte da árvore juntamente com o tempo de execução do algoritmo. Além disso, deve-se considerar esse resultado somente se não houver choque de horário entre turmas alocadas para um mesmo professor, caso contrário, o resultado será diferente ou talvez o algoritmo não chegue a nenhum resultado factível.

5 O PROGRAMA COMPUTACIONAL

5.1 REQUISITOS

A partir do algoritmo desenvolvido na Seção 4 foi elaborada uma ferramenta computacional para o PATP, utilizando um conjunto de classes implementadas em Java. A linguagem Java foi escolhida devido à facilidade no desenvolvimento de interfaces gráficas. E para o desenvolvimento do programa foi utilizado o IDE JCreator.

Para arquivar os dados de professores e turmas, os custos de preferência por disciplinas e similaridade entre disciplinas e a solução do problema foram utilizados arquivos do tipo *.txt*. Uma versão de leitura automática de um banco de dados MySQL também começou a ser desenvolvida, porém o tempo que era gasto para realizar alterações no banco de dados para utilizar em pequenos testes era muito longo. Além disso, como o PATP é somente uma parte de um grande projeto, a forma como deverá ser configurado o banco de dados e suas tabelas dependerá de etapas anteriores do projeto. Assim, decidiu-se trabalhar com os arquivos em *.txt*, dispensando a configuração de um banco de dados e autenticação de usuários, e tornando rápido e prático o transporte do programa e seus dados.

O computador utilizado para o desenvolvimento do programa e seus testes foi um Notebook, com processado Intel Core2 Duo T6500 2.10GHz, com 3 GB de memória RAM. O sistema operacional usado foi o Windows 7 de 32 Bits.

5.2 DESENVOLVIMENTO DAS CLASSES

Na Seção 5.2.1, será explicada cada uma das classes que compõe o programa computacional para o PATP. A Figura 5.1 mostra o diagrama de classes com os atributos e métodos de cada classe e a relação entre elas.

Basicamente, as classes Turma e Professor são responsáveis pelo desenvolvimento dos objetos turma e professor respectivamente. Cada objeto da classe turma é composto por um objeto da classe HourTable que é responsável por criar as tabelas ou grade horária semanal de cada turma. A classe BeamSearch realiza a atribuição dos objetos turma aos objetos professor por meio do algoritmo beam search e cria as diversas soluções para o problema chamando a

classe Solucao. Solucao é a classe na qual cada objeto corresponde a um conjunto de combinações/atribuições de todos os objetos turma a objetos professor.

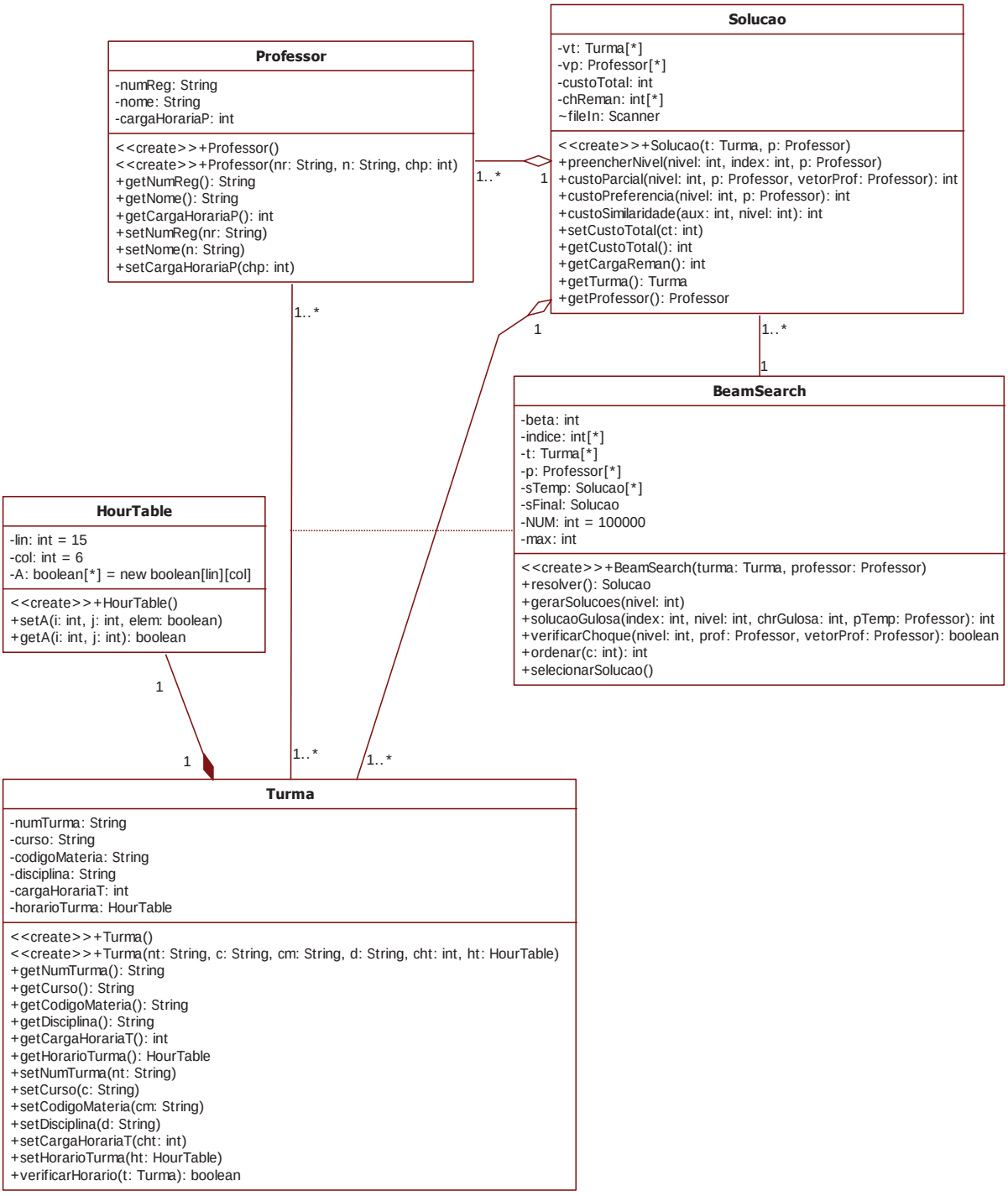


Figura 5.1: Diagrama de Classes.

5.2.1 **Descrição de atributos e métodos das classes**

a) Classe Turma

Atributos: correspondem aos elementos que identificam uma turma – número da turma, curso, código da matéria, disciplina, carga horária semanal, identificação de disciplina quinzenal ou mensal e alocação na grade horária semanal, respectivamente, conforme Figura 5.1.

Métodos:

- "getters": utilizados para recuperação dos valores dos atributos da classe Turma. Abrange os métodos: *getNumTurma()*, *getCurso()*, *getCodigoMateria()*, *getDisciplina()*, *getCargaHorariaT()*, *getQM()* e *getHorarioTurma()*.

- "setters": utilizados para atribuição de valores aos atributos da classe Turma. Abrange os métodos: *setNumTurma(String nt)*, *setCurso(String c)*, *setCodigoMateria(String cm)*, *setDisciplina(String d)*, *setCargaHorariaT(int cht)*, *setQM(String qm)* e *setHorarioTurma(HourTable ht)*.

- *verificarHorario(Turma t)*: usado para checar se existe choque de horário entre duas turmas diferentes, a turma que chamou o método e a turma que é parâmetro de entrada. Para cada *slot* das grades horárias das turmas verifica-se se as turmas estão alocadas no *slot* ou não. Se ambas estiverem alocadas em pelo menos um *slot* em comum, retorna-se *false*, caso contrário não haverá choque de horário e o método retorna *true*.

b) Classe Professor

Atributos: correspondem aos elementos que identificam um professor – número de registro, nome e carga horária semanal, respectivamente.

Métodos:

- "getters": utilizados para recuperação dos valores dos atributos da classe Professor. Abrange os métodos: *getNumReg()*, *getNome()* e *getCargaHorariaP()*.

- "setters": utilizados para atribuição de valores aos atributos da classe Professor. Abrange os métodos: *setNumReg(String nr)*, *setNome(String n)* e *setCargaHorariaP(int chp)*.

c) Classe HourTable

Atributos: correspondem aos elementos necessários para trabalhar com uma grade horária semanal: linhas representam os horários de aula, colunas representam os dias da semana e a matriz booleana representa a grade com cada um dos *slots* (*true* = *slot* cheio/turma alocada; *false* = *slot* vazio/não há turma atribuída), respectivamente.

Métodos:

- *HourTable()*: construtor da classe, inicializa a matriz A com valores *false*.

- *getA(int i, int j)*: utilizada para recuperar valores (*true* ou *false*) da matriz grade horária.

- *setA(int i, int j, boolean elem)*: utilizado para atribuir valores (*true* ou *false*) à matriz da grade horária.

A classe HourTable é usada pela classe Turma para a construção da grade horária semanal de cada turma.

d) Classe BeamSearch

Atributos: correspondem aos elementos necessários para realizar atribuições de turmas a professores utilizando o beam search – largura de busca, vetor auxiliar de armazenamento de índices para ordenar custos, vetor de turmas que serão atribuídas, vetor com todos possíveis professores, vetor de soluções temporárias, vetor de solução final, constante usada para descartar nó e variável usada para descartar solução, respectivamente.

Métodos:

- *BeamSearch(Turma[] turma, Professor[] professor)*: construtor da classe, inicializa os vetores de turma e professores para que seja possível realizar as atribuições, define a largura de busca (beta) e o valor da variável responsável pelo descarte de soluções.

- *resolver(MyTableModel dtm5, MyTableModel dtm6)*: responsável por chamar o método *gerarSolucoes* para cada nível da árvore e o método *selecionarSolucao*. Como resultado deve retornar a solução final para o problema.

- *gerarSolucoes(int nível, MyTableModel dtm5, MyTableModel dtm6)*: responsável por gerar os níveis de atribuição das soluções. Tem como parâmetro principal de

entrada o nível da árvore que está sendo gerado. Caso esteja no primeiro nível da árvore, verificam-se os professores cuja atribuição é factível segundo suas cargas horárias e cria-se um nó para cada professor. Para cada nó produzido cria-se uma solução temporária *sTemp* por meio do construtor *Solucao* e para cada solução temporária cria-se o primeiro nível utilizando o método *preencherNivel*, ambos métodos da classe Solucao. Para cada nó será calculado o custo total *ctTemp* por meio do método *solucaoGulosa*. Os custos serão ordenados usando o método *ordenar* e somente as beta soluções de menor custo continuarão existindo, as demais serão invalidadas ao ser atribuído um valor de custo total muito alto *max* através do método *setCustoTotal*.

Caso não seja o primeiro nível, desenvolvem-se os nós das beta soluções de menor custo do nível anterior, somente para as soluções válidas. Para tanto, verificam-se as cargas horárias remanescentes dos professores e se não há choque de horário entre a turma e as dos níveis anteriores caso seja o mesmo professor, para isso utiliza-se o método *verificarChoque.* Para cada nó filho desenvolvido para cada nó pai é calculado o seu custo parcial *cTemp* pelo método *custoParcial* da classe Solucao e é aplicado o algoritmo guloso, obtendo-se o custo total *ctTemp*. Para cada nó pai, os custos são ordenados e somente o nó factível e com a solução de menor custo total será escolhido para fazer parte das respectivas soluções. Assim, para cada solução temporária cria-se um novo nível, a partir do nó selecionado, utilizando o método *preencherNivel*. Se nenhum nó for factível devido ao choque de horário ou carga horária insuficiente, a solução deverá ser invalidada.

- *solucaoGulosa(int index, int nivel, int[] chrGulosa, Professor[] pTemp, MyTableModel dtm5, MyTableModel dtm6)*: algoritmo guloso usado para calcular os custos totais dos nós. Tem como parâmetro principais de entrada o índice de identificação da solução temporária que está sendo resolvida, o nível da árvore no momento, um vetor para armazenamento das cargas horárias remanescentes dos professores e um vetor com todas as atribuições de professores feitas até o momento. Para o nó em questão desenvolvem-se os próximos níveis. A cada nível criam-se os nós considerando as cargas horárias remanescentes dos professores *chrGulosa* e se não há choque de horário entre a turma do nível e as dos níveis anteriores. Calculam-se os custos parciais *cGuloso* dos nós no nível pelo método

custoParcial e ordenam-se os nós conforme custo. Somente o nó factível com menor custo parcial continuará na árvore. Da mesma forma, todo o processo é repetido para os demais níveis até o último nível da árvore, quando se obtém o custo total. Se em algum nível de desenvolvimento da solução gulosa não existir nenhum nó factível a solução toda deverá ser invalidada por meio da soma de um valor bem alto *NUM* a cada nível seguinte. As cargas horárias remanescentes dos professores deverão ser atualizadas a cada nível e as novas atribuições deverão ser armazenadas no vetor *pTemp*. O método deverá retornar o custo total do nó.

- *verificarChoque(int nivel, Professor prof, Professor[] vetorProf)*: usado para verificar se a turma do nível em questão tem choque de horário com alguma outra dos níveis anteriores. Seus parâmetros de entrada são o nível, o professor que será atribuído ao nó em questão e o vetor de professores já atribuídos às turmas dos níveis anteriores. Comparam-se os professores de cada nível anterior com o professor do nível em questão. Caso seja o mesmo professor, o método *verificarHorario* da classe Turma é chamado para verificar se há choque de horário entre a turma do nível comparado e a turma em questão. O método retorna *false* se houver choque e *true* se não houver choque.

- *ordenar(int[] c)*: utilizado para ordenar vetores. O parâmetro de entrada é o vetor que deverá ser ordenado. A princípio cria-se um vetor com o valor dos índices do vetor de entrada. Comparam-se os valores do vetor de entrada um a um trocando os valores do vetor de índices de lugar. Assim, é possível manter as posições iniciais do vetor de entrada e obter os novos índices desses valores. Os índices cujo valor do vetor de entrada são menores deverão ficar em posições baixas no vetor de índice, ou seja, em ordem crescente. Por exemplo:
  - vetor de entrada: c[0] = 4, c[1] = 1, c[2] = 2
  - vetor de índices: ind[0] = 0, ind[1] = 1, ind[2] = 2
  - após ordenar, o vetor de índices ficará: ind[0] = 1, ind[1] = 2, ind[2] = 0
  - o vetor de entrada em ordem ficará: c[ind[0]] = 1, c[ind[1]] = 2, c[ind[2]] = 4

  Finalmente, o método deverá retornar o vetor de índices.

- *selecionarSolucao()*: método responsável por selecionar a solução de menor custo total. Deverá armazenar os custos de cada solução em um vetor de custo total *ct*. Uma nova solução deverá ser criada por meio do construtor *Solucao*. O vetor *ct* será ordenado e a solução de menor custo será selecionada, caso seja uma solução factível, ou seja, seu custo deverá ser menor que *max*.

e) Classe Solucao

Atributos: correspondem aos elementos que identificam uma solução ao problema – vetor de turmas que deverá guardar as turmas de cada nível, vetor de professores que deverá guardar os professores atribuídos a cada nível, custo total da solução e vetor de carga horária remanescente de cada professor, respectivamente.

Métodos:

- *Solucao(Turma[] t, Professor[] p)*: construtor da classe, inicializa os vetores de turmas, professores e carga horária remanescente para que seja possível realizar as atribuições e o custo total. Tem como parâmetros de entrada o vetor de turmas a serem atribuídas e o vetor de possíveis professores a serem alocados nas turmas.

- *preencherNivel(int nivel, int index, Professor p, MyTableModel dtm5, MyTableModel dtm6)*: responsável por preencher cada nível da solução. Seus parâmetros principais de entrada são o nível em desenvolvimento, o índice da solução em desenvolvimento e o professor a ser atribuído. Deve-se primeiro atribuir o professor ao vetor de professores *vp* na posição correspondente ao nível. A seguir calcula-se o custo total da solução até o nível em questão. Para isso, é necessário utilizar o método *custoParcial* para obter o custo do nó no nível. Finalmente, atualiza-se a carga horária remanescente do professor que foi atribuído.

- *custoParcial(int nivel, Professor p, Professor[] vetorProf, MyTableModel dtm5, MyTableModel dtm6)*: utilizado para o cálculo do custo no nó em questão. Os parêmetros de entrada são o nível em questão, o professor atribuído à turma do nível e o vetor de professores atribuídos às turmas anteriores. Comparam-se os professores de cada nível anterior com o professor do nível em questão. Caso seja o mesmo professor, o método deverá calcular o custo de similaridade entre as disciplinas por meio do método *custoSimilaridade*. Então somam-se o custo de

similaridade ao custo de preferência dado pelo método *custoPreferencia*. O método deverá retornar o resultado da soma.

- *custoPreferencia(int nivel, Professor p, MyTableModel dtm5)*: utilizado para obter o custo de preferência de um dado professor por uma dada matéria. Tem como parâmetros de entrada o nível em que está a turma em questão, o professor que está sendo atribuído à turma e o MyTableModel onde estão os valores de preferência. O método irá buscar no *"MyTableModel dtm5"* o valor correspondente ao custo de preferência do professor da entrada para ministrar a disciplina correspondente à turma do nível em questão. O método deverá retornar o custo de preferência.

- *custoSimilaridade(int aux, int nível, MyTableModel dtm6)*: utilizado para obter o custo de similaridade entre duas disciplinas. Tem como parâmetros de entrada o nível *aux* em que está a turma em comparação, o nível que está a turma em questão e o MyTableModel onde estão os valores de similaridade. O método irá buscar no *"MyTableModel dtm6"* o valor correspondente ao custo de similaridade da disciplina do nível em relação à disciplina do nível em questão. O método deverá retornar o custo de similaridade.

- "setters": utilizados para atribuição de valores aos atributos da classe Solucao. Abrange os método: *setCustoTotal(int c).*

- "getters": utilizados para recuperação dos valores dos atributos da classe Solucao. Abrange os métodos: *getCustoTotal()*, *getCargaReman()*, *getTurma()* e *getProfessor()*.

5.3 ELABARORAÇÃO DA INTERFACE GRÁFICA

Além das classes citadas na Seção 5.3.2, foi elaborada uma classe que implementa uma interface gráfica para o PATP. As figuras que seguirão durante esta seção irão mostrar o visual e o funcionamento da interface, juntamente com algumas explicações.

A Figura 5.2 mostra a primeira tela da interface, onde se pode verificar um *JMenu* (barra de menu) e seus *JMenuItem* (subitens da barra de menu). Os *JMenusItem* são: Novo, Abrir e Salvar arquivos. É possível criar um novo arquivo do zero, abrir e salvar arquivos de cinco tipos: arquivos de dados das turmas, arquivos de dados dos professores, arquivos de valores de preferência por disciplina, arquivos de valores de similaridade entre disciplinas e arquivos com as atribuições (solução do problema de atribuição).

Cada um dos tipos de arquivo possui sua aba correspondente, então o usuário deverá escolher a aba e depois clicar no menu: "novo" para criar um arquivo novo correspondente à aba, "abrir" para que o arquivo seja aberto no local correto e "salvar" para que o arquivo da aba seja salvo.

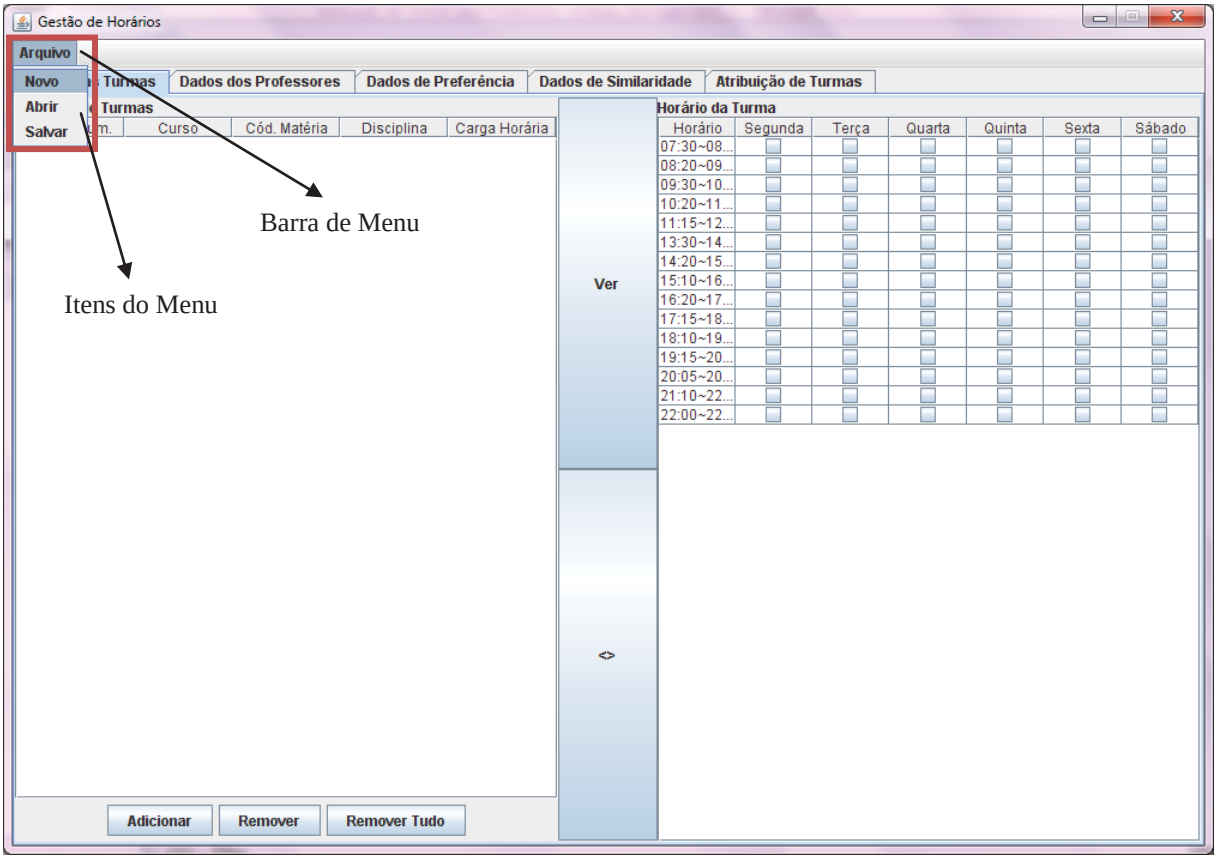


Figura 5.2: Barra de menu - Arquivo e itens do menu – Novo, Abrir e Salvar.

Apenas um arquivo por aba pode ser aberto de cada vez, caso o usuário tente abrir um arquivo em cima de outro surgirá uma caixa de diálogo alertando-o que não é possível realizar a ação. A caixa de seleção para a abertura de arquivo está ilustrada na Figura 5.3.

Além disso, ao abrir um arquivo é possível editar o arquivo clicando sobre a informação que se deseja alterar ou por meio de *JButtons*. Os *JButtons* possibilitam ao usuário a inserção

ou a remoção de linhas e colunas. Pode-se, também, remover o arquivo todo caso o usuário tenha carregado o arquivo errado.

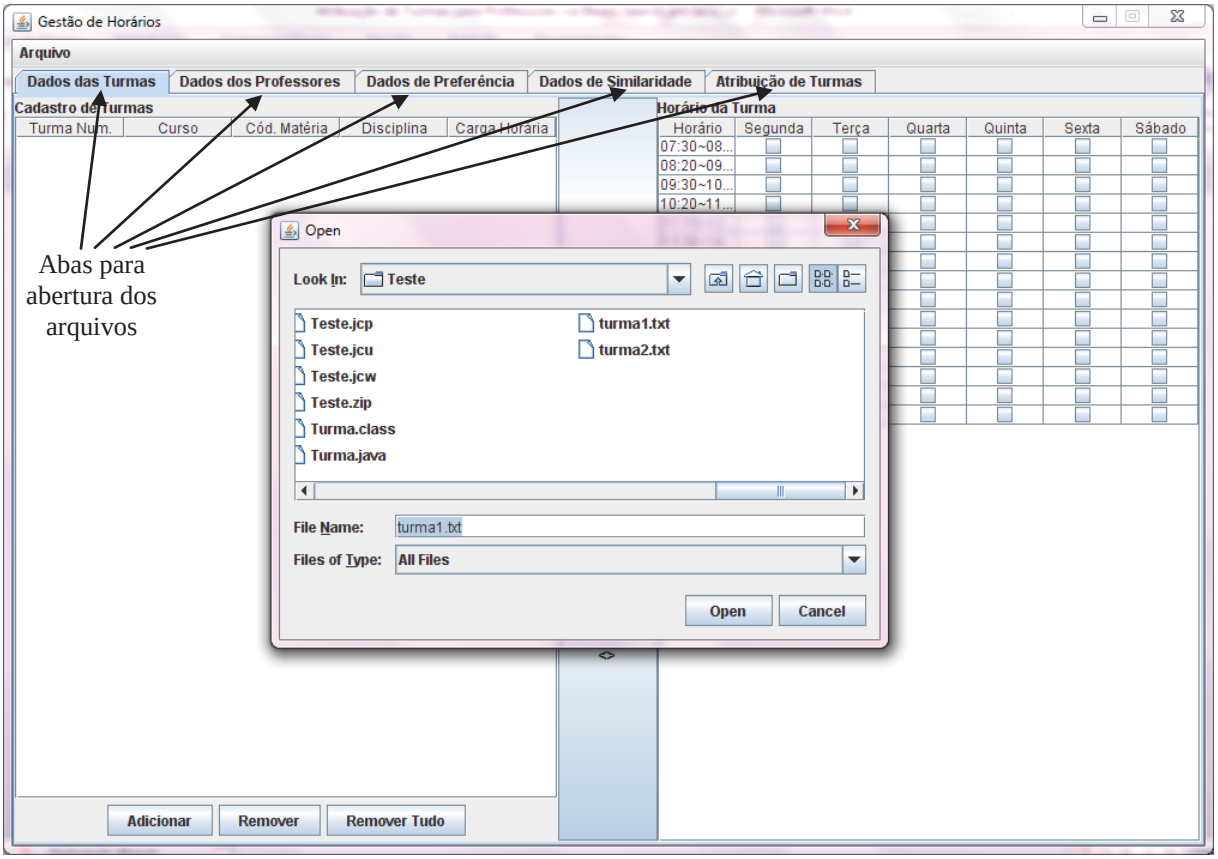


Figura 5.3: Abas e caixa de seleção de abertura de arquivo.

Para visualizar onde está alocada a turma na grade horária semanal deve-se clicar sobre a linha na qual a turma se encontra no painel da esquerda e clicar no botão ver no centro da janela. Os *slots* nos quais a turma está alocada serão sinalizados com um *check* no painel da direita. Pode-se também realizar alterações na grade horária clicando sobre os quadradinhos dos *slots* e, em seguida, clicando no botão de associação < >, para isto, a turma que receberá a nova associação deverá estar previamente selecionada. A Figura 5.4 mostra o arquivo aberto na aba "Dados das turmas" e os *JButtons* citados.

O processo de abertura e edição de arquivo também é possível para a aba "Dados dos Professores". A diferença está no painel da direita que não existe no caso dos professores, logo não há botões de associação nem visualização da grade horária. A Figura 5.5 mostra um arquivo de professores abertos na aba "Dados dos Professores".

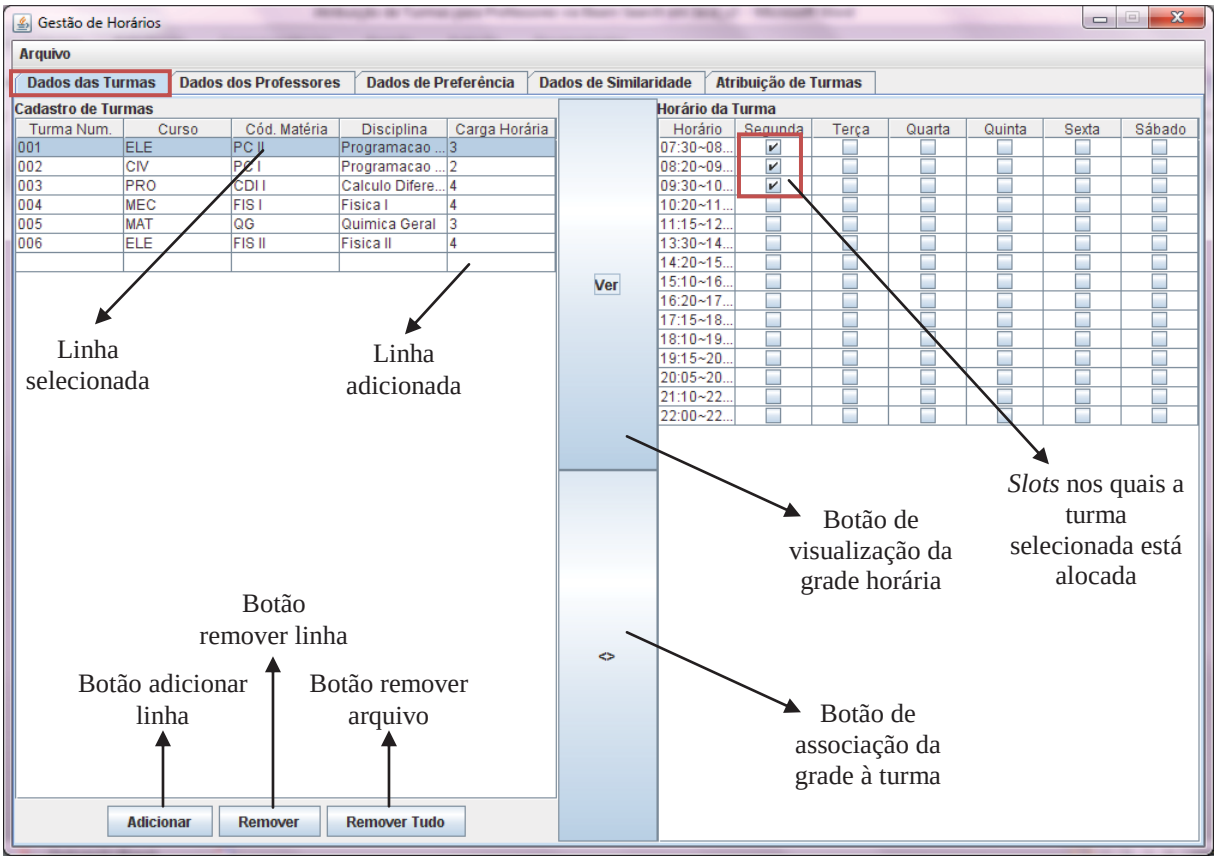


Figura 5.4: Arquivo carregado na aba "Dados das Turmas" e botões de manipulação das informações.

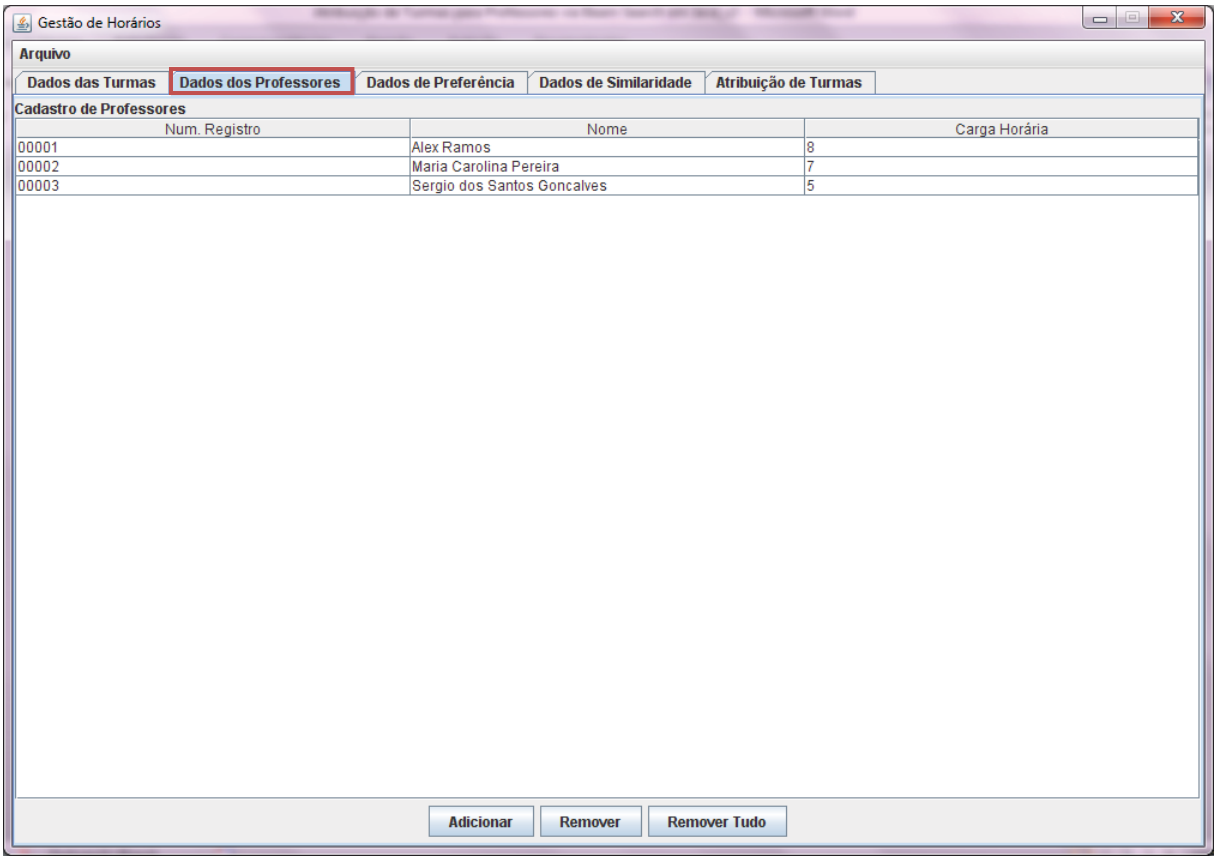


Figura 5.5: Arquivo carregado na aba "Dados dos Professores".

Para as abas “Dados de Preferência” e “Dados de Similaridade”, existem os botões de inserção e remoção de colunas. Ao clicar no botão de inserção o usuário deverá fornecer a sigla correspondente à disciplina à qual quer atribuir valores de preferência ou similaridade. Para excluir uma coluna o usuário deverá selecioná-la e então clicar em remover coluna. A Figura 5.6 ilustra a caixa de diálogo onde o usuário deverá fornecer a sigla e os dados carregados na aba “Dados de Preferência”.

A aba “Dados de Similaridade” é mostrada na Figura 5.7 com seus dados carregados, pode-se notar que a diferença em relação à aba de preferências é a primeira coluna que ao invés de listar professores, lista as diversas disciplinas existentes, já que o intuito é avaliar a similaridades entre disciplinas.

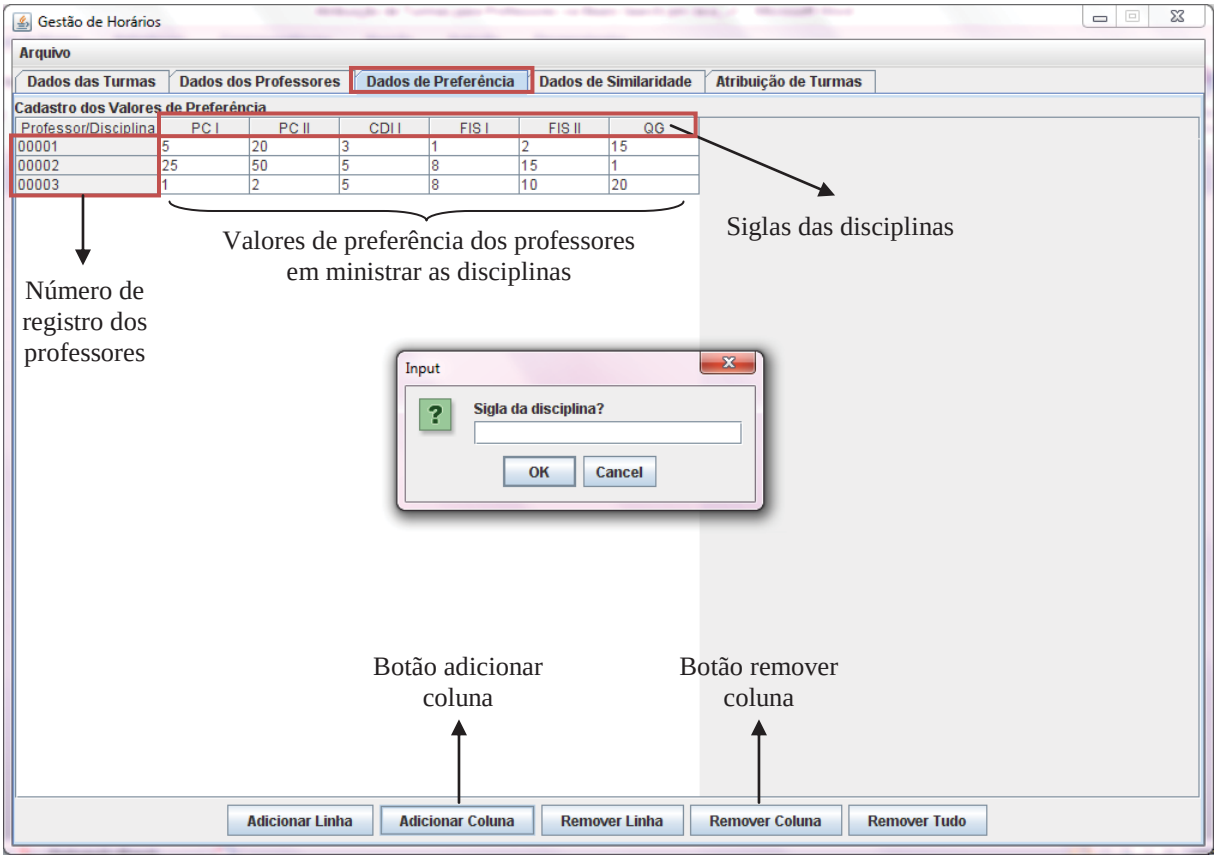


Figura 5.6: Arquivo carregado na aba “Dados de Preferência” e caixa de diálogo para fornecimento da sigla.

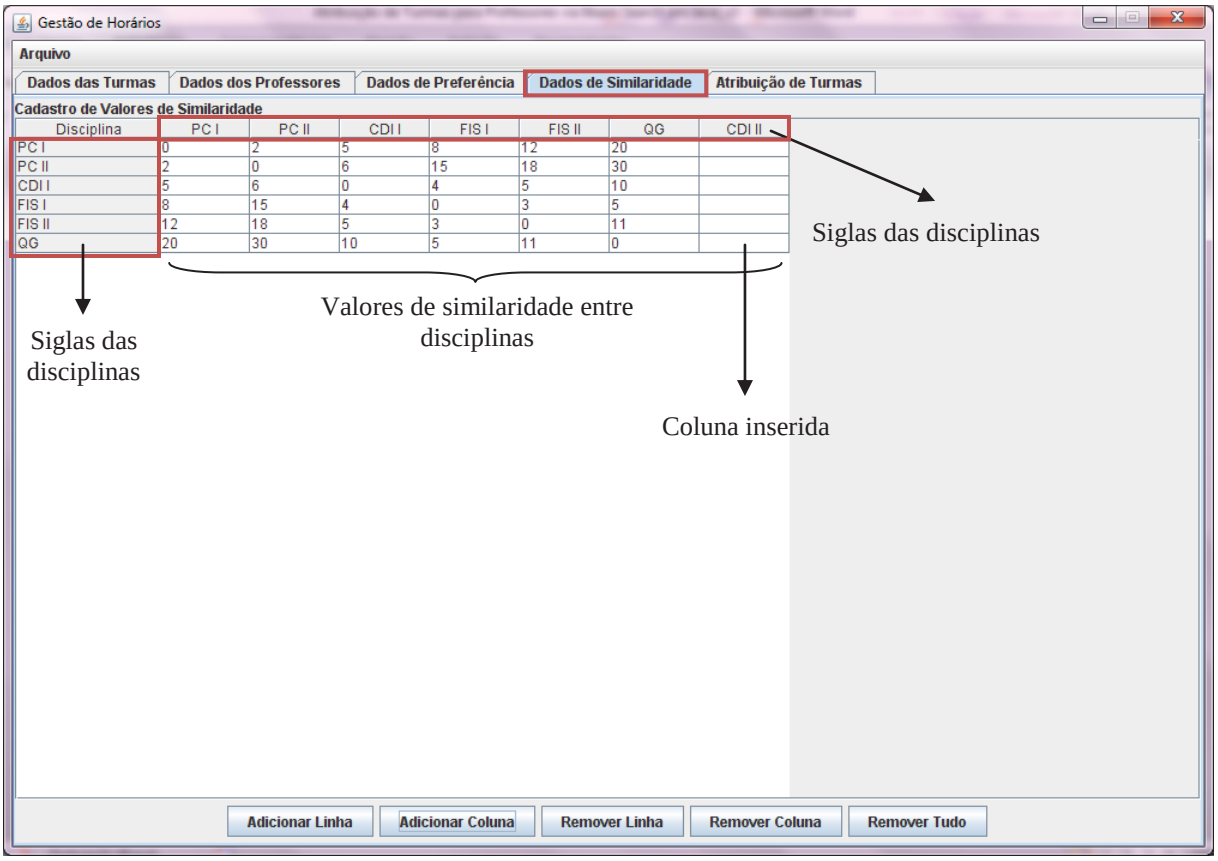


Figura 5.7: Arquivo carregado na aba "Dados de Similaridade".

A última aba de atribuições é bastante similar a aba de dados das turmas, porém junto com os botões de adição e remoção de linhas existe um botão "Atribuir", cuja função é chamar o método BeamSearch para que seja possível realizar as atribuições das turmas aos professores. É importante ressaltar que as atribuições serão feitas utilizando todos os dados inseridos até o momento, ou seja, todas as alterações contam independentemente se foram salvas ou não!

Após clicar no botão de atribuição, o programa automaticamente verifica se há informação nas abas: "Dados das Turmas", "Dados dos Professores", "Dados de Preferência" e "Dados de Similaridade". Caso alguma aba não tiver nenhum dado carregado o programa retorna uma caixa de diálogo sinalizando a falta de dados. Se todas as abas possuírem informações, a interface lançará uma caixa de diálogo solicitando ao usuário que entre com o valor da largura de busca. Isso é mostrado na Figura 5.8.

Ao inserir o valor da largura de busca, o usuário deverá lançar um valor maior que zero e em caracteres numéricos, caso contrário o programa irá solicitar que o usuário insira um novo valor válido.

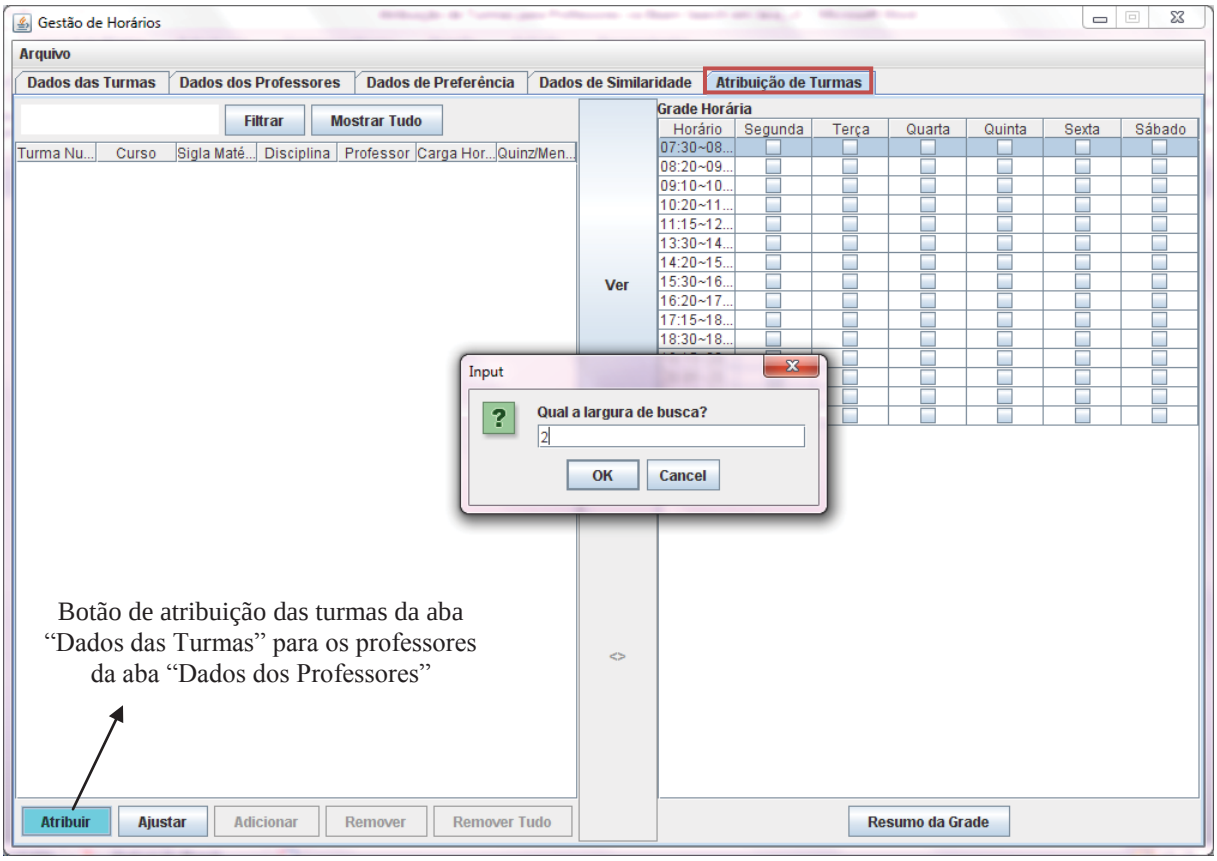


Figura 5.8: Aba "Atribuição de Turmas", caixa de diálogo para inserção da largura de busca e botão de atribuição.

A Figura 5.9 mostra o resultado das atribuições realizadas por meio do algoritmo beam search. No painel à esquerda é mostrada a lista de turmas com seus respectivos professores e no painel direito pode-se verificar as grades horárias das turmas utilizando o botão "Ver". A interface gráfica permite também que o usuário faça algumas alterações no resultado final, para isso, deve-se clicar no botão "Ajustar". Ao clicar no botão "Ajustar", uma caixa de diálogo é lançada com uma mensagem alertando o usuário que isso poderá alterar o resultado final obtido pelo processo e se o usuário deseja continuar mesmo assim. Se a resposta for positiva, os botões inativos passam para o estado ativo e é possível fazer ajustes manuais na solução, como é mostrado na Figura 5.9.

Além do botão de ajuste, a aba de atribuições contém também uma caixa de texto onde o usuário poderá digitar a identificação do professor para realizar um filtro no resultado e facilitar a visualização de todas as turmas que um determinado professor possui. Para filtrar, deve-se clicar no botão "Filtrar" e, para visualizar o resultado completo novamente, deve-se clicar em "Mostrar Tudo". Na região inferior do painel direito existe um último botão "Resumo da Grade", que ao ser clicado mostra a grade com todos os horários da semana do professor filtrado. Isso é mostrado na Figura 5.10.

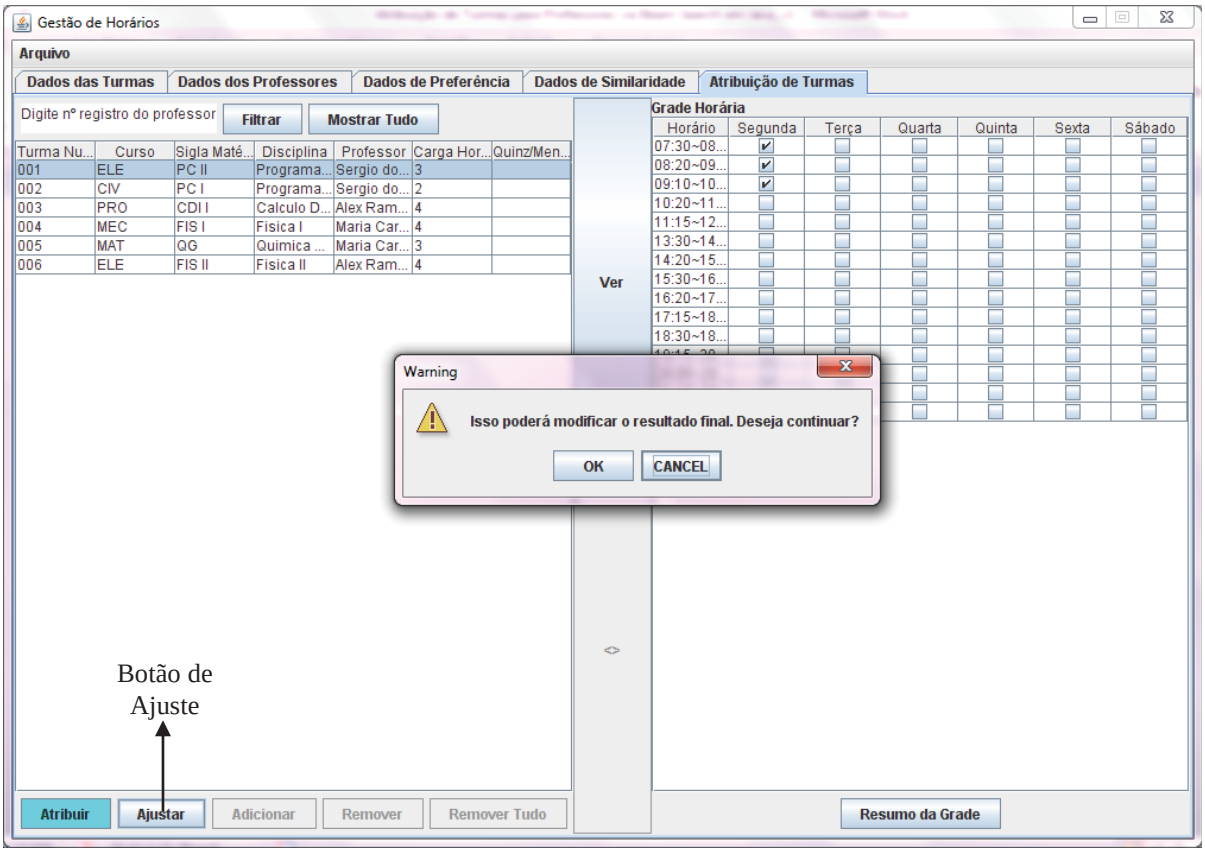


Figura 5.9: Lista com atribuições realizadas, botão de ajuste e caixa de alerta.

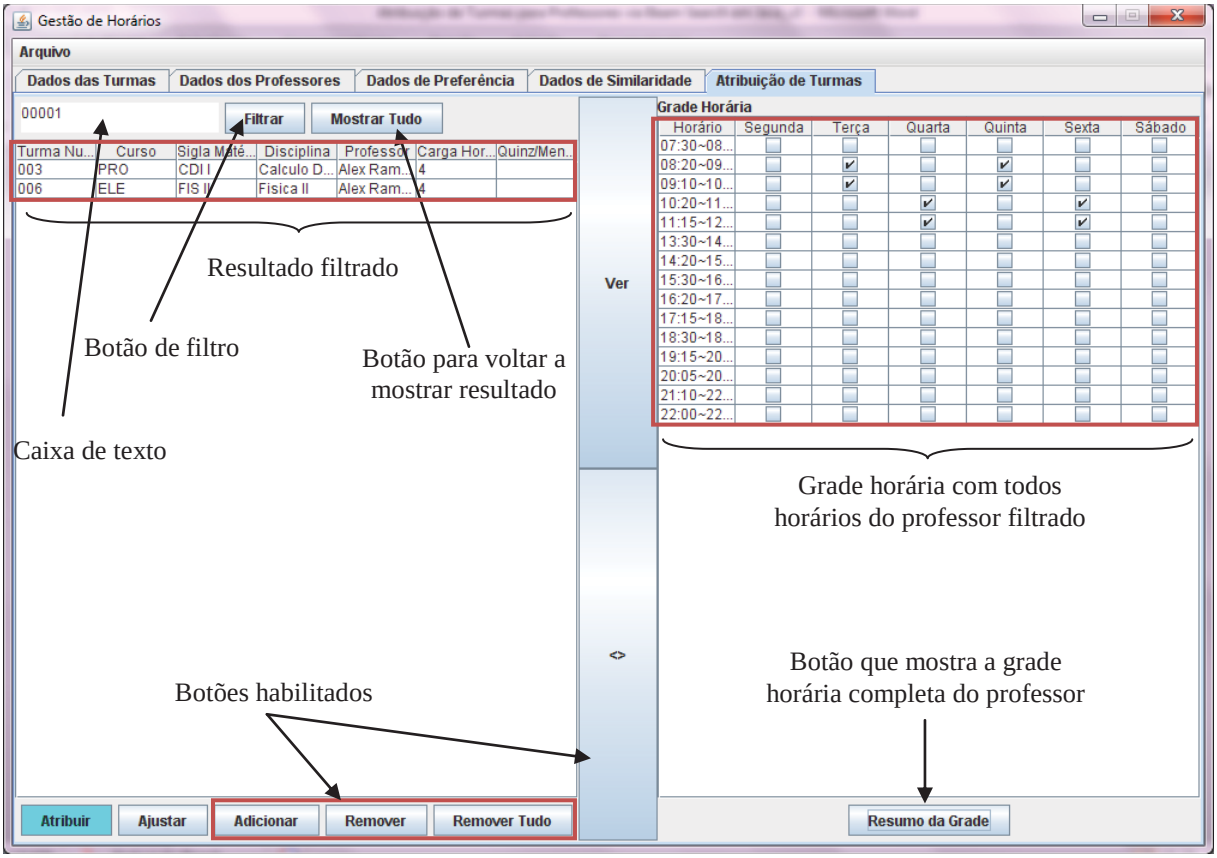


Figura 5.10: Botões de manipulação de informações habilitados, caixa de texto para filtro, botões de filtro e resumo de grade horária.

6 TESTE E VALIDAÇÃO DOS RESULTADOS

Alguns testes utilizando exemplos de turmas e professores foram realizados para comprovar o funcionamento do programa, inclusive as imagens mostradas na seção 5.3 são de uma das simulações.

Para comparação e validação dos resultados obtidos, o mesmo exemplo desenvolvido na Seção 4.2.1 foi executado. A seguir serão mostradas as imagens capturadas durante a simulação.

As Figuras 6.1 e 6.2 mostram os arquivos com as listas de turmas e professores carregadas em suas abas respectivamente. A ordem dos campos dados às colunas é a mesma ordem dos atributos dado no diagrama de classes, como pode ser notado pelos seus nomes. Deve-se notar também que os valores das cargas horárias das turmas e dos professores são os mesmos dados na Seção 4.1 nas Tabelas 4.1 e 4.2. Além disso, as disciplinas das turmas 3 e 4 são idênticas entre si, assim como as do exemplo. Os valores de preferência e similaridade utilizados na simulação são os mesmo das Tabelas 4.3 e 4.4 da Seção 4.1. Da mesma, forma a Figura 6.5 mostra a largura de busca estipulada que é igual a 2.

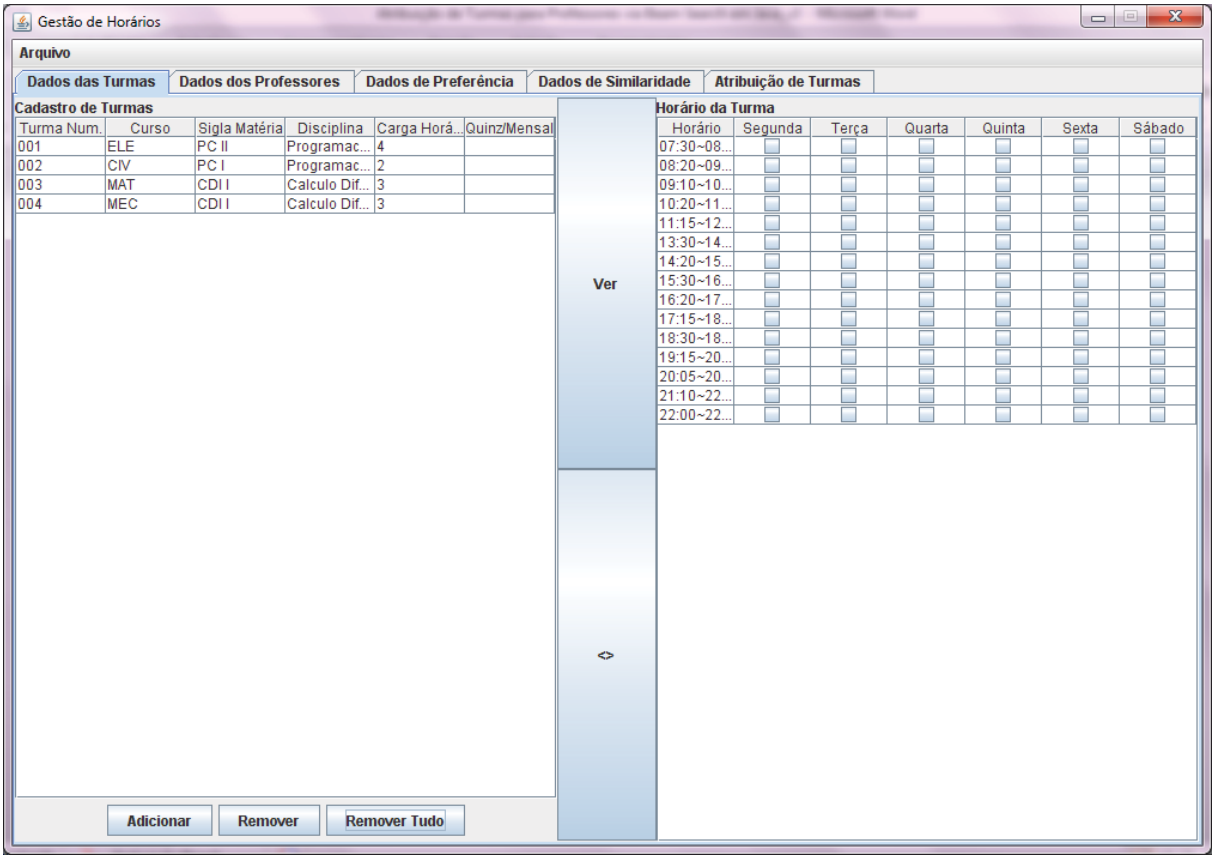


Figura 6.1: Dados das turmas.

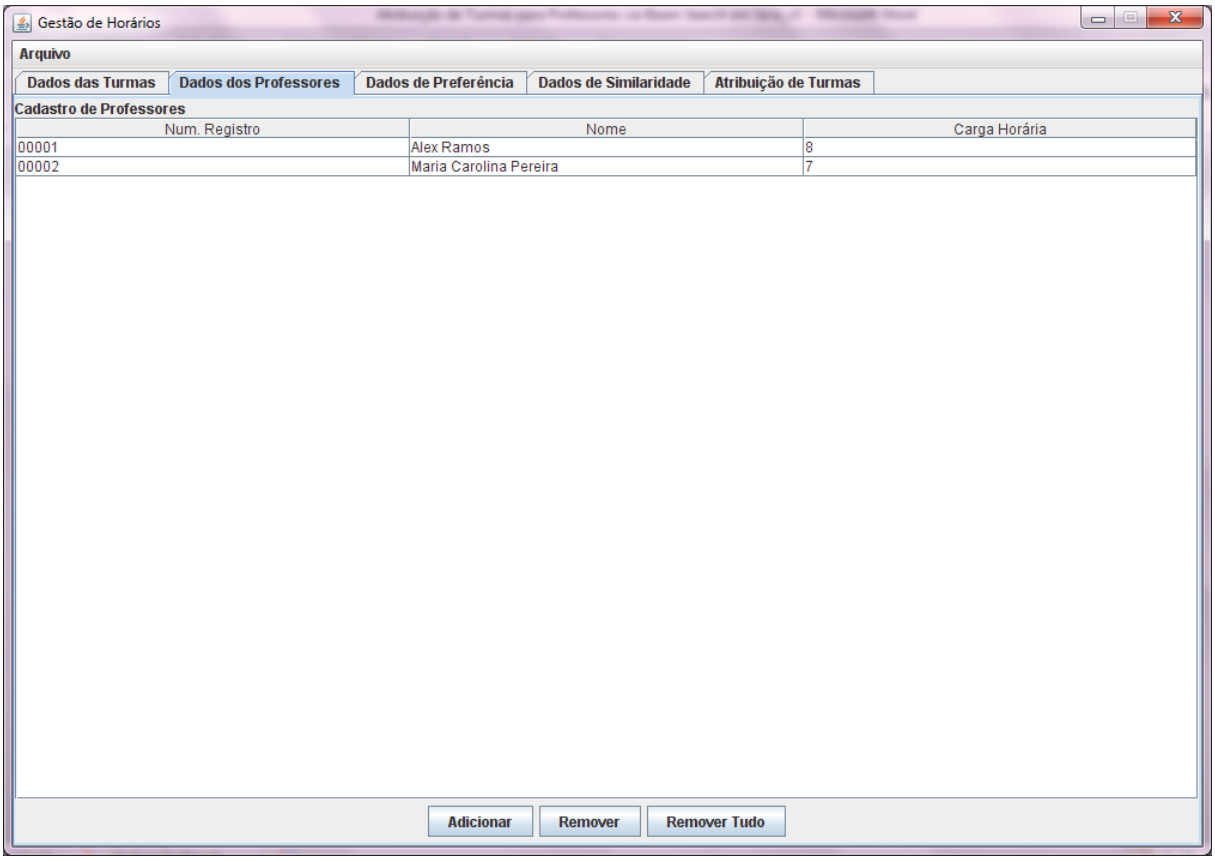


Figura 6.2: Dados dos professores.

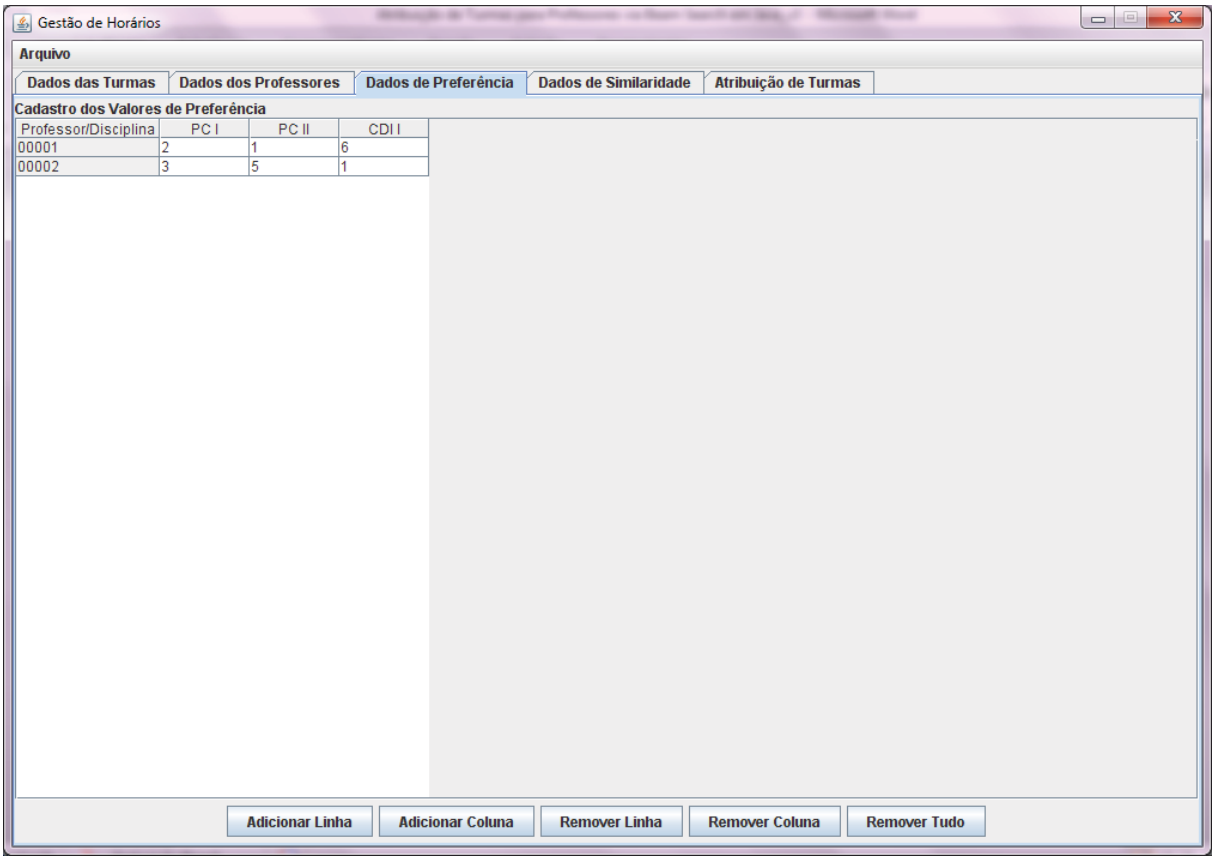


Figura 6.3: Valores de preferência.

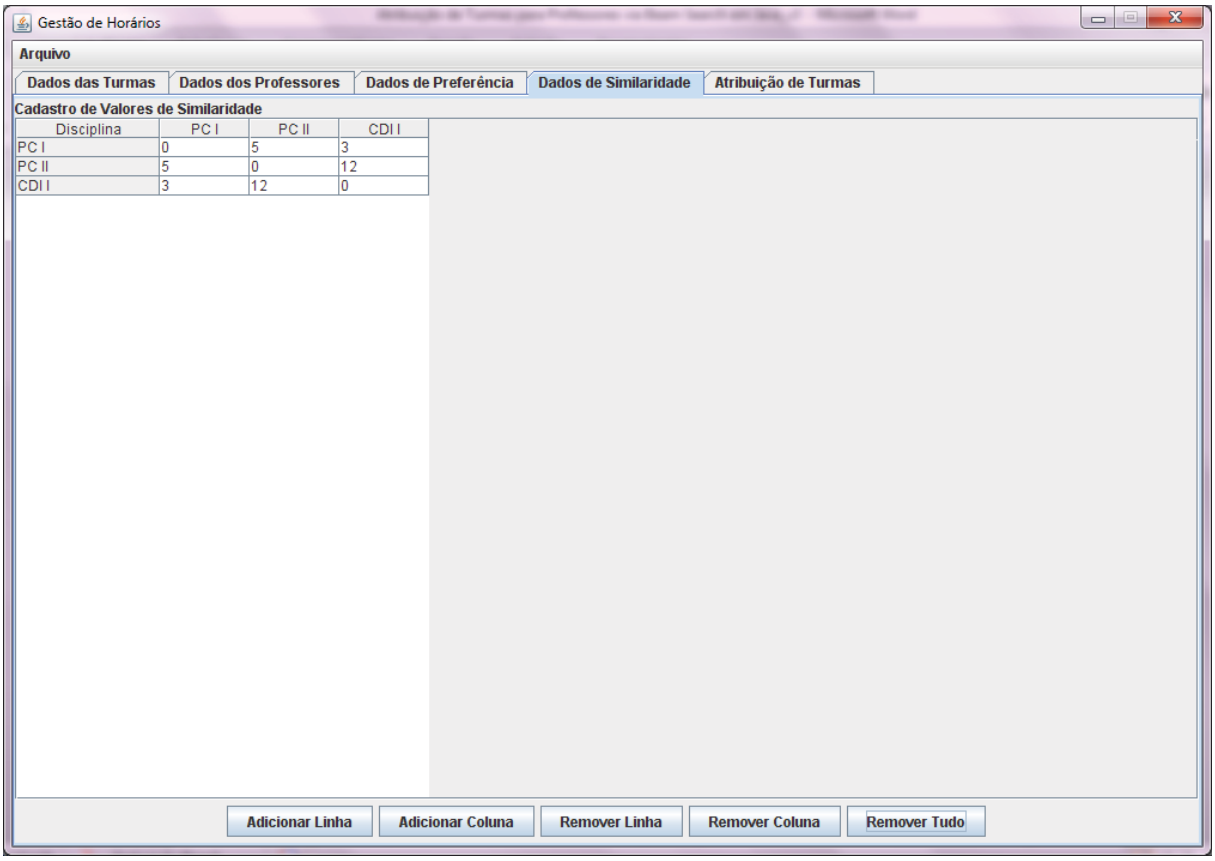


Figura 6.4: Valores de similaridade.

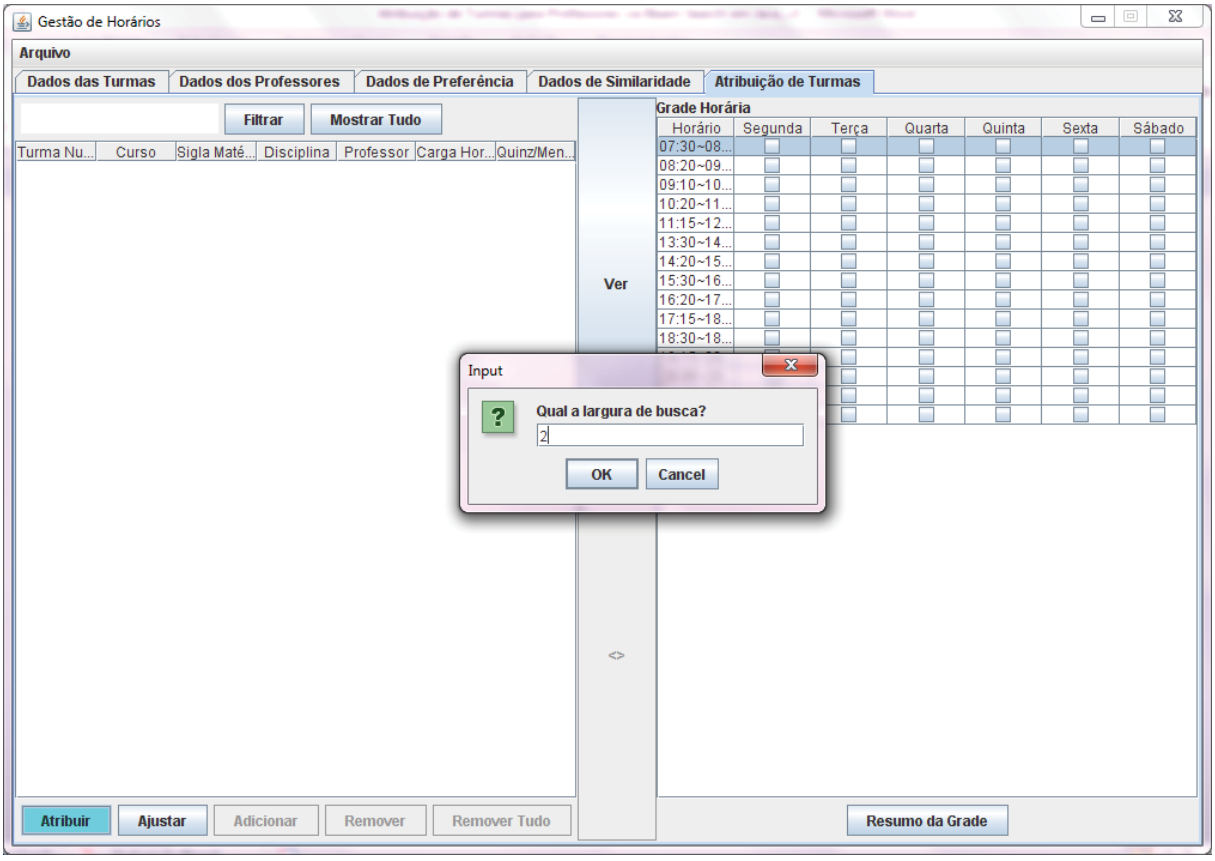


Figura 6.5: Inserção da largura de busca igual a 2.

A Figura 6.6 traz o resultado final da simulação. É possível perceber que os campos das colunas das turmas são mantidos e somam-se a eles uma coluna denominada “Professor”, nesta coluna são feitas as atribuições. Assim, para cada turma tem-se um professor responsável por ministrar suas aulas.

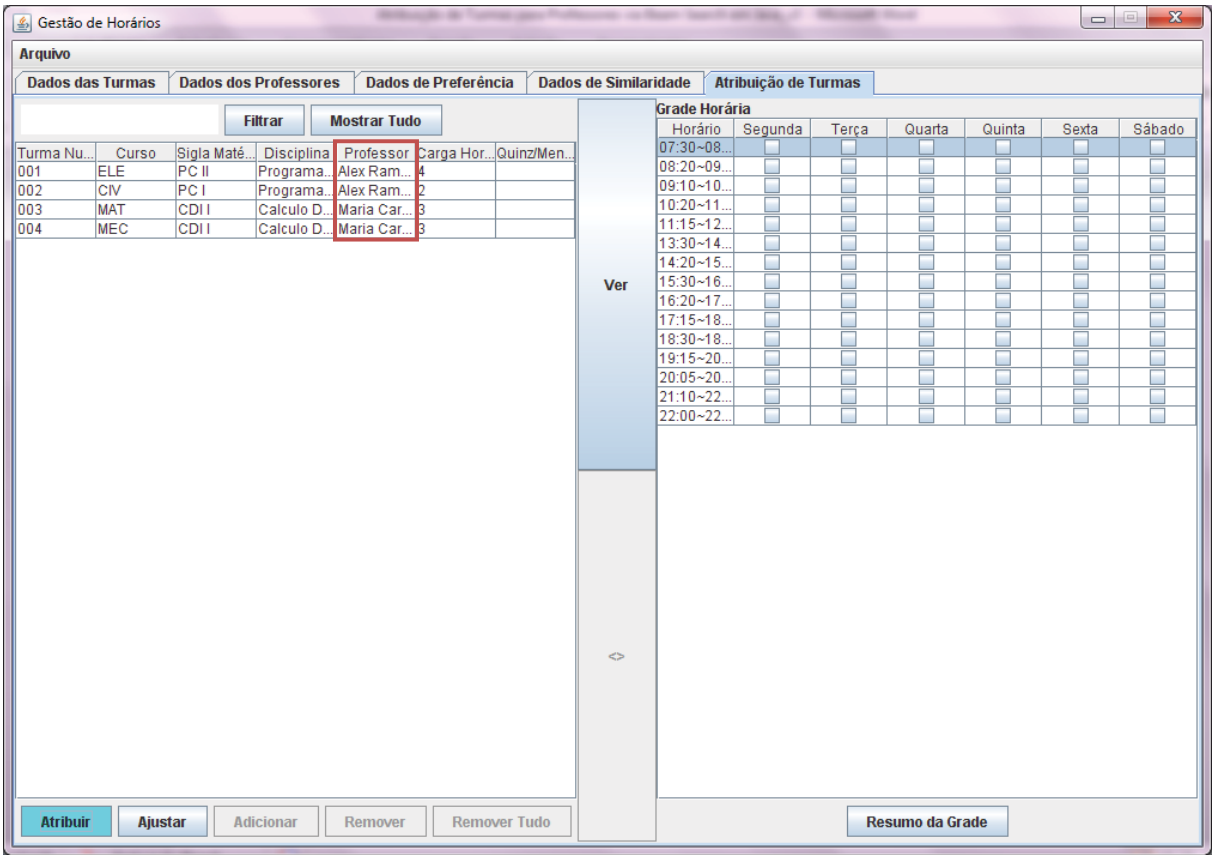


Figura 6.6: Resultado da simulação.

Realizando uma análise do resultado, temos que para as turmas T1 e T2 o professor responsável será P1, e para as turmas T3 e T4 o professor P2. Os números das turmas e dos professores são dados na primeira coluna das tabelas da primeira e segunda aba, respectivamente.

6.1 SIMULAÇÃO COM DADOS REAIS

Comprovado o funcionamento correto da interface e do programa, realizou-se uma simulação de atribuição com dados reais usados na distribuição das disciplinas do Departamento de Matemática na Área de Computação entre os professores do departamento. Os dados utilizados datam de 2011 e estão disponíveis no Anexo no final deste documento.

A simulação foi realizada para turmas do 1º semestre de 2011 e as restrições de carga horária dos professores foi rigorosamente seguida. Os valores de preferência dos professores foram criados baseado no documento em anexo. Assim, os valores de preferência foram criados e manipulados para que o resultado obtido na simulação fosse o mais próximo possível dos realizados à mão. Em outras palavras, adotou-se que o resultado à mão era o mais próximo ao "ideal" para os professores em questão.

Para criar o arquivo com os valores de similaridade entre disciplinas utilizou-se de algumas informações sobre as disciplinas em questão e um pouco de bom senso. A escolha do beta foi tal que pudesse maximizar o range de resultados, minimizando a chance de eliminar a solução ótima e soluções próximas à ótima. A seguir seguem algumas imagens da simulação realizada.

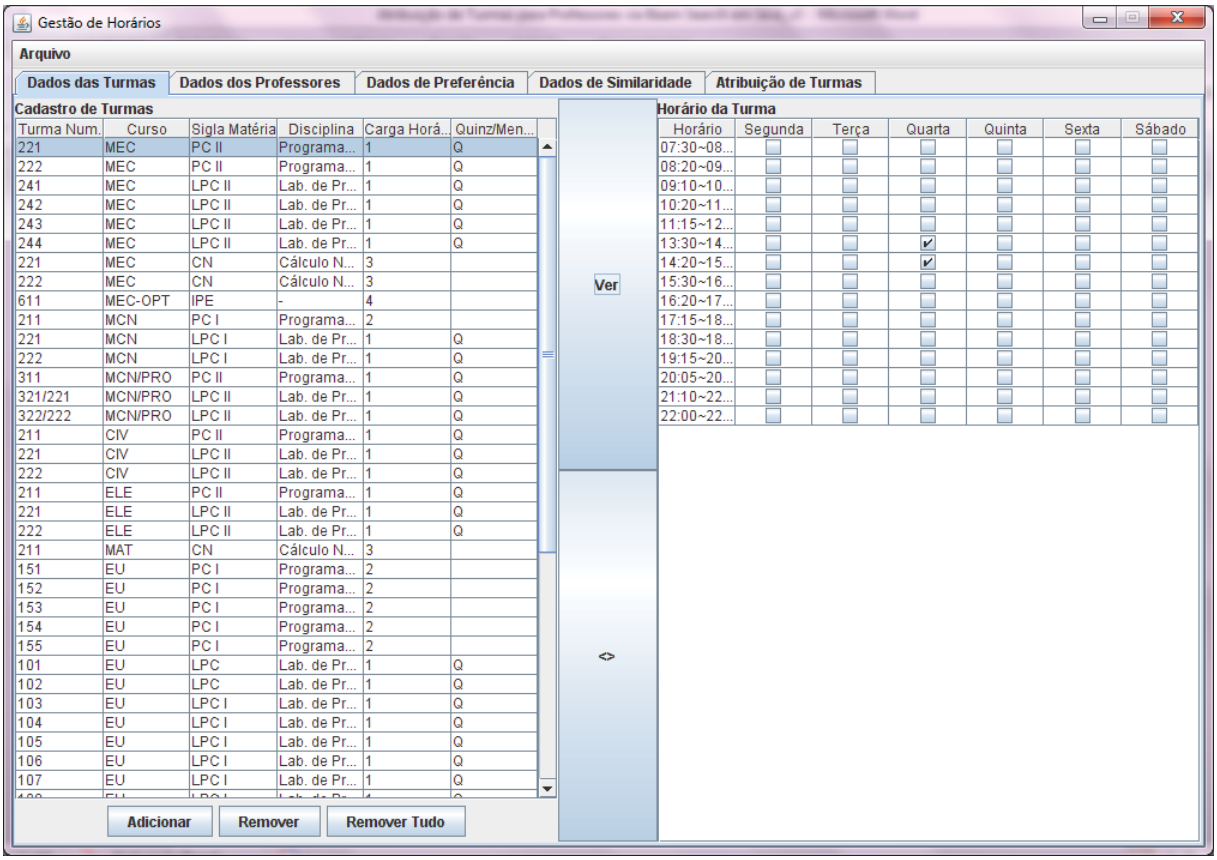


Figura 6.7: Teste com dados reais – turmas.

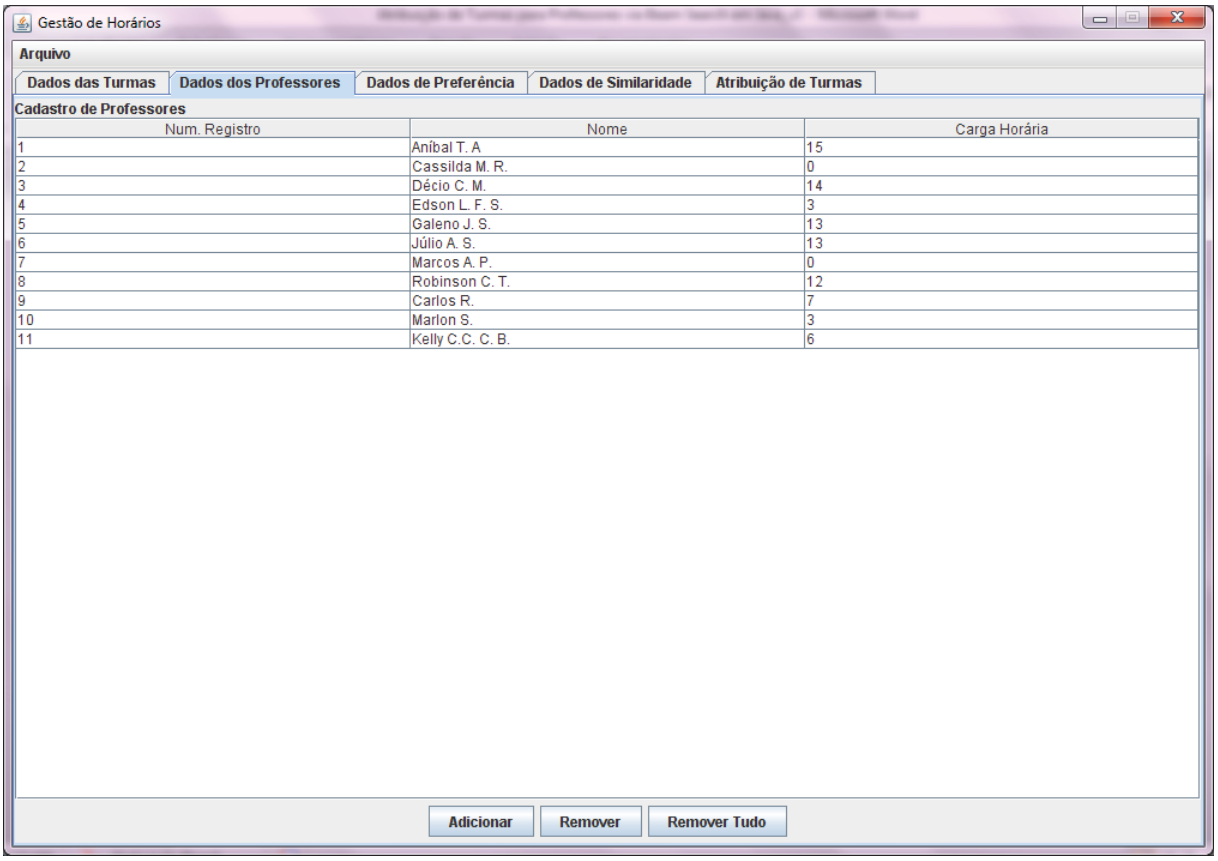


Figura 6.8: Teste com dados reais – professores.

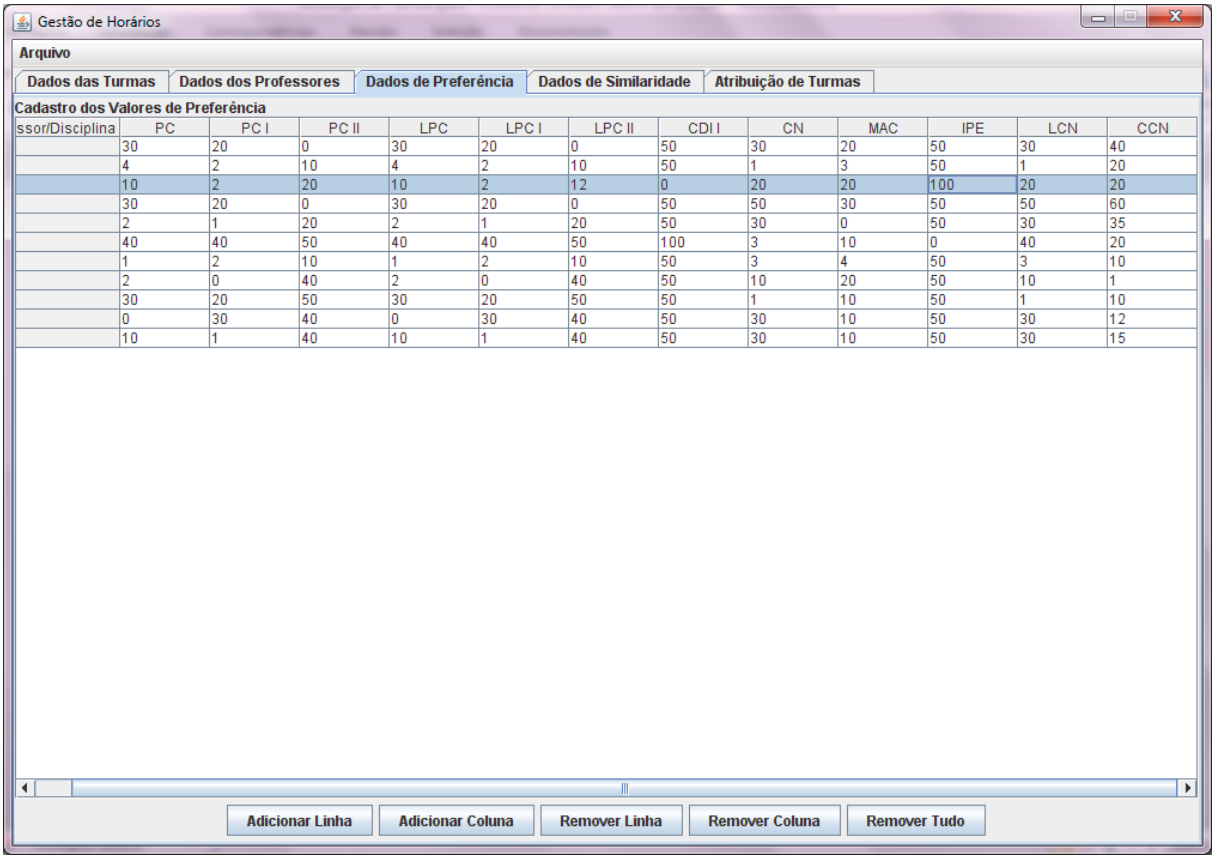
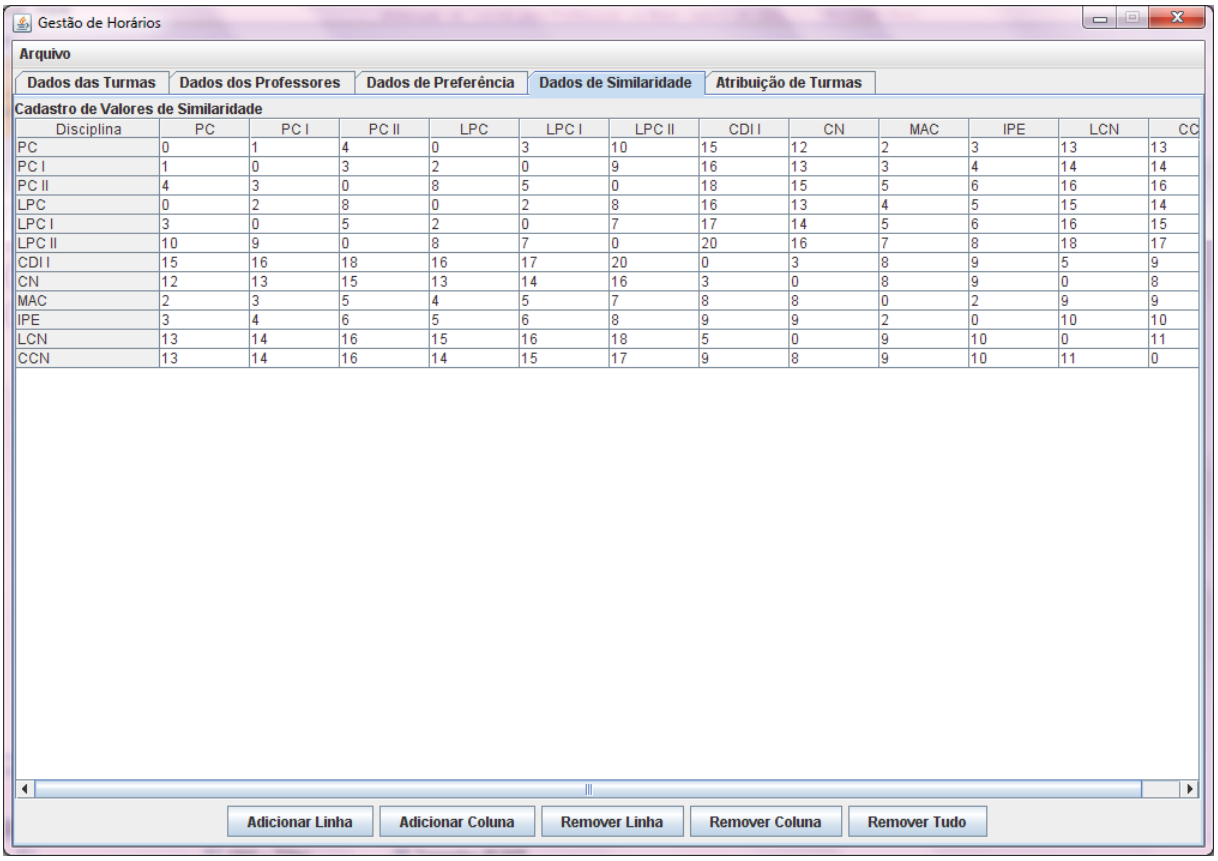
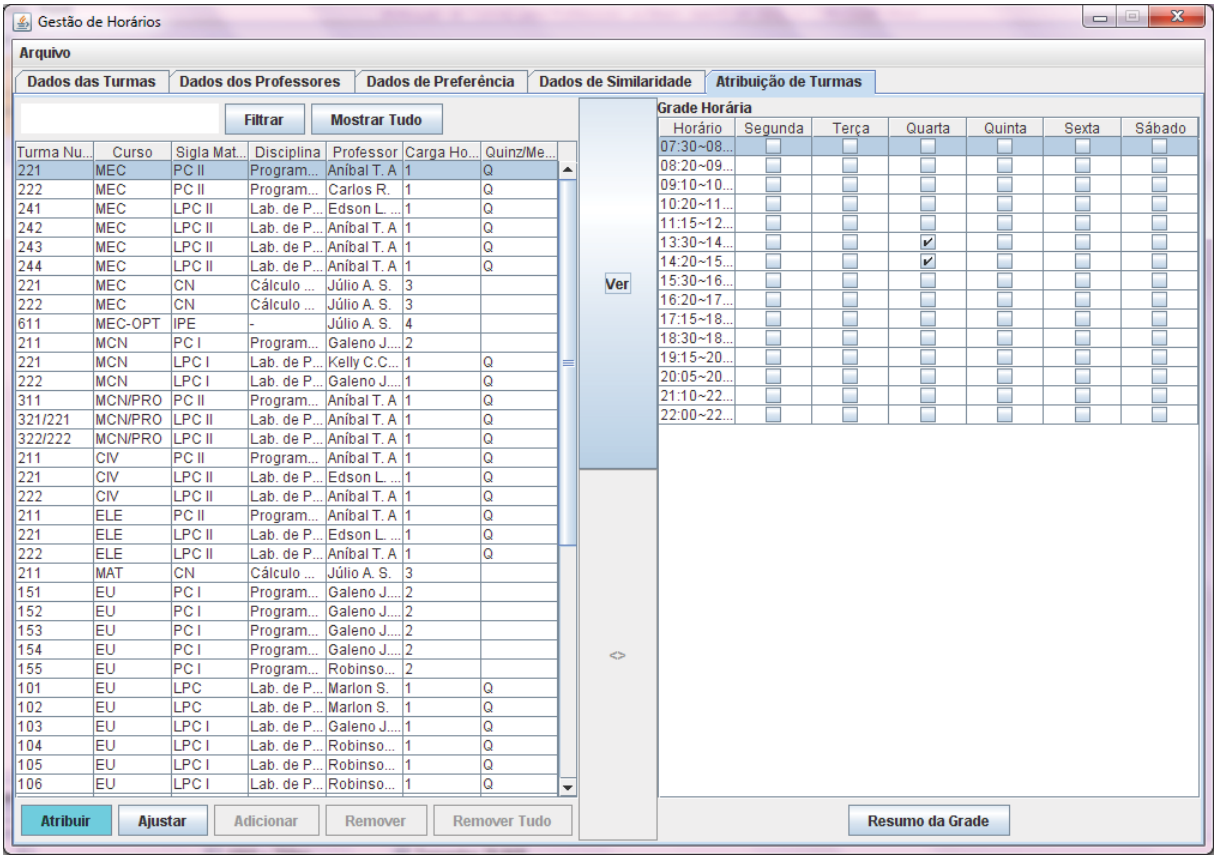


Figura 6.9: Teste com dados reais – valores de preferência.



Disciplina	PC	PC I	PC II	LPC	LPC I	LPC II	CDI I	CN	MAC	IPE	LCN	CC
PC	0	1	4	0	3	10	15	12	2	3	13	13
PC I	1	0	3	2	0	9	16	13	3	4	14	14
PC II	4	3	0	8	5	0	18	15	5	6	16	16
LPC	0	2	8	0	2	8	16	13	4	5	15	14
LPC I	3	0	5	2	0	7	17	14	5	6	16	15
LPC II	10	9	0	8	7	0	20	16	7	8	18	17
CDI I	15	16	18	16	17	20	0	3	8	9	5	9
CN	12	13	15	13	14	16	3	0	8	9	0	8
MAC	2	3	5	4	5	7	8	8	0	2	9	9
IPE	3	4	6	5	6	8	9	9	2	0	10	10
LCN	13	14	16	15	16	18	5	0	9	10	0	11
CCN	13	14	16	14	15	17	9	8	9	10	11	0

Figura 6.10: Teste com dados reais – valores de similaridade.



Turma Nu...	Curso	Sigla Mat...	Disciplina	Professor	Carga Ho...	Quinz/Me...
221	MEC	PC II	Program...	Aníbal T. A	1	Q
222	MEC	PC II	Program...	Carlos R.	1	Q
241	MEC	LPC II	Lab. de P...	Edson L. ...	1	Q
242	MEC	LPC II	Lab. de P...	Aníbal T. A	1	Q
243	MEC	LPC II	Lab. de P...	Aníbal T. A	1	Q
244	MEC	LPC II	Lab. de P...	Aníbal T. A	1	Q
221	MEC	CN	Cálculo ...	Júlio A. S.	3	
222	MEC	CN	Cálculo ...	Júlio A. S.	3	
611	MEC-OPT	IPE	-	Júlio A. S.	4	
211	MCN	PC I	Program...	Galeno J....	2	
221	MCN	LPC I	Lab. de P...	Kelly C.C...	1	Q
222	MCN	LPC I	Lab. de P...	Galeno J....	1	Q
311	MCN/PRO	PC II	Program...	Aníbal T. A	1	Q
321/221	MCN/PRO	LPC II	Lab. de P...	Aníbal T. A	1	Q
322/222	MCN/PRO	LPC II	Lab. de P...	Aníbal T. A	1	Q
211	CIV	PC II	Program...	Aníbal T. A	1	Q
221	CIV	LPC II	Lab. de P...	Edson L. ...	1	Q
222	CIV	LPC II	Lab. de P...	Aníbal T. A	1	Q
211	ELE	PC II	Program...	Aníbal T. A	1	Q
221	ELE	LPC II	Lab. de P...	Edson L. ...	1	Q
222	ELE	LPC II	Lab. de P...	Aníbal T. A	1	Q
211	MAT	CN	Cálculo ...	Júlio A. S.	3	
151	EU	PC I	Program...	Galeno J....	2	
152	EU	PC I	Program...	Galeno J....	2	
153	EU	PC I	Program...	Galeno J....	2	
154	EU	PC I	Program...	Galeno J....	2	
155	EU	PC I	Program...	Robinso...	2	
101	EU	LPC	Lab. de P...	Marlon S.	1	Q
102	EU	LPC	Lab. de P...	Marlon S.	1	Q
103	EU	LPC I	Lab. de P...	Galeno J....	1	Q
104	EU	LPC I	Lab. de P...	Robinso...	1	Q
105	EU	LPC I	Lab. de P...	Robinso...	1	Q
106	EU	LPC I	Lab. de P...	Robinso...	1	Q

Horário	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
07:30~08...	☐	☐	☐	☐	☐	☐
08:20~09...	☐	☐	☐	☐	☐	☐
09:10~10...	☐	☐	☐	☐	☐	☐
10:20~11...	☐	☐	☐	☐	☐	☐
11:15~12...	☐	☐	☐	☐	☐	☐
13:30~14...	☐	☐	☑	☐	☐	☐
14:20~15...	☐	☐	☑	☐	☐	☐
15:30~16...	☐	☐	☐	☐	☐	☐
16:20~17...	☐	☐	☐	☐	☐	☐
17:15~18...	☐	☐	☐	☐	☐	☐
18:30~18...	☐	☐	☐	☐	☐	☐
19:15~20...	☐	☐	☐	☐	☐	☐
20:05~20...	☐	☐	☐	☐	☐	☐
21:10~22...	☐	☐	☐	☐	☐	☐
22:00~22...	☐	☐	☐	☐	☐	☐

Figura 6.11: Teste com dados reais – resultado das atribuições.

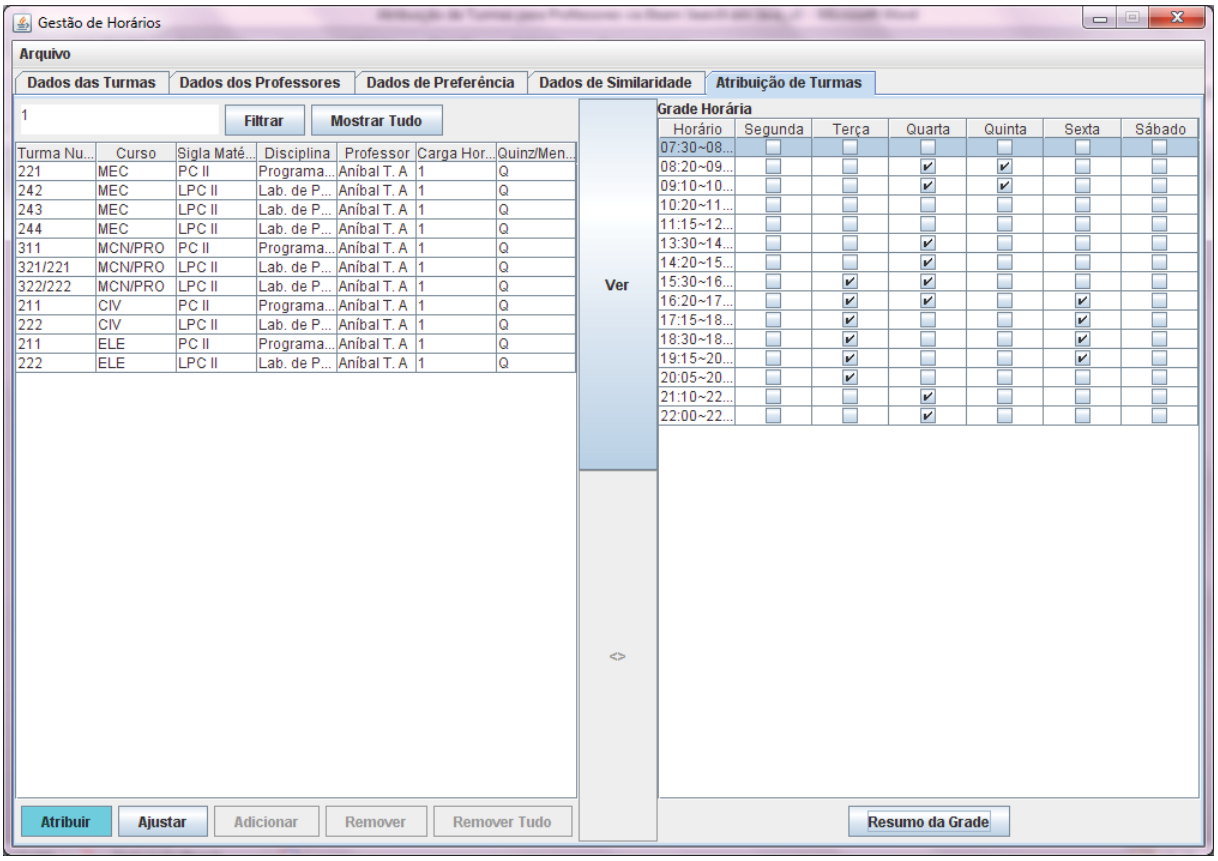


Figura 6.12: Teste com dados reais – filtro.

As figuras comprovam o funcionamento da interface. Em relação ao funcionamento da lógica, chegou-se a um resultado não muito distante do obtido manualmente. As diferenças ocorreram principalmente devido a três fatores:

a) O programa segue rigorosamente a carga horária máxima estipulada para os professores, então professores com carga horária zero, não tiveram nenhuma turma atribuída. Portanto, as turmas manualmente alocadas a professores com carga horária zero tiveram de ser divididas entre os demais professores.

b) O programa não tem 100% de liberdade em alocar turmas quinzenais e mensais num mesmo *slot* da grade horária para um mesmo professor. O que o programa enxerga são algumas turmas do mesmo curso e mesma disciplina (quinzenais ou mensais) alocados na mesma grade, e sendo assim, eles podem eventualmente alocar no mesmo *slot*. No entanto, esta técnica depende das turmas previamente formadas: se quinzenal, no máximo duas turmas poderão estar alocadas no mesmo horário, se não isso poderá causar choque na grade de algum professor; se mensal, no máximo quatro turmas.

c) O programa não agrupa teoria e laboratório da mesma disciplina para serem ministrados pelo mesmo professor.

Na sequência estão os dados obtidos na simulação e os dados obtidos manualmente para o primeiro semestre de 2011, devidamente filtrados para que seja possível uma comparação entre as disciplinas atribuídas a cada professor.

Manual		Simulado em Computador	
Disciplina	**CHD**	**Disciplina**	**CHD**
Aníbal Tavares de Azevedo	**15/9h**		**15/15h**
PC II - MEC(222)	2*	PC II - MEC(222)	2*
LPC II - MEC(243)	2*	LPC II - MEC(241)	2*
LPC II - MEC(244)	2*	LPC II - MEC(242)	2*
LPC II - MCN(321)/PRO(221)	2*	LPC II - MEC(244)	2*
LPC II - MCN(322)/PRO(222)	2*	PC II - MCN(311)/PRO(222)	2*
LPC II - CIV(222)	2*	LPC II - MCN(321)/PRO(221)	2*
PC II - MCN(311)/PRO(222)	2*	LPC II - MCN(322)/PRO(222)	2*
PC II -CIV (211)	2*	PC II -CIV (211)	2*
LPC II - CIV (221)	2*	LPC II - CIV(222)	2*
		PC II - ELE(211)	2*
		LPC II - ELE(222)	2*
		PC I - EU(151)	2
		PC I - EU(152)	2
Cassilda Maria Ribeiro	**0/8h**		**0/0h**
PC I - EU(155)	2		
PC I - ESP(131)	2		
LPC I - EU(109)	2*		
LPC I - EU(110)	2*		
LPC I - EU(161)	2*		
LPC I - EU(162)	2*		
Décio Cardozo Mourão	**14/10h**		**14/13h**
CDI	2	CN - MEC (221)	3
CDI	2	PC I - MCN (211)	2
PC I - MCN (211)	2	PC - LMN(111)	2
CDI	2	CDI	2
LPC I - MCN(221)	2*	CDI	2
LPC I -MCN(222)	2*	CDI	2
Edson Luiz França Senne	**3/3h**		**3/3h**
PC II - ELE(211)	2*	LPC II - MEC(243)	2*
LPC II - ELE(221)	2*	LPC II - CIV (221)	2*
LPC II - ELE(222)	2*	LPC II - ELE(221)	2*

Manual		Simulado em Computador	
Galeno José de Sena	**13/9h**		**13/13h**
MAC - ESP(111)	2	LPC I - MCN(221)	2*
PC I - ESP(132)	2	LPC I -MCN(222)	2*
PC II - MEC(221)	2*	PC I - EU(153)	2
LPC II - MEC(241)	2*	PC I - EU(154)	2
LPC II - MEC(242)	2*	LPC I - EU(103)	2*
LPC I - ESP(163)	2*	PC I - ESP(131)	2
LPC I - ESP(164)	2*	PC I - ESP(133)	2
		LPC I - ESP(163)	2*
		LPC I - ESP(164)	2*
Júlio Antunes Santana	**13/13h**		**13/12h**
CN - MEC(222)	3	PC II - MEC(221)	2*
CN - MAT(211)	3	CN - MEC(222)	3
IPE - MEC-OPT(611)	4	IPE - MEC-OPT(611)	4
CN - MEC (221)	3	CN - LMN(211)	2
		LCN - LMN(211)	2
Marcos Antônio Pereira	**0/8h**		**0/0h**
PC I - EU(151)	2		
PC I - EU(152)	2		
LPC - EU(101)	2*		
LPC I - EU(104)	2*		
LPC - EU(102)	2*		
LPC I - EU(103)	2*		
Substituto: Robinson Carlos Teixeira	**12/12h**		**12/11h**
PC I - EU(154)	2	LPC I - EU(104)	2*
LPC I - EU(108)	2*	LPC I - EU(105)	2*
CCN - ESP(211)	2	LPC I - EU(106)	2*
LPC I - EU(107)	2*	LPC I - EU(107)	2*
CCN - ESP(211)	2	LPC I - EU(108)	2*
PC I - ESP(133)	2	LPC I - EU(109)	2*
LPC I - ESP(165)	2*	LPC I - EU(110)	2*
LPC I - ESP(166)	2*	LPC I - EU(161)	2*
		LPC I - EU(162)	2*
		LPC I - ESP(165)	2*
		LPC I - ESP(166)	2*
Substituto: Carlos Renato	**7,5/4h**		**7,5/7h**
CN - LMN(211)	2	CN - MAT(211)	3
LCN - LMN(211)	2	CCN - ESP(211)	2
		CCN - ESP(211)	2
Bolsista 1: Marlon da Silva	**3/3h**		**3/3h**
PC - LMN(111)	2	LPC - EU(101)	2*
LPC - LMN(111)	2*	LPC - EU(102)	2*
		LPC - LMN(111)	2*
Bolsista 2: Kelly Cristina Coelho de Carvalho Benini	**6/4h**		**6/6h**
PC I - EU(153)	2	PC I - EU(155)	2
LPC I - EU(105)	2*	MAC - ESP(111)	2
LPC I - EU(106)	2*	PC I - ESP(132)	2

Figura 6.13: Comparação da distribuição das disciplinas pelo método computacional e manual.

7 CONCLUSÕES E TRABALHOS FUTUROS

O problema de Atribuição de Turmas a Professores é um problema de natureza combinatória que faz parte de um problema mais complexo de *Course Timetabling* onde é feita a gestão eficiente dos recursos educacionais. O PATP tem por objetivo realizar a distribuição dos professores entre as turmas de uma universidade de modo a respeitar a carga horária semanal da turma, as restrições de carga horária dos professores e as preferências dos mesmos pelas matérias a serem ministradas, atribuir disciplinas similares a um mesmo professor, não atribuir uma turma para mais de um professor e não permitir que um professor seja alocado em turmas diferentes num mesmo horário.

Para resolver o problema foi desenvolvida uma ferramenta computacional que é uma heurística para a resolução automática do problema combinatório. A ferramenta contribui principalmente pela rapidez e eficiência na tomada de decisão para a atribuição dos professores, além de evitar que alguns professores fiquem sobrecarregados. O algoritmo desenvolvido é uma heurística baseada na técnica de enumeração completa por meio da busca em árvore e do algoritmo guloso.

Desenvolveu-se, também, uma interface gráfica para facilitar a captura de dados para a resolução do problema. É possível, através da interface, criar, abrir, salvar e editar arquivos contendo dados dos professores, dados das turmas, valores de preferência por disciplinas e valores de similaridade entre disciplinas. Outra facilidade que a interface gráfica proporciona é a possibilidade de simular diversas atribuições com diversas larguras de busca em pouco tempo. Essa facilidade traz uma vantagem muito grande que é a visualização e comparação de diferentes configurações da grade horária, inclusive realizar mudanças rápidas nos valores de preferência e similaridade para verificar o efeito nas atribuições. Ao final das atribuições, o programa permite também realizar ajustes manuais no resultado, caso algum professor queira realizar uma mudança simples que não resultará em choques de horário ou extrapolação da carga horária.

Esta interface gráfica poderá ser expandida no futuro para integrar a captura de dados e resolução dos demais problemas de gestão de recursos educacionais relacionados ao PATP. Além disso, podem-se adicionar outros recursos como a impressão de arquivos em formatos ideais para visualização e distribuição entre docentes, e ainda, aperfeiçoar a lógica de atribuição para obter resultados mais próximos da distribuição manual.

REFERÊNCIAS BIBLIOGRÁFICAS

AZEVEDO, A.T.; RIBEIRO, C.M.; LIMA, F.M.B. **Resolução do problema de carregamento e descarregamento de contêineres em terminais portuários via Beam Search**. In: Simpósio de Engenharia de Produção, 16., 2009, Bauru, SP, Novembro de 2009.

BERGHMAN, L.; LEUS, R. **Practical solutions for a dock assignment problem with trailer transportation**. Katholieke Universiteit Leuven. Faculty of Business and Economics, 2011. Disponível em: < http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1974492>. Acesso em: 20 abr. 2012.

DEITEL, H. M.; DEITEL, P. J. **Java: como programar**. 6. ed. Bookman, 2006.

FURCY, D.; KOENING, S. **Limited Discrepancy Beam Search**. University of Wisconsin Oshkosh e University of Southern California, 2005. Disponível em: < http://www.ijcai.org/papers/ 0596.pdf>. Acesso em: 20 abr. 2012.

GERSTING, J. L. **Fundamentos matemáticos para a ciência da computação: um tratamento moderno de matemática discreta**. 5. ed. Rio de Janeiro: LTC, 2008.

GOLDBARG, M.C.; LUNA, P. L. **Otimização combinatória e programação linear: modelos e algoritmos**. 2. ed. Rio de Janeiro: Elsevier, 2005.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. New Jersey: Prentice Hall, 1998.

PARBERRY, I. **Problems on Algorithms**. University of North Texas, 1994. Disponível em: < http://larc.unt.edu/ian/books/free/poa.pdf>. Acesso em: 20 mai. 2012.

PROJETOS EM OTIMIZAÇÃO. **Projeto Gestão Eficiente de Recursos Educacionais.** Disponível em: <https://sites.google.com/site/projetosemotimizacao/home/projeto-gestao-eficiente-de-recursos-educacionais>. Acesso em: 1 ago. 2011.

RIBEIRO, C.M.; AZEVEDO, A. T.; TEIXEIRA, R.F. **Problem of assignment cells to switches in a cellular mobile network via Beam Search method**. In: World Scientific and Engineering Academy and Society, 2010, São Paulo. Transaction on Communications Systems, v. 9, n. 1, USA: World Scientific and Engineering Academy and Society, 2010. p.11-21. ISSN 1109-2742.

ROCHA, A.; DORINI, L. B. **Algoritmos gulosos: definições e aplicações**. Universidade Estadual de Campinas, 2004. Disponível em: < http://www.ic.unicamp.br/~rocha/msc/complex/algoritmosGulososFinal.pdf>. Acesso em: 19 mai. 2012.

SABUNCUOGLU, I.; BAYIZ, M. **Jop shop scheduling with beam search**. Department of Industrial Engineering, Bilkent University, Turquia, 1998. Disponível em: < http://www.graco.unb.br/alvares/Doutorado_Disciplinas/feature/jobshop.pdf>. Acesso em: 2 abr. 2012.

SCHAERF, A. A **Survey of Automated Timetabling**. Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1999. Disponível em:<http://www.diegm.uniud.it/satt/papers/Scha99.pdf>. Acesso em: 16 jun. 2011

SENNE, E.L.F. **Linguagem de Programação Java**. Faculdade de Engenharia de Guaratinguetá – UNESP. Disponível em: <http://www.feg.unesp.br/~elfsenne/cursos/FEG/LPJ/>. Acesso em: 1 abr. 2012.

SUN MICROSYSTEMS. **Java™ Platform, Standard Edition 6 API Specification**. Disponível em: <http://java.sun.com/javase/6/docs/api/>. Acesso em: 20 jan. 20012.

VALENTE, J. M. S.; ALVES, R. A. F. S. **Filtered and recovering beam search algorithms for early/tardy scheduling problem with no idle time**. Faculdade de Economia. Universidade do Porto, 2004. Disponível em: < http://wps.fep.up.pt/wps/wp142.pdf>. Acesso em: 18 mai. 2012.

WILLEMEN, R. J. **School timetable construction: algorithms and complexity**. Technische Universiteit Eindhoven, 2002. Disponível em: < http://alexandria.tue.nl/extra2/200211248.pdf >. Acesso em: 10 jul.2011.

APÊNDICE – Código-fonte desenvolvido em Java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import java.io.*;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Scanner;
import java.util.Formatter;

public class GestaoDeHorario extends JFrame{

    //Interface-------------------------------------------------------------------//

    private final int WIDTH=1000, HEIGHT=700; //ajuste de tamanho da janela
    private final int NUM=1000; //p/ inicializar vetores de turmas e professores
    private JMenu arqMenu;
    private JMenuItem itemNovo, itemAbrir, itemSalvar;
    private JTabbedPane tabPane;
    private JPanel p1, p11, p111, p12, p13, p2, p21, p211, p3, p31, p311, p4, p41,
    p411, p5, p51, p511, p512, p52, p53, p531;
    private JLabel l1, l2, l3, l4, l5, l6, l8;
    private JButton b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13, b14,
    b15, b16, b17, b18, b19, b20, b21, b22, b23, b24, b25, b26, b27, b28, b29,
    b30;
    private JTable t1, t2, t3, t4, t5, t6, t7, t8;
    private DefaultTableModel dtm1, dtm3, dtm7;
    private JScrollPane jsp1, jsp2, jsp3, jsp4, jsp5, jsp6, jsp7, jsp8;
    private JTextArea jta;
    MyTableModel2 dtm5, dtm6;
    Turma vetorTurma[], turma[];
    Professor vetorProfessor[], prof[];
    HourTable vetorHourT[], vetorHourP[], vetorHour[], filtro;

    // Itens para tratar a abertura e leitura dos dados de um arquivo texto.
    File fileDir;       // Diretorio ou arquivo especificado pelo usuário.
    int response;       // resposta do usuario na GUI.
    String output = ""; // lista de arquivos com tamanhos.
    JFileChooser chooser = new JFileChooser(".");
    // Itens para abrir e ler conteudo de arquivos ou abrir e escrever dados.
    Scanner fileIn;
    Formatter fileOut;

    public GestaoDeHorario() {
         setTitle("Gestão de Horários");
         setSize(WIDTH,HEIGHT);
         setDefaultCloseOperation(EXIT_ON_CLOSE);
         setLocation(150,30);
         setResizable(false);
         createContents(); //método que adiciona os componentes na frame
         setVisible(true);
    }

    public void createContents(){
         //Criando barra e menus
```

```
        JMenuBar barra=new JMenuBar();
        arqMenu=createJMenu();
        barra.add(arqMenu);
        setJMenuBar(barra);

        //Criando Panel com informações carregadas do cadastro de turmas
        p1=createJPanel1();
        //Criando Panel com informações carregadas com cadastro de professores
        p2=createJPanel2();
        //Criando Panel com informações carregadas com valores de preferência
        p3=createJPanel3();
        //Criando Panel com informações carregadas com valores de similaridade
        p4=createJPanel4();
        //Criando Panel com informações carregadas com atribuição turma/professor
        p5=createJPanel5();

        //Criando Tabs
        tabPane=new JTabbedPane();
        tabPane.addTab("Dados das Turmas",null,p1,"Inserir os dados de cada
        turma");
        tabPane.addTab("Dados dos Professores",null,p2,"Inserir dados dos
        professores");
        tabPane.addTab("Dados de Preferência", null, p3, "Inserir valores de
        preferência dos professores pelas disciplinas");
        tabPane.addTab("Dados de Similaridade", null, p4, "Inserir valores de
        similaridade entre as disciplinas");
        tabPane.addTab("Atribuição de Turmas",null,p5,"Obter resultados da
        atribuições");
        add(tabPane);
    }

    private JMenu createJMenu(){
        arqMenu=new JMenu("Arquivo");

        itemNovo=new JMenuItem("Novo");
        itemAbrir=new JMenuItem("Abrir");
        itemSalvar=new JMenuItem("Salvar");

        itemNovo.addActionListener(new TrataArqMenu());
        itemAbrir.addActionListener(new TrataArqMenu());
        itemSalvar.addActionListener(new TrataArqMenu());

        arqMenu.add(itemNovo);
        arqMenu.add(itemAbrir);
        arqMenu.add(itemSalvar);
        return arqMenu;
    }

    private JPanel createJPanel1(){
        p1=new JPanel();

        //-----------------------------------------------------------------//
        //Panel de cadastro de turmas
        p11=new JPanel();
        l1=new JLabel("Cadastro de Turmas");

        //Construindo JTable para cadastro de turmas
        String[] colunas=new String[]{"Turma Num.", "Curso", "Sigla Matéria",
        "Disciplina", "Carga Horária", "Quinz/Mensal"};
```

```
        Object[][] dados=new Object[][]{};
        //passando os dados para uma tabela modelo antes
        dtm1=new DefaultTableModel(dados, colunas);
        t1= new JTable(dtm1);
        jsp1=new JScrollPane(t1);
        t1.setFillsViewportHeight(true);

        //Criando JButton
        b1=new JButton("Adicionar");
        b1.addActionListener(new TrataBotao());
        b2=new JButton("Remover");
        b2.addActionListener(new TrataBotao());
        b3=new JButton("Remover Tudo");
        b3.addActionListener(new TrataBotao());
        p111=new JPanel();
        p111.setLayout(new FlowLayout());
        p111.add(b1);
        p111.add(b2);
        p111.add(b3);


        //Montando Panel
        p11.setLayout(new BorderLayout());
        p11.add(l1, BorderLayout.NORTH);
        p11.add(jsp1, BorderLayout.CENTER);
        p11.add(p111, BorderLayout.SOUTH);

        //-------------------------------------------------------------------//
        //Panel com botões de associação turma/horário
        p12=new JPanel();

        //Criando JButton
        b4=new JButton("Ver");
        b4.addActionListener(new TrataBotao());
        b5=new JButton("<>");
        b5.addActionListener(new TrataBotao());

        //Montando Panel
        p12.setLayout(new GridLayout(2,1));
        p12.add(b4);
        p12.add(b5);

        //-------------------------------------------------------------------//
        //Panel com horário das turmas
        p13=new JPanel();
        l2=new JLabel("Horário da Turma");

        //Criando JTable com Matriz de Horários
        t2=new JTable(new MyTableModel());
        jsp2=new JScrollPane(t2);
        t2.setFillsViewportHeight(true);

        //Montando Panel
        p13.setLayout(new BorderLayout());
        p13.add(l2, BorderLayout.NORTH);
        p13.add(jsp2, BorderLayout.CENTER);

        //-------------------------------------------------------------------//
        //Montando panel de cadastro de turmas
```

```
        p1.setLayout(new BorderLayout());
        p1.add(p11, BorderLayout.WEST);
        p1.add(p12, BorderLayout.CENTER);
        p1.add(p13, BorderLayout.EAST);

        return p1;
    }

    private JPanel createJPanel2(){
        p2=new JPanel();

        //-----------------------------------------------------------------------//
        //Panel de cadastro de professores
        p21=new JPanel();
        l3=new JLabel("Cadastro de Professores");

        //Construindo JTable para cadastro de professores
        String[] colunas3=new String[]{"Num. Registro", "Nome", "Carga Horária"};
        Object[][] dados3=new Object[][]{};
        dtm3=new DefaultTableModel(dados3,colunas3);
        t3= new JTable(dtm3);
        jsp3=new JScrollPane(t3);
        t3.setFillsViewportHeight(true);

        //Criando JButton
        b6=new JButton("Adicionar");
        b6.addActionListener(new TrataBotao());
        b7=new JButton("Remover");
        b7.addActionListener(new TrataBotao());
        b8=new JButton("Remover Tudo");
        b8.addActionListener(new TrataBotao());
        p211=new JPanel();
        p211.setLayout(new FlowLayout());
        p211.add(b6);
        p211.add(b7);
        p211.add(b8);

        //Montando Panel
        p21.setLayout(new BorderLayout());
        p21.add(l3, BorderLayout.NORTH);
        p21.add(jsp3, BorderLayout.CENTER);
        p21.add(p211, BorderLayout.SOUTH);

        //-----------------------------------------------------------------------//
        //Montando panel de cadastro de professores
        p2.setLayout(new BorderLayout());
        p2.add(p21, BorderLayout.CENTER);

        return p2;
    }

    private JPanel createJPanel3(){
        p3=new JPanel();

        //-----------------------------------------------------------------------//
        //Panel de valor de preferência
        p31=new JPanel();
        l5=new JLabel("Cadastro dos Valores de Preferência");
```

```
        //Construindo JTable de valores de preferência
        String[] colunas5=new String[]{"Professor/Disciplina"};
        Object[][] dados5=new Object[][]{};
        dtm5=new MyTableModel2(dados5,colunas5);
        t5=new JTable(dtm5);
        t5.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        jsp5=new JScrollPane(t5, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        t5.getColumnModel().getColumn(0).setPreferredWidth(120);
        t5.setFillsViewportHeight(true);

        //Criando JButton
        b12=new JButton("Adicionar Linha");
        b12.addActionListener(new TrataBotao());
        b13=new JButton("Adicionar Coluna");
        b13.addActionListener(new TrataBotao());
        b14=new JButton("Remover Linha");
        b14.addActionListener(new TrataBotao());
        b15=new JButton("Remover Coluna");
        b15.addActionListener(new TrataBotao());
        b16=new JButton("Remover Tudo");
        b16.addActionListener(new TrataBotao());
        p311=new JPanel();
        p311.setLayout(new FlowLayout());
        p311.add(b12);
        p311.add(b13);
        p311.add(b14);
        p311.add(b15);
        p311.add(b16);

        //Montando Panel
        p31.setLayout(new BorderLayout());
        p31.add(l5, BorderLayout.NORTH);
        p31.add(jsp5, BorderLayout.CENTER);
        p31.add(p311, BorderLayout.SOUTH);

        //-----------------------------------------------------------------//
        //Montando panel de valores de preferência
        p3.setLayout(new BorderLayout());
        p3.add(p31, BorderLayout.CENTER);

        return p3;
    }

    private JPanel createJPanel4(){
        p4=new JPanel();

        //-----------------------------------------------------------------//
        //Panel de valor de similaridade
        p41=new JPanel();
        l6=new JLabel("Cadastro de Valores de Similaridade");

        //Construindo JTable de valores de similaridade
        String[] colunas6=new String[]{"Disciplina"};
        Object[][] dados6=new Object[][]{};
        dtm6=new MyTableModel2(dados6,colunas6);
        t6=new JTable(dtm6);
        t6.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
```

```
        jsp6=new JScrollPane(t6, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        t6.getColumnModel().getColumn(0).setPreferredWidth(120);
        t6.setFillsViewportHeight(true);

        //Criando JButton
        b17=new JButton("Adicionar Linha");
        b17.addActionListener(new TrataBotao());
        b18=new JButton("Adicionar Coluna");
        b18.addActionListener(new TrataBotao());
        b19=new JButton("Remover Linha");
        b19.addActionListener(new TrataBotao());
        b20=new JButton("Remover Coluna");
        b20.addActionListener(new TrataBotao());
        b21=new JButton("Remover Tudo");
        b21.addActionListener(new TrataBotao());
        p411=new JPanel();
        p411.setLayout(new FlowLayout());
        p411.add(b17);
        p411.add(b18);
        p411.add(b19);
        p411.add(b20);
        p411.add(b21);

        //Montando Panel
        p41.setLayout(new BorderLayout());
        p41.add(l6, BorderLayout.NORTH);
        p41.add(jsp6, BorderLayout.CENTER);
        p41.add(p411, BorderLayout.SOUTH);

        //-----------------------------------------------------------------------//
        //Montando panel de valores de preferência
        p4.setLayout(new BorderLayout());
        p4.add(p41, BorderLayout.CENTER);

        return p4;
    }

    private JPanel createJPanel5(){
        p5=new JPanel();

        //-----------------------------------------------------------------------//
        //Panel de atribuição de turmas
        p51=new JPanel();
        jta=new JTextArea("Digite nº registro do professor");
        jta.setPreferredSize(new Dimension(165,25));
        b28=new JButton("Filtrar");
        b28.addActionListener(new TrataFiltro());
        b29=new JButton("Mostrar Tudo");
        b29.addActionListener(new TrataFiltro());
        p511=new JPanel();
        p511.setLayout(new FlowLayout(FlowLayout.LEFT));
        p511.add(jta);
        p511.add(b28);
        p511.add(b29);

        //Construindo JTable para atribuição de turmas
        String[] colunas7=new String[]{"Turma Num.", "Curso", "Sigla Matéria",
        "Disciplina", "Professor", "Carga Horária", "Quinz/Mensal"};
```

```
        Object[][] dados7=new Object[][]{};
        dtm7=new DefaultTableModel(dados7,colunas7);
        t7=new JTable(dtm7);
        jsp7=new JScrollPane(t7);
        t7.setFillsViewportHeight(true);

        //Criando JButton
        b9=new JButton("Atribuir");
        b9.setBackground(Color.CYAN);
        b9.addActionListener(new TrataAtribuir());
        b22=new JButton("Ajustar");
        b22.addActionListener(new TrataBotao());
        b23=new JButton("Adicionar");
        b23.setEnabled(false);
        b23.addActionListener(new TrataBotao());
        b24=new JButton("Remover");
        b24.setEnabled(false);
        b24.addActionListener(new TrataBotao());
        b25=new JButton("Remover Tudo");
        b25.setEnabled(false);
        b25.addActionListener(new TrataBotao());
        p512=new JPanel();
        p512.setLayout(new FlowLayout());
        p512.add(b9);
        p512.add(b22);
        p512.add(b23);
        p512.add(b24);
        p512.add(b25);

        //Montando Panel
        p51.setLayout(new BorderLayout());
        p51.add(p511, BorderLayout.NORTH);
        p51.add(jsp7, BorderLayout.CENTER);
        p51.add(p512, BorderLayout.SOUTH);

        //-------------------------------------------------------------------//
        //Panel com botões de associação professor/horário
        p52=new JPanel();

        //Criando JButton
        b26=new JButton("Ver");
        b26.addActionListener(new TrataBotao());
        b27=new JButton("<>");
        b27.setEnabled(false);
        b27.addActionListener(new TrataBotao());

        //Montando Panel
        p52.setLayout(new GridLayout(2,1));
        p52.add(b26);
        p52.add(b27);

        //-------------------------------------------------------------------//
        //Panel com horário dos professores
        p53=new JPanel();
        l8=new JLabel("Grade Horária");

        //Criando JTable com Matriz de Horários
        t8=new JTable(new MyTableModel());
        jsp8=new JScrollPane(t8);
```

```
        t8.setFillsViewportHeight(true);
        b30=new JButton("Resumo da Grade");
        b30.addActionListener(new TrataFiltro());
        p531=new JPanel();
        p531.add(b30);

        //Montando Panel
        p53.setLayout(new BorderLayout());
        p53.add(l8, BorderLayout.NORTH);
        p53.add(jsp8, BorderLayout.CENTER);
        p53.add(p531, BorderLayout.SOUTH);

        //-------------------------------------------------------------------//
        //Montando panel de cadastro de professores
        p5.setLayout(new BorderLayout());
        p5.add(p51, BorderLayout.WEST);
        p5.add(p52, BorderLayout.CENTER);
        p5.add(p53, BorderLayout.EAST);

        return p5;
    }

    //Evento Menu Arquivo-----------------------------------------------------//

    private class TrataArqMenu implements ActionListener{

        public void actionPerformed(ActionEvent e){
         //abertura da caixa para seleção
         chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

         //Tratamento do menu novo-----------------------------------------//
         if(e.getSource()==itemNovo){
             //Descobrir qual aba está aberta
             int aux=tabPane.getSelectedIndex();
                          output="Arquivo em branco.";
             if (aux==0){
                 if(t1.getRowCount()==0){
                 Turma t=new Turma();
                  vetorHourT=new HourTable[NUM];
                 dtm1.insertRow(0, new Object[]{});
                 dtm1.fireTableStructureChanged();
                 vetorHourT[0]=new HourTable();
             }
             else
             output="Já existe um arquivo aberto. Para abrir novo arquivo é
                  necessário remover os dados.";
         }

         else if (aux==1){
             if(t3.getRowCount()==0){
                  Professor p=new Professor();
                  dtm3.insertRow(0, new Object[]{});
                  dtm3.fireTableStructureChanged();
             }
             else
                  output="Já existe um arquivo aberto. Para abrir novo arquivo é
                  necessário remover os dados.";
         }
```

```
        else if (aux==2){
            if(t5.getRowCount()==0){
                dtm5.insertRow(0, new Object[]{});
                dtm5.fireTableStructureChanged();
                dtm5.formatarColuna(t5);
            }
            else
                output="Já existe um arquivo aberto. Para abrir novo arquivo é
                necessário remover os dados.";
        }

        else if (aux==3){
            if(t6.getRowCount()==0){
                 dtm6.insertRow(0, new Object[]{});
                 dtm6.fireTableStructureChanged();
                 dtm6.formatarColuna(t6);
            }
            else
                 output="Já existe um arquivo aberto. Para abrir novo arquivo é
                 necessário remover os dados.";
        }

        else if (aux==4){
            if(t7.getRowCount()==0){
                 Turma turmaA=new Turma();
                 Professor profA=new Professor();
                 vetorHour=new HourTable[NUM];
                 dtm7.insertRow(0, new Object[]{});
                 dtm7.fireTableStructureChanged();
                 vetorHour[0]=new HourTable();
            }
            else
                 output="Já existe um arquivo aberto. Para iniciar novo arquivo
                 é necessário remover os dados.";
        }
        JOptionPane.showMessageDialog(null, output, "Arquivo de Dados",
                 JOptionPane.INFORMATION_MESSAGE);
    }

    //Tratamento do menu abrir-------------------------------------------------//

    if (e.getSource()==itemAbrir){
        response=chooser.showOpenDialog(null);
            //verificando se o usuário selecionou e pressionou Open
            if(response==JFileChooser.APPROVE_OPTION){
                fileDir=chooser.getSelectedFile();
                //Verificando se algum arquivo foi selecionado
                    if (fileDir.isFile()){
                    //Tenta abrir o arquivo
                    try{
                        //Descobrir qual aba está aberta
                        int aux=tabPane.getSelectedIndex();
                        fileIn=new Scanner(new FileReader(fileDir));

                        if (aux==0){
                            if(t1.getRowCount()==0){
                                Turma t=new Turma();
                                vetorHourT=new HourTable[NUM];
```

```
                    int k=0;
                        while (fileIn.hasNext()){
                        String str=fileIn.nextLine();
                        String[] temp=str.split(";");
                        //leitura dos dados do arquivo
                        t.setNumTurma(temp[0]);
                        t.setCurso(temp[1]);
                        t.setCodigoMateria(temp[2]);
                        t.setDisciplina(temp[3]);
                        t.setCargaHorariaT(Integer.parseInt
                        (temp[4]));
                        t.setQM(temp[5]);
                        dtm1.insertRow(t1.getRowCount(), temp);
                        vetorHourT[k]=new HourTable();
                        int z=6;
                        for(int i=0; i<15; i++)
                            for(int j=0; j<6; j++){
                                vetorHourT[k].setA(i, j,
                                Boolean.parseBoolean(temp[z]));
                                z++;
                            }
                        k++;
                    }
                    fileIn.close();
                    output="Dados recuperados com sucesso";
                }
                else
                    output="Já existe um arquivo aberto. Para abrir
                    novo arquivo é necessário remover os dados.";
            }

            else if (aux==1){
                if(t3.getRowCount()==0){
                    Professor p=new Professor();
                    vetorHourP=new HourTable[NUM];
                    int k=0;
                    while (fileIn.hasNext()){
                        String str=fileIn.nextLine();
                        String[] temp=str.split(";");
                        p.setNumReg(temp[0]);
                        p.setNome(temp[1]);
                        p.setCargaHorariaP(Integer.parseInt
                        (temp[2]));
                        dtm3.insertRow(t3.getRowCount(), temp);
                    }
                    fileIn.close();
                    output="Dados recuperados com sucesso";
                }
                else
                output="Já existe um arquivo aberto. Para abrir novo
                arquivo é necessário remover os dados.";
            }

            else if (aux==2){
                if(t5.getRowCount()==0){
                    t5.setAutoCreateColumnsFromModel(true);
                    String str=fileIn.nextLine();
                    String[] temp=str.split(";");
                    for(int k=0; k<temp.length; k++)
```

```
                    dtm5.addColumn(temp[k]);
                while(fileIn.hasNext()){
                    str=fileIn.nextLine();
                    temp=str.split(";");
                    dtm5.insertRow(t5.getRowCount(), temp);
                }
                dtm5.formatarColuna(t5);
                fileIn.close();
                output="Dados recuperados com sucesso";
            }
            else
            output="Já existe um arquivo aberto. Para abrir
            novo arquivo é necessário remover os dados.";
        }

        else if (aux==3){
            if(t6.getRowCount()==0){
                t6.setAutoCreateColumnsFromModel(true);
                String str=fileIn.nextLine();
                String[] temp=str.split(";");
                for(int k=0; k<temp.length; k++)
                    dtm6.addColumn(temp[k]);
                while(fileIn.hasNext()){
                    str=fileIn.nextLine();
                    temp=str.split(";");
                    dtm6.insertRow(t6.getRowCount(), temp);
                }
                dtm6.formatarColuna(t6);
                fileIn.close();
                output="Dados recuperados com sucesso";
            }
            else
            output="Já existe um arquivo aberto. Para abrir novo
            arquivo é necessário remover os dados.";
        }

        else if (aux==4){
            if(t7.getRowCount()==0){
                Turma turmaA=new Turma();
                Professor profA=new Professor();
                vetorHour=new HourTable[NUM];
                int k=0;
                while (fileIn.hasNext()){
                    String str=fileIn.nextLine();
                    String[] temp=str.split(";");
                    turmaA.setNumTurma(temp[0]);
                    turmaA.setCurso(temp[1]);
                    turmaA.setCodigoMateria(temp[2]);
                    turmaA.setDisciplina(temp[3]);
                    profA.setNome(temp[4]);
                    turmaA.setCargaHorariaT(Integer.parseInt
                    (temp[5]));
                    turmaA.setQM(temp[6]);
                    dtm7.insertRow(t7.getRowCount(), temp);
                    vetorHour[k]=new HourTable();
                    int z=7;
                    for(int i=0; i<15; i++)
                        for(int j=0; j<6; j++){
```

```
                                        vetorHour[k].setA(i, j,
                                        Boolean.parseBoolean(temp[z]));
                                        z++;
                                    }
                                    k++;
                                }
                                fileIn.close();
                                output="Dados recuperados com sucesso";
                            }
                            else
                            output="Já existe um arquivo aberto. Para abrir
                            novo arquivo é necessário remover os dados.";
                        }
                    }
                    catch (FileNotFoundException ex){
                        JOptionPane.showMessageDialog(null, "Arquivo não
                        encontrado.","File Sizes",
                        JOptionPane.INFORMATION_MESSAGE);
                    }
                }

                else if (fileDir.isDirectory())
                      output="Diretório selecionado. Selecione o arquivo.";

                else
                      output="Escolha inválida. Selecione um diretório ou um
                      arquivo.";

                JOptionPane.showMessageDialog(null, output,"Recuperação de
                Dados", JOptionPane.INFORMATION_MESSAGE);
            }
        }

        //Tratamento do menu salvar----------------------------------------//
        else if (e.getSource()==itemSalvar){
            response=chooser.showSaveDialog(null);
            //verificando se o usuário selecionou e precionou Save
            if(response==JFileChooser.APPROVE_OPTION){
                fileDir=chooser.getSelectedFile();
                //criando arquivo
                    try{
                       //Descobrir qual aba está aberta
                       int aux=tabPane.getSelectedIndex();
                       fileOut=new Formatter(fileDir);
                       if (aux==0){
                           //percorre tabela salvando cada elemento
                           for (int i=0; i<dtm1.getRowCount(); i++){
                               for (int j=0; j<dtm1.getColumnCount(); j++){
                                   if (j!=5)
                                       fileOut.format("%s;",
                                       dtm1.getValueAt(i,j));
                                   else if (j==5)
                                       fileOut.format("%s",
                                       dtm1.getValueAt(i,j));
                               }
                               //percorre cada elemento da grade horaria
                               for (int k=0; k<t2.getRowCount(); k++)
                                   for (int j=0; j<t2.getColumnCount()-1; j++)
```

```
                                fileOut.format(";%b",
                                vetorHourT[i].getA(k,j));
                        fileOut.format("\r\n");
                    }
                }

                else if (aux==1){
                    for (int i=0; i<dtm3.getRowCount(); i++){
                        for (int j=0; j<dtm3.getColumnCount(); j++){
                            if (j!=2)
                                fileOut.format("%s;",
                                dtm3.getValueAt(i,j));
                            else if (j==2)
                                fileOut.format("%s",
                                dtm3.getValueAt(i,j));
                        }
                        fileOut.format("\r\n");
                    }
                }

                else if (aux==2){
                    for(int i=1; i<dtm5.getColumnCount(); i++){
                        if (i!=(dtm5.getColumnCount()-1))
                            fileOut.format("%s;", dtm5.getColumnName(i));
                        else if (i==(dtm5.getColumnCount()-1))
                            fileOut.format("%s", dtm5.getColumnName(i));
                    }
                    fileOut.format("\r\n");
                    for (int i=0; i<dtm5.getRowCount(); i++){
                        for (int j=0; j<dtm5.getColumnCount(); j++){
                            if (j!=(dtm5.getColumnCount()-1))
                                fileOut.format("%s;",
                                dtm5.getValueAt(i,j));
                            else if (j==(dtm5.getColumnCount()-1))
                                fileOut.format("%s",
                                dtm5.getValueAt(i,j));
                        }
                        fileOut.format("\r\n");
                    }
                }

                else if (aux==3){
                    for(int i=1; i<dtm6.getColumnCount(); i++){
                        if (i!=(dtm6.getColumnCount()-1))
                            fileOut.format("%s;",
                            dtm6.getColumnName(i));
                        else if (i==(dtm6.getColumnCount()-1))
                            fileOut.format("%s",
                            dtm6.getColumnName(i));
                        }
                        fileOut.format("\r\n");
                        for (int i=0; i<dtm6.getRowCount(); i++){
                            for (int j=0; j<dtm6.getColumnCount(); j++){
                                if (j!=(dtm6.getColumnCount()-1))
                                    fileOut.format("%s;",
                                    dtm6.getValueAt(i,j));
                                else if (j==(dtm6.getColumnCount()-1))
                                     fileOut.format("%s",
                                     dtm6.getValueAt(i,j));
```

```
                                }
                            fileOut.format("\r\n");
                            }
                        }

                        else if (aux==4){
                            for (int i=0; i<dtm7.getRowCount(); i++){
                                for (int j=0; j<dtm7.getColumnCount(); j++){
                                     if (j!=6)
                                        fileOut.format("%s;",
                                        dtm7.getValueAt(i,j));
                                    else if (j==6)
                                        fileOut.format("%s",
                                        dtm7.getValueAt(i,j));
                                }
                                for (int k=0; k<t8.getRowCount(); k++)
                                    for (int j=0; j<t8.getColumnCount()-1; j++)
                                        fileOut.format(";%b",
                                        vetorHour[i].getA(k,j));
                                fileOut.format("\r\n");
                            }
                        }
                        fileOut.close();
                        output="Dados salvos com sucesso.";
                    }
                    catch (FileNotFoundException ex){
                        JOptionPane.showMessageDialog(null, "Arquivo não e
                          ncontrado.","File Sizes",
                        JOptionPane.INFORMATION_MESSAGE);
                    }

                    JOptionPane.showMessageDialog(null, output,"Recuperação de
                    Dados", JOptionPane.INFORMATION_MESSAGE);
                }
            }
        }
    }

    //Evento Tratamento de Botões-------------------------------------------------//

    private class TrataBotao implements ActionListener{

          public void actionPerformed(ActionEvent e){

          //Tratamento do botão adicionar linha-----------------------------------//
              if (e.getSource()==b1){
                    // Obtendo qual linha da tabela foi selecionada.
                    int rowIndex=t1.getSelectedRow();
                    // Verificando se realmente uma linha foi selecionada.
                    if (rowIndex!=-1){
                              dtm1.insertRow(rowIndex+1, new Object[]{});
                              dtm1.fireTableStructureChanged();
                              HourTable[] auxVetorHourT = new
                              HourTable[dtm1.getRowCount()];
                              for (int i=0; i < dtm1.getRowCount(); i++){
                                        if(i<=rowIndex)
                                        auxVetorHourT[i] = vetorHourT[i];
                              else if(i>rowIndex+1)
                                        auxVetorHourT[i] = vetorHourT[i-1];
```

```
                        else
                                auxVetorHourT[i] = new HourTable();
                        }
                        vetorHourT=new HourTable[dtm1.getRowCount()];
                        for (int i=0; i < dtm1.getRowCount(); i++)
                                vetorHourT[i]=auxVetorHourT[i];
                }
        }

        if (e.getSource()==b6){
                int rowIndex=t3.getSelectedRow();
                if (rowIndex!=-1){
                        dtm3.insertRow(rowIndex+1, new Object[]{});
                        dtm3.fireTableStructureChanged();
                }
        }

        if (e.getSource()==b12){
                int rowIndex=t5.getSelectedRow();
                if (rowIndex!=-1){
                        dtm5.insertRow(rowIndex+1, new Object[]{});
                        dtm5.fireTableStructureChanged();
                        dtm5.formatarColuna(t5);
                }
        }

        if (e.getSource()==b17){
                int rowIndex=t6.getSelectedRow();
                if (rowIndex!=-1){
                        dtm6.insertRow(rowIndex+1, new Object[]{});
                        dtm6.fireTableStructureChanged();
                        dtm6.formatarColuna(t6);
                }
        }

        if (e.getSource()==b23){
                int rowIndex=t7.getSelectedRow();
                if (rowIndex!=-1){
                        dtm7.insertRow(rowIndex+1, new Object[]{});
                        dtm7.fireTableStructureChanged();
                        HourTable[] auxVetorHour = new
                        HourTable[dtm7.getRowCount()];
                        for (int i=0; i < dtm7.getRowCount(); i++){
                                if(i<=rowIndex)
                                        auxVetorHour[i] = vetorHour[i];
                                else if(i>rowIndex+1)
                                        auxVetorHour[i] = vetorHour[i-1];
                                else
                                        auxVetorHour[i] = new HourTable();
                        }
                        vetorHour=new HourTable[dtm7.getRowCount()];
                        for (int i=0; i < dtm7.getRowCount(); i++)
                                vetorHour[i]=auxVetorHour[i];
                }
        }

        //Tratamento do botão remover linha--------------------------------//
        if (e.getSource()==b2){
                // Obtendo qual linha da tabela foi selecionada.
```

```
                int rowIndex=t1.getSelectedRow();
                // Verificando se realmente uma linha foi selecionada.
                if (rowIndex!=-1){
                        dtm1.removeRow(rowIndex);
                        dtm1.fireTableStructureChanged();
                        HourTable[] auxVetorHourT = new
                        HourTable[dtm1.getRowCount()];
                        for (int i=0; i < dtm1.getRowCount(); i++){
                                if(i < rowIndex)
                                        auxVetorHourT[i] = vetorHourT[i];
                                else
                                        auxVetorHourT[i] = vetorHourT[i+1];
                        }
                        vetorHourT=new HourTable[dtm1.getRowCount()];
                        for (int i=0; i < dtm1.getRowCount(); i++)
                                vetorHourT[i] = auxVetorHourT[i];
                }
        }

        if (e.getSource()==b7){
                int rowIndex=t3.getSelectedRow();
                if (rowIndex!=-1){
                        dtm3.removeRow(rowIndex);
                        dtm3.fireTableStructureChanged();
                }
        }

        if (e.getSource()==b14){
                int rowIndex=t5.getSelectedRow();
                if (rowIndex!=-1){
                        dtm5.removeRow(rowIndex);
                        dtm5.fireTableStructureChanged();
                        dtm5.formatarColuna(t5);
                }
        }

        if (e.getSource()==b19){
                int rowIndex=t6.getSelectedRow();
                if (rowIndex!=-1){
                        dtm6.removeRow(rowIndex);
                        dtm6.fireTableStructureChanged();
                        dtm6.formatarColuna(t6);
                }
        }

        if (e.getSource()==b24){
                int rowIndex=t7.getSelectedRow();
                if (rowIndex!=-1){
                        dtm7.removeRow(rowIndex);
                        dtm7.fireTableStructureChanged();
                        HourTable[] auxVetorHour = new
                        HourTable[dtm7.getRowCount()];
                        for (int i=0; i < dtm7.getRowCount(); i++){
                                if(i < rowIndex)
                                        auxVetorHour[i] = vetorHour[i];
                                else
                                        auxVetorHour[i] = vetorHour[i+1];
                        }
                        vetorHour=new HourTable[dtm7.getRowCount()];
```

```
                        for (int i=0; i < dtm7.getRowCount(); i++)
                                vetorHour[i] = auxVetorHour[i];
                }
        }

        //Tratamento do botão adicionar coluna-----------------------------//
        if (e.getSource()==b13){
                dtm5=(MyTableModel2)t5.getModel();
                TableColumn col=new
                TableColumn(dtm5.getColumnCount());
                t5.setAutoCreateColumnsFromModel(false);
                String header=JOptionPane.showInputDialog("Sigla da
                disciplina?");
                if(!(header==null)){
                        col.setHeaderValue(header);
                        t5.addColumn(col);
                        dtm5.addColumn(header);
                        dtm5.fireTableStructureChanged();
                        dtm5.formatarColuna(t5);
                }
        }

        if (e.getSource()==b18){
                dtm6=(MyTableModel2)t6.getModel();
                TableColumn col=new
                TableColumn(dtm6.getColumnCount());
                t6.setAutoCreateColumnsFromModel(false);
                String header=JOptionPane.showInputDialog("Sigla da
                disciplina?");
                if(!(header==null)){
                        col.setHeaderValue(header);
                        t6.addColumn(col);
                        dtm6.addColumn(header);
                        dtm6.fireTableStructureChanged();
                        dtm6.formatarColuna(t6);
                }
        }

        //Tratamento do botão remover coluna-------------------------------//
        if (e.getSource()==b15){
                if(t5.getSelectedColumn()!=0){
                        t5.setAutoCreateColumnsFromModel(false);
                        dtm5=(MyTableModel2)t5.getModel();
                        TableColumn col=
                        t5.getColumnModel().getColumn(t5.getSelectedColumn());
                        int colModelIndex=col.getModelIndex();
                        t5.removeColumn(col);
                        Enumeration enumeracao=
                        t5.getColumnModel().getColumns();
                        dtm5.removeColumn(colModelIndex, enumeracao);
                        dtm5.formatarColuna(t5);
                }
        }

        if (e.getSource()==b20){
                if(t6.getSelectedColumn()!=0){
                        t6.setAutoCreateColumnsFromModel(false);
                        dtm6=(MyTableModel2)t6.getModel();
```

```
                        TableColumn col=
                        t6.getColumnModel().getColumn(t6.getSelectedColumn());
                        int colModelIndex=col.getModelIndex();
                        t6.removeColumn(col);
                        Enumeration enumeracao=
                        t6.getColumnModel().getColumns();
                        dtm6.removeColumn(colModelIndex, enumeracao);
                        dtm6.formatarColuna(t6);
                }
        }

        //Tratamento do botão remover tudo---------------------------------//
        if (e.getSource()==b3){
                int rowCount=dtm1.getRowCount();
                // Percorrendo todas as linhas da tabela e eliminando todas.
                for (int i=0; i<rowCount; i++){
                        //Removendo cada linha da tabela
                        dtm1.removeRow(0);
                        dtm1.fireTableStructureChanged();
                }
        }

        if (e.getSource()==b8){
                int rowCount=dtm3.getRowCount();
                for (int i=0; i<rowCount; i++){
                        dtm3.removeRow(0);
                        dtm3.fireTableStructureChanged();
                }
        }

        if (e.getSource()==b16){
                dtm5=(MyTableModel2)t5.getModel();
                int rowCount=dtm5.getRowCount();
                for (int i=dtm5.getColumnCount(); i>1; i--){
                        t5.removeColumn(t5.getColumnModel().getColumn(i-1));
                        Enumeration enumeracao=
                        t5.getColumnModel().getColumns();
                        dtm5.removeColumn(i-1, enumeracao);
                }
                for (int i=0; i<rowCount; i++){
                        dtm5.removeRow(0);
                        dtm5.fireTableStructureChanged();
                }
        }

        if (e.getSource()==b21){
                int rowCount=dtm6.getRowCount();
                dtm6=(MyTableModel2)t6.getModel();
                for (int i=dtm6.getColumnCount(); i>1; i--){
                        t6.removeColumn(t6.getColumnModel().getColumn(i-1));
                        Enumeration enumeracao=
                        t6.getColumnModel().getColumns();
                        dtm6.removeColumn(i-1, enumeracao);
                }
                for (int i=0; i<rowCount; i++){
                        dtm6.removeRow(0);
                        dtm6.fireTableStructureChanged();
                }
        }
```

```
            if (e.getSource()==b25){
                  int rowCount=dtm7.getRowCount();
                  for (int i=0; i<rowCount; i++){
                  dtm7.removeRow(0);
                  dtm7.fireTableStructureChanged();
                  }
                  b23.setEnabled(false);
                  b24.setEnabled(false);
                  b25.setEnabled(false);
                  b27.setEnabled(false);
            }

            //Tratamento do botão de vizualização------------------------------//

            if (e.getSource()==b4){
                  // Obtendo qual linha da tabela foi selecionada.
                  int rowIndex=t1.getSelectedRow();
                  // Verificando se realmente uma linha foi selecionada.
                  if (rowIndex!=-1){
                  // Percorrer toda a tabela de horário do professor
                           for (int i=0;i<t2.getRowCount(); i++)
                                    for (int j=1; j<t2.getColumnCount(); j++)
                                    //Modificando os valores da tabela de acordo
                                    //com a tabela de horário de cada prof.
                                             t2.setValueAt(vetorHourT[rowIndex].
                                             getA(i,j-1),i,j);
                  }
            }

            if (e.getSource()==b26){
                  int rowIndex=t7.getSelectedRow();
                  if (rowIndex!=-1){
                           for (int i=0;i<t8.getRowCount(); i++)
                                    for (int j=1; j<t8.getColumnCount(); j++)
                                             t8.setValueAt(vetorHour[rowIndex].
                                             getA(i,j-1),i,j);
                  }
            }

            //Tratamento do botão de associação--------------------------------//

            if (e.getSource()==b5){
                  // Obtendo qual linha da tabela de turmas foi selecionada
                  int rowIndex = t1.getSelectedRow();
                  // Verificando se realmente uma linha foi selecionada
                  if (rowIndex != -1){
                  // Percorrer toda a tabela de horário da turma
                           for (int i=0; i < t2.getRowCount(); i++)
                                    for (int j=1; j < t2.getColumnCount(); j++)
                                    // Modificar o estado do vetor que armazena
                                    //os horarios para cada turma
                                             vetorHourT[rowIndex].setA(i,j-
                                             1,(Boolean)
                                    t2.getValueAt(i,j));
                  }
            }

            if (e.getSource()==b27){
```

```
                int rowIndex = t7.getSelectedRow();
                if (rowIndex != -1){
                        for (int i=0; i < t8.getRowCount(); i++)
                                for (int j=1; j < t8.getColumnCount(); j++)
                                        vetorHour[rowIndex].setA(i,j-
                                        1,(Boolean)
                                t8.getValueAt(i,j));
                }
        }

        //Tratamento do botão de ajuste-------------------------------------//

        if (e.getSource()==b22){
                if(dtm7.getRowCount()!=0 && b23.isEnabled()==false){
                        Object[] options = { "OK", "CANCEL" };
                        int pane=JOptionPane.showOptionDialog(null, "Isso
                        poderá modificar o resultado final. Deseja
                        continuar?", "Warning", JOptionPane.YES_NO_OPTION,
                        JOptionPane.WARNING_MESSAGE, null, options,
                        options[1]);
                        if(pane==0){
                                b23.setEnabled(true);
                                b24.setEnabled(true);
                                b25.setEnabled(true);
                                b27.setEnabled(true);
                        }
                }
        }
    }
}

//Evento Tratamento do Botão Atribuir--------------------------------------//

private class TrataAtribuir implements ActionListener{

    public void actionPerformed(ActionEvent e){

        if (e.getSource()==b9){
        //verificar se há dados nas abas (turma, professor, preferência e
        //similaridade)
                if(dtm1.getRowCount()!=0 && dtm3.getRowCount()!=0 &&
                        dtm5.getRowCount()!=0 && dtm6.getRowCount()!=0){
                        //verificar se tem dados no último tabPane
                            if (t7.getRowCount()==0){
                                vetorTurma=new Turma[dtm1.getRowCount()];
                                for (int i=0; i<dtm1.getRowCount(); i++){
                                //criando o objeto turma
                                    String nt=(String)dtm1.getValueAt(i,0);
                                    String c=(String)dtm1.getValueAt(i,1);
                                    String cm=(String)dtm1.getValueAt(i,2);
                                    String d=(String)dtm1.getValueAt(i,3);
                                    int cht=Integer.parseInt
                                    (dtm1.getValueAt(i,4).toString());
                                    String qm=(String)dtm1.getValueAt(i,5)
                                    HourTable ht=vetorHourT[i];
                                    Turma t=new Turma(nt, c, cm, d, cht, qm,
                                                                        ht
                                    vetorTurma[i]=t;
                                }
```

```
                                    vetorProfessor=new
                                    Professor[dtm3.getRowCount()];
                                    for (int i=0; i<dtm3.getRowCount(); i++){
                                        String nr=(String)dtm3.getValueAt(i,0);
                                        String n=(String)dtm3.getValueAt(i,1);
                                        int chp=Integer.parseInt
                                        (dtm3.getValueAt(i,2).toString());
                                        Professor p=new Professor(nr, n, chp);
                                        vetorProfessor[i]=p;
                                    }
                                    tabPane.setSelectedIndex(4);
                                    BeamSearch bs=new BeamSearch(vetorTurma,
                                                                 vetorProfessor);
                                    Solucao solucao=bs.resolver((MyTableModel2)
                                    t5.getModel(), (MyTableModel2)t6.getModel());
                                    turma=solucao.getTurma();
                                    prof=solucao.getProfessor();
                                    vetorHour=new HourTable[turma.length];
                                    for(int k=0; k<turma.length; k++){
                                    dtm7.insertRow(t7.getRowCount(),
                                    new Object[]{turma[k].getNumTurma(),
                                    turma[k].getCurso(),
                                    turma[k].getCodigoMateria(),
                                    turma[k].getDisciplina(),
                                    prof[k].getNome(),
                                    turma[k].getCargaHorariaT(),
                                    turma[k].getQM()});
                                    vetorHour[k]=turma[k].getHorarioTurma();
                                }
                            }
                            else
                                JOptionPane.showMessageDialog(null, "Já existem
                                dados na aba de atribuição. É necessário deletá-
                                los.","Atribuir Dados",
                                JOptionPane.INFORMATION_MESSAGE);
                        }
                        else
                                JOptionPane.showMessageDialog(null, "Não há dados a
                                serem atribuídos.","Atribuir Dados",
                                JOptionPane.INFORMATION_MESSAGE);
                }
        }
    }

    //Evento Tratamento do Botão Filtrar e Mostrar Tudo-------------------------//

    private class TrataFiltro implements ActionListener{

        public void actionPerformed(ActionEvent e){

            if (e.getSource()==b28){
                int rowCount=dtm7.getRowCount();
                for (int i=0; i<rowCount; i++)
                        dtm7.removeRow(0);
                String numero=jta.getText();
                filtro=new HourTable();
                int aux=0;
                for(int k=0; k<turma.length; k++){
                        if(numero.equals(prof[k].getNumReg())){
```

```
                                    dtm7.insertRow(t7.getRowCount(),
                                    new Object[]{turma[k].getNumTurma(),
                                    turma[k].getCurso(),
                                    turma[k].getCodigoMateria(),
                                    turma[k].getDisciplina(), prof[k].getNome(),
                                    turma[k].getCargaHorariaT(),
                                    turma[k].getQM()});
                                    for (int i=0; i<15; i++)
                                        for (int j=0; j<6; j++)
                                            if(vetorHour[k].getA(i, j)==true)
                                                filtro.setA(i, j, true);
                                    vetorHour[aux]=turma[k].getHorarioTurma();
                                    aux++;
                            }
                        }
                }

                if (e.getSource()==b29){
                    int rowCount=dtm7.getRowCount();
                            for (int i=0; i<rowCount; i++)
                                    dtm7.removeRow(0);
                            for(int k=0; k<turma.length; k++){
                                    dtm7.insertRow(t7.getRowCount(),
                                    new Object[]{turma[k].getNumTurma(),
                                    turma[k].getCurso(),
                                    turma[k].getCodigoMateria(),
                                    turma[k].getDisciplina(), prof[k].getNome(),
                                    turma[k].getCargaHorariaT(),
                                    turma[k].getQM()});
                                    vetorHour[k]=turma[k].getHorarioTurma();
                            }
                }

                if (e.getSource()==b30){
                    for(int i=0;i<t8.getRowCount(); i++)
                            for(int j=1; j<t8.getColumnCount(); j++)
                                    t8.setValueAt(filtro.getA(i,j-1),i,j);

                }
            }
        }

        public static void main(String[] args) {
            GestaoDeHorario gt1=new GestaoDeHorario();
        }
}
```

```
public class Turma {

    private String numTurma;
    private String curso;
    private String codigoMateria;
    private String disciplina;
    private int cargaHorariaT;
    private HourTable horarioTurma;
    private String quiMen;

    public Turma() {
          numTurma="";
          curso="";
          codigoMateria="";
          disciplina="";
          cargaHorariaT=0;
          quiMen="";
          horarioTurma=new HourTable();
    }

    public Turma(String nt, String c, String cm, String d, int cht, String qm,
    HourTable ht){
          setNumTurma(nt);
          setCurso(c);
          setCodigoMateria(cm);
          setDisciplina(d);
          setCargaHorariaT(cht);
          setHorarioTurma(ht);
          setQM(qm);
    }

    public String getNumTurma(){
          return numTurma;
    }

    public String getCurso(){
          return curso;
    }

    public String getCodigoMateria(){
          return codigoMateria;
    }

    public String getDisciplina(){
          return disciplina;
    }

    public int getCargaHorariaT(){
          return cargaHorariaT;
    }

    public String getQM(){
          return quiMen;
    }

    public HourTable getHorarioTurma(){
          return horarioTurma;
    }
```

```
    public void setNumTurma(String nt){
       numTurma=nt;
    }

    public void setCurso(String c){
         curso=c;
    }

    public void setCodigoMateria(String cm){
         codigoMateria=cm;
    }

    public void setDisciplina(String d){
         disciplina=d;
    }

    public void setCargaHorariaT(int cht){
         cargaHorariaT=cht;
    }

    public void setQM(String qm){
         quiMen=qm;
    }

    public void setHorarioTurma(HourTable ht){
         horarioTurma=ht;
    }

    public boolean verificarHorario(Turma t){
    //verificar se os horários das turmas dentro do vetor não batem entre elas
         boolean aux=true;
         HourTable m1=t.getHorarioTurma();
         HourTable m2=this.getHorarioTurma();
         for (int i=0; i<15 && aux; i++)
          for (int j=0; j<6 && aux; j++){
             //verificar se para ambas turmas as células estão ocupadas
              if(m1.getA(i, j)==true && m2.getA(i, j)==true)
                   aux=false;
              else
                            aux=true;
          }
         return aux;
    }
}
```

```
public class Professor {

    private String numReg;
    private String nome;
    private int cargaHorariaP;

    public Professor() {
       numReg="";
          nome="";
          cargaHorariaP=0;
    }

    public Professor(String nr, String n, int chp){
          setNumReg(nr);
          setNome(n);
          setCargaHorariaP(chp);
    }

    public String getNumReg(){
          return numReg;
    }

    public String getNome(){
          return nome;
    }

    public int getCargaHorariaP(){
          return cargaHorariaP;
    }

    public void setNumReg(String nr){
          numReg=nr;
    }

    public void setNome(String n){
          nome=n;
    }

    public void setCargaHorariaP(int chp){
          cargaHorariaP=chp;
    }
}
```

```
import javax.swing.JOptionPane;

public class BeamSearch {

    private int beta;                    //largura de busca
    private int[] indice;                //vetor ordenação das soluções
    private Turma[] t;                   //vetor turmas a serem atribuídas
    private Professor[] p;               //vetor professores atribuidos às turmas
    private Solucao[] sTemp;             //vetor solução temporario
    private Solucao sFinal;                   //solução final
    private final int NUM=100000;        //constante usada no descarte de nós
    private int max;                     //variável usada no descarte de soluções

    public BeamSearch(Turma[] turma, Professor[] professor){
       boolean aux=false; beta=0;
       while(beta<=0 || !aux){
           try{
           //definição da largura de busca
               String b=JOptionPane.showInputDialog("Qual a largura de busca?");
               if(b==null)
                           break;
               else{
                           beta=Integer.parseInt(b);
                           aux=true;
               }
           }
           catch(NumberFormatException nfEx){
               JOptionPane.showMessageDialog(null, "Insira um valor válido.(Somente
               Números)", null, JOptionPane.INFORMATION_MESSAGE);
           }
          }
          t=turma;
          p=professor;
          max=NUM*t.length*10;
    }

    public Solucao resolver(MyTableModel2 dtm5, MyTableModel2 dtm6){
          for(int nivel=0; nivel<t.length; nivel++)
           //realizar atribuição para cada turma (nivel)
           gerarSolucoes(nivel, dtm5, dtm6);
          //selecionar melhor solução
          selecionarSolucao();
          return sFinal;
    }

    public void gerarSolucoes(int nivel, MyTableModel2 dtm5, MyTableModel2 dtm6){
       if(nivel==0){
          //indice da solução nao é necessariamente igual ao do professor então é
          //utilizada uma variável auxiliar para identificar o número de soluções
          //factíveis
           int i=0;
           for(int k=0; k<p.length; k++)
               //verificar se a atribuição é factível
               if(t[nivel].getCargaHorariaT()<=p[k].getCargaHorariaP())
                   i++;
                   //inicializar vetor solução temporária
                   sTemp=new Solucao[i];
                   i=0;
                   for(int k=0; k<p.length; k++)
```

```
                    //verificar se a atribuição é factível
                    if(t[nivel].getCargaHorariaT()<=p[k].getCargaHorariaP()){
                        //inicializar cada uma das soluções
                          sTemp[i]=new Solucao(t, p);
                          //preencher solução com valores de atribuição do nível
                          sTemp[i].preencherNivel(nivel, k, p[k], dtm5, dtm6);
                          i++;
                    }
                //criar vetor para armazenar custo total temporário gerado na
                //solucao gulosa
                int[] ctTemp=new int[sTemp.length];
                //para cada solução criada desenvolver algoritmo guloso
                for(i=0; i<sTemp.length; i++){
                    //criar vetor para guardar professores das soluções gulosas
                    Professor[] pTemp=sTemp[i].getProfessor();
                    //criar vetor das cargas horarias remanescentes até o
                          //momento para gerar os próximos níveis
                    int[] chrTemp=sTemp[i].getCargaReman();
                    //clonar vetor das cargas horarias remanescentes para
                          //utilizar no algoritmo guloso
                    int[] chrGulosa=chrTemp.clone();
                    //calcular custo total temp. do nó pelo algoritmo guloso
                    ctTemp[i]=solucaoGulosa(i, nivel, chrGulosa, pTemp, dtm5,
                    dtm6)+sTemp[i].getCustoTotal();
                }
                //ordenar vetor de custo total temporário das soluções gulosas
                indice=ordenar(ctTemp);
                for(i=beta; i<sTemp.length; i++)
                    //Invalidar/descartar as demais soluções
                    sTemp[indice[i]].setCustoTotal(max);
    }
    if(nivel>0){
    //desenvolver os níveis somente para as beta melhores soluções
        for(int i=0; i<beta; i++){
            //verificar se a solução é factível
            if(sTemp[indice[i]].getCustoTotal()<max){
                //criar vetor para guardar profs das soluções gulosas
                Professor[] pTemp=sTemp[indice[i]].getProfessor();
                //criar vetor para guardar custos parciais temporários
                int[] cTemp=new int[p.length];
                //criar vetor das cargas horarias remanescentes até o
                //momento para gerar os próximos níveis
                int[] chrTemp=sTemp[indice[i]].getCargaReman();
                for(int k=0; k<p.length; k++){
                //para cada prof verificar se atribuição é factível
                    if(t[nivel].getCargaHorariaT()<=chrTemp[k] &&
                            verificarChoque(nivel, p[k], pTemp))

                    //se factível, calcular custo parcial até o nó
                            cTemp[k]=sTemp[indice[i]].custoParcial(nivel,
                            p[k], pTemp, dtm5, dtm6) +
                                sTemp[indice[i]].getCustoTotal();
                    else
                    //se não for factível, inserir um custo bem grande
                    //para descarte do nó
                        cTemp[k]=NUM;
                }
                //criar vetor para armazenar custo total temporário
                 //gerado na solucao gulosa
```

```
                int[] ctTemp=new int[p.length];
                //para cada nó aplicar algoritmo guloso
                        for(int k=0; k<p.length; k++){
                    //verificar se nó não foi descartado (é factível)
                    if(cTemp[k]<NUM){
                    //inserir professor temporário do nível no vetor
                    //temporário de professores
                    pTemp[nivel]=p[k];
                    //clonar vetor das cargas horarias remanescentes
                    //para utilizar no algoritmo guloso
                    int[] chrGulosa=chrTemp.clone();
                    //atualizar carga horaria do professor temporário no nível
                    chrGulosa[k]=chrGulosa[k]-t[nivel].getCargaHorariaT();
                    //calcular custo total temp. do nó pelo algoritmo guloso
                    ctTemp[k]=solucaoGulosa(indice[i], nivel, chrGulosa, pTemp,
                    dtm5, dtm6)+cTemp[k];
                    }
                    else
                    //descartar solução gulosa se o nó foi descartado
                        ctTemp[k]=max;
                }
                //ordenar vetor de custo total temp das soluções gulosas
                int[] ind=ordenar(ctTemp);
                //verificar se solução gulosa de menor custo é factível
                if(ctTemp[ind[0]]<max)
                //se factível, desenvolver o próximo nível da  solução
                    sTemp[indice[i]].preencherNivel(nivel, ind[0], p[ind[0]],
                    dtm5, dtm6);
                else
                //se não for factível, invalidar a solução
                            sTemp[indice[i]].setCustoTotal(max);
            }
        }
    }
}

public int solucaoGulosa(int index, int nivel, int[] chrGulosa, Professor[]
pTemp, MyTableModel2 dtm5, MyTableModel2 dtm6){
   int custo=0;
   //desenvolver cada nivel da solução gulosa
   for(int i=nivel+1; i<t.length; i++){
       //criar vetor de custo parcial temporário usado no algoritmo guloso
       int[] cGuloso=new int[p.length];
       for(int k=0; k<p.length; k++){
           //para cada professor verificar se atribuição é factível
           if(t[i].getCargaHorariaT()<=chrGulosa[k] && verificarChoque(i, p[k],
           pTemp))
               //se factível, calcular custo parcial do nó
               cGuloso[k]=sTemp[index].custoParcial(i, p[k], pTemp, dtm5,
                dtm6);
           else
               //se não factível, inserir um custo grande para descarte do nó
               cGuloso[k]=NUM;
       }
       //ordenar vetor de custo parcial dos nós
       int[] guloso=ordenar(cGuloso);
       //obtendo custo parcial da solução de menor custo ate o nivel
       custo=custo+cGuloso[guloso[0]];
       //verificar se o nó de menor custo parcial é factível
```

```
          if(cGuloso[guloso[0]]<NUM){
                  //se factível, atualizar professor atribuído no nível e sua
carga                       //horária remanescente
                  pTemp[i]=p[guloso[0]];
                  chrGulosa[guloso[0]]=chrGulosa[guloso[0]]-
t[i].getCargaHorariaT();
          }
          //invalidar solução caso não seja factível o desenvolvimento a partir de
          //algum nivel
          else{
                  i++;
                  while(i<t.length){
                  //somar constante a cada nível p/ que possibilita comparar os
valores
                  custo=custo+NUM;
                  i++;
                  }
                  break;
          }
      }
      //retornar valor do custo total temporário
      return custo;
    }

    public boolean verificarChoque(int nivel, Professor prof, Professor[]
    vetorProf){
          boolean aux=true;
          for(int i=0; i<nivel; i++)
           //virificar se há choque somente se as turmas tiverem o mesmo professor
           if(vetorProf[i].getNumReg().equals(prof.getNumReg())){
               //retornar falso caso alguma turma tenha choque de horário com a
               //turma em questão
                   aux=t[nivel].verificarHorario(t[i]);
                   if(aux==false)
                       //retornar false somente se as turmas não forem quinzenais
                       //ou mensais, de mesmo curso e disciplina
                       if(t[nivel].getQM().equals("Q")){
                           if(t[nivel].getCurso().equals(t[i].getCurso()) &&

    t[nivel].getCodigoMateria().equals(t[i].getCodigoMateria()))
                                 aux=true;
                               else
                                 break;
                       }
          }
      return aux;
    }

    public int[] ordenar(int[] c){
          int aux;
          int[] ind=new int[c.length];
          for(int i=0; i<c.length; i++)
           //criar indice auxiliar
           ind[i]=i;
          for(int i=0; i<c.length-1; i++)
           for (int j=0; j<c.length-i-1; j++)
           //comparar custos e ordená-los por meio do índice auxiliar
               if (c[ind[j]]>c[ind[j+1]]){
                   aux=ind[j];
```

```
                    ind[j]=ind[j+1];
                    ind[j+1]=aux;
                }
        return ind;
    }

    public void selecionarSolucao(){
        //criar vetor para os custos totais das soluções
        int[] ct=new int[sTemp.length];
        sFinal=new Solucao(t, p);
        for(int i=0; i<sTemp.length; i++)
         //extrair custos totais das soluções
         ct[i]=sTemp[i].getCustoTotal();
        //ordenar custos
        int ind[]=ordenar(ct);
        //verificar se solução com menor custo total é factível
        if(ct[ind[0]]<max)
         //se factível selecionar
         sFinal=sTemp[ind[0]];
        //se não for factível, imprimir mensagem
        else
         JOptionPane.showMessageDialog(null, "Não foi possível obter uma
                 solução.", null, JOptionPane.INFORMATION_MESSAGE);
    }
}
```

```
import java.io.*;
import java.util.Scanner;

public class Solucao {

    private Turma[] vt;                    //vetor armazena turmas
    private Professor[] vp;                //vetor armazena professores
    private int custoTotal                 //custo total da solução beta
    private int[] chReman;                 //carga horária remanescente
    Scanner fileIn;

    public Solucao(Turma[] t, Professor[] p){
          vt=t;
          vp=new Professor[vt.length];
          custoTotal=0;
          chReman=new int[p.length];
          for(int i=0; i<p.length; i++)
           chReman[i]=p[i].getCargaHorariaP();
    }

    public void preencherNivel(int nivel, int index, Professor p, MyTableModel2
    dtm5, MyTableModel2 dtm6){
          //atribuir professor à turma correspondente ao nível
          vp[nivel]=p;
          //calcular o custo parcial da solução até o nível
          custoTotal=custoTotal+custoParcial(nivel, vp[nivel], vp, dtm5, dtm6);
          //Atualizando carga horaria remanescente dos professores
          chReman[index]=chReman[index]-vt[nivel].getCargaHorariaT();
    }

    public int custoParcial(int nivel, Professor p, Professor[] vetorProf,
    MyTableModel2 dtm5, MyTableModel2 dtm6){
          int      custoSimil=0;
          if(nivel!=0)
           for(int aux=0; aux<nivel; aux++){
               //verificar similaridade entre disciplinas somente para turmas com
               //mesmo professor
               if(vetorProf[aux].getNumReg().equals(p.getNumReg()))
                    //somar os custos de similaridade entre as turmas
                    custoSimil=custoSimil+custoSimilaridade(aux, nivel, dtm6);
           }
          //somar custo de pref. com custo de simil. para obter o custo parcial no
nó
          int custoParcial=custoPreferencia(nivel, p, dtm5)+custoSimil;
          return custoParcial;
    }

    public int custoPreferencia(int nivel, Professor p, MyTableModel2 dtm5){
          int i, j;
          String nr="";
          String cm="";
          int cp=0;
          for(j=1; j<dtm5.getColumnCount(); j++){
           cm=dtm5.getColumnName(j);
           if(cm.equals(vt[nivel].getCodigoMateria()))
               break;
          }
          for (i=0; i<dtm5.getRowCount(); i++){
           nr=dtm5.getValueAt(i,0).toString();
```

```
            if(nr.equals(p.getNumReg()))
                break;
            }
            cp=Integer.parseInt(dtm5.getValueAt(i,j).toString());
            return cp;
    }

    public int custoSimilaridade(int aux, int nivel, MyTableModel2 dtm6){
            int i, j;
            String cm1="";
            String cm2="";
            int cs=0;
            for(j=1; j<dtm6.getColumnCount(); j++){
            cm1=dtm6.getColumnName(j);
            if(cm1.equals(vt[aux].getCodigoMateria()))
                break;
            }
            for (i=0; i<dtm6.getRowCount(); i++){
             cm2=dtm6.getValueAt(i,0).toString();
             if(cm2.equals(vt[nivel].getCodigoMateria()))
                break;
            }
            cs=Integer.parseInt(dtm6.getValueAt(i,j).toString());
            return cs;
    }

    public void setCustoTotal(int ct){
            custoTotal=ct;
    }

    public int getCustoTotal(){
            return custoTotal;
    }

    public int[] getCargaReman(){
            return chReman;
    }

    public Turma[] getTurma(){

            return vt;
    }

    public Professor[] getProfessor(){
            return vp;
    }
}
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import java.io.*;
import java.util.Scanner;

public class HourTable {

    // Matriz com os dados de horarios
    // marcados para cada turma.
    private int lin = 15;
    private int col = 6;
    private boolean A[][] = new boolean[lin][col];

    public HourTable(){
       // inicializando os elementos da Table.
       for (int i=0; i < lin; i++)
           for (int j=0; j < col; j++)
               A[i][j] = false;
    }

    public void setA(int i, int j, boolean elem){
       if (((i < lin)&&(i >= 0))&&((j < col)&&(j >= 0)))
            A[i][j] = elem;
    }

    public boolean getA(int i, int j){
         return A[i][j];
    }
}
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import java.io.*;
import java.util.Scanner;

class MyTableModel extends AbstractTableModel{

    private boolean DEBUG=false; //se fosse true ativaria o modo debug

    String[] colunas=new String[]{"Horário", "Segunda", "Terça", "Quarta",
    "Quinta", "Sexta", "Sábado"};
    Object[][] dados=new Object[][]{
    {"07:30~08:20", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"08:20~09:10", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"09:10~10:00", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"10:20~11:10", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"11:15~12:05", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"13:30~14:20", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"14:20~15:10", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"15:30~16:20", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"16:20~17:10", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"17:15~18:10", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"18:30~18:55", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"19:15~20:05", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"20:05~20:55", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"21:10~22:00", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)},
    {"22:00~22:50", new Boolean(false), new Boolean(false), new Boolean(false),
    new Boolean(false), new Boolean(false), new Boolean(false)}};

    public int getColumnCount(){
       return colunas.length;
    }

    public int getRowCount(){
         return dados.length;
    }

    public String getColumnName(int col){
         return colunas[col];
    }
```

```
    public Object getValueAt(int row, int col){
          return dados[row][col];
    }

    public Class getColumnClass(int c){
          return getValueAt(0, c).getClass();
    }

    public boolean isCellEditable(int row, int col){
          if (col<1)
           return false;
       else
           return true;
    }//para o caso de células editáveis

    public void setValueAt(Object value, int row, int col){
          if (DEBUG)//no modo debug imprime as alterações que estão sendo feitas na
          //tabela de horários
           System.out.println("Alterando o valor de" + row + "," + col + "para" +
           value);
          dados[row][col]=value;//alterando dado da tabela
          fireTableCellUpdated(row, col);//atualizando valor

          if (DEBUG){
           System.out.println("Novo valor:");
           printDebugData();//no modo debug imprime os novos valores de horários
           //que estão sendo associados
          }
    }//usado para o caso de mudança em algfum valor dos dados da tabela

    private void printDebugData(){
          int numRows=getRowCount();
          int numCols=getColumnCount();

          for (int i=0; i<numRows; i++){
           System.out.println("   linha" + i + ":");
           for (int j=0; j<numCols; j++)
                           System.out.println("  " + dados[i][j]);
           System.out.println();
          }
          System.out.println("----------------------------------------");
    }//impressão no modo debug de cada linha da tabela de horário
}
```

```
import java.util.Vector;
import java.util.Enumeration;
import javax.swing.table.*;
import javax.swing.JTable;
import java.awt.*;

public class MyTableModel2 extends DefaultTableModel{

    public MyTableModel2(Object[][] dados, String[] colunas){
         super(dados, colunas);
    }

    public void removeColumn(int colModelIndex, Enumeration enumeracao){
         columnIdentifiers.removeElementAt(colModelIndex);
         for(int r=0; r<dataVector.size(); r++){
          Vector row=(Vector)dataVector.get(r);
          row.removeElementAt(colModelIndex);
         }
         this.setDataVector(dataVector, columnIdentifiers);
         //Corrigir indices do table model
         for (; enumeracao.hasMoreElements(); ) {
          TableColumn c=(TableColumn)enumeracao.nextElement();
          if (c.getModelIndex()>=colModelIndex)
              c.setModelIndex(c.getModelIndex()-1);
         }
    }

    public void formatarColuna(JTable t){
         TableColumn pd=t.getColumnModel().getColumn(0);
         pd.setPreferredWidth(120);
         DefaultTableCellRenderer dtcr=new DefaultTableCellRenderer();
         dtcr.setBackground(new Color(240, 240, 240));
         pd.setCellRenderer(dtcr);
    }
}
```

ANEXO – Distribuição de disciplinas no Departamento de Matemática em 2011

Distribuição de disciplinas 2011 – Depto. de Matemática – Área de Computação – Prof. Efetivos

Disciplina	Sala/Lab(Cap)	Semestre	Dia	Horário	CHD
Aníbal Tavares de Azevedo					**15/9h**
PC-II (222) – MEC (62)	501 (63)	A	3ªf	15:30-17:10	2*
LPC-II (243) – MEC	LDC5 (34)	A	3ªf	15:30-17:10	2*
LPC-II (244) – MEC	LDC5 (34)	A	3ªf	17:15-19:00	2*
LPC-II (321) – MCN/LPC-II (221) – PRO	LDC5 (34)	A	3ªf	19:15-20:05	2*
LPC-II (322) – MCN/LPC-II (222) – PRO	LDC5 (34)	A	3ªf	19:15-20:05	2*
LPC-II (222) – CIV	LDC7 (21)	A	4ªf	08:20-10:00	2*
PC-II (311) – MCN/ (222) – PRO (55)	102 (45)	A	4ªf	21:10-22:50	2*
PC-II (211) – CIV (32)	509 (48)	A	5ªf	08:20-10:00	2*
LPC-II (221) – CIV	LDC7 (21)	A	5ªf	08:20-10:00	2*
Cassilda Maria Ribeiro					**0/11h**
PC-I (155) – EU	ANF I (84)	A	2ªf	15:30-17:10	2
PC-I (131) – ESP (49)	503 (63)	A	2ªf	17:15-18:55	2
LPC-I (109) – EU	LDC7 (21)	A	3ªf	10:20-12:05	2*
LPC-I (110) – EU	LDC7 (21)	A	3ªf	10:20-12:05	2*
LPC-I (161) – EU	LDC5 (34)	A	5ªf	13:30-15:10	2*
LPC-I (162) – EU	LDC3 (21)	A	5ªf	15:30-17:10	2*
CN - (211) – CIV	CTI 11 (60)	2	6ªf	09:30-12:05	3
Décio Cardozo Mourão					**14/10h**
CDI		A	3ªf	13:30-15:10	2
CDI		A	4ªf	08:20-10:00	2
PC-I (211) – MCN (28)	508 (48)	A	4ªf	21:10-22:50	2
CDI		A	5ªf	13:30-15:10	2
LPC-I (221) – MCN	LDC2 (21)	A	5ªf	19:15-20:55	2*
LPC-I (222) – MCN	LDC2 (21)	A	5ªf	19:15-20:55	2*
Edson Luiz França Senne					**3/3h**
PC-II (211) – ELE (40)	102 (45)	A	6ªf	16:20-18:10	2*
LPC-II (221) – ELE	LDC2 (21)	A	6ªf	16:20-18:10	2*
LPC-II (222) – ELE	LDC2 (21)	A	6ªf	18:30-20:10	2*
Galeno José de Sena					**13/10h**
MAC (111) – ESP	-	1	2ªf	17:15-19:00	2
PC-I (132) – ESP (26)	505 (68)	A	3ªf	17:15-19:00	2
PC-II (221) – MEC (63)	501 (63)	A	4ªf	13:30-15:10	2*
LPC-II (241) – MEC	LDC5 (34)	A	4ªf	13:30-15:10	2*
LPC-II (242) – MEC	LDC5 (34)	A	4ªf	15:30-17:10	2*
BD * OPT *		2	5ªf	09:30-12:00	3
LPC-I (163) – ESP	LDC3 (21)	A	5ªf	13:30-15:10	2*
LPC-I (164) – ESP	LDC3 (21)	A	5ªf	15:30-17:10	2*
Júlio Antunes Santana					**13/6h**
CN (211) – ELE	501 (63)	2	2ªf	09:30-12:05	3
CN (222) – MEC	508 (48)	1	2ªf	13:30-16:20	3
CN (211) – MAT	508 (48)	1	3ªf	13:30-16:20	3
IPE 611 – MEC *OPT*	LDC5 (34)	1	3ªf 4ªf	21:10-22:50 19:15-20:55	2 2
CN - (221) – MEC	501 (63)	1	6ªf	15:30-18:00	3
IPE 611 – MEC *OPT	LDC5 (34)	2	6ªf	15:30-18:00	3
Marcos Antônio Pereira					**0/11h**
CN (221) – PRO	CTI 11 (60)	2	2ªf	09:30-12:05	3
PC-I (151) – EU	503 (63)	A	2ªf	13:30-15:10	2
PC-I (152) – EU	Anf. III (136)	A	2ªf	15:30-17:10	2
LPC (101) – EU	LDC2 (21)	A	3ªf	08:20-10:00	2*
LPC (102) – EU	LDC2 (21)	2	3ªf	10:20-12:05	2*
LPC-I (103) – EU	LDC1 (21)	2	3ªf	08:20-10:00	2*
LPC-I (104) – EU	LDC1 (21)	A	3ªf	10:00-12:05	2*
LPC (102) – EU	LDC2 (21)	1	4ªf	13:30-15:10	2*
LPC-I (103) – EU	LDC1 (21)	1	4ªf	13:30-15:10	2*
Carga Horária Semanal					**58/59h**

Distribuição de disciplinas 2011 – Depto. de Matemática – Área de Computação – Prof. Emergenciais

Disciplina	Sala/Lab(Cap)	Semestre	Dia	Horário	CHD
Substituto: Robinson Carlos Teixeira (Supervisor: Prof. Galeno)					**12/12h**
PC-I (154) – EU	506 (75)	A	2ªf	15:30-17:10	2
LPC-I (107) – EU	LDC6 (21)	2	3ªf	08:20-10:00	2*
LPC-I (108) – EU	LDC6 (21)	A	3ªf	10:20-12:05	2*
CCN 211 ESP / CN (311) – FIS	-	A/2	3ªf	19:15-20:55	2
LPC-I (107) – EU	LDC6 (21)	1	4ªf	13:30-15:10	2*
CCN 211 ESP / LCN (311) – FIS	-	A/2	4ªf	21:10-22:50	2
PC-I (133) – ESP (39)	502 (63)	A	6ªf	13:30-15:10	2
LPC-I (165) – ESP	LDC5 (34)	A	6ªf	15:30-17:10	2*
LPC-I (166) – ESP	LDC5 (34)	A	6ªf	13:30-17:10	2*
Substituto: Carlos Renato (Supervisor: Prof. Galeno)					**7,5/11h**
CN (211) – LMN	502 (63)	A	2ªf	19:15-20:55	2
IC (111) – FIS	507 (72)	2	4ªf	19:15-20:55	2
LCN (211) – LMN	LDC1 (21)	A	4ªf	21:10-22:55	2
IC (111) – FIS	507 (72)	2	5ªf	21:10-22:50	2
CN (311) – MCN	104 (45)	2	6ªf	21:10-22:50	2
CN (311) – MCN	508 (48)	2	Sáb	10:20-12:05	2*
Bolsista 1: Marlon da Silva (Supervisor: Prof. Senne)					**3/3h**
PC (111) – LMN	501 (63)	A	6ªf	21:10-22:50	2
LPC (111) – LMN	LDC5 (34)	A	Sab.	10:20-12:05	2*
Bolsista 2: Kelly Cristina Coelho de Carvalho Benini (Supervisor: Prof. Anibal)					**6/4h**
PC-I (153) – EU	506 (75)	A	2ªf	13:30-15:10	2
LPC-I (105) – EU	LDC7 (21)	A	3ªf	08:20-10:05	2*
LPC-I (106) – EU	LDC3 (21)	2	3ªf	10:20-12:05	2*
LPC-I (106) – EU	LDC7 (21)	1	4ªf	13:30-15:10	2*
Carga Horária Semanal					**28,5/28h**