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

ARTORIAS: SOLUÇÃO PARA O PROBLEMA DE ALOCAÇÃO DE HORÁRIOS UNIVERSITÁRIOS

Formiga - MG

2017

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

ARTORIAS: SOLUÇÃO PARA O PROBLEMA DE ALOCAÇÃO DE HORÁRIOS UNIVERSITÁRIOS

Monografia do trabalho de conclusão de curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Cláudio Jose Menezes Junior

Orientador: Walace de Almeida Rodrigues

Formiga - MG
2017

Menezes Junior, Cláudio Jose.
005 Artorias : solução para o problema de alocação de horários universitários / Cláudio Jose Menezes Junior. – Formiga : IFMG, 2017.
49p. : il.

Orientador: Prof. Msc. Walace de Almeida Rodrigues
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Simulated Annealing. 2. Tabu Search. 3. TimeTabling. 4. Grasp. 5. Gerador de horários. I. Título.

CDD 005

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

Cláudio Jose Menezes Junior

ARTORIAS: SOLUÇÃO PARA O PROBLEMA DE ALOCAÇÃO DE HORÁRIOS UNIVERSITÁRIOS

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

Aprovado em: 13 de novembro de 2017.

BANCA EXAMINADORA

Prof.º Walace de Almeida Rodrigues

Prof.º Diego Mello da Silva

Prof.º Denise Ferreira Garcia Resende

Este trabalho é dedicado ao meu pái e minha mãe,
que sempre me apoiaram nas minhas decisões.

Agradecimentos

Primeiramente gostaria de agradecer meus pais por tudo que fizeram para que eu estivesse aqui e a pessoa que me tornei.

A morena dondoka, Priscila, minha companheira nesta caminhada, e um pilar que sempre pude contar.

Ao Willa, o vigilante, pelas horas de conversa sobre *Dark Souls*, um amigo das aulas entediantes.

Agradeço tambem a toda equipe da *PullUp*, principlamente ao Thyaguinho pela compreenção, ao meu grande irmão Tiago o mago da computação que me salvou em uma hóra difícil, e ao Guilherme *meninin* por me ajudar e incentivar o estudo do *Laravel*.

"E a única forma que pode ser norma,
é nenhuma regra ter,
é nunca fazer nada que o mestre mandar
Sempre desobedecer
Nunca reverenciar
(Belchior)

Resumo

A alocação de horários é um problema clássico de computação, chamado tabela-horário ou timetabling. O problema consiste em alocar um conjunto de aulas em um número pré-determinado de horários satisfazendo diversas restrições envolvendo professores, alunos e o espaço físico disponível. O presente projeto propõe desenvolver um protótipo de um produto de software para o problema de alocação de horários de instituições universitárias. Esse problema pertence a classe dos problemas NP-difícil. A proposta consiste em aplicar a meta-heurística *GRASP* fazendo variações em sua busca local com as meta-heurísticas de *Simullated Annealing* e de *Tabu Search*.

Palavras-chave: Simulated Annealing. Tabu Search. TimeTabling. Grasp. Gerador de horários.

Abstract

Time allocation is a classic computing problem, called time-table or timetabling. The problem is to allocate a set of classes in a predetermined number of schedules satisfying several restrictions involving teachers, students and the available physical space. The present project proposes to develop a prototype of a software product for the problem of time allocation of university institutions. This problem belongs to the class of NP-difficult problems. The proposition is to apply the meta-heuristic GRASP by making variations on its local search with the meta-heuristics of *Simullated Annealing* and *Tabu Search*.

textbf Keywords: Simulated Annealing. Tabu Search. TimeTabling. Grasp.

Keywords: latex. abntex. text editoration.

Lista de ilustrações

Sumário

1 Introdução

A alocação de horários é um problema clássico de computação, chamado tabela-horário ou timetabling. O problema consiste em alocar um conjunto de aulas em um número pré-determinado de horários satisfazendo diversas restrições envolvendo professores, alunos e o espaço físico disponível. A solução manual deste problema não é uma tarefa trivial e as instituições de ensino precisam resolvê-lo anualmente ou semestralmente. É difícil encontrar uma solução satisfatória para este problema, pois contemplar todos os anseios das partes envolvidas não é uma tarefa simples. Ademais, esse problema pertence a classe dos problemas NP-difícil (ROCHA, 2013).

Tendo em vista essa situação, o presente projeto propõe desenvolver um protótipo de um produto de software para o problema de alocação de horários de instituições universitárias. Para efeito de testes e avaliações será utilizada como instância real o IFMG campus Formiga. A proposta consiste em aplicar a meta-heurística *GRASP* (*Greed Randomize Adaptative Search Procedures*) (ROCHA, 2013), fazendo variações em sua busca local com as meta-heurísticas de *Simulated Annealing* (CESCHIA S.; GASPERO, 2011) e de *Tabu Search* (GLOVER F. E LAGUNA, 1997).

1.1 Justificativa

Existe a demanda real de que em cada semestre letivo em uma universidade é necessário gerar uma tabela horário atendendo às restrições da instituição. O problema de tabela-horário é de natureza combinatória, aumentando sua dificuldade na medida em que aumenta o número de restrições. Geralmente problemas com essas características pertencem à classe de problemas NP-completo (SCHAERF, 1995), para os quais a busca por soluções exatas é praticamente viável somente para instâncias pequenas. Instâncias reais como aquelas geradas por universidades frequentemente extrapolam o limite aceitável de tempo quando se buscam soluções exatas.

Resultados satisfatórios podem ser obtidos em tempo hábil com a utilização de meta-heurísticas. Para esse problema, duas meta-heurísticas que foram utilizadas obtendo bons resultados são: o Simulated Annealing (CESCHIA S.; GASPERO, 2011) e a Tabu Search (ELLOUMI, 2008) e (SOUZA M. J. F.; MACULAN, 2004).

O nicho de otimização em alocação de horários envolve uma séries de subáreas abordadas no curso de bacharelado em Ciência da Computação como Algoritmos e Estruturas de Dados, Projeto e Análise de Algoritmos, Pesquisa Operacional e Inteligência Artificial. Essas disciplinas motivaram o interesse por tais subáreas e a oportunidade de

aplicar seus conhecimentos e técnicas em um único projeto é satisfatória e encorajadora.

1.2 Objetivos

1.2.1 Objetivo Geral

Tem-se como objetivo geral obter um protótipo de um produto de software para o problema de alocação de horários usando a abordagem GRASP com duas variações na busca local, a saber: o Simulated Annealing e a Tabu Search. Avaliar tais algoritmos através de análise experimental. Comparar os resultados obtidos usando instâncias reais do IFMG campus Formiga e instâncias do benchmark International Timetabling Competition (ITC- 2007).

1.2.2 Objetivos Específicos

- Obter uma implementação do algoritmo GRASP para o problema de alocação de horários, com busca local Simulated Annealing, seguindo o modelo proposto por Rocha (ROCHA, 2013).
- Obter uma implementação do algoritmo GRASP para o problema de alocação de horários, com busca local Tabu.
- Definir uma função objetivo para avaliar a qualidade das soluções geradas;
- Minimizar a função objetivo definida;

2 Fundamentação Teórica

Para entendimento do problema de alocação de horários é necessário investigar as principais abordagens encontradas na literatura, bem como os principais métodos computacionais aplicados para otimização de soluções para esse problema.

2.1 TimeTabling

O problema de tabela horário educacional apresenta diversas formulações na literatura de acordo com o ambiente aplicado e seu conjunto de restrições. Schaerf apresenta três classes para o problema: Tabela horário de Escola, Universidade e Exames (SCHAERF, 1995).

A Tabela horário de universidade se reduz a uma programação semanal de aulas, minimizando a sobreposição de aulas e atendendo todas restrições envolvidas no ambiente. Para avaliar a qualidade das soluções geradas utilizam-se coleções de instâncias mundialmente conhecidas. Essas coleções são denominadas benchmarks. Dentre os benchmarks disponíveis, há o International Timetabling Competition (ITC - versões 2002, 2007 e 2011), com instâncias para o problema de Tabela horário para universidades bem próximas à realidade.

A versão três do ITC-2007 é fundamentada no ambiente real das universidades que aborda dois grupos de restrições: fracas e fortes, em que o objetivo é encontrar uma tabela horário viável e que minimize a quantidade de violações fracas.

2.2 Modelo padrão do Padrão ITC - 2007

A formulação matemática adotada pelo International Timetabling Competition (ITC-2007) constitui em cinco blocos de parâmetros. Estes são baseados em casos reais de algumas universidades européias. Dado que o ITC-2007 possui três formulações, uma para tabela-horário de exames e duas para tabela-horário de universidades. A descrição abaixo compete a formulação três.

O modelo de um arquivo para uma instância do ITC-2007 é escrita da seguinte forma, em cinco partes:

- A primeira parte, contém os dados gerais que descrevem a instância e que são: nome da instância, número de disciplinas, número de salas, número de dias, número de períodos por dia, número de currículos (um currículo denota um conjunto de

disciplinas que compõem um determinado período de um curso), e por último o número de restrições.

- A segunda parte, contém a descrição de cada disciplina: nome da disciplina e do professor, número de aulas por semana, mínimo de dias na semana que devem conter as aulas, número de alunos.
- A terceira parte, contém a descrição das salas: nome da sala e sua lotação.
- A quarta parte, contém a descrição dos currículos: nome do currículo, número de disciplinas, seguido do nome de cada disciplina.
- quinta parte, contém a descrição das indisponibilidades (ou restrições), cada uma contendo: nome da disciplina, dia e horário que esta não pode ocorrer.

A figura 1 ilustra um arquivo exemplo para uma instância de teste chamada Toy. Esta instância é bem reduzida e foi fornecida pela organização do ITC-2007 apenas para demonstrar o formato das instâncias oficiais do campeonato.

```
Name: Toy
Courses: 4
Rooms: 3
Days: 5
Periods_per_day: 4
Curricula: 2
Constraints: 8

COURSES:
SceCosC Ocra 3 3 30
ArcTec Indaco 3 2 42
TecCos Rosa 5 4 40
Geotec Scarlatti 5 4 18

ROOMS:
rA 32
rB 50
rC 40

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec

UNAVAILABILITY_CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3
ArcTec 4 0
ArcTec 4 1
ArcTec 4 2
ArcTec 4 3

END.
```

Figura 1 – Arquivo com dados de instância Toy no formato ITC - 2007

Uma solução consiste na alocação de cada aula em um horário e uma sala. A partir dos requisitos do problema são estabelecidas as restrições fortes e fracas: as restrições fortes devem ser sempre respeitadas (a violação de uma restrição forte torna uma tabela-horário inviável, que na prática não pode ser utilizada); as restrições fracas devem ser satisfeitas o máximo possível (a violação de uma restrição fraca não torna uma tabela-horário inviável, mas quanto menos violações houver, melhor é a tabela-horário). As restrições do problema

são descritas a seguir:Uma solução consiste na alocação de cada aula em um horário e uma sala. A partir dos requisitos do problema são estabelecidas as restrições fortes e fracas: as restrições fortes devem ser sempre respeitadas (a violação de uma restrição forte torna uma tabela-horário inviável, que na prática não pode ser utilizada); as restrições fracas devem ser satisfeitas o máximo possível (a violação de uma restrição fraca não torna uma tabela-horário inviável, mas quanto menos violações houver, melhor é a tabela-horário). As restrições do problema são descritas a seguir:

1. **Restrições Fortes**
    - Aulas: Todas as aulas das disciplinas devem ser alocadas e em períodos diferentes. Uma violação ocorre se uma aula não é alocada.
    - Conflitos: Aulas de disciplinas do mesmo currículo ou lecionadas pelo mesmo professor devem ser alocadas em períodos diferentes.
    - Ocupação de Sala: Duas aulas não podem ocupar uma sala no mesmo horário.
    - Disponibilidade: Uma aula não pode ser alocada num horário em que a disciplina é indisponível.
2. **Restrições Fracas**
    - Dias Mínimos de Trabalho: As aulas de cada disciplina devem ser espalhadas por uma quantidade mínima de dias. Cada dia abaixo do mínimo é contado como uma violação.
    - Aulas Isoladas: Aulas do mesmo currículo devem ser alocadas em períodos adjacentes. Cada aula isolada é contada como uma violação.
    - Capacidade da Sala: O número de alunos da disciplina deve ser menor ou igual ao número de assentos da sala em que a aula for alocada. Cada aluno excedente contabiliza uma violação.
    - Estabilidade de Sala: Todas as aulas de uma disciplina devem ser alocadas na mesma sala. Cada sala distinta é contada como uma violação.

2.3 Grasp

O algoritmo GRASP, como é descrito por (FEO T. A.; RESENDE, 1995), consiste em um processo iterativo em duas fases: fase construtiva e busca local. Um pseudocódigo geral pode ser visto na Figura 2.

A linha 1 corresponde a entrada do problema. As linhas 2 até 6 correspondem ao processo iterativo que termina de acordo com algum critério de parada, como quantidade de iterações ou qualidade da solução encontrada. A linha 3 é a fase construtiva. A linha 4

```
procedure grasp()
1    InputInstance();
2    for GRASP stopping criterion not satisfied →
3        ConstructGreedyRandomizedSolution(Solution);
4        LocalSearch(Solution);
5        UpdateSolution(Solution,BestSolutionFound);
6    rof;
7    return(BestSolutionFound)
end grasp;
```

Figura 2 – Pseudocódigo genérico para o GRASP

é a fase busca local. Se uma solução melhor do que a atual é encontrada, então a solução atual é atualizada na linha 5.

A fase construtiva consiste em obter-se uma solução factível de forma gradativa, adicionando um elemento à solução em cada iteração. A cada iteração, a escolha do próximo elemento a ser adicionado na solução é determinada pela ordenação de todos os elementos de acordo com alguma função gulosa, tal função mede o benefício de adicionar o elemento à solução atual. O processo é adaptativo pois a cada iteração, o benefício de inserção do elemento na solução é atualizado e pode ser diferente. Uma lista dos melhores candidatos é construída, contendo aqueles elementos de melhor benefício, chamada de Lista de Candidatos Restrita (RCL) e depois, um dos elementos é selecionado randomicamente para ser inserido na solução, provendo o fator probabilístico do método. Em suma, a fase construtiva pode ser vista na Figura 3.

```
procedure ConstructGreedyRandomizedSolution(Solution)
1    Solution = {};
2    for Solution construction not done →
3        MakeRCL(RCL);
4        s = SelectElementAtRandom(RCL);
5        Solution = Solution ∪ {s};
6        AdaptGreedyFunction(s);
7    rof;
end ConstructGreedyRandomizedSolution;
```

Figura 3 – Fase construtiva do GRASP

O método de busca local é utilizado no intuito de melhorar a solução encontrada na fase construtiva, pois essa pode não ser a melhor localmente. A fase de busca local perturba soluções na vizinhança fazendo alterações simples na solução atual e a substitui se encontrar alguma solução vizinha de melhor qualidade. O método de busca local utiliza como critério de parada o momento em que se encontra a melhor solução na vizinhança ou o número de iterações especificado é alcançado. Um exemplo de método de busca local pode ser visto na Figura 8, onde N corresponde à vizinhança da solução atual e t corresponde a solução vizinha selecionada.

```
procedure local(P,N(P),s)
1     for s not locally optimal →
2           Find a better solution t ∈ N(s);
3           Let s = t;
4     rof;
5     return(s as local optimal for P)
end local;
```

Figura 4 – Fase de busca local do GRASP

2.4 Simulated Annealing

O algoritmo Simulated Annealing (SA), também conhecido como Recozimento Simulado consiste em uma meta-heurística desenvolvida por (KIRKPATRICK, 1983), projetada para lidar com problemas de otimização combinatória. Consiste em uma simulação de processo termodinâmico que recozimento, onde alguns tipos de materiais são expostos inicialmente a um ambiente de alta temperatura, que é reduzida gradativamente de forma que o arranjo molecular dos materiais expostos se aproxime de um arranjo cristalino, com custo energético mínimo. Usando tal analogia Kirkpatrick propôs um método inspirado neste processo em que um espaço de busca de um problema combinatório é visitado através desta dinâmica que considera a temperatura e energia do sistema.

O método explora a vizinhança de uma solução inicial para o problema e altera o sentido da busca no espaço de soluções em direção à uma solução vizinha de menor energia, se houver alguma, ou na direção de uma solução de maior 'energia' com uma probabilidade que depende da temperatura e da variação energética que o sistema sofre ao fazer tal movimento. O procedimento se repete até a temperatura do sistema atingir um limiar mínimo, de onde a solução com menor estado energético visitada ao longo do processo é devolvida como solução da busca. Para melhor compreensão dos conceitos, o pseudocódigo básico de um algoritmo SA é dado na Figura 5.

Algoritmo 1 SIMULATED-ANNEALING($f(.)$, $N(.)$, α, $SAMax$, T_0, s)

```
1: s* ← s; Iter ← 0; T ← T0
2: while (T > 0) do
3:     while (Iter < SAMax) do
4:         Iter ← Iter + 1
5:         Gere um vizinho qualquer s' ∈ N(s)
6:         Δ ← f(s') − f(s)
7:         if (Δ < 0) then
8:             s ← s'
9:             if (f(s') < f(s*)) then
10:                s* ← s'
11:            end if
12:        else
13:            Tome x ∈ [0, 1]
14:            if (x < e^(−Δ/T)) then
15:                s ← s'
16:            end if
17:        end if
18:    end while
19:    T ← αT; Iter ← 0
20: end while
21: return s*
```

Figura 5 – Pseudocódigo genérico para o Simulated Annealing

2.5 Tabu Search

O Tabu Search é uma meta-heurística de busca local originada nos trabalhos independentes (GLOVER F. E LAGUNA, 1997), é uma técnica para guiar uma meta-heurísticas de busca local tradicional na exploração do espaço de soluções além da otimalidade local, usando para isso basicamente estruturas de memória. Ela é uma das meta-heurísticas mais usadas e seu sucesso decorre de sua eficiência em produzir soluções de alta qualidade para vários problemas combinatórios. O Pseudocódigo é mostrado na figura 6.

Os seguintes parâmetros enumerados impactam diretamente na eficiência da Tabu Search:

1. Delimitação da vizinhança: seleção dos movimentos vizinhos que são examinados, entendendo por movimento uma pequena alteração na solução corrente que gera uma nova solução vizinha;
2. Escolha do critério de seleção da próxima solução vizinha;
3. Escolha da estrutura de dados que implementa a memória de curto prazo que armazena as regras de proibição: normalmente implementada como uma lista (lista tabu);
4. Número de iterações que um atributo selecionado permanecerá proibido: se a estrutura de dados escolhida foi uma lista, define o tamanho da lista tabu;
5. Escolha dos critério de aspiração: conjunto de regras que permitem a escolha de soluções vizinhas geradas que estariam proibidas naquele momento. Um critério de aspiração pode retirar o status de tabu de uma solução proibida.

Algorithm 1 Busca Tabu

Entrada: $f(.), N(.), |T|, |V|, s$
Saída: $s^\star$
$s^\star \leftarrow s$ {Melhor solução encontrada até então}
$Iter \leftarrow 0$ {Contador do número de iterações}
$MelhorIter \leftarrow 0$ {Iteração mais recente que forneceu $s^\star$}
$T \leftarrow \emptyset$ {Lista Tabu}
while Critério de parada não satisfeito **do**
  $Iter \leftarrow Iter + 1$
  Seja $s' \leftarrow s \oplus m$ o melhor elemento de $V \subseteq N(s)$ tal que o movimento m não seja tabu ($m \notin T$) ou s' atenda a um critério de aspiração
  Atualize a lista tabu T
  $s \leftarrow s'$
  if $f(s) < f(s^\star)$ **then**
    $s^\star \leftarrow s$
    $MelhorIter \leftarrow Iter$
  end if
end while
Retorne $s^\star$

Figura 6 – Pseudocódigo genérico para o Tabu Search

3 Desenvolvimento

O núcleo do sistema foi desenvolvido em Python 2.7, com a interface em Laravel 5.5, HTML 5 e CSS 3.

Para auxiliar na implementação do sistema de alocação de horários foi desenvolvido um conjunto de estruturas de dados que armazenam todas as informações necessárias para o manuseio e consulta eficiente dos dados fornecidos que definem a instância do problema. A eficiência no acesso é muito importante, na medida que os métodos heurísticos utilizados na busca da solução consultam intensivamente os dados do problema e esses dados são muito interdependentes. As estruturas de dados criadas são apresentadas na seção 3.1, e as seções 3.2 e 3.3 discutem como essas estruturas são utilizadas para busca da solução. A seção 3.4 apresenta a interface desenvolvida para o sistema.

3.1 Estrutura de dados

Para abstração do problema de alocação de horários (baseado no ITC - 2007) foi utilizado o diagrama de classes mostrado na figura 7:

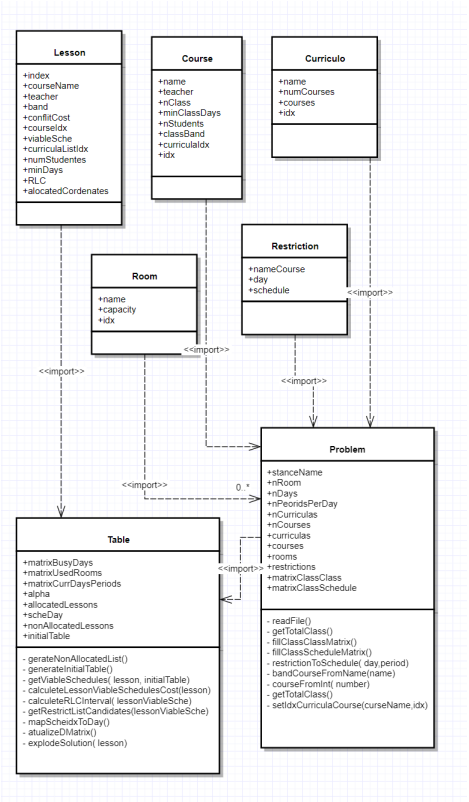

Figura 7 – Diagrama de Classes para o problema

A estrutura geral foi dividida em classes primordiais para a modularização do problema. As classes *Course*, *Lesson*, *Curriculo*, *Restriction* e *Rooms* são utilizadas para representar objetos dos parâmetros fornecidos pelo formato do ITC-2007, que tratam respectivamente: disciplinas, aulas, currículos, restrições (ou indisponibilidades) e salas.

Estas classes são bastantes simples isoladamente e bem parecidas. Sua importância está em armazenar os dados referentes a cada um dos parâmetros da instância, oferecendo diferentes possibilidades de acesso e manipulação facilitados. Uma descrição resumida de cada classe é apresentada a seguir:

- Classe *Course* - Criada para armazenar todos os parâmetros necessários para definir a disciplina. Além dos atributos citados pelo arquivo padrão de uma instância do ITC-2007, essa classe armazena dois outros atributos auxiliares gerados internamente. O atributo textitclassBand é uma tupla (início, fim), onde início e fim são números que denotam respectivamente o primeiro e o último identificadores para as aulas da disciplina. O atributo textitcurriculaIdx é uma lista contendo todos os currículos aos quais a disciplina pertence.

  Considere, por exemplo, uma instância contendo a disciplina X que tem 4 aulas e pertence aos currículos A e B; e a disciplina Y que tem 3 aulas e pertence ao currículo B. Assim, as disciplinas X e Y terão identificadores 0 e 1, respectivamente. Os currículos A e B terão identificadores 0 e 1, respectivamente. Analogamente, cada aula terá um identificador, e o classBand de X será (0,3) e o classBand de Y será (4,6). O curriculaIdx de X será (0, 1) e o curriculaIdx de Y será (1,).

- Classes *Restriction*, Rooms e Curriculo - Contém somente atributos citados pelo arquivo padrão de uma instância do ITC-2007.

- Classe *Lesson* - Criada para armazenar todos os parâmetros necessários para definir a aula (que pertence a uma disciplina). Cada aula da disciplina é representada por um objeto desta classe. Seus atributos são index, que armazena o índice da aula (associado ao classBand de uma disciplina), que é o dado que aparecerá na tabela horário final. Os atributos *courseName* e *courseIdx* , *teacher*, *numStudentes*, *minDays* são respectivamente: o nome do disciplina e o seu índice, nome do professor da disciplina, número de estudantes da disciplina, e o mínimo de dias em que as aulas desta disciplina devem ocorrer. Os dados desses atributos são citados no arquivo padrão de uma instância do ITC-2007.

  Além dos atributos já citados, são gerados outros atributos auxiliares. O atributo viableSche é uma lista de tuplas (sala, horário, custo) que armazena todos os possíveis escalonamentos viáveis para a aula - cada par sala/horário define uma célula na tabela de horários, e o custo de alocar uma aula numa célula é calculado pelo número de restrições fracas violadas. O atributo cost é um inteiro que representa a

dificuldade inicial de alocação da aula, calculada como o número de células disponíveis inicialmente para a alocação da aula. (O sistema vai tentar alocar primeiro as aulas com menos opções de alocação, ou seja, com menor custo. Esse custo está associado também com as indisponibilidades do professor que leciona a disciplina.). Os atributos *band* e *curriculaListIdx* replicam os atributos *classBand* e *curriculaIdx* da disciplina por razões de eficiência. O atributo *RLC*, de *restrict list candidates*, mantém uma lista restrita de candidatos (possíveis células sala e horário), maiores detalhes na seção 3.3 Por fim, o atributo alocatedCordenates é uma lista que armazena as coordenadas da célula sala e horário da tabela onde esta aula foi alocada.

As classes já apresentadas possibilitam a leitura de um arquivo padrão de uma instância do ITC-2007 e a representação dos dados que descrevem o problema. Mais uma classe foi desenvolvida para oferecer suporte para o sistema:

- Classe *Problem* - Criada para representar uma instância do problema. Esta classe contém o método *readFile* e ao ler o arquivo instancia os objetos que representam as disciplinas, salas, currículos e indisponibilidades. Dois atributos auxiliares são também instanciados nessa classe para facilitar o acesso às informações, uma matriz estática aula/aula e uma matriz estática aula e horário:

  A matriz classClass tem como dimensões o número total de aulas pelo número total de aulas. Esta estrutura é criada para realizar consultas rápidas referente às restrições fortes, para avaliar se duas aulas têm conflito entre si, ou se pertencem a mesma disciplina ou mesmo professor. Dadas duas aulas a1 e a2 , adota-se a seguinte convenção:

  1. AA[a1][a2] = 2 se as duas aulas são da mesma disciplina;
  2. AA[a1][a2] = 1 se as duas aulas possuem conflitos entre si, seja por estarem num mesmo currículo ou por serem lecionadas pelo mesmo professor;
  3. AA[a1][a2] = 0 se não há conflitos entre as aulas e elas não pertencem à mesma disciplina.

  A matriz classSchedule tem com dimensões o número total de aulas pelo número total de horários na semana. Essa estrutura também é criada para realizar consultas rápidas, nesse caso referentes às indisponibilidades de horários de cada aula de uma disciplina. Dado os índices de uma aula e um horário, temos duas possibilidades:

  1. AH[a][h] = 1 se a aula a é indisponível no horário h;
  2. AH[a][h] = 0 se a aula a pode ser alocada no horário h.

Vale lembrar que somente esta checagem não garante a disponibilidade de uma aula a em um determinado horário h, sendo necessário olhar na tabela horário corrente se o horário h já não está alocado por outra aula.

3.2 A construção da tabela de horários

Para representar a tabela de horários foi desenvolvida a classe Table que é armazena a solução para a instância do problema descrito na classe Problem. Na classe Table também são implementados métodos utilizados para busca da solução, como o encarregado da construção da tabela inicial, e outros apresentados neste tópico.

- Classe *Table* - Criada para representar uma solução para uma instância do problema descrito na classe Problem. Os atributos restritos desta classe são: *alpha*, *initialTable*, *scheDay*, *allocatedLessons* e *nonAllocatedLessons*. O atributo *initialTable* é uma matriz com dimensões o número total de salas pelo número total de horários na semana, sendo esse último número obtido do produto entre o total de períodos por dia e o total de dias da semana (nos quais serão ministradas aulas). Essa matriz é inicializada pelo método *generateInitialTable* que será apresentado a seguir. O atributo *alpha* é uma constante que influencia o comportamento guloso do algoritmo de construção da tabela inicial. Essa constante assume um valor entre 0 e 1. Os valores próximos de 0 fazem o algoritmo produzir soluções com menor valor da função objetivo, porém menos diversificadas. Rocha afirma que os valores no intervalo [0.01, 0.5] são recomendáveis para produzir soluções com alguma diversidade e qualidade razoável (ROCHA, 2013).

  O atributo *scheDay* é uma lista que tem como tamanho a quantidade de colunas da tabela inicial, ou seja, o número de todas de aulas na semana. Cada elemento dessa lista é uma tupla (dia, horário) associada com a tabela de horários. Essa lista replica a informação contida na tabela inicial para facilitar seu processamento pelos algoritmos heurísticos. Por exemplo, numa tabela de horários com cincos dias e quatro horários por dia temos vinte horários na semana, então, assumindo a contagem dos horários partindo do zero, o elemento *scheDay[16]* representa a aula do primeiro horário do quinto dia.

  O atributo *allocatedLesson* é uma lista com todas as aulas alocadas, e é utilizada para contagem das restrições fracas. É mais fácil e eficiente percorrer uma lista que uma tabela bidimencional. Também esta lista é que preenche o arquivo .json no retorno para a interface gráfica.

  O atributo *nonAllocatedLessons* é uma lista que contém todas as aulas que ainda não foram alocadas na tabela de horários. É utizada como critério para a construção da tabela inicial.

Além dos atributos restritos já citados, um objeto da classe *Table* armazena outros atributos que são utilizados internamente durante as buscas pela solução do problema, bem como mantém métodos para auxiliar nessas buscas:

1. São criadas três matrizes dinâmicas utilizadas para facilitar a contagem das restrições fracas, a saber: *buzyDays, usedRooms e currDaysPeriods*. Sempre que uma aula é alocada ou movida da tabela, essas matrizes devem ser atualizadas:

   - A matriz dinâmica *buzyDays* tem como dimensões o número de disciplinas pelo número de dias. Dados uma disciplina c e um dia d, a matriz *buzyDays*[c][d] retorna a quantidade aulas da disciplina c no dia d. Essa matriz facilita a verificação da restrição fraca de dias mínimos para que um disciplina ocorra na semana.
   - A matriz dinâmica *usedRooms* tem como dimensões o número de disciplinas pelo número de salas. Essa matriz facilita a contagem de quantas salas estão sendo utilizadas por uma dada disciplina e, dada uma disciplina c e uma sala s, podemos facilmente verificar a restrição fraca de estabilidade de sala.
   - A matriz de três dimensões *currDaysPeriods* tem como dimensões o número de currículos, pelo número de dias e pelo número de períodos em um dia. Assim, dado um currículo c, um dia d e um período p, a matriz currDaysPeriods[c][d][p] retorna a quantidade de aulas do currículo c alocadas no dia d e horário p. fraca de aulas isoladas, Se um currículo c possui alguma aula no dia d e horário p, mas *currDaysPeriods*[c][d][$p-1$] = 0 e *currDaysPeriods*[c][d][p + 1] = 0, então o currículo c não tem nenhuma aula no período anterior e nem no próximo, o que gera uma violação de aulas isoladas.

2. O método *generateInitialTable* é encarregado de gerar a tabela inicial, de acordo com o algoritmo proposto por (ROCHA, 2013). Quando uma posição da tabela inicial está preenchida com o valor -1 significa que a sala naquele horário está sem nenhuma aula alocada.

3. O método *gerateNonAllocatedList* é uma função gera e retorna numa lista todos os objetos aulas (classe *Lesson*) segundo cada disciplina. O retorno dessa função é utilizado pelo método generateInitialTable para atribuir um valor ao atributo *nonAllocatedLessons*.

4. O método *mapScheidxToDay* é utilizado para inicializar o atributo *scheDay*, gerando o mapeamento das posições da tabela de horários para as tuplas (dia, horário) da lista.

5. O método *getViableSchedules* é uma função que retorna as possíveis tuplas (sala, horário) onde uma dada aula pode ser alocada. As tuplas que indicam as disponibilidades para a aula são encontradas percorrendo, em conjunto, a tabela de horários e a matriz estática *classSchedule*. A matriz *classSchedule* relaciona aulas com horários indicando as possíveis disponibilidades: a linha da aula é percorrida para encontrar as colunas contendo valor zero, valor zero indica disponibilidade. Em seguida, uma vez encontrado um horário disponível para aquela aula, o método procura uma sala disponível para aquele horário: a coluna do horário em questão na tabela de horários corrente é percorrida procurando as salas contendo valor -1, valor -1 indica sala e horário não alocados. Desse modo, o método encontra todas as tupla (sala, horário) viáveis para se alocar a aula sem violar nenhuma restrição forte.

   Se não houver um espaço sala e horário na tabela corrente para uma dada aula, em uma determinada iteração, pois o algoritmo propõe que todas as aulas serão alocadas sem nenhuma violação das restrições fortes, o método *explodeSolution* será invocado para resolver a situação de impasse.

6. O método *explodeSolution* consiste em retirar uma aula já alocada da tabela de horário para em seguida tentar novamente a alocação da aula que gerou a situação de impasse - o método utiliza uma estratégia de tentativa e erro. A tabela de horários parcial é explorada até obter a construção de uma tabela de horários completa e viável.

   A aula que será retirada da tabela de horários é escolhida aleatoriamente. Uma vez escolhida essa aula é necessário checar se o novo par sala e horário disponível é viável para aquela aula que gerou o impasse. Essa checagem é feita acessando a matriz *matrixClassSchedule* na linha da aula que gerou o impasse, na coluna do horário da aula que foi retirada, sendo viável a alocação se a célula contém zero. Este processo é repetido até que uma tupla (sala, horário) de uma aula retirada não viole nenhuma indisponibilidade para a aula que gerou o impasse. A aula retirada volta para a lista de aulas não alocadas. A aleatoriedade na escolha da aula retirada evitou o problema de ciclagem no trabalho de Rocha (ROCHA, 2013), mas ele trabalhou com instâncias do ITC-2007, onde encontrar uma solução viável de má qualidade não era um problema custoso.

7. O método *calculeteLessonViableSchedulesCost* calcula o custo de cada tupla (sala, horário, custo) obtida pelo método *getViableSchedules* para uma dada aula, segundo a lista de restrições. O custo é computado somando a penalidade de cada violação de restrição fraca que ocorre se a aula for alocada naquele par sala e horário. As violações fracas são checadas com o auxílio das três matrizes *dinâmicas buzyDays, usedRooms* e *currDaysPeriods*, e da capacidade da sala que é fornecida na instância do problema.

8. O método *getRestrictListCandidates* preenche a lista de candidatos restritos da aula (possiveis células sala e horário da tabela de horários) que está sendo alocada (atributo *RLC* do objeto *Lesson*), que será utilizada na construção da tabela inicial. Esse método seleciona todas as tuplas (sala, horário) disponíveis para a aula cujo custo esteja compreendido no intervalo calculado pela função *calculeteRLCInterval*, que acessa o atributo *viableSche* do objeto *Lesson*, percorre a lista de tuplas (sala, horário, custo), e calcula o intervalo (Linf, Lsup). Chamando mínimo e máximo, respectivamente, o menor custo e o maior custo encontrados na lista *viableSche*, o valor de Linf é definido para ser o mínimo, e o valor Lsup é definido para ser o resultado do cálculo (mínimo + alpha * (máximo - mínimo))

9. O método *atualizeDMatrix* percorre e atualiza as matrizes dinâmicas *buzyDays, usedRooms* e *currDaysPeriods*. Após uma aula ter sido alocada ou esta ter passada pelo método *explodeSolution* é necessário atualizar as três matrizes dinâmicas de acordo com as posições da aula alocada ou retirada da tabela inicial, e o método *atualizeDMatrix* é que realiza este trabalho.

3.3 Algoritmo para geração da tabela inicial viável

O algoritmo de geração da tabela initial adotado neste trabalho foi proposto por Rocha (ROCHA, 2013). Partindo de uma tabela de horários vazia, as aulas são acrescentadas uma a uma até que todas estejam alocadas. A escolha da aula que será alocada na tabela é tanto gulosa (para produzir soluções de boa qualidade) quanto aleatória (para produzir soluções diversificadas). As aulas mais difíceis de alocar são alocadas primeiro, avaliando essa dificuldade segundo as indisponibilidades de dias e horários cadastrados na instância.

A construção da tabela inicial é feita de tal forma que seja factível e atenda todas as restrições fortes e se possível com poucas violações das restrições fracas, assim diminuindo o esforço computacional da busca local que será executada posteriormente.

A dificuldade de alocação de uma aula é avaliada de acordo com o número de tuplas (sala, horário) disponíveis para sua alocação na tabela de horários. Quanto menor o número de tuplas disponíveis, maior é considerada a dificuldade de alocação da aula. Esta contagem envolve:

- Contar a quantidade de horários disponíveis (desocupados);
- Retirar os horários em o que o professor da disciplina já leciona alguma aula;
- Retirar os horários em que estão alocadas disciplinas do mesmo currículo;
- Retirar os horários que são indisponíveis para a disciplina segundo as restrições.

Em cada iteração, a aula mais difícil (a que possui menos horários viáveis) é escolhida para ser alocada. Existem diferentes combinações de horários e salas para a alocação. Os custos de todas essas combinações são calculados levando-se em conta as penalizações das restrições fracas. As combinações que possuem horários inviáveis são descartadas. Com base no menor e maior custo de adição de um elemento à solução (c^{min} e c^{max}) é construída a lista restrita de candidatos (LRC). Pertencem a LRC as aulas cujos custos estejam no intervalo [c^{min} , $c^{min} + \alpha(c^{max} - c^{min})$]. Uma aula é escolhida aleatoriamente da LRC e acrescentada à solução.

É possível que em alguns casos, ao escolher uma aula para alocar, não haja um horário que mantenha a viabilidade da solução. Para contornar esta situação foi implementado o procedimento de tentativa e erro, considerando que essas situações são pouco comuns e nem muito custosas de contornar. A figura 8 mostra o algoritmo proposto por (ROCHA, 2013).

Algorithm 1 Algoritmo construtivo para geração de tabela horário inicial

```
1:  T ← 0
2:  ListaNaoAlocadas ← GeraListaNaoAlocadas(A)
3:  OrdenaAulasPorConflitos(ListaNaoAlocadas)
4:  enquanto |ListaNaoAlocadas| < 0 faça
5:      a ← ListaNaoAlocadas[0]
6:      H ← hé viável para h
7:      se H = ∅ então
8:          ExplodeSolução(T, a)
9:          H ← h é viável para h
10:     fim se
11:     para (s, h) ∈ S x H, T[s, h] = ∅ ←
12:         até faça computar o custo de alocação f (a, s, h)
13:     fim para
14:     C^min ← min[ f (a, s, h) : (s, h) ∈ S x H]
15:     C^max ← max[ f (a, s, h) : (s, h) ∈ S x H]
16:     LRC ← [(s, h) ∈ S x H : f(a, s, h) ≤ C^min + α(C^max − C^min)]
17:     Escolhaaleatoriamente(s, h) ∈ LRC
18:     T[s, h] = a
19:     RetiraAula(ListaNaoAlocadas, a)
20:     OrdenaAulasPorConflitos(ListaNaoAlocadas)
21: fim enquanto
```

Figura 8 – Algoritmo para construção da tabela inicial

3.4 Heurísticas e a busca local

Foram implementadas três heurísticas para solucionar o problema de alocação de horários. A heurística *GRASP* é o motor principal que explora o espaço de soluções do problema encontrando as soluções iniciais geradas pelo método guloso aleatório de construção da tabela inicial. As duas outras heurísticas são o *Simulated Annealing* e o *Tabu Search* utilizadas para refinar as soluções encontradas pelo *GRASP* utilizando busca local.

A competência da busca local é, uma vez obtida uma solução inicial, melhorar gradativamente essa solução, gerando novas soluções vizinhas através de pequenas modificações, para selecionar as melhores variações. A busca local busca melhores soluções, minimizando as violações das restrições fracas e mantendo sua viabilidade a cada iteração até que atinja um mínimo global ou local. Cada uma das duas heurísticas têm suas próprias

estratégias para fugir de resultados mínimos locais. O método para gerar vizinhos é de extrema importância na busca local e influencia diretamente o resultado das heurísticas de busca. Este trabalho contempla duas estratégias para geração de soluções vizinhas: mover aulas e trocar aulas.

- Mover aulas - Se a tabela de horários corrente possui células sem aula alocada, podemos mover alguma aula já alocada para uma dessas células. Tanto a célula vazia quanto a aula que será movida para ela são escolhidas aleatoriamente. A aula selecionada para o movimento não pode violar nenhuma restrição forte se ocupar a célula vazia. Caso o movimento escolhido não seja viável, são escolhidos aleatoriamente uma nova célula vazia e uma nova aula para mover.

- Trocar aulas - A segunda estratégia faz um swap entre duas aulas já alocadas. Consiste em escolher aleatoriamente duas aulas entre as aulas já alocadas para realizar uma troca entre seus lugares alocados na tabela horário. A troca de aulas não pode violar nenhuma restrição forte. Caso a troca escolhida não seja viável, ou seja, alguma das duas aulas viola uma restrição forte com a troca, então uma das dessas aulas é escolhida aleatoriamente para ser pivô, e uma outra nova aula já alocadas é escolhida aleatoriamente para a realização da troca.

Os algoritmos que implementam as heurísticas apresentadas são clássicos e estão apresentados no referencial teórico.

3.5 Interface

Para o cadastro das instâncias foi desenvolvida uma interface gráfica para a entrada e saída de dados do sistema, assumindo que os dados que definem uma instância do problema seguem o formato especificado para a terceira formulação proposta pelo ITC (ITC - 2007). Uma vez cadastrados os dados de uma instância, a interface permite gerar um arquivo .ctt que contém todos os dados relevantes que definem a instância. O arquivo .ctt depois pode ser lido pelo motor do sistema, desenvolvido em python, que vai interpretar os dados da entrada, calcular a solução para o problema e retornar o resultado num arquivo .json que será novamente tratado pela interface.

A experiência do usuário com a interface inicia na tela de login, onde ele pode realizar o cadastro de um novo usuário, resgatar sua senha e entrar no sistema, se já tiver um cadastro. A figura 9 ilustra a tela de *login*.

Uma vez logado no sistema, aparece a tela inicial para cadastro de informações gerais de uma instância. No lado esquerdo da tela inicial aparece um menu com todas as instâncias cadastradas, e quando o usuário clica na instância desejada o sistema mostra

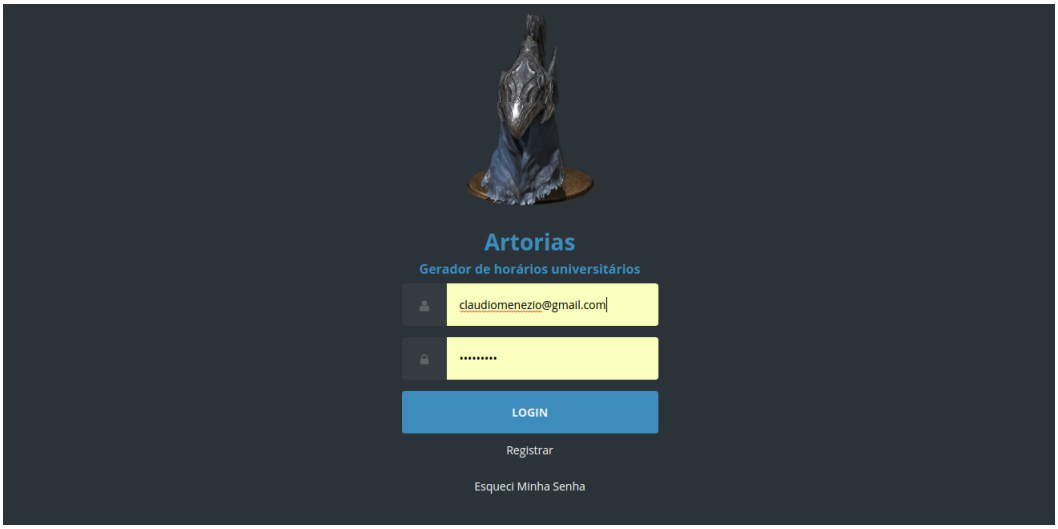

Figura 9 – Tela de login

as telas relativas àquela instância. A figura 10 mostra, ao lado direito da tela, o label "Instâncias" seguido abaixo de duas caixas: a primeira com o label "Criar nova instância", e a segunda com o label "Instâncias cadastradas".

A caixa com o label "Criar nova instância" permite cadastrar os dados gerais de uma nova instância de acordo com o formato padrão do ITC - 2007. Os campos para cadastro são: nome, número de disciplinas, número de salas, o número de períodos por dia, número de currículos e o número de restrições. Abaixo existe um botão para salvar (cadastrar) estes dados.

Figura 10 – Tela inicial e de cadastro de dados gerais de uma instância

A caixa com o label "Instâncias cadastradas" contém todas as instâncias já cadastradas, com os seus dados apresentados, sendo possível editar ou deletar uma instância. No campo nome pode-se clicar no nome da instância para entrar em suas dependências.

Ao entrar nas dependências de uma instância temos uma nova tela com cinco abas que são: disciplinas, salas, curriculas, restrições e resultados. As quatro primeiras abas contém duas caixas uma para um novo cadastro e a outra para mostrar o que já foi cadastrado, sendo possível editar ou deletar entradas no cadastro. A figura 11 mostra a aba de cadastro das disciplinas, a primeira caixa com o label "Adicionar disciplinas" que contém os campos para cadastro de uma disciplina que são: nome, nome do professor, número de aulas por semana, mínimo de dias para que a disciplina ser lecionada na semana e o número de alunos. Abaixo tem o botão para salvar esta nova disciplina cadastrada. Na caixa com o label "Disciplinas cadastradas" temos todas as disciplinas já cadastradas com os botões de editar e deletar.

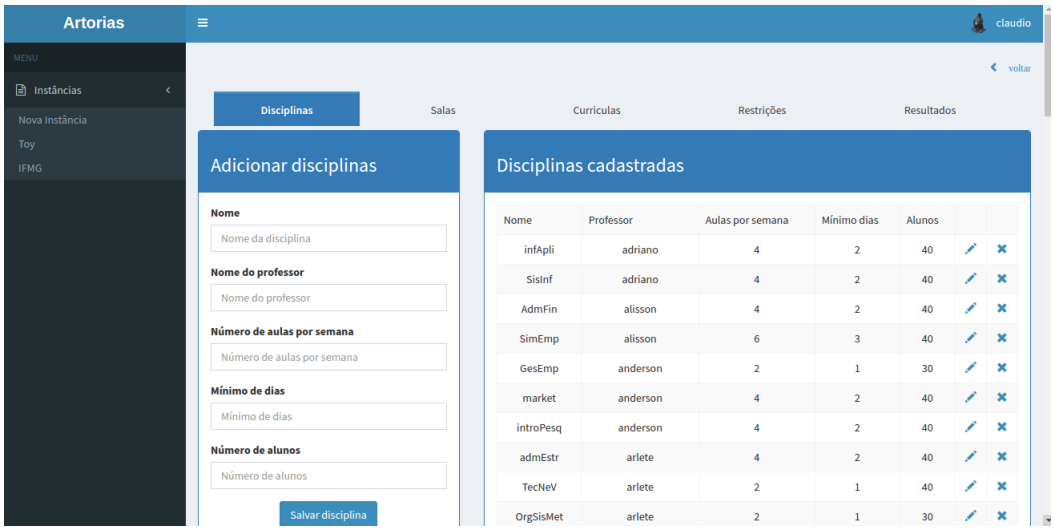

Figura 11 – Aba de cadastro das disciplinas de uma instância

A próxima aba é para o cadastro das salas da instância. A figura 12 mostra a aba de cadastro das salas, a primeira caixa com o label "Adicionar salas" que contém os campos para cadastro de uma sala que são: nome e lotação da sala. Abaixo tem o botão para salvar esta nova sala cadastrada. Na caixa com o label "Salas cadastradas" temos todas as salas já cadastradas com os botões de editar e deletar.

A próxima aba é para o cadastro dos currículos da instância. A figura 13 mostra a aba de cadastro dos currículos, a primeira caixa com o label "Adicionar curriculas" que contém os campos para cadastro de um currículos que são: nome da curricula, número de curriculas e as disciplinas pertencente a este currículo. Abaixo tem o botão para salvar esta nova curricula cadastrada. Na caixa com o label "Curriculas cadastradas" temos todas os currículos já cadastrados com os botões de editar e deletar.

A próxima aba é para o cadastro das restrições da instância. A figura 14 mostra a aba de cadastro das restrições ou indisponibilidades, a primeira caixa com o label "Adicionar restrições" que contém os campos para cadastro das restrições que são: seleciona uma disciplina já cadastrada em seguida o dia e o período do dia com a indisponibilidade.

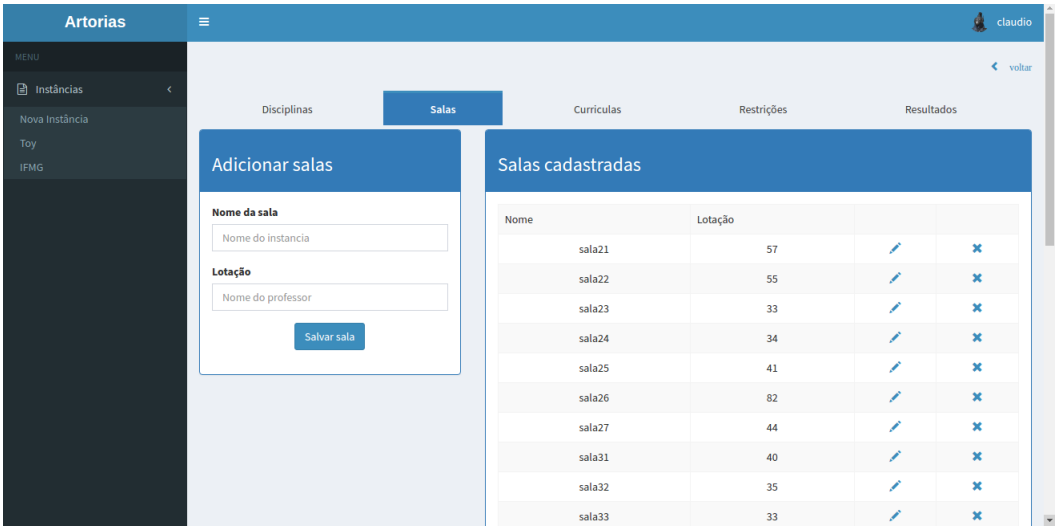

Figura 12 – Aba de cadastro das salas de uma instância

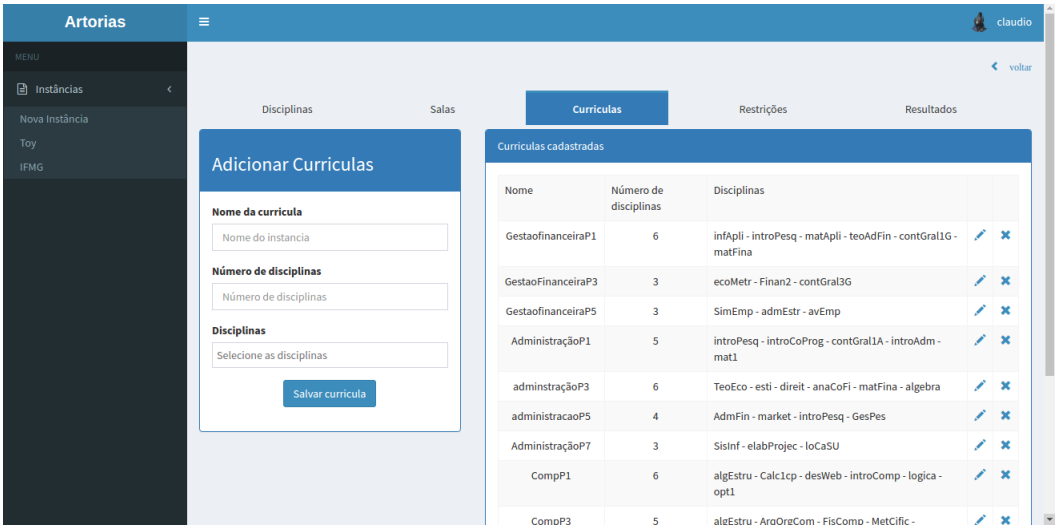

Figura 13 – Aba de cadastro dos currículos de uma instância

Abaixo tem o botão para salvar esta nova restrição cadastrada. Na caixa com o label "Restrições cadastradas" temos todas as restrições já cadastrados com os botões de editar e deletar.

As abas citadas acima e a tela inicial contém o botão de edição em seus dados já cadastrados, assim levando a uma nova tela onde se possa editar os dados já cadastrados em cada aba e da tela inicial. A figura 15 demonstra uma tela de edição, neste caso editando um currículo como exemplo.

Assim pode-se editar qualquer campo já cadastrado e salvar esta edição com o botão abaixo e voltar ao cadastro da instância.

Na última aba resultados, temos a renderização dos resultados obtidos para solução da instância. A figura 16 mostra as tabela horários para cada currículo cadastrado. Nas

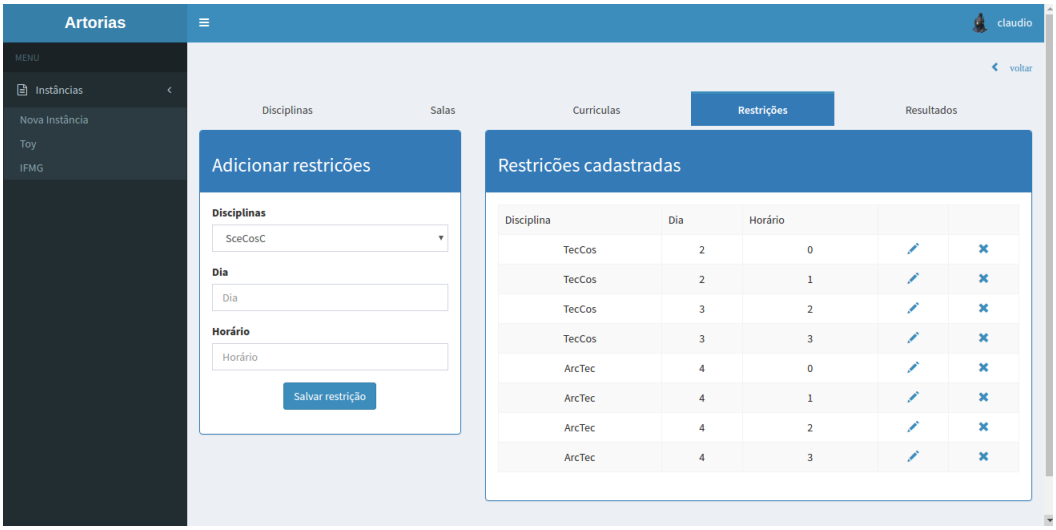

Figura 14 – Aba de cadastro das restrições de uma instância

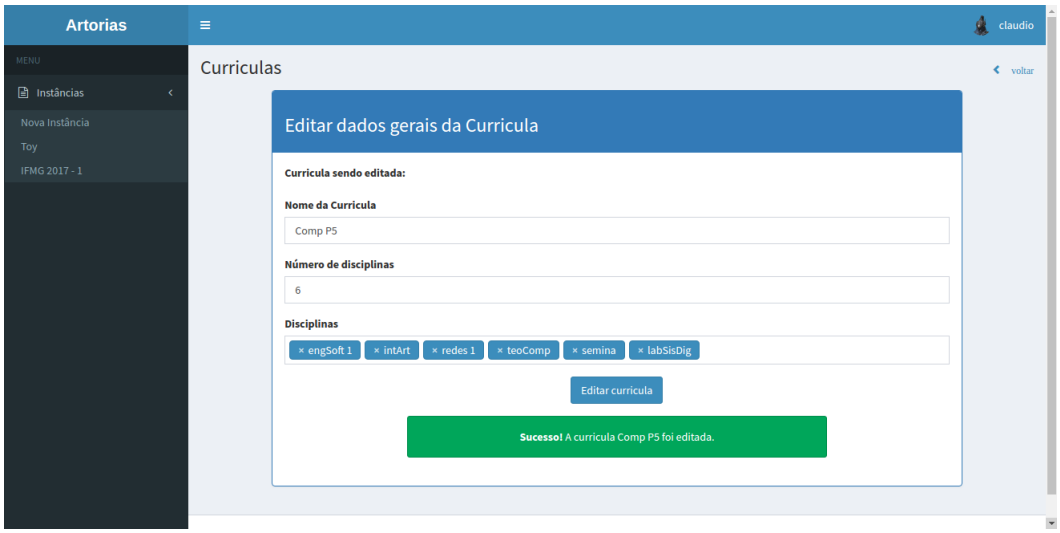

Figura 15 – Tela de edição

linhas temos as salas e nas colunas temos os dias com seus respectivos números dos horários neste dia. Nas células contém o nome da disciplina a ser lecionada. Este são gerados e salvos em um arquivo .json de modo que são persistentes.

3.5.1 Persistência do cadastro da instância

Para que um cadastro se mantivesse salvo no sistema foi desenvolvido um banco de dados, a figura 17 mostra as tabelas com suas dependências e relacionamentos. São duas tabelas relacionadas ao usuário e sua autenticação.

As tabelas salas, disciplina, restrições, curriculas estão amarradas a tabela instância por meio de uma chave estrangeira da instância, relacionamento um para n. As restrições têm relacionamento n para 1 com as disciplinas, e curriculas e disciplinas têm um

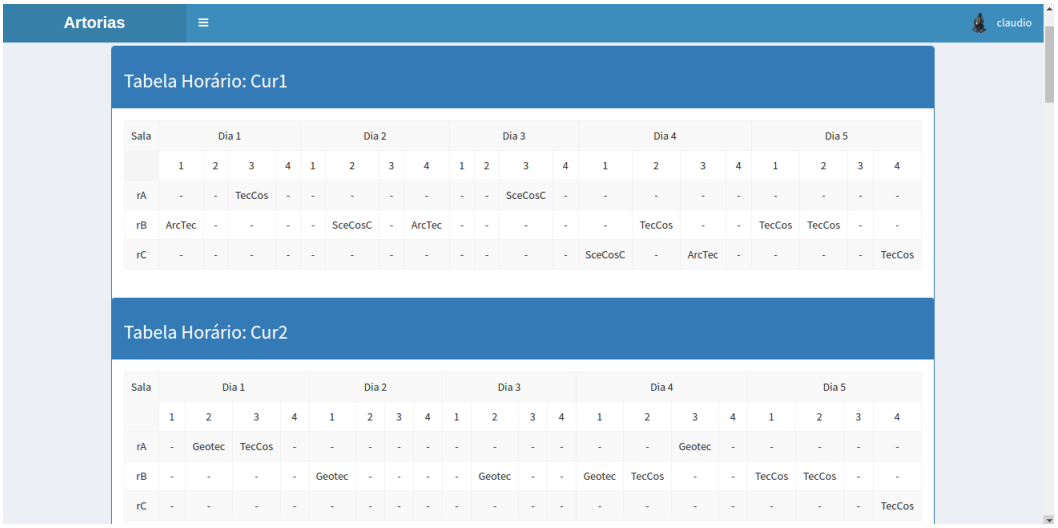

Figura 16 – Aba com tabelas horários por currículos

relacionamento n para n necessitando de uma tabela auxiliar tabela curricula_disciplina.

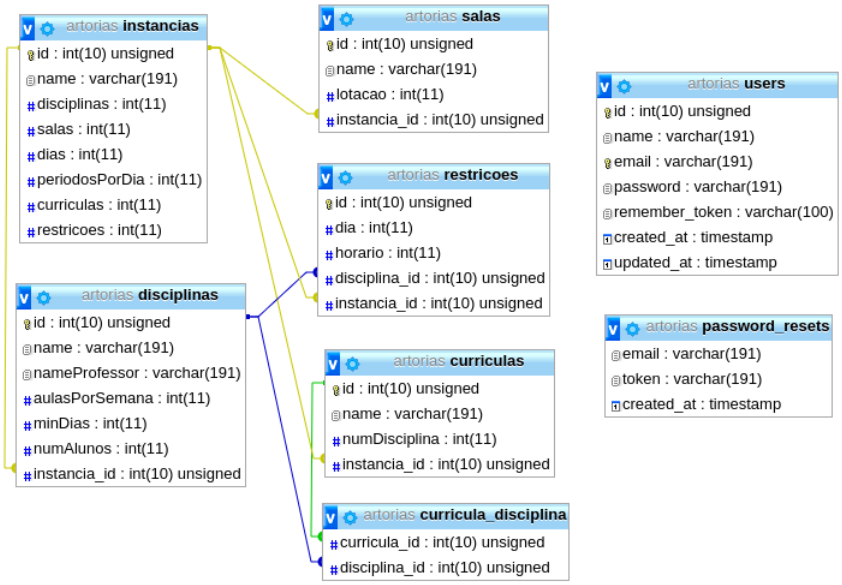

Figura 17 – Diagrama de tabelas do banco de dados

3.5.2 Modularização do sistema

O sistema é composto de duas partes totalmente desacopladas sendo a interface desenvolvida em *php* e o *solver* desenvolvido em *python*. Para que estas parte se comuniquem é utilizado como intermediador um *framework php* para aplicações *web* chamado *Symfony*. O módulo de interface importa a biblioteca do *Symfony* e simplesmente instancia um objeto *process*, que recebe como parâmetro uma *string* com a linha de comando contendo os parâmetros para executar a aplicação solver quando o usuário ordena a geração da

tabela de horários. O uso do *Symfony* independente da linguagem em que a aplicação chamada foi desenvolvida.

Assim, ao final da execução, a aplicação solver vai gerar um arquivo .json escrito na saída padrão, que será capturado pelo *Symphony* e exibido para o usuário pelo módulo interface. A aplicação solver vai utilizar a biblioteca *json* do *python*, através do método *dump*, para serializar o objeto que contém as informações relativas à tabela de horários apresentada como solução ao problema.

3.5.3 Instância do IFMG - *Campus* Formiga

Para gerar uma instância no formato do ITC - 2007 de um período ímpar do IFMG - *Campus* Formiga, foi realizada uma pesquisa junto a secretaria do campus. Foi apurado nesta pesquisa que o campus possui 96 disciplinas e 19 curriculas pertencente aos cursos de Administração, Ciência da Computação, Engenharia Elétrica e Gestão Financeira. As 27 salas são distribuídas em três prédios e são 13 períodos por dia contando os três turnos manhã, tarde e noite. As restrições foram obtidas junto a secretaria acordado o sigilo completo destas informações.

4 Materiais e métodos

O solver foi excutado em um Dell Inspirion 14r processador i5 quarta geração, 6 gigas de memoria ram e hd de 1 terabyte. Para o desenvolvimento do solver foi utilizado a linguagem python em sua versão 2.7 . Para o front - end, onde se pode cadastrar as instâncias foi utilizado o *framework* Laravel na vesão 5.5 juntamente com *Html* 5 e *CSS* 3.

As instâncias utilizadas são as do ITC - 2007, sendo estas as *comp.ctt* que vão do número 1 a 6. Para testes em desenvolvimeto foi utilazada a instância *toy.ctt*.

5 Conclusão

O trabalho cumpriu os objetivos de implementar as heurísticas propostas para solucionar o problema de alocação de horários numa universidade. Foi implementado o algoritmo *GRASP* para exploração do espaço de soluções, utilizando o *Simulated Annealing* para busca local, seguindo o modelo proposto por Rocha (ROCHA, 2013). Como método alternativo, também foi experimentado o algoritmo *GRASP* utilizando o *Tabu search* para busca local. Nos dois casos, a função objetivo definida e adotada neste trabalho foi o somatório das penalidades das restrições fortes e fracas definidas pelo ITC-2007, com as heurísticas buscando sua minimização. Todos os resultados obtidos tem como resultado do somatório das penalidades das restrições fortes o valor zero, isto é, nenhuma restrição forte foi violada.

A minimização da função objetivo perante as restrições fracas não foi satisfatória neste trabalho, comparando com os resultados obtidos no ITC-2007. A avaliação da implementação do sistema foi efetuada empiricamente, comparando os resultado da execução de várias instâncias do *benchmark* do ITC-2007 com os resultados obtidos pelas equipes que participaram do ITC. Os resultados obtidos neste trabalho ficaram bem aquém do esperado e bem distantes dos valores obtidos pelos dez primeiros participantes do ITC-2007, citados por Rocha (ROCHA, 2013).

Partindo do valor da função objetivo obtido de tabelas de horários iniciais geradas pelo *GRASP*, realizando testes informais obtivemos um decréscimo no valor da função objetivo de no máximo 20% utilizando o *Simulated Annealing* para busca local. Os resultados obtidos com a utilização do *Tabu search* foram ainda piores, quase não melhorando os resultados. Nossos resultados foram visivelmente piores que os participantes do ITC-2007. Tendo em vista essa diferença marcante, julgamos desnecessário efetuar avaliações estatísticas mais precisas para constatar o problema.

As instâncias *comp.ctt* foram executadas 1000 vezes cada, com 500 vizinhos na busca local. O tempo médio de execução para das instancias 17735,37 segundos. A tabela 1 mostra a média dos resultados obtidos para função objetivo.

Para a instância real do real do IFMG - *Campus* Formiga não foi possível executar o sistema, devido ao número absurdo de restrições fortes que, possivelmente, até tornam o problema insolúvel. O algoritmo de geração da tabela inicial entrou em ciclagem, o que nunca havia acontecido em nenhuma instância do ITC. Todavia, a eliminação de algumas restrições propostas permitiu ao sistema encontrar uma solução viável.

Tabela 1 – Resultados das heurísticas

Instância	Grasp - Simulated Annealing	Grasp - Tabu Search
comp1	485	1037
comp2	784	1465
comp3	655	1335
comp4	845	1956
comp5	1005	1598
comp6	512	1732

6 Trabalhos futuros

O trabalho desenvolvido envolveu o aprendizado de diversas disciplinas cursadas na graduação atendendo a proposta do TCC. Espera-se que este trabalho seja útil para a sociedade acadêmica e algum dia venha a ser utilizado para a geração de horários no IFMG. Para que isso aconteça da melhor forma, recomendamos que os seguintes trabalhos futuros sejam realizados:

- Implementar o path-relink (ROCHA, 2013) para manter uma memória dos melhores resultados obtidos com as heurística *GRASP* e *tabu serach*, e explorar a conexão entre soluções iniciais e elites;
- Fazer estudos estatísticos para melhorar a heurística *tabu search*, possivelmente desenvolvendo critérios de seleção não aleatória para a escolha dos movimentos e trocas para geração de vizinhos melhores;
- Geração de informações para o usuário do sistema de quais restrições estão dificultando mais a obtenção de boas soluções: indicar as piores restrições e sugerir estratégias;
- Tornar a interface mais amigável e disponível para os professores na *web*;

Referências

CESCHIA S.; GASPERO, L. D. S. A. Design,engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. In: *Computers & Operations Research, v. 39, n. 7, p.1615–1624* . [S.l.: s.n.], 2011. Citado na página 17.

ELLOUMI, A. A tabu search procedure for course timetabling at a tunisian university. In: *Proceedings of the 7th PATAT Conference, 2008. [S.l.: s.n.].* [S.l.: s.n.], 2008. Citado na página 17.

FEO T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. journal of global optimization. In: *Journal of Global Optimization, [S.I.] n. 2, p. 109-133.* [S.l.: s.n.], 1995. Citado na página 21.

GLOVER F. E LAGUNA, M. Tabu search. *Kluwer Academic Publishers, Boston.*, 1997. Citado 2 vezes nas páginas 17 e 24.

KIRKPATRICK. Optimization by simulated annealing. science. In: *New Series, Vol. 220, No. 4598.* [S.l.: s.n.], 1983. Citado na página 23.

ROCHA, W. d. S. *Algoritmo GRASP para o Problema de Tabela-horário de Universidades.* Dissertação (Mestrado) — Departamento de Informática, Universidade Federal do Espírito Santo, Vitória., 2013. Citado 9 vezes nas páginas 17, 18, 28, 29, 30, 31, 32, 43 e 45.

SCHAERF, A. A survey of automated timetabling. In: *ARTIFICIAL INTELLIGENCE REVIEW, v. 13, p. 87–127.* [S.l.: s.n.], 1995. Citado 2 vezes nas páginas 17 e 19.

SOUZA M. J. F.; MACULAN, N. O. L. S. Metaheuristics. In: _____. [S.l.]: Norwell, MA, USA: Kluwer Academic Publishers, 2004. cap. A GRASP-tabu search algorithm for solving school timetabling problems, p. 659–672. Citado na página 17.