Programação de Horários e Alocação de Salas de Aula no IME/UERJ com Simulated Annealing e LAHC

Daniel Henrique da Silva Costa[1], **Igor Machado Coelho**[2], **Paulo Eustáquio Duarte Pinto**[3]
Instituto de Matemática e Estatística - Universidade do Estado do Rio de Janeiro (UERJ)
Rua São Francisco Xavier, 254, CEP 20550-900, Rio de Janeiro, RJ
daniel.henrique@ime.uerj.br[1], igor.machado@ime.uerj.br[2],
pauloedp@ime.uerj.br[3]

RESUMO

A maioria das instituições de ensino pelo mundo possuem problemas semelhantes no início de cada semestre letivo. Entre estes problemas, destacam-se a programação de horários e a alocação de salas de aulas, que devem respeitar restrições, fornecer recursos necessários e maximizar preferências de alunos e professores. Atualmente, na maioria das instituições de ensino superior do Brasil, este processo é realizado manualmente, demandando dias de trabalho, e nem sempre gerando uma solução satisfatória. Este trabalho apresenta um estudo de caso do curso de Ciência da Computação do IME/UERJ - composto por 39 salas de aula, 37 professores, 38 disciplinas, 53 turmas e 116 aulas - e a solução dos seus problemas de programação de horários e de alocação de salas de aula de forma simultânea, utilizando as técnicas Simulated Annealing e Late Acceptance Hill Climbing. Nos experimentos, foram obtidos resultados que custam menos de 25% do custo total da solução manual referencial.

PALAVRAS CHAVE. problema de programação de horários, problema de alocação de salas, metaheurísticas

Metaheurísticas, Otimização Combinatória

ABSTRACT

Most of the world's educational institutions have similar problems at the beginning of each school term. Among these problems, features the course timetabling and classroom assignment, which must respect constraints, provide necessary resources and maximize the preferences of students and teachers. Currently, in most higher education institutions of Brazil, this process is carried out manually, requiring days of work, and not always generating a satisfactory solution. This work presents a case study of the Computer Science course of the IME/UERJ - composed of 39 classrooms, 37 teachers, 38 disciplines, 53 classes and 116 lessons - and the solution of its course timetabling and classroom assignment problems simultaneously, using Simulated Annealing and LAHC techniques. In the experiments, were found out results that cost less than 25% of the total cost of the reference manual solution.

KEYWORDS. course timetabling, classroom assignment, metaheuristics

Metaheuristics, Combinatorial Optimization

1. Introdução

Em todo início de semestre letivo, o Instituto de Matemática e Estatística da Universidade do Estado do Rio de Janeiro (IME/UERJ), responsável pelo curso de graduação em Ciência da Computação, passa por dois problemas enfrentados praticamente por todas as instituições de ensino pelo mundo: o Problema de Programação de Horário (PPH) e o Problema de Alocação de Salas de Aula (PAS). Tais problemas são comumente definidos como *Problema de Timetabling Educacional* ou como subproblemas dele [Carter e Laporte, 1998; Schaerf, 1999; Burke et al., 2004; Wren, 1996; Bardadym, 1996; Dimopoulou e Miliotis, 2001; Souza e Santos, 2007].

Segundo Wren [1996], estes problemas podem ser definidos como a alocação, sujeita a restrições, de recursos a objetos que estão sendo colocados no espaço-tempo, de modo que se satisfaça o máximo possível um conjunto de objetivos desejáveis.

Estas restrições são normalmente classificadas como restrições graves e restrições leves. As graves são as restrições que uma solução factível deve obrigatoriamente satisfazer. Uma solução que não atenda a todas as restrições graves deve ser rejeitada. As restrições leves são menos importantes do que as graves, e normalmente é impossível obter uma solução que não quebre ao menos uma delas. Então, o que deve ser feito é minimizar estas quebras, de acordo com uma função de penalidade. Algumas restrições leves são mais importantes do que outras, e podem ser especificadas com um valor de prioridade [Burke et al., 1997; Abdullah, 2006].

Os problemas mencionados, atualmente, são solucionados de forma manual tanto no IME/UERJ quanto em muitas outras instituições pelo mundo, o que causa um grande esforço e demanda dias de trabalho manual. Ainda assim, nem sempre é gerada uma solução satisfatória ao fim do processo, em relação às preferências de alunos e professores e a outros aspectos, promovendo possíveis incômodos, como constante mudança de professores em uma mesma sala de aula, grande repasse das chaves das salas entre os professores, desperdício de tempo de aula com arrumação do material de aula pelos professores, entre outros. Além do mais, a solução manual também é restringida pelo crescimento do curso ou instituição, isto é, não é facilmente escalável.

Normalmente, estes problemas são resolvidos separadamente (tanto manualmente, quanto computacionalmente), sendo primeiramente resolvido o PPH, e logo após, já com os horários fixados, sendo resolvido o PAS. Esta separação é feita para diminuir a complexidade do problema. Porém, isto também diminui o espaço de busca de soluções, onde poderia ser encontrada uma solução melhor. Neste trabalho, os dois problemas são resolvidos simultaneamente, aumentando o espaço de busca e consequentemente aumentando a capacidade de encontrar melhores soluções.

Dadas todas estas limitações, a automação deste problema é alvo de inúmeras pesquisas e projetos. Esta não é uma tarefa simples, pois este problema é NP-dificil [Even et al., 1975; Carter e Tovey, 1992], e portanto, em casos reais, é muito difícil ou até mesmo impossível de ser solucionado de forma exata. Por este motivo, normalmente utilizam-se técnicas heurísticas para sua solução, o que resulta em soluções geralmente satisfatórias em um tempo razoável. Dentre as estratégias heurísticas existentes, utilizaremos neste trabalho as metaheurísticas Simulated Annealing (SA) e Late Acceptance Hill-Climbing (LAHC).

O restante deste trabalho está organizado como segue: a Seção 2 detalhará o caso de uso do curso de Ciência da Computação do IME/UERJ; a Seção 3 apresentará a modelagem utilizada; a Seção 4 apresentará a implementação dos algoritmos utilizados; a Seção 5 apresentará as experimentações realizadas e resultados obtidos;e, por último, a Seção 6 apresentará as conclusões obtidas.

2. Problema Abordado

A UERJ é uma das maiores instituições de ensino superior do Brasil e do mundo, e assim como todas as outras IESs, também enfrenta os problemas de timetabling. Este processo ainda é realizado manualmente na maioria dos departamentos e cursos da Universidade, demandando tempo e esforço, e nem sempre gerando um resultado satisfatório.

Em prol da melhoria desse processo, este trabalho visa resolver este problema de forma automatizada, obtendo resultados mais satisfatórios, e com economia de tempo.

Neste trabalho, o problema será restringido ao curso de Ciência da Computação do IME/UERJ, que, no cenário do primeiro semestre de 2016, possui 39 salas de aulas (entre salas comuns e laboratórios), 37 professores, 38 disciplinas, 53 turmas e 116 aulas, que devem ser distribuídas em intervalos de tempo, nos quais existem 18 em dias de semana e 6 aos sábados, totalizando 3456 intervalos de tempo por semana (96 intervalos por sala).

2.1. Objetivo

O objetivo da aplicação consiste em alocar as aulas em salas, obedecendo um conjunto de restrições, de forma que não haja conflitos que gerem a infactibilidade da solução.

Entre estas restrições, temos a necessidade da sala atender todos os recursos necessários para a realização da aula, como capacidade, computadores, projetores, etc.

Outra restrição importante é que as aulas devem ser alocadas em horários que estejam de acordo com um conjunto de horários pré-definidos para tais aulas.

Alguns dos requisitos buscados para a aplicação foram os seguintes:

- Tratar de forma integrada o **Problema de Programação de Horários** com o **Problema de Alocação de Salas**.
- Usar o conceito de flexibilidade de recursos, onde a necessidade de um recurso para uma aula pode não ser sempre **obrigatório**, mas também **altamente necessário**, **necessário** ou **dispensável**.
- Considerar que aulas podem ter seus professores especificados, assim como proibir que um mesmo professor dê duas aulas ao mesmo tempo.
- Considerar que disciplinas pertecem a um período recomendado de acordo com o programa, e, assim, proibir que aulas de disciplinas do mesmo período ocorram simultaneamente, pois senão, não seria possível assistir aulas de ambas disciplinas em um mesmo semestre, o que vai contra o próprio programa.
- Considerar que restrições leves possuem peso de importância.

Para este cenário, são relevantes as seguintes características:

- Disciplinas têm N turmas.
- Disciplinas pertecem a cursos, onde um deles é o curso preferencial, e neste curso, a disciplina possui um período recomendado, de acordo com um programa de formação.
- Turmas têm M aulas.
- Aulas possuem tempos, que são o número de intervalos de tempo que elas precisam em sequência.
- Aulas possuem um conjunto de possíveis horários, devendo ser alocadas em qualquer um destes.
- Aulas podem ter salas prefixadas.
- Aulas podem ter professores prefixados.
- Aulas pertencem exclusivamente a uma turma, porém, podem ser agregadas com aulas de outra turma, como se fossem uma única aula. Exemplo: Turma 1 de Física I para Computação com Turma 2 de Física I para Matemática podem ter suas aulas teóricas agregadas, sendo lecionadas em uma sala maior, por um só professor e em um único horário.

- Salas possuem capacidade para A alunos e possuem R recursos, como quadro branco, projetor, computadores e outros.
- Aulas precisam de salas com capacidade para C alunos e com F recursos, como quadro branco, projetor, computadores e outros.

2.2. Restrições

As restrições, levadas em conta em todos os problemas de *timetabling*, são essenciais para a qualidade de uma solução. Existem dois tipos de restrições, as **restrições graves**, que devem sempre ser respeitadas para se ter uma solução factível, e as **restrições leves**, que devem ser respeitadas sempre que possível.

As **restrições leves** possuem penalidades variáveis que definem quais são mais preferíveis em relação às outras. Normalmente, não é possível satisfazer todas as restrições leves, pois algumas se contradizem com outras, como por exemplo, o fato de ser preferível que aulas de disciplinas do mesmo período se mantenham na mesma sala e que aulas de um mesmo professor se mantenham na mesma sala, gerando uma contradição sempre que um professor der aula para disciplinas de períodos diferentes.

As **restrições graves** deste cenário são:

G1 Todas as aulas devem ser alocadas.

G2 Uma sala só pode ter uma aula por vez (uma agregação é considerada como uma única aula).

G3 Aula deve ser alocada em sala com capacidade maior ou igual ao total de alunos que a assistará.

G4 Duas aulas de uma mesma turma não podem ocorrer simultaneamente.

G5 Duas aulas de disciplinas de um mesmo período, segundo o programa, não podem ocorrer simultaneamente.

G6 Duas aulas lecionadas por um mesmo professor não podem ocorrer simultaneamente.

G7 Todos os recursos obrigatórios devem ser atendidos.

G8 Aula pode ter uma sala predefinida.

G9 Uma sala não pode ter aulas fora de seu horário de funcionamento.

G10 Aula deve ser alocada em um dos horários predefinidos como possíveis para a realização da mesma.

As **restrições leves** deste cenário são:

L1 Aulas devem, preferencialmente, ser alocadas em salas com capacidade igual ou o mais próximo possível da sua necessidade.

L2 Aulas devem, preferencialmente, ser alocadas em salas com recursos iguais ou o mais próximo possível da sua necessidade.

L3 Aulas de disciplinas do mesmo período, segundo o programa, devem, preferencialmente, ser alocadas na mesma sala ou em salas próximas, evitando o deslocamento em grandes distâncias.

L4 Recursos não-obrigatórios devem, preferencialmente, ser atendidos.

L4.1 Recurso altamente necessário deve, preferencialmente, ser atendido.

L4.2 Recursos necessário deve, preferencialmente, ser atendido.

L4.3 Recursos dispensável deve, preferencialmente, ser atendido.

L5 Aulas lecionadas por um mesmo professor devem, preferencialmente, ser alocadas na mesma sala, evitando desperdício de tempo com organização de material e com devoluções de chaves.

L6 Deve, preferencialmente, ser evitada a alocação de aulas nos dois últimos intervalos de tempo da noite.

L7 Devem, sempre que possível, ser deixados intervalos de tempo livre nas salas, para ser realizada a limpeza das mesmas.

3. Modelagem

Uma parte primordial na solução dos problema é como este é representado no algoritmo. Esta representação por si só pode automaticamente solucionar algumas restrições e mudar consideravelmente a qualidade do processamento e da otimização.

3.1. Estruturas de Dados

Entre diversas estruturas auxiliares, o algoritmo é composto principalmente por uma estrutura bidimensional, que representa o quadro **Salas x Intervalos de Tempo** e uma estrutura linear onde são armazenadas as informações de cada sala. Para armazenar estas informações, existem classes que as organizam de forma estruturada.

A estrutura **Salas x Intervalos de Tempo** é uma estrutura bidimensional onde as colunas representam as salas (além das salas reais, podem existir salas virtuais que auxiliam nos movimentos de troca) e as linhas representam os intervalos de tempo existentes. Estas estruturas são preenchidas com inteiros que representam uma informação, ou com uma referência a um objeto da classe *Agregacao* que está alocada naquela sala e naquele horário específico.

Os possíveis inteiros que representam informações são:

- **0** - Espaço disponível para ser alocado.
- **-1** - Espaço já alocado para agregação definida linhas acima, na mesma coluna. Por exemplo, se a Agregação X tem 3 tempos e está alocada na posição 20x32, a posição 20x33 e 20x34 estará preenchida com -1, pois são tempos da Agregação X, que começa em 20x32 e vai até 20x34.
- **-2** - Sala está indisponível neste horário.

Esta estrutura, por si só, já resolve duas restrições graves, que são *(G2) Uma sala só pode ter uma aula por vez*, visto que não se pode colocar mais de uma agregação em um único espaço da estrutura e *(G9) Uma sala não pode ter aulas fora do seu horário de funcionamento*, visto que não se pode mover uma agregação para um espaço preenchido com o código -2.

Vale ressaltar que as agregações alocadas nestes espaços contêm uma referência a um objeto *Agregacao* completo, com todas as suas propriedades, tais como seus possíveis horários, as aulas que a compõe, total de alunos, sala na qual esta agregação deve ser obrigatoriamente alocada (caso seja definida), entre outras informações. Estas informações são utizadas para verificar o valor daquela agregação naquela sala e horário, a partir da apuração do atendimento às restrições do problema, utilizando uma função de cálculo de valor da solução.

Além da estrutura detalhada anteriormente, temos também uma estrutura linear que armazena os detalhes de cada sala, utilizando os mesmos índices das salas na estrutura anterior. Por exemplo, supondo-se que a Sala 3 seja o índice 2 (na dimensão de salas) na estrutura Salas x Intervalos de Tempo, a mesma também terá o índice 2 na estrutura de salas.

3.2. Movimentos de troca

O processo de busca por uma solução cada vez melhor e mais satisfatória consiste basicamente de começar a partir de uma solução inicial, que será a primeira solução atual, gerar soluções vizinhas a partir de um movimento de troca na solução atual, verificar se esta solução será aceita como a nova solução atual, e realizar este processo iterativamente até atingir um critério de parada.

Visto esse processo, tais movimentos de troca são parte essencial para a solução do problema de forma consistente e de qualidade, e devem ser tratados com muita atenção.

Os movimentos de troca utilizados neste trabalho são:

- **Alocação por movimentação de horário** - move-se uma agregação para um dos seus outros horários possíveis (caso exista outro horário possível além do atual), alterando ou não sua sala. Em termos de estrutura de dados, é realizado o movimento vertical para troca de horário, e possivelmente, uma movimentação horizontal para troca de sala, sendo o destino um espaço necessariamente vazio.

- **Alocação por movimentação de sala** - move-se uma agregação para uma outra sala aleatória, caso a mesma esteja livre para alocação, mantendo esta agregação no mesmo horário. Em termos de estrutura de dados, é realizado apenas o movimento horizontal, isto é, a troca de uma coluna para outra, sendo o destino um espaço necessariamente vazio.

- **Troca por movimentação de sala** - inverte-se a posição entre duas agregações que iniciam-se no mesmo horário, trocando suas salas. Em termos de estrutura de dados, é realizada a troca horizontal entre duas agregações, invertendo as colunas entre si.

3.3. Comparando soluções

Após a realização de um movimento de troca, passamos a ter duas soluções, a solução corrente **s** e a candidata a nova solução corrente **s'**, vizinha de **s**.

A partir destas duas soluções, é preciso decidir qual deverá seguir em frente como solução corrente no prodecimento, isto é, decidir se **s** continua como solução corrente ou se **s'** passa a ser a nova solução corrente e **s** é substituída.

Esta decisão é baseada no valor total de cada solução, que é obtido a partir de uma *função de cálculo de valor da solução*. Esta função, a partir de uma solução como entrada, verifica todas as quebras de restrições existentes, sendo elas graves ou leves, e atribui penalidades a elas de acordo com seus pesos e prioridades, retornando assim um valor inteiro, que é a soma total das penalidades desta solução. Tendo estes valores para as duas soluções **s** e **s'**, cabe ao algoritmo decidir, utilizando sua lógica e seus parâmetros correntes, qual solução deverá prosseguir.

4. Implementação

Após definida todas as estruturas que serão utilizadas no projeto, é preciso implementá-las de modo eficiente, a fim de obter soluções satisfatórias em um período aceitável de tempo. Nesta seção consideraremos somente a implementação dos algoritmos de otimização (SA e LAHC). Os algoritmos implementados podem ser encontrados em `https://github.com/danhenriquesc/schdlr-cpp`.

4.1. Obtendo a solução inicial

Após todos os dados necessários para o início do processamento estarem preparados, devemos iniciar os algoritmos de busca local para refinar a solução, que neste projeto são os algoritmos Simulated Annealing (SA) e Late Acceptance Hill-Climbing (LAHC).

Porém, estes algoritmos geram novas soluções candidatas a partir de uma solução corrente, e por isso é necessária uma solução inicial para ser adotada como solução corrente, possibilitando o início da tais gerações de novas soluções candidatas.

Devido à sua importância, para se ter uma solução inicial de qualidade, foi elaborado um algoritmo onde cada agregação é alocada inicialmente em um dos seus horários possíveis e em uma

sala com capacidade que cumpra seus requisitos, assim gerando uma solução inicial que já cumpra duas das restrições mais complexas do problema, possibilitando os algoritmos de refinamento focarem na solução e minimização de outras restrições. Vale salientar que podem ser utilizadas salas virtuais no auxílio da criação da solução corrente inicial e posteriormente no refinamento das soluções, porém tais salas não devem ter aulas alocadas ao fim do algoritmo, pois o tornaria teoricamente infactível.

O Algoritmo 1 apresenta o pseudocódigo da geração de uma Solução Inicial.

Algoritmo 1: Geração de uma Solução Inicial

```
1  para cada agregacao ag faça
2      obtém um intervalo de tempo possível de ag; //ag.temposAula[aleatorio(0,
         total − 1)]
3      busca uma sala disponível neste horário com capacidade maior ou igual a
         necessária para ag;
4      se encontrou uma sala então
5          aloque ag nesta sala e horário;
6      senão
7          aloque ag em uma sala virtual qualquer, neste horário;
8      fim
9  fim
10 Retorna solução inicial
```

4.2. Busca Local

Após obtermos a solução inicial, devemos refiná-la em um processo metaheurístico, a fim de obter novas soluções que sejam cada vez melhores, removendo todas as quebras de restrições graves e minimizando a quebra de restrições leves.

Para isto, utilizamos as metaheurísticas SA e LAHC, que basicamente são um processo iterativo, que gera novas soluções **s'**, vizinhas a solução corrente **s**, e decide se as aceita como nova solução corrente **s** a partir do valor de cada solução, comparando-as de acordo com propriedades, critérios e parâmetros particulares de cada algoritmo, sempre guardando a melhor solução **s*** obtida durante estas iterações.

4.2.1. Simulated Annealing

O Algoritmo Simulated Annealing, por vezes traduzido como *Arrefecimento Simulado*, é um algoritmo que simula o processo físico análogo à termodinâmica, onde as partículas e um sólido se organizam entre si em um equilíbrio térmico.

O algoritmo é baseado no princípio das heurísticas de busca local e usa uma estrutura de vizinhança prédefinida no espaço de pesquisa S. Um parâmetro de controle que é chamado de *temperatura*, em analogia ao processo de recozimento físico, governa o comportamento da busca. Em cada iteração é calculada uma solução vizinha y para a solução atual x. Se y tem um melhor custo do que x, a solução y é *aceita*, ou seja, a solução atual x é substituída por y. Se, por outro lado, y tem um pior custo em relação à x, a solução y pode ou não ser aceita de acordo com uma certa probabilidade, dependendo de (I) a diferença dos valores da função objetivo de x e y, e (II) a temperatura vigente no momento. O Algoritmo 2 apresenta o pseudocódigo do Simulated Annealing.

4.2.2. Late Acceptance Hill-Climbing

A ideia inicial do LAHC é bastante simples: o parâmetro de controle da condição de aceitação é tomado a partir de um histórico de soluções. Esta metaheurística pode ser vista como

uma extensão do *Hill Climbing*, com apenas uma diferença: como o *Hill Climbing* é guloso, uma solução candidata é comparada com a solução imediata, mas, no LAHC, uma solução candidata é comparada com uma solução que foi a corrente em algumas iterações atrás, e foi guardada em uma lista de um comprimento fixo L (Comprimento do Histórico). Exceto este critério de condição de aceitação, os outros detalhes do LAHC são iguais a outros métodos de busca local, isto é, o algoritmo é iniciado a partir de uma solução inicial aleatória e aceita ou rejeita iterativamente os candidatos até que ocorra uma condição de parada.

O Algoritmo 3 apresenta o pseudocódigo do LAHC.

Algoritmo 2: SA

```
1  S ← S_0; T ← T_0; // S_0 = Solução Inicial, T_0 = Temperatura Inicial
2  enquanto T > 0 faça
3      iteracoes ← 0;
4      enquanto iteracoes < maximo_iteracoes faça
5          S' ← geraVizinho(S);
6          Δ ← custo(S') − custo(S);
7          se Δ < 0 então
8              S ← S';
9              se custo(S') < custo(S*) então
10                 S* ← S';
11             fim
12         senão
13             se e^{−Δ/T} > aleatório(0,1) então
14                 S ← S';
15             fim
16         fim
17         iteracoes ← iteracoes + 1
18     fim
19     T ← T ∗ α
20 fim
21 Retorna S*; //S* = Melhor Solução
```

Algoritmo 3: LAHC

```
1  S ← S_0;
2  ∀k ∈ {0 ... T_lh − 1}, L[k] ← f(S);
3  iter ← 0; iterOciosas ← 0;
4  enquanto (iterOciosas < iterMax) faça
5      S' ∈ N(S);
6      se f(S') ≥ f(S*) então
7          iterOciosas = iterOciosas + 1;
8      senão
9          S* = S'; iterOciosas = 0;
10     fim
11     v ← iter mod T_lh;
12     se (f(S') < L[v]) ou (f(S') ≤ f(S)) então
13         S ← S';
14     fim
15     se f(S) < L[v] então
16         L[v] ← f(S);
17     fim
18     iter ← iter+1;
19 fim
20 retorna S*;
```

4.3. Função de cálculo de valor da solução

Uma das partes mais importantes de ambos os algoritmos é a **função de cálculo de valor da solução**, que é capaz de obter um valor inteiro que representa o custo de uma solução **s**.

Este valor é o total de penalidades que esta solução obteve em relação ao descumprimento de restrições. Isto é, quanto maior o valor, mais restrições foram infringidas, e pior é esta solução.

Neste trabalho, todos os infrigimentos de restrições graves foram penalizados com um valor muito alto em relação às penalidades de restrições leves.

Para as restrições leves, são aplicadas penalidades de valores muito menores em relação às penalidades de restrições graves. Cada problema particular pode ter algumas restrições leves mais importantes do que outras, e por esse motivo, cada restrição leve tem um valor de penalidade. Além disso, o não atendimento de uma restrição leve pode ser penalizado de forma variável, em relação a quanto esta restrição não está sendo respeitada. É realizado o somatório das penalidades de **todas** as restrições infringidas, individualmente.

Uma abordagem matemática deste problema pode ser apresentada como, sendo **s** uma solução, temos uma função **f(s)** na qual calcula o custo total desta solução. Com isso, temos a seguinte função objetivo:

$$\min f(s)$$

Onde:

$$f(s) = g(s) + h(s)$$

Sendo:

g(s): Função que calcula o custo de quebra de restrições graves.
h(s): Função que calcula o custo de quebra de restrições leves.

A função **g(s)** é detalhada pela seguinte expressão:

$$g(s) = \sum_{i=1}^{S} \sum_{j=1}^{H} \sum_{k=1}^{G} w(i,j,k)\alpha$$

Onde:

S: Número de salas.
H: Número de horários.
G: Número de restrições graves.
$w(i,j,k)$: Função que retorna 1 caso a k-ésima restrição grave tenha sido infringida pela alocação de uma aula na i-ésima sala e no j-ésimo horário, e 0, caso contrário.
α: Penalidade por quebra de uma restrição grave.

A função **h(s)** é detalhada pela seguinte expressão:

$$h(s) = \sum_{i=1}^{S} \sum_{j=1}^{H} \sum_{k=1}^{L} w(i,j,k)P(i,j,k)\beta_k$$

Onde:

S: Número de salas.
H: Número de horários.
L: Número de restrições leves.
$w(i,j,k)$: Função que retorna 1 caso a k-ésima restrição leve tenha sido infringida pela alocação de uma aula na i-ésima sala e no j-ésimo horário, e 0, caso contrário.
$P(i,j,k)$: Função que retorna o peso associado ao infringimento da *k*-ésima restrição leve pela alocação de uma aula na i-ésima sala e no j-ésimo horário.
β_k: Penalidade por quebra da *k*-ésima restrição leve.

5. Experimentação e Resultados

Os testes foram realizados em um computador com o processador Intel® Core™i7-4820 3.70GHz e 16GB de memória RAM.

Para estes experimentos, foram obtidas todas as informações do instituto, referentes ao período de 2016/1, incluindo os dados pré-alocação e a alocação manual finalizada. Os testes foram realizados utilizando dois cenários:

1. **Cenário 1** - Foram fixados os horários utilizando o resultado obtido pela programação de horários manual, deixando o sistema processar somente a alocação de salas de aula, caracterizando um problema clássico de *alocação de salas de aula*.

2. **Cenário 2** - Foram utilizados os dados pré-alocação reais do IME/UERJ, caracterizando os problemas clássicos de *alocação de salas de aula* e *programação de horários*, que neste trabalho são resolvidos de forma simultânea.

Tabela 1: Penalidade das restrições leves nas experimentações

Restrição	L1	L2	L3	L4.1	L4.2	L4.3	L5	L6	L7
Penalidade	6	7	8	9	5	1	4	3	2

Todas as restrições graves foram penalizadas com o custo 10.000 cada. As restrições leves foram penalizadas de acordo com a Tabela 1.

Todos os experimentos utilizaram 70% de movimentos de salas e 30% de movimentos de horário. No SA foram utilizados como parâmetro *T: 5*; α*: 0,9*; *iterações por estado: 20.000*; *critério de parada: T*$\leq$*0,001*; e 6 salas virtuais. No LAHC foram utilizados como parâmetro *tamanho da lista: 1.000*; *critério de parada: iterações ociosas = 1.000.000*; e 4 salas virtuais.

Foram realizados 30 testes para cada algoritmo, em cada cenário, totalizado 60 testes por cenário. Devido à complexidade de resolver ambos os problemas simultaneamente, alguns experimentos deram resultados infactíveis com estes parâmetros. No **Cenário 1**, 3,33% dos resultados utilizando o SA e 13,33% dos resultados utilizando LAHC foram infactíveis. No **Cenário 2** estes números foram 53,33% e 63,33%, respectivamente. Para os cálculos de média, as soluções infactíveis foram desconsideradas. Estes resultados podem ser vistos na Tabela 2.

Tabela 2: Resultados das soluções por experimentações e manual

	LAHC					
	CM	**MC**	**TM(s)**	**MT(s)**	**IM**	**MNI**
Cenário 1	787	747	3.757	3.268	1.791.413	1.521.749
Cenário 2	759	745	5.636	3.130	2.473.240	1.791.498

	SA						**Manual**
	CM	MC	TM(s)	MT(s)	IM	MNI	MC
Cenário 1	792	731	3.750	3.727	1.620.000	1.620.000	-
Cenário 2	749	717	4.197	3.188	1.620.000	1.620.000	2.942

CM = Custo Médio; MC = Menor Custo; TM = Tempo Médio; MT = Menor Tempo; IM = Iterações Médias; MNI = Menor Número de Iterações.

Também foi comparada a eficiência dos dois algoritmos em relação à velocidade para alcançar um custo alvo predefinido, utilizando a técnica TTTplots [Aiex et al., 2007].

Para isto, foi utilizado o **Cenário 1**, onde o custo da melhor solução conhecida até então é de 731. Para a comparação, foi utilizado um alvo de dificuldade média, com 20% de custo acima da melhor solução conhecida, ou seja, o alvo é uma solução que tenha o custo de 877 ou menos.

Foram realizados 50 testes para cada algoritmo, com um limite de tempo de 1.800s para cada execução. Dentre os testes com o algoritmo SA, 1 teste não alcançou o alvo dentro do tempo limite. Dentre os testes com o algoritmo LAHC, 4 testes não alcançaram o alvo dentro do tempo limite.

O gráfico da comparação pode ser visto abaixo, na Figura 1, e mostra a curva azul, referente ao algoritmo SA, mais à esquerda no eixo x (tempo) em relação à curva vermelha, referente ao algoritmo LAHC, o que sugere que a primeira atinge o alvo escolhido mais rapidamente.

6. Conclusão

Baseado nos experimentos, conclui-se que ambos os algoritmos geraram soluções altamente satisfatórias, em custo e em tempo, em relação à solução manual, onde no **Cenário 2** -

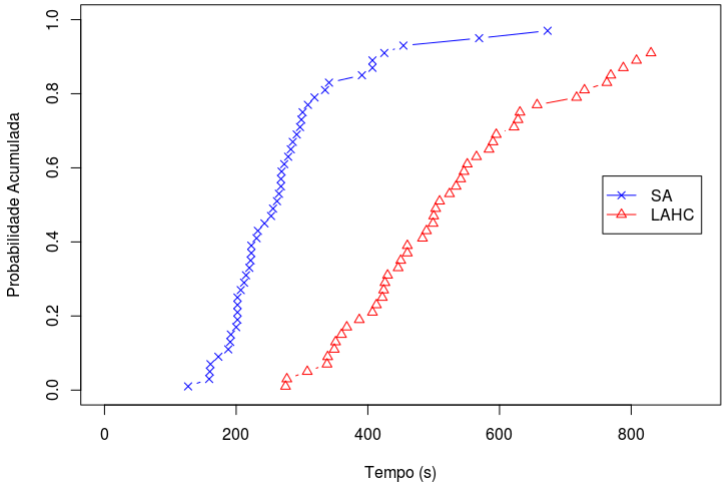

Figura 1: Comparação de tempo para atingir um alvo predeterminado utilizando SA x LAHC

cenário real do IME/UERJ - o custo da melhor solução encontrada foi de 717, sendo o custo da solução manual igual à 2.942, isto é, custando apenas 24,37% da solução manual.

Percebe-se também que, no **Cenário 2**, houve um alto índice de infactibilidade, devido à complexidade da solução dos problemas de *alocação de salas de aula* e de *programação de horários* de forma simultânea. Tal índice de infactibilidade foi quase nulo no **Cenário 1**, pois este era menos complexo devido ao fato do mesmo abranger somente o problema de *alocação de salas de aula*. Ambos menores custos foram obtidos pelo algoritmo SA, porém o algoritmo LAHC obteve um custo médio melhor no **Cenário 1**, o que indica um equilibrio em relação à satisfabilidade de ambos.

Em relação ao tempo, apesar do LAHC ter as execuções de menor tempo em ambos os cenários - o que não implica em que tais execuções geraram as melhores soluções -, a média dos tempos do SA é consideravelmente melhor do que a do LAHC no cenário mais complexo, onde o SA obteve o menor custo e o menor custo médio. Além do mais, o SA converge mais rapidamente à uma boa solução do que o LAHC, como pode ser visto na Figura 1.

Com isso, os resultados de ambos os algoritmos são demasiadamente superiores ao do processo manual. Porém, o SA mostrou-se levemente superior ao LAHC, pois foi mais consistente em relação à factiblidade, e atingiu melhores médias de custo e tempo.

Referências

Abdullah, S. (2006). *Heuristic Approaches for University Timetabling Problems*. PhD thesis.

Aiex, R. M., Resende, M. G. C., e Ribeiro, C. C. (2007). Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366.

Bardadym, V. A. (1996). Computer-aided school and university timetabling: The new wave. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1153, p. 22–45.

Burke, E. K., Jackson, K. S., Kingston, J. H., e Weare, R. F. (1997). Automated Timetabling: The State of the Art. *The Computer Journal*, 40(9):565–571.

Burke, E. K., Silva, J. D. L., e Petrovic, S. (2004). An introduction to multiobjective metaheuristics for scheduling and timetabling. *Metaheuristics for multiobjective optimisation*, p. 91–129.

Carter, M. W. e Laporte, G. (1998). Recent Developments in Practical Course Timetabling. *Practice and Theory of Automated Timetabling II*, 1408:3–19.

Carter, M. W. e Tovey, C. A. (1992). When Is the Classroom Assignment Problem Hard?

Dimopoulou, M. e Miliotis, P. (2001). Implementation of a university course and examination timetabling system. *European Journal of Operational Research*, 130(1):202–213.

Even, S., Itai, A., e Shamir, A. (1975). On the complexity of time table and multi-commodity flow problems. *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, p. 184–193.

Schaerf, A. (1999). Survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127.

Souza, M. J. F. e Santos, H. G. (2007). Programação de Horários em Instituições Educacionais: Formulações e Algoritmos. *XXXIX SBPO - A Pesquisa Operacional e o Desenvolvimento Sustentável*, p. 2827–2882.

Wren, A. (1996). Scheduling, timetabling and rostering - A special relationship? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1153:46–75.