SIMULATED ANNEALING APLICADO AO PROBLEMA DE ALOCAÇÃO DE SALAS DE AULA

Hanniere de Faria Veloso dos Santos, Dinesh Atul Rodrigues Trivedi

Instituto de Ciência e Tecnologia

Universidade Federal de São Paulo (UNIFESP)

Rua Talim, 330, São José dos Campos, São Paulo,

CEP 12231-280

São José dos Campos, SP

Resumo

Este artigo vem com a proposta de tratar do problema de alocação de salas, modelando-o de forma a utilizar *Simulated Annealing* (SA) para sua resolução. Este problema é considerado como NP-Difícil e por isso muitos métodos heurísticos, como: *Simulated Annealing*, Busca Tabu, Algoritmo Genético, tem sido propostos para resolve-lo. Sendo assim, serão abordadas apenas algumas restrições do problema que possibilite a geração de resultados de qualidade. O objetivo é otimizar a utilização das salas de aula de uma instituição de acordo suas capacidades bem como eliminar inviabilidades, no caso, a alocação de turmas a salas cujas capacidades não a comportem. Este trabalho foi implementado utilizando C++ ANSI juntamente com a ferramenta Qt Creator 2.7.0 32 bit.

Palavras-Chave: Problema de Programação de Horários, Problema de Alocação de Sala, Simulated Annealing, *timetabling*.

Sumário

1 Introdução

Problemas de Programação de Horários (*Timetabling*) , também conhecidos como (PPH) vêem motivando diversas pesquisas na área de Otimização Combinatória e Inteligência Artificial (SCHAEFER, 1999). Uma vez que estes problemas são classificados como NP-Difícil e não se conhece um algoritmo que os resolvam em tempo satisfatório, suas resoluções vêem sendo obtidas por heurísticas e metaheurísticas que não oferecem soluções ótimas, mas que possuem boa qualidade.

Na literatura há uma divisão de Problemas de Programação de Horários (PPH) em: Programação de Horários em Escolas (PPHE),Problemas Programação de Horários de Cursos em Universidades (PPHU). Essa classificação foi proposta por Schaefer (1999) sendo a mais utilizada na literatura.

Para essa classe de problemas inicialmente são definidos os requisitos que são expressos em forma de restrições do problema a ser atacado, que normalmente podem ser de dois tipos:

- Restrições fortes ou essenciais: restrições que devem ser satisfeitas a qualquer custo, sendo que não é possível uma solução factível que não as satisfaça.
- Restrições fracas ou não essenciais: restrições que são desejáveis de serem satisfeitas afim de que a solução obtida tenha melhor qualidade.

Segundo Souza (2007), o Problema de Alocação de Salas (PAS) é uma derivação dos Problemas de Programação de Horários de Cursos em Universidades (PPHU), ou seja, é tido como resolvido o problema PPHU desconsiderando a alocação de salas o que é resolvido por meio do PAS. Sendo assim, este artigo tem o objetivo de obter soluções factíveis para o Problema de Alocação de Salas por meio de sua modelagem e posterior utilização da metaheurística *Simulated Annealing.* Com o intuito de validar o sistema implementado, será utilizado dados reais obtidos da própria Universidade Federal de São Paulo (UNIFESP).

2 Descrição do problema

2.1 O problema de alocação de Salas

Conhecido na literatura inglesa como *Classroom Assignment Problem* e aqui chamado de Problema de Alocação de Salas (PAS), consiste em alocar aulas, com horários previamente conhecidos à um numero fixo de salas de forma que requisitos considerados essenciais sejam respeitados (CARTER; TOVEY, 1992).

Na literatura são observadas várias restrições essenciais e não-essenciais, sendo os mais comuns listados a seguir. Como restrições fortes, considera-se:

- Em uma mesma sala e horário não pode haver mais de uma aula;
- Uma sala não pode receber uma turma cuja quantidade de alunos seja superior à sua capacidade;
- Uma turma só pode ser alocada somente em uma sala de aula em um determinado horário.

Como restrições fracas mais comuns, têm-se:

- Algumas aulas precisam de recursos especiais, tais como : projetores, alto-falantes o que obriga alocá-las em salas que possuam esses recursos.
- Algumas salas possuem restrições de uso e a utilização delas deve ser evitada, por exemplo: auditórios, salas de defesa de tese e etc.
- Sempre que possível alocar alunos de um mesmo curso e período a uma mesma sala.
- Alocar os alunos de modo a otimizar o espaço das salas.
- Sempre que possível deixar diariamente um horário vazio em cada sala, para possibilitar sua limpeza.

O Problema de Alocação considera que existem horários pré-estabelecidos de inicio e término das aulas e suas respectivas demandas de alunos, bem como um conjunto de salas onde as turmas devem ser alocadas com suas respectivas capacidades.

Sendo assim, neste artigo serão abordados os requisitos essenciais para a validação da implementação aqui apresentada.

2.2 Representação da solução

A solução do problema é representada por uma matriz S_{mxn} com m representando o número de horários que as salas estão disponíveis para as aulas e n representando o numero de salas. Cada célula $S_{i,j}$ da matriz S representa um horário *i* disponível numa sala *j*, sendo que cada matriz representa um dia da semana. Sendo assim uma célula vazia indica que a sala *j* está desocupada no horário *i*. Caso uma célula esteja ocupada, indica que uma turma T com horário *i* está alocada a uma sala *j*.

Para exemplificar, é dada a seguinte tabela que possui seis salas (1, 2, 3, 4, 5, 6), sete horários disponíveis (1, 2, 3, 4, 5, 6,7) e dez turmas (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

Horários \ Salas	1	2	3	4	5	6
1	1	3	5	10	6	5
2	1	3	4	1	7	2
3			7		2	6
4			7	2	10	
5		8		2		
6	2	10				1
7	3				5	2

Tabela 1 – Exemplo de uma solução

2.3 Geração da solução inicial

A geração da solução inicial foi dada de forma aleatória. Inicialmente é construída a lista de turmas e outra de salas, sendo preenchidas a partir de um arquivo de entrada. De acordo com as listas formadas, escolhe-se uma sala e turma aleatoriamente, verifica-se a disponibilidade da sala no horário requerido pela turma, verificando também se a sala comporta a demanda de alunos da turma. Caso a sala esteja disponível e comporte a

demanda, essa turma é então alocada nessa sala e respectivo horário. Neste processo, caso alguma turma não consiga ser alocada a alguma sala, é criada uma sala virtual para ser alocada, de forma que durante a execução do método, turmas em salas virtuais também sejam consideradas para a geração de vizinhos e cálculos do custo.

Formato do arquivo de entrada:

```
<quantidade de salas n>
<capacidade sala 1> <descrição sala 1>
.
.
.
<capacidade sala n><descrição sala 1>
<quantidade turmas t>
<demanda turma 1> <horário> <dia> <descrição turma 1>
.
.
.
<demanda turma t> <horário> <dia> <descrição turma t>
```

Figura 1 - Formato de arquivo de entrada

Onde horário varia de 0 até a quantidade de horários desejada, dia da semana varia de 1, segunda-feira, a 7, domingo.

2.4 Estrutura de vizinhança

Foram definidos quatro tipo de movimentos para definir uma vizinhança N(*s*) de uma determinada solução *s*. O primeiro movimento se caracteriza pela realocação de determinada turma de uma sala para outra sala que esteja vazia, respeitando a capacidade e disponibilidade de horários da sala.

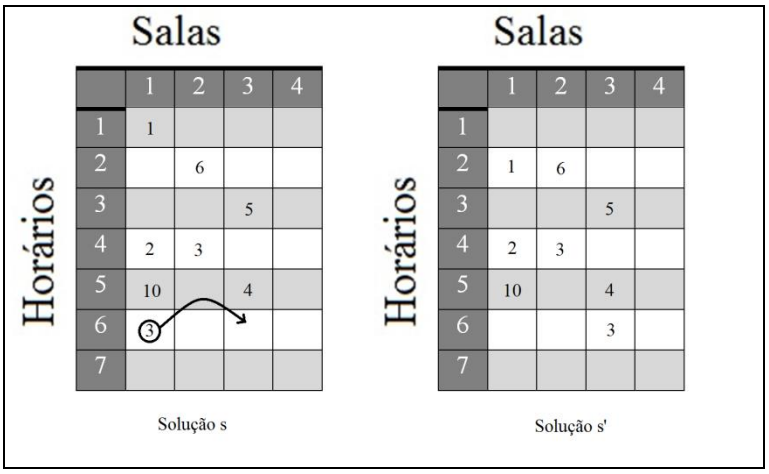

Figura 2 - Movimento de realocação

Nesta figura, a aula da turma três realizada no horário 6 da sala 1 foi transferida para a sala 3.

O segundo movimento é baseado na troca de salas entre aulas de duas turmas em um mesmo horário.

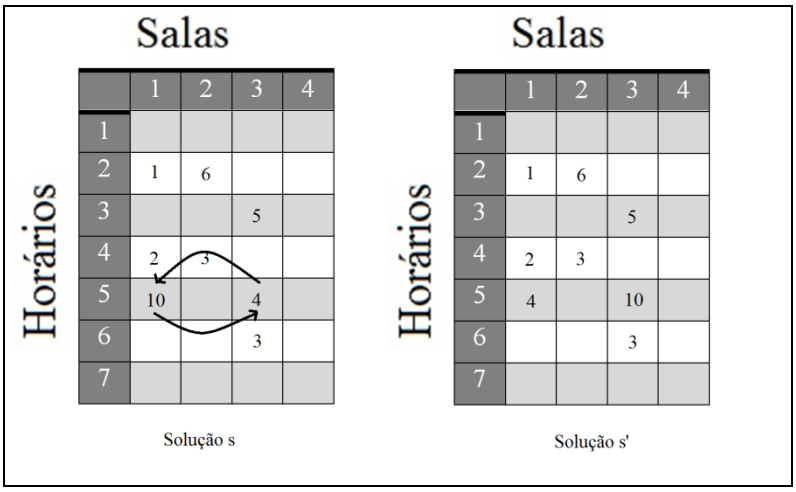

Figura 3 - Movimento de troca

Nesta figura, as aulas das turmas 10 e 4 que se encontram nas salas 1 e 3 respectivamente são permutadas. A aula da turma 10 é transferida para a sala 3 e a aula da turma 4 é transferida para sala 1.

O terceiro movimento é baseado nas salas virtuais existentes, tenta-se alocar uma turma que esteja em sala virtual em alguma sala que está vaga, pois após a realocação essa turma que não foi alocada inicialmente poderá ser colocada em alguma sala.

O quarto, e último movimento, também está em função de salas virtuais e é similar ao segundo movimento, mas a troca é feita entre uma turma já alocada e outra que está na sala virtual.

2.5 Função objetivo

A função objetivo considerada neste trabalho é baseada na penalização dos requisitos essenciais não atendidos, dando um peso para cada requisito. O cálculo é feito em função da quantidade de salas virtuais e da diferença entre a capacidade da sala e a demanda da turma alocada.

Esse cálculo considera que a existência de sala virtual com uma penalidade maior do que a diferença da capacidade e demanda, com peso 50 e 1 respectivamente. Portanto temos:

$$f(s) = (Qsv * 500) + \sum(capacidadeSala[j] - demandaTurma[i][j])$$ Onde:

i. Qsv é a quantidade de salas virtuais

ii. capacidadeSala[j] é a capacidade da sala j, tal que j = 0,1,2,... Número sala

iii. demandaTurma[i][j] é demanda da turma alocada no horário i na turma j, tal que i = 0,1,2,... quantidade de horários e j = 0,1,2,... Número sala

iv. A diferença da capacidade pela demanda da turma só entra no somatório caso uma turma esteja alocada na sala no horário em questão.

3 Metodologia

3.1 Ferramentas Utilizadas

Para o desenvolvimento da aplicação foram utilizadas as seguintes ferramentas:

1. QtCreator: IDE de desenvolvimento, parte do Qt Project. Possui diversas funcionalidades similares ao eclipse além de integração com o qmake, gerador de makefile do QtProject, valgrind e outras ferramentas (DIGIA, 2014).

2. Valgrind: É um software livre que auxilia na depuração de sistemas. Faz a análise durante a execução, consequentemente deixando a aplicação mais lenta, mas apenas durante a análise. Suas principais funcionalidades são: detectar *memory leak*, vazamento de memória, alocação e desalocação incorretas e acessos a áreas inválidas (VALGRIND,2014).
3. Gprof: É uma ferramenta de análise dinâmica da execução de sistemas desenvolvida pela GNU. O propósito principal é determinar quanto de recurso computacional está sendo utilizado por cada parte do código, com o a finalidade de otimizar o tempo de execução e, em alguns casos, consumo de memória (GNU, 2014).

3.2 Simulated Annealing

Simulated Annealing é uma heurística de busca local que aceita movimentos que pioram o custo de forma a escapar de ótimos locais por meio de uma probabilidade. Este método foi proposto inicialmente por (Kirkpatrick et al., 1983), e tem como fundamento o principio da termodinâmica ao simular o resfriamento de um conjunto de átomos aquecidos.

Esta técnica inicia sua busca a partir de uma solução inicial qualquer, de forma que posteriormente em cada loop gera-se um vizinho de forma aleatória da solução corrente *s*.A cada vizinho gerado s' de s, testa-se o valor de Δ, ou seja, $\Delta = f(s') - f(s)$. Caso $\Delta < 0$, o método aceita a solução e o vizinho *s'*. *Caso* $\Delta \geq 0$, é calculado a probabilidade $e^{-\Delta/T}$ desse vizinho ser aceito ou não. T é um parâmetro do método, chamado de temperatura e regula a probabilidade de aceitação de soluções de pior custo.

Inicialmente a temperatura T assume um valor elevado T_0 . Após determinado número de iterações a temperatura é gradativamente reduzia por uma taxa de resfriamento α, ou seja, $T_{n} \leftarrow \alpha * T_{n-1}$ $(0 < \alpha < 1)$. Desta forma faz-se com que no inicio tenha-se uma chance de sair de mínimos locais e quando T aproxima-se de zero, o algoritmo comporta-se como o método de descida, dado que é diminuída a probabilidade de aceitar movimentos que pioram o custo.

O algoritmo para quando a temperatura chega a um valor próximo de zero e nenhuma solução de pior custo seja mais aceita.

Os parâmetros deste algoritmo são: a taxa de resfriamento α, o numero de iterações para cada temperatura (IT*max*) e a temperatura inicial T_0.

```
        Algoritmo SA. (f(.), N(.), α,  ITmax, T0, s)
1.  s* ← s;                  {Melhor solução obtida}
2.  IterT ← 0;               {Número de iterações na temperatura T}
3.  T ← T0;                  {Temperatura corrente}
4.  enquanto (T > 0) faça
5.     enquanto (IterT < ITmax ) faça
6.        IterT ← IterT + 1;
7.        Gere um vizinho qualquer s' ∈ N(s);
8.        Δ = f(s') – f(s);
9.        se (Δ < 0)
10.          então s ← s';
11.                se (f(s') < f(s*)) então s* ← s';
12.          senão Tome x ∈ [0,1];
13.                se (x < e-Δ/T) então s ← s';
14.       fim-se;
15.    fim-enquanto;
16.    T ← α * T;
17.    IterT ← 0;
18. fim-enquanto;
19. s ← s*;
20. Retorne s;
    Fim S.A.:
```

Figura 4 - Algoritmo *Simulated Annealing*

3.3 Sistema Desenvolvido

O Sistema aqui desenvolvido é basicamente divido em duas etapas. Na primeira constrói-se a solução inicial e na segunda etapa essa solução é submetida ao algoritmo *Simulated Annealing*, onde o objetivo é gerar outras soluções de forma que o valor da função objetivo fique o menor possível (ou maior possível dependendo do problema abordado). No caso do Problema de Alocação de Salas, minimizar a função objetivo é minimizar a relação de diferença entre a capacidade de uma sala e a demanda das turmas a ela alocadas além de tentar evitar ao máximo o uso de salas virtuais, cumprindo assim com os requisitos essenciais apresentados neste artigo.

Nesse sistema foi utilizado conceitos de orientação a objeto para que fosse possível o reaproveitamento do algoritmo *Simulated Annealing* para outros problemas, ou seja, o sistema foi desenvolvido de forma genérica criando interfaces para abstrair o

método implementado e suas soluções, além disso, foi utilizados padrões de projetos como:

i. Template Method: Para separar as características comuns da solução e do método implementado, *Simulated Annealing*, das características específicas do problema atacado, PAS;

ii. Factory Method: Como dito a forma de implementação abordou a ideia de generalidade. Esse padrão tem como função minimizar o impacto no resto do sistema para implementar e executar um novo tipo de problema a ser atacado;

iii. Singleton: Garantir que tenha apenas uma factory, no sistema todo, para os tipos de problemas atacados;

iv. MVC: Utilizamos o padrão Model View Controller para não ficar amarrada a IDE QtCreator. Assim a troca da view, seja até mesmo pelo console, se torna simples e faz com que seja necessário passar apenas o arquivo de entrada e os parâmetros do algoritmo.

Esses padrões permitem que a implementação de um novo problema que utilize *Simulated Annealing* de forma simples e rápida, os passos são:

i. Implementar as classes de domínio;

ii. Implementar a interface ISolucao com as características da nova solução;

iii. Implementar a interface IParse para parsear os novos dados de entrada;

iv. Implementar a interface ISimulatedAnnealing para o novo problema.

v. Adicione a classe implementada no item iv no factory do simulated annealing

Todas as implementações acima são métodos claros e simples, facilitando a implementação e legibilidade.

3.4 Sistema Aplicado ao Problema de Alocação de Salas

Durante o desenvolvimento do sistema utilizamos a idéia de TDD, *Test Driven Development*. Isso foi possível devido aos padrões de projetos utilizados e interfaces

criadas. Os testes unitários foram criados utilizando a biblioteca cassert do c++ para que não ficasse dependente de sistema operacional ou framework.

Abaixo explicaremos a forma de implementação do problema seguindo os passos de implementação citadas anteriormente.

No primeiro passo, as classes de domínio, baseiam-se em basicamente duas entidades: Turma e Sala.

1. Turma: Essa entidade tem a responsabilidade de representar as turmas que precisam ser alocadas. Sua representação é feita a partir de uma classe chamada Turma descrita a seguir.

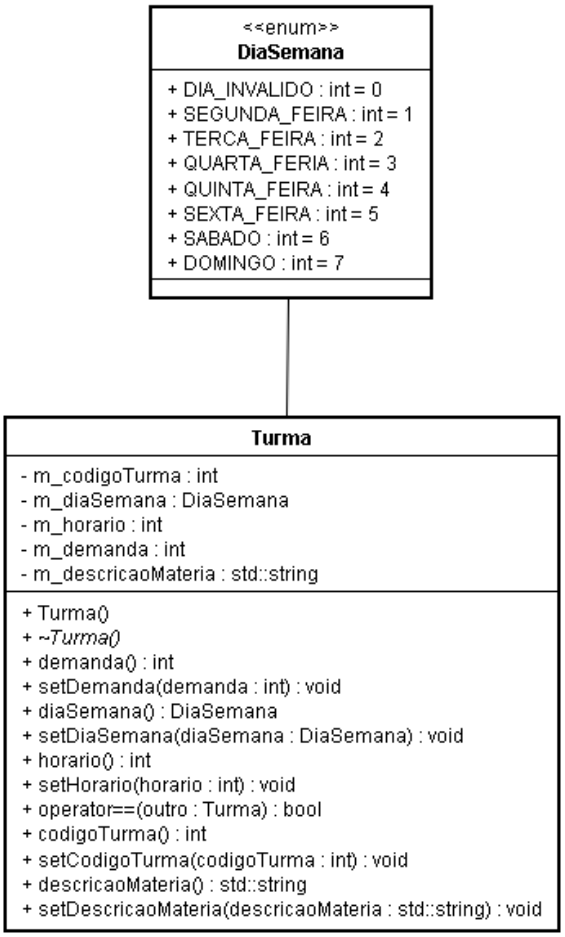

Figura 5 - Diagrama de classes da entidade Turma

2. Sala: Essa entidade tem a responsabilidade de representar as salas disponíveis. Sua representação é feita a partir de uma classe chamada Sala descrita a seguir.

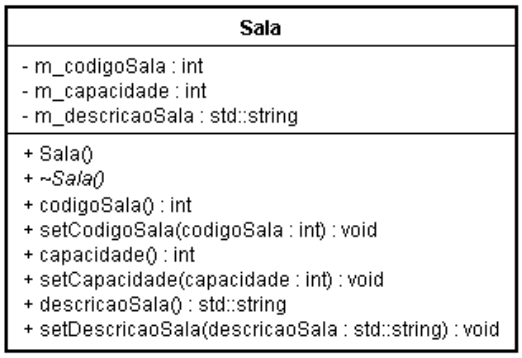

Figura 6 - Diagrama de classes da entidade Sala

No segundo passo, responsável pela definição da solução, baseia-se em basicamente em duas entidades: SolucaoSaAlocacaoSala, SolucaoSaAlocacaoSalaCompleta.

1. SolucaoSaAlocacaoSala: entidade que representa a solução de apenas um dia.
2. SolucaoSaAlocacaoSalaCompleta: entidade que representa a solução dos dias letivos da semana.

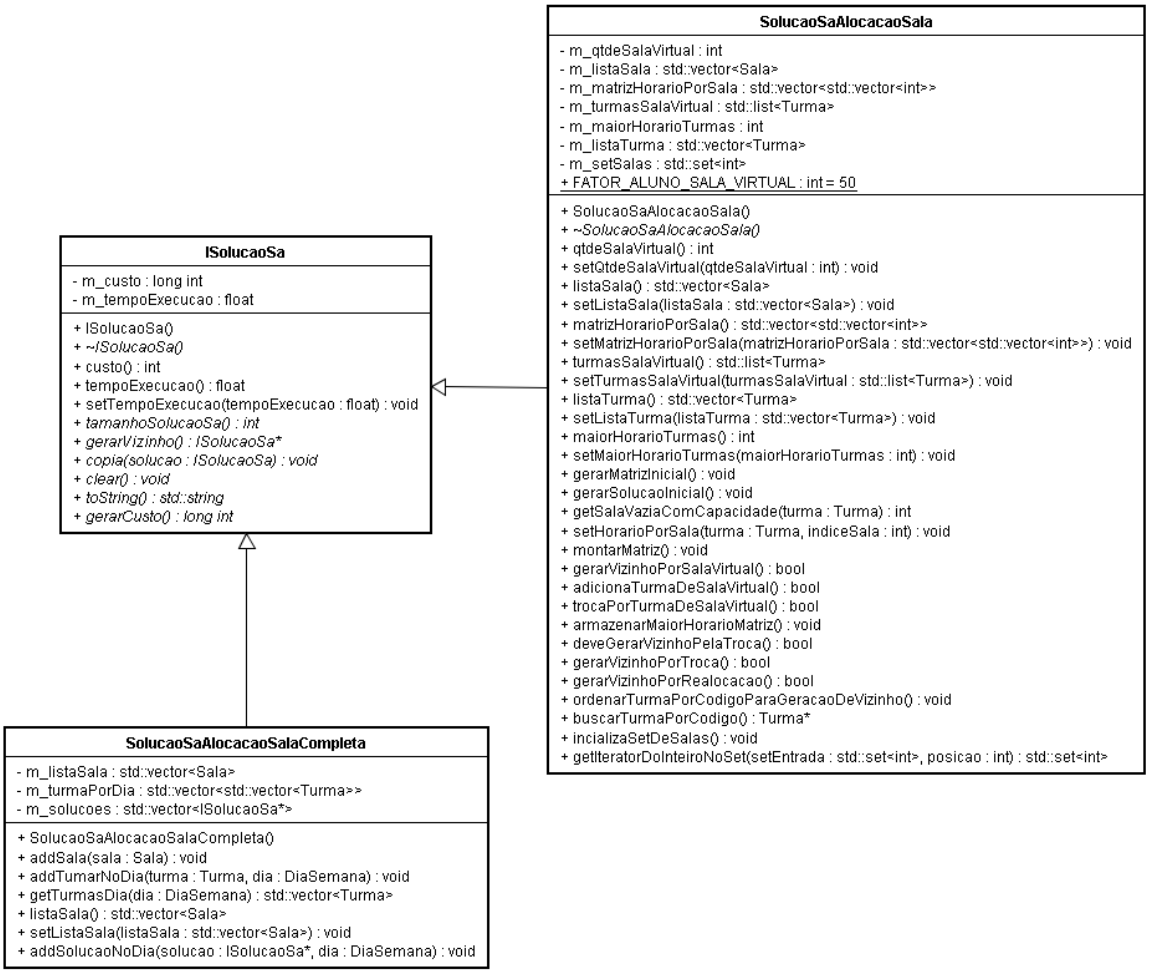

Figura 7 - Diagrama de classes das entidades Soluções

O terceiro passo consiste em implementar a interface IParse e baseia em basicamente em uma classe: ParserSimulatedAnnelingAlocacaoSalaArquivo, que tem a responsabilidade de fazer o parse da entrada de dados de acordo com o que foi definido.

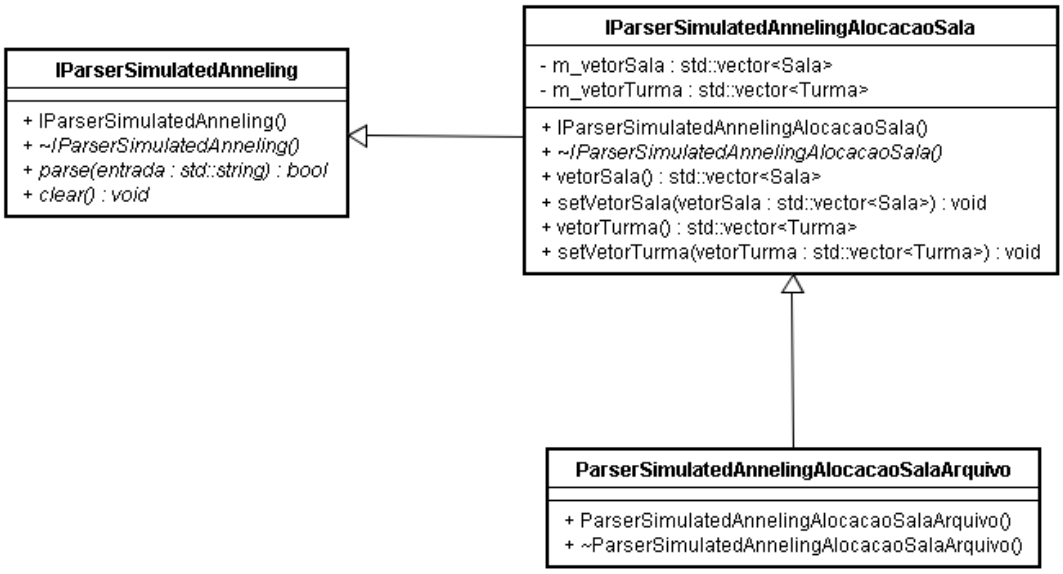

Figura 8 - Diagrama de classes do Parser

Já o quarto passo, responsável pelo algoritmo, baseia-se em basicamente em uma classe: SimulatedAnnelingAlocacaoSala, classe com a implementação e as características específicas do problema abordado.

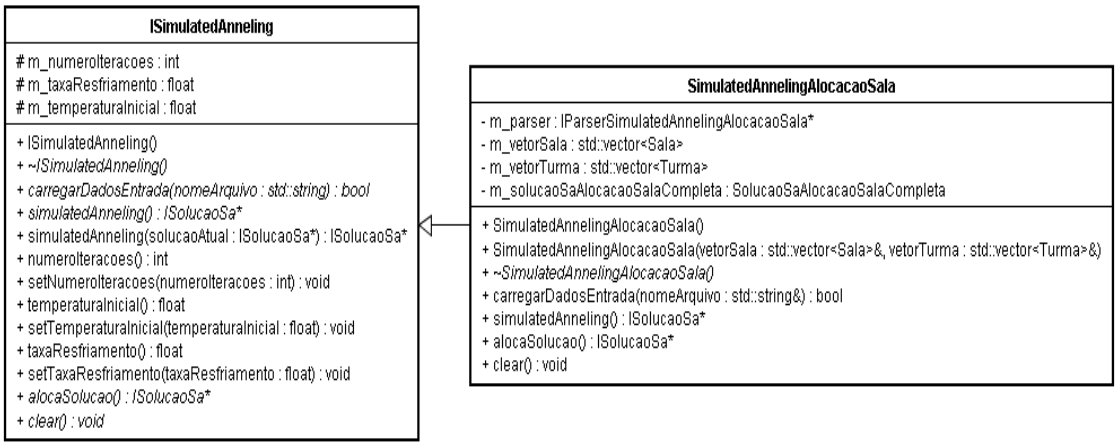

Figura 9 - Diagrama de classe Simulated Annealing

Por fim no quinto passo, consiste em possibilitar o uso do algoritmo para realizar a execução do problema e baseia-se em basicamente em uma classe:

SimulatedAnnelingFactory, factory responsável por devolver a implementação do algoritmo solicitada pelo usuário.

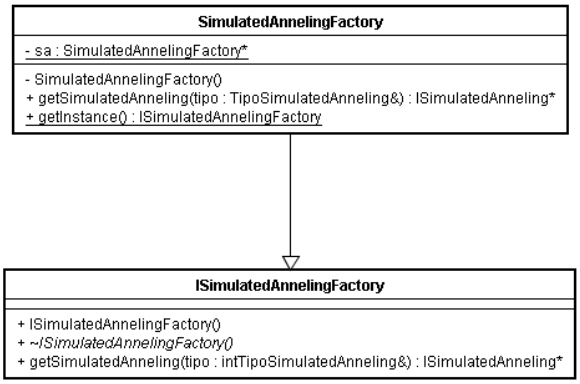

Figura 10 - Diagrama de classes do factory

4 Resultados

O algoritmo foi implementado na linguagem C++ ANSI juntamente com a ferramenta Qt Creator. Utilizou-se em uma maquina virtual com sistema operacional Ubuntu 12.04 core i7, 2.5Ghz, com 4GB de RAM.

Para a calibragem dos parâmetros foram feitos diversos testes para obtenção de resultados coerentes. Os parâmetros adotados foram: ITmax = 1000 (número máximo de iterações), $\alpha = 0.9$ (taxa de decaimento da temperatura), $T_0 = 100$ (temperatura inicial). Estes valores foram apenas adotados como valores padrões, pois na interface gráfica podem ser modificados.

Para os testes, foram utilizados dados da Universidade Federal de São Paulo, campus São José dos Campos. Os seguintes dados foram:

- Salas: 9 salas e suas capacidades.

Sala	Capacidade
101	50
102	50
103	62
104	60
106	90
107	80
108	40
109	45
110	38

Tabela 2 - Segunda-feira

- Turmas: 63 turmas divididas em cinco dias, como se segue.

Horário	Professor da matéria	Demanda
0	IPO_Chaves	30
0	QG_Fernando	36
0	IAR_Juliana	45
0	MD_Gabriel	50
1	MM_Arlindo	55
0	PP_Vinicius	60
1	AOC_Tiago	60
0	INTRONANO_Passador	60
0	COMPIL_Galvao	62
0	AED_Regina	62
0	FA_Vilhena	80

Tabela 3 - Segunda-feira

Horário	Professor da matéria	Demanda
0	POS_BIO_MOLEC_Claudia	4
1	CN2_Erwin	8
1	COMPIL_Galvao	18
1	MICROBIOL_Elisa	22
0	ONL_Bueno	26
1	CN_Bueno	29
0	BIOMAT_Mariana	32
1	EM_Vanessa	32
0	BD_Daniela	48
0	FIS3_Rossano	51
0	CVV_Elisangela	53
1	LPI_Luisa	78

Tabela 4 - Terça-feira

Horário	Professor da matéria	Demanda
0	SS_Juliana	50
0	INTROMAT_Quinteiro	36
0	IPO_Chaves	30
0	Prob2_Mirela	22
0	AOC_Tiago	60
0	CTM_Katia	39
0	POOI_Lorena	42
0	CVV_Eudes	37
0	CVV_Elisangela	29
1	IAR_Juliana	18
1	PAA_Juranry	26
1	MD_Gabriel	21
1	AB_Reginaldo	58
1	FEME_Fabiano	34

Tabela 5 - Quarta-feira

Horário	Professor da matéria	Demanda
0	AED_Marcio	40
0	MM_Arlindo	38
0	PP_Vinicius	62
0	TNC_Angelo	41
0	FA_Vilhena	27
0	TG_Maria	36
0	QG_Fernando	49
1	LPI_Luisa	42
1	EM_Vanessa	32
1	CN2_Erwin	18

Tabela 6 - Quinta-feira

Horário	Professor da matéria	Demanda
0	METODOLOGIA_Silvio	36
0	PosBIOTEC_Renato	48
0	CN_Bueno	34
0	FIS3_Rossano	14
0	AB_Reginaldo	58
0	RG_maria	36
0	TNC_Angelo	41
0	BBMF_Luciane	29
1	ONL_Bueno	32
1	POOI_Lorena	42
1	PAA_Jurandy	26
1	PCD_Alvaro	28
1	PI_Cappabianco	38
1	FEMEC_Fabiano	34

Tabela 7 - Sexta-feira

A seguir seguem dois gráficos de desempenho do algoritmo. A Figura 11 representa um gráfico de tempo, em segundos no eixo y, por número de iterações no eixo x. Os valores de taxa de decaimento da temperatura e temperatura inicial foram os valores default, 0.9 e 100 respectivamente.

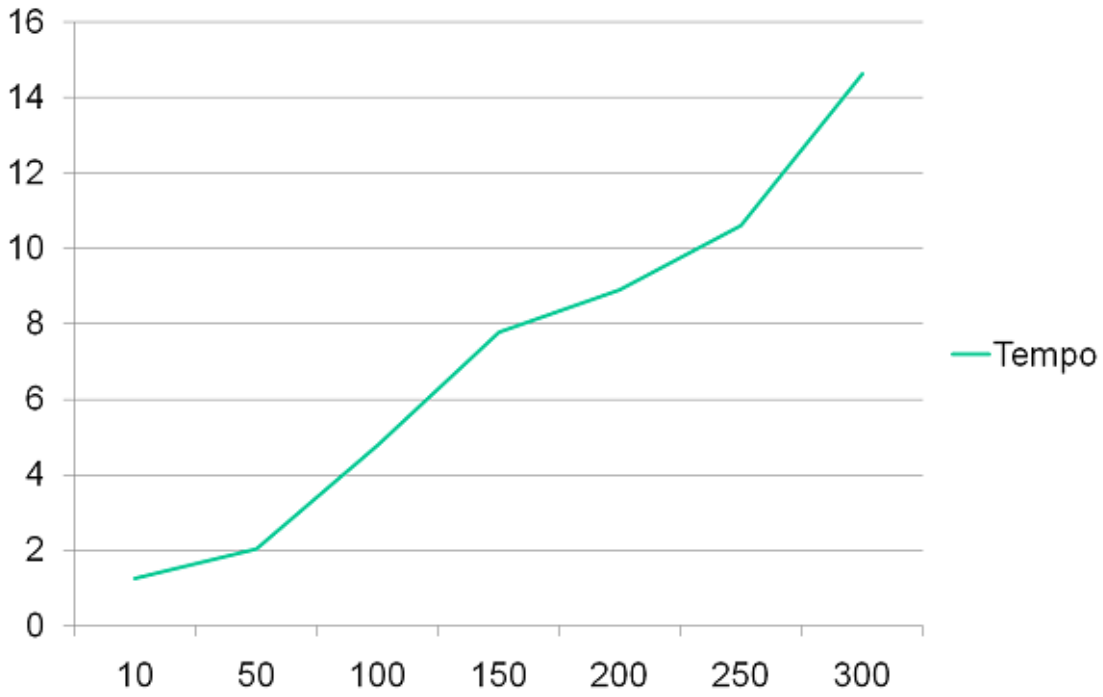

Figura 11 – Gráfico tempo por iterações

A Figura 12 representa um gráfico de tempo, em segundos no eixo y, por número de repetições no eixo x. Os valores de taxa de decaimento da temperatura, temperatura inicial e numero de iterações foram: 0.9, 100 e 100 respectivamente.

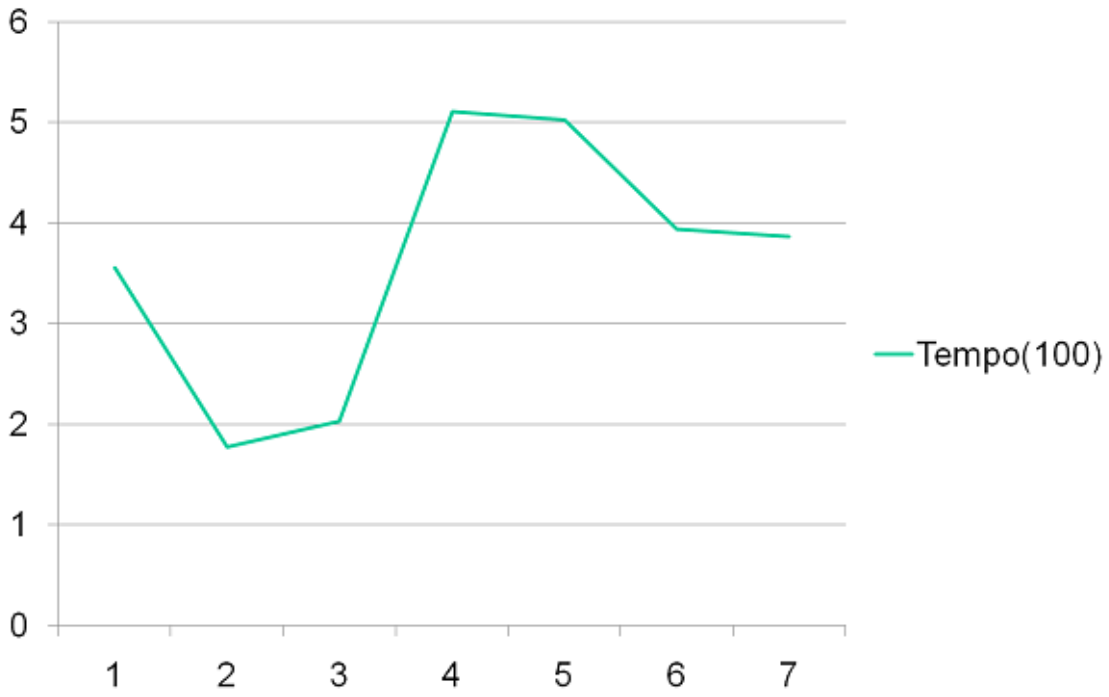

Figura 12 – Gráfico tempo por repetição

Por fim segue a tabela, Tabela 8, com os melhores resultados alcançados com os valores default.

Dia	Segunda	Terça	Quarta	Quinta	Sexta
Custo	31077	187	246	88	212
Sala Virtual	1	0	0	0	0
Tempo(s)	1.61	4.4	3.21	6.46	3.82

Tabela 8 – Tabela de Melhores Resultados

Referências

SCHAEFER, A. **,A survey of automated timetabling**, Artificial Intelligence Review, 13, 87- 127, (1999).

SOUZA, M.J.F., MARTINS, A.X. E ARAÚJO, C.R. (2002a), **Experiências com a utilização de Simulated Annealing e Busca Tabu na resolução do Problema de Alocação de Salas**. In: XXXIV Simpósio Brasileiro de Pesquisa Operacional - SBPO, Instituto Militar de Engenharia, Rio de Janeiro, Brasil. *Anais do XXXIV SBPO*, 1100-1110, 2007.

M.W. CARTER; C.A. TOVEY. **When is the classroom assignment problem hard?** Operations Research Supplement 1, 40:2839, 1992.

C. J. H. MCDIARMID, **The solution of a timetabling problem**, J. Inst. Math 9 (1972) 23–34.

DIGIA, QtCreator. Disponível em: < http://qt-project.org/wiki/Category:Tools::QtCreator >. Acesso em: 02 de janeiro 2014.

VALGRIND, valgrind. Disponível em: < http://valgrind.org/>. Acesso em: 02 de janeiro 2014.

GNU, gprof. Disponível em: < https://sourceware.org/binutils/docs-2.24/gprof/index.html >. Acesso em: 02 de janeiro 2014.