

DEPARTAMENTO DE COMPUTAÇÃO  
**D E C O M**

Resolução do Problema de alocação de salas de aula via  
*Simulated Annealing*

Otávio de Mello Castro – 99.1.4125

Marcone Jamilson Freitas Souza  
Orientador

**Monografia Projeto Orientado**  
Fevereiro 2003

**U F O P**  
UNIVERSIDADE FEDERAL DE OURO PRETO

# Resumo

Este trabalho trata do problema de alocação de salas. Uma vez que este problema é NP-difícil, muitos métodos heurísticos tais como *Simulated Annealing*, Busca Tabu, Algoritmo Genético, têm sido propostos para resolvê-lo.

Esta classe de heurísticas encontra uma boa solução melhorando uma alocação inicial através de técnicas de pesquisa em vizinhança. Contrariamente às heurísticas convencionais, SA permite movimentos de piora como forma de escapar de ótimos locais. Neste projeto relata-se uma experiência com a utilização desta técnica na resolução do problema de alocação de salas do Instituto de Ciências Exatas e Biológicas (ICEB) da Universidade Federal de Ouro Preto (UFOP).

Neste trabalho, o problema de alocação de salas foi modelado levando-se em consideração a metaheurística *Simulated Annealing* (SA), utilizando-se como movimento de vizinhança a realocação de turmas a salas. A função objetivo utilizada pelo método visa principalmente a eliminação de inviabilidades, no caso, a alocação de turmas a salas cujas capacidades não as comportem. A função objetivo procura também encontrar uma utilização eficiente do espaço minimizando a quantidade de carteiras não utilizadas nas salas, além de tentar agrupar todas as aulas semanais de uma dada turma em uma mesma sala de aula.

O modelo foi implementado utilizando-se a linguagem de programação C++, com a ferramenta C++ Builder 6.0. Para melhor tratamento do problema, tentou-se fazer com que o sistema abordasse também uma boa solução para a interface com os usuários, fazendo assim com que o sistema além de eficiente fosse também funcional e prático em termos de usabilidade. Com o intuito de validar o sistema implementado, testou-se este com dados reais obtidos da própria Universidade Federal de Ouro Preto (UFOP) que espera implantar o sistema e comprovar o seu funcionamento e eficácia já no próximo período de aulas.

**PALAVRAS-CHAVE:** Problema de Alocação de Salas, Metaheurísticas, Otimização Combinatória

# Agradecimentos

Agradeço, principalmente, à Deus que me deu força para seguir meu objetivo e sem o qual nada seria possível.

Aos meus pais, irmãs, vó e toda a família pela confiança e carinho.

Aos irmãos xequemateanos e a eterna república (família) XEQUE-MATE pela escola de vida e por todas as razões que nos fazem tão unidos, .

E também ao orientador do projeto, Marcone Jamilson Freitas Souza, pela atenção e disponibilidade sempre que foi solicitado, dando incentivo, força e exemplo de vida e de profissionalismo para continuar o meu aprendizado.

E às demais pessoas que colaboraram para o desenvolvimento deste trabalho, ficam os meus sinceros agradecimentos.

# Índice

<b>RESUMO.....</b>	<b>2</b>
<b>AGRADECIMENTOS .....</b>	<b>3</b>
<b>ÍNDICE.....</b>	<b>4</b>
<b>LISTA DE FIGURAS.....</b>	<b>6</b>
<b>LISTA DE TABELAS .....</b>	<b>7</b>
<b>1 INTRODUÇÃO .....</b>	<b>8</b>
1.1 OBJETIVOS DO TRABALHO .....	8
1.2 IMPORTÂNCIA DO TRABALHO .....	8
1.3 ESTRUTURA DA MONOGRAFIA .....	8
<b>2 REVISÃO BIBLIOGRÁFICA .....</b>	<b>9</b>
2.1 METAHEURÍSTICAS.....	9
2.1.1 <i>Simulated Annealing</i> .....	10
2.2 DESENVOLVIMENTO DE SOFTWARE.....	11
2.3 ORIENTAÇÃO A OBJETOS .....	13
2.3.1 <i>Objeto</i> .....	13
2.3.2 <i>Classe</i> .....	13
2.3.3 <i>Relacionamentos</i> .....	14
2.3.4 <i>Princípios básicos</i> .....	14
2.3.5 <i>Benefícios da Orientação a Objetos</i> .....	14
2.4 ARQUITETURA CLIENTE SERVIDOR .....	15
<b>3 METODOLOGIA.....</b>	<b>17</b>
3.1 DESCRIÇÃO DO PROBLEMA.....	17
3.1.1 <i>O Problema de Alocação de Salas Abordado</i> .....	17
3.2 MODELAGEM DO PROBLEMA .....	17
3.2.1 <i>Representação</i> .....	17
3.2.2 <i>Estrutura de vizinhança</i> .....	18
3.2.3 <i>Função objetivo</i> .....	19
3.3 DESCRIÇÃO DO MÉTODO UTILIZADO.....	20
3.3.3 <i>Geração de uma solução inicial</i> .....	20
3.3.4 <i>Algoritmo proposto para a resolução do Problema de Alocação de Salas</i> .....	21
3.3.5 <i>A Arquitetura Cliente/Servidor do PAS</i> .....	21
<b>4 SISTEMA DESENVOLVIDO.....</b>	<b>27</b>
4.1 DESCRIÇÃO DO SISTEMA .....	27
4.2 SISTEMA APLICADO AO PAS.....	28
4.3 INTERAÇÃO SISTEMA X SA .....	29
4.4 RECURSOS OFERECIDOS PELO SISTEMA .....	30
4.4.1 <i>Iniciação do Sistema</i> .....	30
4.4.2 <i>Entrada Automática de Dados</i> .....	31
4.4.3 <i>Controle Manual dos Dados</i> .....	33
4.4.4 <i>Limpar todos os dados</i> .....	36
4.4.5 <i>Agregação de Turmas</i> .....	36
4.4.6 <i>Especificação dos Parâmetros</i> .....	37

4.4.7	<i>Especificação das Penalizações</i> .....	39
4.4.8	<i>Iniciar uma Nova Solução do SA</i> .....	39
4.4.9	<i>Interromper Execução</i> .....	40
4.4.10	<i>Continuar Executando o SA</i> .....	40
4.4.11	<i>Salvar Solução</i> .....	40
4.4.12	<i>Carregar Solução</i> .....	40
4.4.13	<i>Troca manual de salas</i> .....	41
4.4.14	<i>Visualização da solução no Microsoft Excel</i> .....	42
4.4.15	<i>Resultados da Execução</i> .....	42
4.4.16	<i>Finalização do Sistema</i> .....	43
<b>5</b>	<b>RESULTADOS COMPUTACIONAIS</b> .....	<b>44</b>
<b>6</b>	<b>CONCLUSÕES E TRABALHOS FUTUROS</b> .....	<b>45</b>
<b>7</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS</b> .....	<b>46</b>

# Lista de Figuras

Figura 2.1 - Algoritmo do simulated annealing .....	11
Figura 2.2 - C++ Builder 6.0 .....	<b>Erro! Indicador não definido.</b> 2
Figura 2.3 - Esquema de protocolo de requisição .....	<b>Erro! Indicador não definido.</b> 5
Figura 3.1 - Exemplo de uma alocação .....	18
Figura 3.2 - Movimento de alocação .....	18
Figura 3.3 - Movimento de troca .....	19
Figura 3.4 - Algoritmo de construção da solução inicial .....	21
Figura 3.5 - Algoritmo proposto: solução inicial + simulated annealing .....	21
Figura 3.6 - Arquitetura de software utilizada .....	23
Figura 3.7 - Camada de interface .....	24
Figura 3.8 - Camada de negócio. ....	24
Figura 3.9 - Camada de banco de dados .....	25
Figura 3.10- Camada de comunicação .....	26
Figura 4.1 - Representação de uma turma. ....	28
Figura 4.2 - Representação de uma sala. ....	28
Figura 4.3 - Representação de uma solução. ....	29
Figura 4.4 - Algoritmo utilizado no sistema. ....	30
Figura 4.5 - Tela inicial do sistema. ....	31
Figura 4.6 - Tela Entrada Automática de Dados. ....	32
Figura 4.7 - Tela Controle de Disciplinas. ....	34
Figura 4.8 - Tela Controle de Turmas. ....	35
Figura 4.9 - Tela Controle de Salas. ....	36
Figura 4.10 - Tela de Confirmação de limpeza dos dados do banco de dados. ....	36
Figura 4.11 - Tela Agregação de Turmas. ....	37
Figura 4.12 - Tela Parâmetros do Sistema. ....	38
Figura 4.13 - Tela Penalidades do Sistema. ....	39
Figura 4.14 - Tela da Troca Manual de Salas. ....	41
Figura 4.15 - Tela Resultados da Execução do Sistema. ....	42

# Lista de Tabelas

Tabela 5.1 - Características das instâncias consideradas.....	44
Tabela 5.2 - Resultados Computacionais.....	44

# **1 Introdução**

## **1.1 Objetivos do trabalho**

O objetivo deste trabalho foi de desenvolver um sistema computacional amigável e eficiente na resolução do problema de alocação de salas do ICEB.

## **1.2 Importância do trabalho**

A solução manual do problema de alocação de salas é uma tarefa árdua e normalmente requer vários dias de trabalho. Além do mais, a solução obtida pode ser insatisfatória com relação a vários aspectos. Por exemplo, em função da alocação feita, pode haver em um dado horário um fluxo acentuado de alunos deslocando-se de salas com conseqüente perturbação no ambiente.

Em função de situações como essa, uma atenção especial vem sendo dada à automação deste problema. Entretanto, sua automação não é uma tarefa das mais simples, pois o problema é NP-difícil (Even et al. 1976, Carter and Tovey 1992), o que, em casos reais, dificulta ou até mesmo impossibilita sua resolução por técnicas de programação matemática, ditas exatas.

Em vista desta dificuldade, este problema é normalmente abordado por técnicas heurísticas.

Assim, além dos benefícios que se esperam obter para a UFOP com a implementação do sistema que se propõe, o presente trabalho contribui com o estado da aplicação de uma técnica heurística baseada em Simulated Annealing à resolução do problema.

## **1.3 Estrutura da monografia**

Este trabalho está organizado como segue.

No capítulo 2 é feita uma breve revisão bibliográfica sobre a forma de abordagem do problema bem como da técnica SA, do ambiente C++ Builder, de orientação a objetos e da Arquitetura Cliente/Servidor.

No capítulo 3 descreve-se o problema abordado, como ele foi modelado aplicando-se a técnica SA e qual foi a arquitetura de software utilizada para acesso ao Banco de Dados.

No capítulo 4 relata-se detalhes da implementação, interface e recursos oferecidos pelo sistema. Na capítulo seguinte (4) mostra-se detalhes do sistema desenvolvido.

Na capítulo 5 são mostrados os resultados obtidos pelo sistema

Na capítulo 6 apresentam-se as conclusões obtidas, bem como as sugestões para trabalhos futuros.



## 2 Revisão Bibliográfica

O problema de alocação de salas (PAS) diz respeito à distribuição de aulas, com horários previamente estabelecidos, a salas, respeitando-se um conjunto de restrições de várias naturezas (Schaefer 1999).

A alocação de salas é tratada ou como parte integrante do problema de programação de cursos universitários (*course timetabling*) ou como um problema derivado dele (*classroom assignment*) (Bardadym 1996). Nesta última situação, considera-se que as aulas dos cursos universitários já estejam programadas, isto é, que já estejam definidos os horários de início e término das aulas de cada turma de cada disciplina e o problema, então, é o de alocar essas aulas às salas.

Trata-se de um problema NP-difícil (Even et. Al. 1976) o que, em casos reais dificulta ou mesmo impossibilita sua resolução por métodos exatos. Assim sendo, esse problema é normalmente abordado através de técnicas heurísticas. Apesar de estas técnicas não garantirem a otimalidade, elas conseguem, em geral, gerar soluções de boa qualidade sem um elevado custo computacional e são relativamente fáceis de serem implementadas. Dentre as heurísticas, destacam-se as chamadas metaheurísticas, as quais, ao contrário das heurísticas convencionais, têm caráter geral e são providas de mecanismos para escapar de ótimos locais.

Dentre as metaheurísticas que vêm sendo aplicadas com relativo sucesso em problemas de programação de horários, destacamos, dentre outras: *Simulated Annealing* (Dowland 1998, Abramson 1991), Busca Tabu (Burke et al. 2001, Costa 1994, Hertz 1992) e Programação Genética (Ueda et al. 2001, Santos et al. 1997, Erben and Keppler 1996, Rich 1996).

A seguir, é feita uma breve revisão das técnicas heurísticas, em particular ao método SA e ao funcionamento e construção do banco de dados, bem como uma revisão de análise de sistema para melhor compreensão do sistema desenvolvido.

### 2.1 Metaheurísticas

Muitos problemas práticos são modelados da seguinte forma: Dado um conjunto  $S$  de variáveis discretas  $s$  (chamadas soluções) e uma função objetivo  $f: S \leftarrow R$ , que associa cada solução  $s \in S$  a um valor real  $f(s)$ , encontre a solução  $s^* \in S$ , dita ótima, para a qual  $f(s)$  é mínima.

Grande parte desses problemas são classificados na literatura como NP-difíceis. isto é, são problemas para os quais não existem algoritmos que os resolvam em tempo polinomial. Tais problemas são enquadrados como “problemas de otimização combinatória”.

Em problemas dessa natureza, o uso de métodos exatos se torna bastante restrito. Por outro lado, na prática, em geral, é suficiente encontrar uma “boa” solução para o problema, ao invés de um ótimo, o qual somente pode ser encontrado após um considerável esforço computacional.

Este é o motivo pelo qual os pesquisadores têm concentrado esforços na utilização de heurísticas para solucionar problemas desse nível de complexidade. Definimos heurística como sendo uma técnica que procura boas soluções (próximas da otimalidade) a um custo computacional razoável, sem, no entanto, estar capacitada a garantir a otimalidade, bem como garantir quão próximo uma determinada solução está da solução ótima.

O desafio é produzir, em tempo mínimo, soluções tão próximas quanto possível da

solução ótima. Muitos esforços têm sido feitos nessa direção e heurísticas muito eficientes foram desenvolvidas para diversos problemas. Entretanto, a maioria dessas heurísticas desenvolvidas é muito específica para um problema particular, não sendo eficientes (ou mesmo aplicáveis) na solução de uma classe mais ampla de problemas.

Somente a partir da década de 1980 intensificaram-se os estudos no sentido de se desenvolver procedimentos heurísticos, com uma certa estrutura teórica e de caráter mais geral, sem, no entanto prejudicar a sua principal característica, que é a flexibilidade.

Essa meta tornou-se mais realista a partir da reunião de conceitos das áreas de Otimização e Inteligência Artificial, viabilizando a construção das chamadas melhores estratégias ou dos métodos “inteligentemente flexíveis”, comumente conhecidos como “metaheurísticas”.

Esses métodos, situados em domínios teóricos ainda pouco explorados pela literatura, possuem como característica básica estruturas com uma menor rigidez que as encontradas nos métodos clássicos de otimização sem, contudo, emergir em uma flexibilidade caótica.

Metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada a qual tem que ser modelada para cada problema específico.

Contrariamente as heurísticas convencionais, as metaheurísticas são de caráter geral e têm condições de escapar de ótimos locais.

Dentre os procedimentos enquadrados como metaheurísticas que surgiram ao longo das últimas décadas, destacam-se: Algoritmos Genéticos (AGs), Redes Neurais, *Simulated Annealing* (SA), Busca Tabu (BT), GRASP, VNS, Colônia de Formigas, etc.

As duas primeiras metaheurísticas fundamentam-se em analogias com processos naturais, sendo que os AG's são procedimentos inspirados em princípios da evolução natural.

O método SA explora uma possível analogia com a termodinâmica. enquanto o método BT faz uso de uma memória flexível para tornar o processo de busca mais eficaz.

### 2.1.1 Simulated Annealing

Simulated Annealing é um método de busca local que aceita movimento de piora para escapar de ótimos locais. Ele foi proposto originalmente por (Kirkpatrick et al., 1983), e se fundamenta em uma analogia com a termodinâmica, ao simular o resfriamento de um conjunto de átomos aquecidos.

Esta técnica começa sua busca a partir de uma solução inicial qualquer. O procedimento principal consiste em um *loop* que gera aleatoriamente, em cada iteração, um único vizinho  $s'$  da solução corrente  $s$ .

A cada geração de um vizinho  $s'$  de  $s$ , é testada a variação  $\Delta$  do valor da função objetivo, isto é,  $\Delta = f(s') - f(s)$ . Se  $\Delta < 0$ , o método aceita a solução e  $s'$  passa a ser a nova solução corrente. Caso  $\Delta \geq 0$  a solução vizinha candidata também poderá ser aceita, mas neste caso, com uma probabilidade  $e^{-\Delta/T}$ , onde  $T$  é um parâmetro do método, chamado de *temperatura* e que regula a probabilidade de aceitação de soluções com custo pior.

A temperatura  $T$  assume, inicialmente, um valor elevado  $T_0$ . Após um número fixo de iterações (o qual representa o número de iterações necessárias para o sistema atingir o equilíbrio térmico em uma dada temperatura), a temperatura é gradativamente diminuída por uma razão de resfriamento  $\alpha$ , tal que  $T_n \leftarrow \alpha * T_{n-1}$ , sendo  $0 < \alpha < 1$ . Com esse procedimento, dá-se, no início uma chance maior para escapar de mínimos locais e, à medida que  $T$

aproxima-se de zero, o algoritmo comporta-se como o método de descida, uma vez que diminui a probabilidade de se aceitar movimentos de piora ( $T \rightarrow 0 \Rightarrow e^{-\Delta/T} \rightarrow 0$ ).

O procedimento pára quando a temperatura chega a um valor próximo de zero e nenhuma solução que piore o valor da melhor solução é mais aceita, isto é, quando o sistema está estável. A solução obtida quando o sistema encontra-se nesta situação evidencia o encontro de um mínimo local.

Os parâmetros de controle do procedimento são a razão de resfriamento  $\alpha$ , o número de iterações para cada temperatura ( $SA_{max}$ ) e a temperatura inicial  $T_0$ . A Figura 3.5 apresenta o algoritmo *Simulated Annealing* básico.

```

Procedimento SA. ( $f(\cdot)$ ,  $N(\cdot)$ ,  $\alpha$ ,  $SA_{max}$ ,  $T_0$ ,  $s$ )
1.  $s^* \leftarrow s$ ;           {Melhor solução obtida até então}
2.  $IterT \leftarrow 0$ ;       {Número de iterações na temperatura T}
3.  $T \leftarrow T_0$ ;        {Temperatura corrente}
4. enquanto ( $T > 0$ ) faça
5.   enquanto ( $IterT < SA_{max}$ ) faça
6.      $IterT \leftarrow IterT + 1$ ;
7.     Gere um vizinho qualquer  $s' \in N(s)$ ;
8.      $\Delta = f(s') - f(s)$ ;
9.     se ( $\Delta < 0$ )
10.      então  $s \leftarrow s'$ ;
11.      se ( $f(s') < f(s^*)$ ) então  $s^* \leftarrow s'$ ;
12.      senão Tome  $x \in [0,1]$ ;
13.      se ( $x < e^{-\Delta/T}$ ) então  $s \leftarrow s'$ ;
14.    fim-se;
15.  fim-enquanto;
16.   $T \leftarrow \alpha * T$ ;
17.   $IterT \leftarrow 0$ ;
18. fim-enquanto;
19.  $s \leftarrow s^*$ ;
20. Retorne  $s$ ;
Fim S.A.:

```

Figura 2.1: Algoritmo *Simulated Annealing*

## 2.2 Desenvolvimento de Software

Várias tecnologias têm emergido nos últimos anos para atender às necessidades dos desenvolvedores de *software*, cada qual com seus problemas e vantagens. Isso torna a escolha de uma tecnologia um fator determinante na produção desse tipo de aplicação.

Como a manutenção e continuação do sistema desenvolvido ficarão provavelmente sob a responsabilidade de pessoas ligadas à UFOP, foram avaliadas as tecnologias mais populares num meio acadêmico e que atendessem bem aos propósitos de um sistema de grande sobrecarga computacional e que tivesse uma interface visual amigável. Desta forma, a escolha foi pelo ambiente de programação Borland C++ Builder Professional, Versão 6.0, que pode ser visto na Figura abaixo:

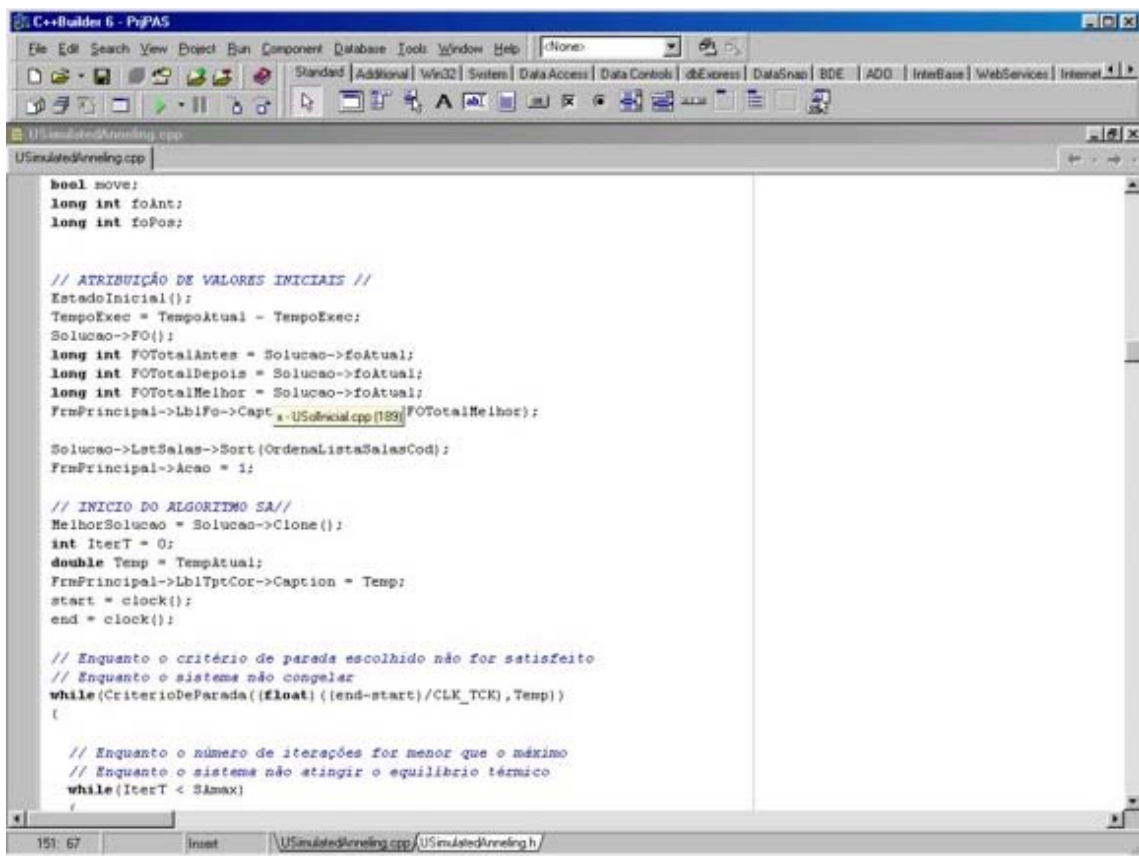


Figura 2.2 - C++ Builder 6.0

Como características importantes do C++ Builder pode-se citar:

- É um ambiente visual para o desenvolvimento de aplicações, com um vasto e ampliável conjunto de componentes visuais e funcionais que agilizam o processo de desenvolvimento.
- É uma poderosa linguagem de programação orientada a objetos.
- Possui integração com os principais SGBDs utilizados no mercado.

Além disso, o C++ Builder oferece um conjunto de facilidades como:

- O compilador C++, que cria módulos executáveis e DLLs, os quais rodam nos sistemas operacionais Windows 9x, Windows 2000 e Windows NT.
- O Object Inspector que auxilia a definição de propriedades e eventos relacionados aos componentes.
- A biblioteca de componentes VCL (*Visual Component Library*), com mais de 90 componentes prontos para as mais variadas funções.
- Vários assistentes para edição e construção de aplicações de forma rápida e simples como, por exemplo, o assistente para criação de aplicativos de banco de dados.
- O aplicativo Database Explorer, usado na criação e manutenção de tabelas de banco de dados.

- Acesso nativo a tabelas dos banco de dados Paradox, Dbase, Microsoft Access e FoxPro.
- O aplicativo BDE (*Borland Database Engine*) responsável pelo acesso ao banco de dados. É através do BDE que os componentes do C++ Builder se comunicam com o banco de dados.

No C++ Builder pode-se também criar componentes próprios, e integrá-los ao ambiente. Os novos componentes podem ser usados facilmente, da mesma forma que os já distribuídos com o C++ Builder.

A maioria dos intrincados aspectos da programação Windows fica encapsulada na implementação dos componentes reutilizáveis e distribuídos com o produto. Isso oferece um nível de abstração que simplifica, em alto grau, o trabalho de programação. Alguns tipos de aplicações podem ser desenvolvidos sem que seja necessário digitar uma única linha de código.

## 2.3 Orientação a Objetos

Orientação a objetos não é novidade no mundo da informática. Ainda na década de 70 já se estudava e aplicava este paradigma, especialmente na programação de sistemas. Entretanto seu uso como metodologia básica para o desenvolvimento de sistemas, abrangendo todo o ciclo desde a análise até a construção de códigos, é uma prática recente. Apenas na década de 80 surgiram os primeiros estudos sobre o uso da OO para especificação de projetos de sistemas. [LimaLima98]

A proposta da orientação a objetos baseia-se em conceitos importantes que definem novas formas de entender como se estruturam sistemas e programas, e como seus diversos módulos interagem entre si.

Para discutir esta técnica, são descritos inicialmente, seus conceitos básicos, conforme apresentados em [Rumbaugh97]:

### 2.3.1 Objeto

Objeto é qualquer coisa, real ou abstrata que possui um **estado**, determinado por um conjunto de dados, e possui um conjunto de **operações**, para consultar ou modificar esse estado. A implementação de cada uma dessas operações é codificada como um **método**. Uma solicitação ou chamada de um método ativa a operação correspondente, usando um ou mais objetos como parâmetros. Por exemplo, uma garrafa é um objeto que tem um estado (ela pode estar vazia ou cheia e pode ter conteúdos diferentes) e que permite algumas operações como enche-la, esvazia-la, abri-la, fecha-la e mudar seu conteúdo.

### 2.3.2 Classe

Uma classe é uma descrição de um grupo de objetos, cada um com um estado específico, mas capazes de realizar as mesmas operações. Representa a implementação de um determinado tipo de objeto, incluindo o conjunto de dados que representa o seu estado e os métodos que implementam as operações definidas sobre estes objetos.

### 2.3.3 Relacionamentos

As classes de um sistema, em geral colaboram entre si de várias maneiras. Essa colaboração entre classes é chamada de relacionamento.

Por exemplo, ao construir uma casa poderiam ser identificadas classes de objetos como parede, janela e porta, que não funcionam separadamente. Uma porta está ligada a uma parede e uma janela também. Nesse caso existem relacionamentos entre as classes parede, porta e janela.

### 2.3.4 Princípios básicos

A orientação a objetos é baseada em quatro princípios básicos [Rumbaugh97], que são:

- **Abstração de Dados:** é a habilidade de ignorar os aspectos de um assunto não relevantes para o propósito em questão, tornando possível uma concentração maior nos assuntos principais. [Oxford 86]
- **Encapsulamento:** ou ocultamento de informações é restringir o escopo ou visibilidade da informação deixando exposto somente aquilo que é necessário.
- **Herança:** é o mecanismo para expressar a similaridade entre classes, simplificando a definição de classes iguais a outras que já foram definidas. A herança permite reutilizar classes acrescentando funcionalidades ou modificando o seu comportamento.
- **Polimorfismo:** consiste em criar tipos de dados que oferecem serviços de mesmo nome, ou seja, diferentes classes podem definir métodos de mesmo nome. O polimorfismo é importante por tornar as classes mais independentes entre si e por permitir que novas classes sejam incluídas num sistema com mínimo impacto sobre as classes já existentes.

### 2.3.5 Benefícios da Orientação a Objetos

Os sistemas orientados a objetos podem representar melhor o mundo real, uma vez que a percepção e o raciocínio do ser humano estão relacionados diretamente com o conceito de objetos. Esse fato permite uma modelagem mais perfeita e natural. Além dessa característica, a Orientação a Objetos oferece outros benefícios descritos por [LimaLima98]:

- A mesma notação é usada desde a análise até o projeto e a implementação, de modo que a informação adicionada em uma etapa do desenvolvimento não é necessariamente perdida ou traduzida para a etapa seguinte.
- Esta abordagem incentiva os desenvolvedores a trabalharem e pensarem em termos do domínio da aplicação durante a maior parte do ciclo de vida, ou seja, dedicação maior à fase de análise. A parte mais difícil do desenvolvimento de software é a manipulação de sua essência face à inerente complexidade do problema, e não os acidentes de seu mapeamento em uma determinada linguagem, que são devidos a imperfeições temporárias das ferramentas, que estão sendo rapidamente corrigidas.
- Ocorre uma redução na quantidade de erros com conseqüente diminuição do tempo despendido nas etapas de codificação e teste, visto que os problemas são detectados mais cedo e corrigidos antes da implementação.

- Melhora a comunicação entre desenvolvedores e usuários já que o sistema refletirá o modelo do negócio. Os modelos na orientação a objetos espelham a estrutura e o comportamento dos objetos do negócio, diminuindo o abismo existente nas outras abordagens que tratam dados e funções separadas.
- No desenvolvimento baseado em objetos há uma redução no tempo de manutenção, pois as revisões são mais fáceis e mais rápidas já que o problema é mais bem localizado.
- Favorece a reutilização, já que ocorre a construção de componentes mais gerais, estáveis e independentes.
- A cada dia os sistemas estão mais complexos e sujeitos a alterações freqüentes. Essa tecnologia suporta bem a construção desse tipo de sistema.
- Facilidade de extensão, visto que objetos têm sua interface bem definida. A criação de novos objetos que se comunicam com os já existentes não obriga o desenvolvedor a conhecer o interior destes últimos.

## 2.4 Arquitetura Cliente Servidor

Essa arquitetura é bem detalhada por [Queiroz00]. Em suma tem-se:

A arquitetura cliente/servidor é baseada na divisão do trabalho a ser realizado por uma aplicação entre dois processos, chamados de **cliente** e **servidor**. É necessário ressaltar que os termos cliente e servidor não devem ser confundidos com os computadores que rodam os processos clientes e servidores respectivamente.

O processo cliente é caracterizado por rodar numa estação cliente e fornecer a interface gráfica com o usuário, conhecida como *Graphics User Interface* (GUI), fazer requisições a serviços específicos a serem realizados em máquinas remotas e executar parte do código da aplicação. Enquanto o processo servidor caracteriza-se por rodar em máquinas, geralmente mais poderosas do que as estações clientes, tendo grande quantidade de memória, executar um conjunto de funcionalidades relacionadas a serviços, que geralmente necessitam de componentes de hardware/software especializados. Os servidores são, via de regra, passivos, apenas “escutando” as requisições dos clientes, executando-as e então respondendo.

O protocolo utilizado entre clientes e servidores é conhecido como requisição-resposta, desta maneira, um cliente envia uma requisição e o servidor a responde. A figura abaixo ilustra a idéia do protocolo requisição-resposta.

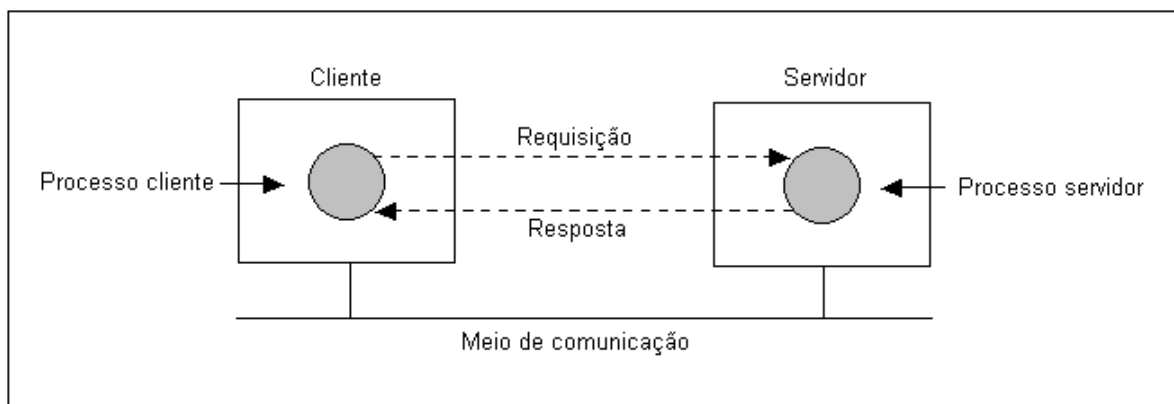


Figura 2.3 - Esquema do protocolo de requisição

Como exemplos de servidores pode-se citar:

- **Servidor de banco de dados:** fornece acesso a um ou mais bancos de dados. As requisições do cliente são da forma de declaração SQL (*Structured Query Language*) – uma linguagem padrão para manipulação de dados em banco de dados relacionais. O servidor de banco de dados recebe a declaração SQL e envia ao Sistema Gerenciador de Banco de Dados (SGBD), que consulta ou atualiza o banco de dados, dependendo do tipo de requisição. Mesmo que o SGBD tenha que ler muitos registros para satisfazer a uma consulta do cliente de poucos registros, apenas o resultado é enviado de volta ao cliente. Isso faz com que o tráfego do meio de comunicação diminua.
- **Servidor de aplicação:** fornece acesso a procedimentos invocados remotamente pelos clientes através de mecanismos de chamada remota a procedimentos (RPC). Esses procedimentos remotos encapsulam a lógica do negócio da aplicação e o acesso ao banco de dados.
- **Servidor Web:** fornece acesso a documentos do serviço Web da Internet como, por exemplo, textos, imagens, vídeos e sons.

Costuma-se classificar a arquitetura cliente/servidor em sistemas de duas ou arquitetura de três camadas.

Em duas camadas, a GUI e a lógica da aplicação rodam no cliente enquanto que o servidor de banco de dados roda no servidor. Em uma organização de 3 camadas, a GUI roda no cliente, a lógica da aplicação roda em um servidor de aplicação que atua como um cliente para o servidor de banco de dados.



## 3 Metodologia

### 3.1 Descrição do Problema

#### 3.1.1 O Problema de Alocação de Salas Abordado

O problema considerado para análise é o do Instituto de Ciências Exatas e Biológicas (ICEB) da Universidade Federal de Ouro Preto (UFOP). Trata-se de um instituto que recebe, em média, 1200 alunos por semestre e que oferece cerca de 250 turmas de disciplinas em aulas nos horários matutino, vespertino e noturno. As aulas são realizadas de segunda a sábado à tarde, mas a maioria delas está concentrada de terça a quinta-feira. Para a realização das aulas das turmas estão disponíveis 20 salas de aulas e 29 laboratórios específicos. Os horários de aula das turmas são confeccionados previamente pelos departamentos e encaminhados à diretoria do instituto para que esta faça a alocação das turmas às salas. As aulas que requerem laboratórios são alocadas pelos próprios departamentos e não fazem parte do problema objeto deste trabalho.

No processo de alocação de aulas a salas no ICEB, são observados os seguintes requisitos, dentre outros de menor importância que ainda não foram implementados: (a) Em uma mesma sala e horário não pode haver mais de uma aula; (b) Uma sala não pode receber uma turma cuja quantidade de alunos seja superior à sua capacidade; (c) Algumas salas têm alguns horários previamente reservados para a realização de outras atividades e nesses horários elas ficam indisponíveis; (d) Certas salas têm restrições de uso e a utilização delas deve ser evitada tanto quanto possível; (e) Sempre que possível, alocar a uma mesma sala alunos de um mesmo curso e período; (f) Utilizar o espaço das salas eficientemente, isto é, evitar alocar aulas de turmas pequenas em salas de maior capacidade; (g) Se possível, cada uma das salas deve ser deixada vazia em pelo menos um horário ao longo do dia, de forma a possibilitar sua limpeza.

A alocação de salas no ICEB é feita manualmente e é um processo que se inicia normalmente duas semanas antes do início do período letivo, com base em uma pré-matrícula feita pela própria instituição. Dois dias antes do início das aulas, os alunos têm direito a fazer um ajuste de matrícula. Esta é uma situação que agrava o problema, uma vez que com este ajuste muitas turmas são reduzidas ou ampliadas ou mesmo criadas e as alocações anteriormente feitas têm que ser modificadas. Como não há tempo hábil para efetuar manualmente todas essas correções, em geral o semestre letivo se inicia com uma distribuição de salas que não agrada. Em muitos casos, turmas de 50 alunos são alocadas em salas com capacidade para 40 alunos! A correção dessas distorções ocorre ao longo das primeiras semanas do período letivo e é sempre motivo de transtorno, porque para fazer essas correções é necessário realocar as aulas de várias outras turmas.

### 3.2 Modelagem do problema

#### 3.2.1 Representação

Uma alocação (solução) do problema é representada por uma matriz  $S = (s_{ij})_{m \times n}$ , onde  $m$  representa o número de horários reservados para a realização das aulas e  $n$  o número de salas disponíveis. Em cada célula  $s_{ij}$  é colocado o número da turma  $t$  alocada ao horário  $i$  e sala  $j$ . Uma célula vazia indica que a sala  $j$  está desocupada no horário  $i$ . Um exemplo simples de representação é dado pela Figura 4.1.

Esta figura mostra, por exemplo, que na sala 4 os horários 2 a 4 e 6 e 7 estão ocupados com aulas das turmas 6 e 9, respectivamente. Nos demais horários esta sala está desocupada.

		Salas				
		1	2	3	4	5
Horários	1	3				4
	2	3		1	6	4
	3	3	5		6	7
	4		5	2	6	7
	5	12		2		
	6	12	13	11	9	
	7		13	11	9	10
	8	8		11		10
	9	8				10

Figura 3.1 – Exemplo de uma alocação

### 3.2.2 Estrutura de vizinhança

Dada uma solução  $s$ , para atingir uma solução  $s'$ , onde  $s'$  é dito vizinho de  $s$ , são usados dois tipos de movimento: Realocação e Troca.

O movimento de realocação consiste em realocar as aulas de uma dada turma e sala a uma outra sala que esteja vazia nos horários de realização das aulas. Para a realização desse movimento é exigido que a sala que receberá as aulas de uma turma esteja disponível nos horários das aulas. Este tipo de movimento é ilustrado na Figura 3.2.

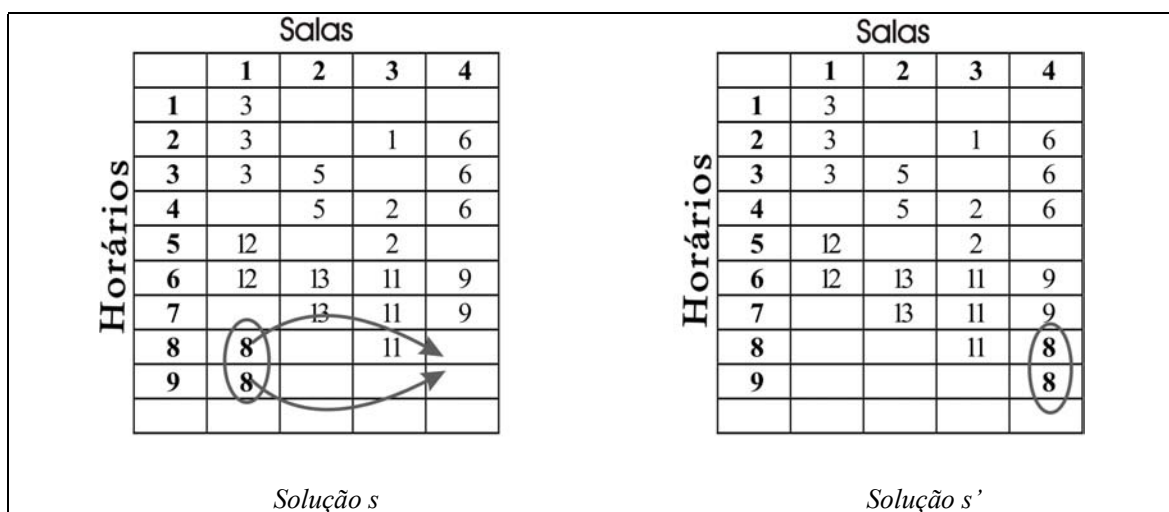


Figura 3.2 - Movimento de Alocação

Nesta figura, as aulas da turma 8 realizadas nos horários 8 e 9 na sala 1 são transferidas para a sala 4.

Já o movimento de troca consiste em trocar de sala as aulas de duas turmas realizadas em um mesmo bloco de horários. Este tipo de movimento é ilustrado na Figura 3.3.

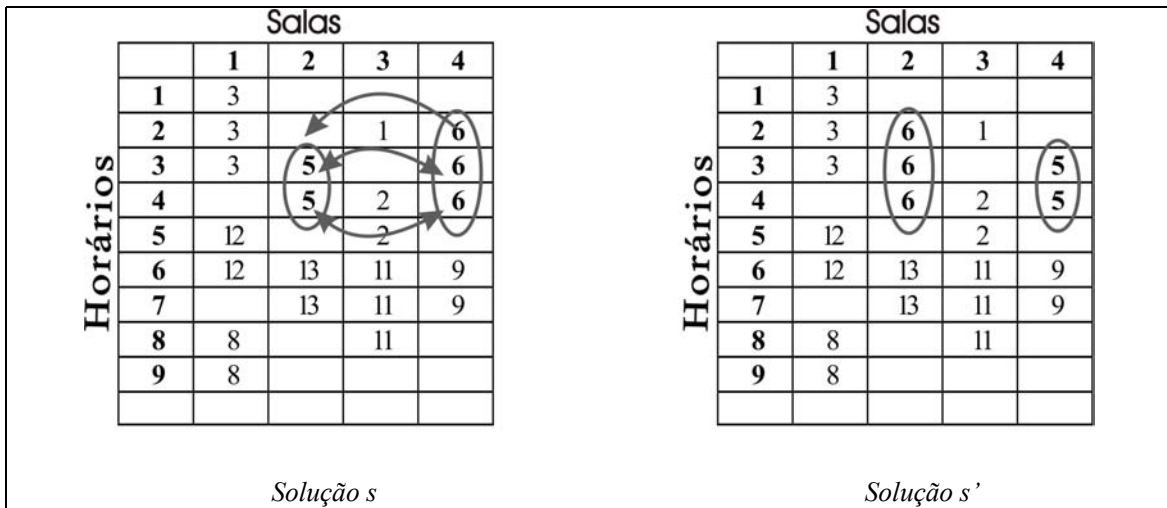


Figura 3.3 - Movimento de Troca

Nesta figura, as aulas das turmas 5 e 6 são permutadas de sala. As aulas da turma 5 realizadas na sala 2 nos horários 3 e 4 são transferidas para a sala 4, enquanto que as aulas da turma 6 realizadas na sala 4 nos horários 2, 3 e 4 são transferidas para a sala 2.

Para a realização desse movimento exige-se que nos horários envolvidos as salas estejam vazias ou com aulas apenas das turmas relacionadas com a operação. Desta forma, não é permitido, por exemplo, permutar as aulas das turmas 2 e 6, porque no horário 2 a sala 3 está ocupada com uma aula da turma 1, impedindo que as aulas da turma 6 sejam transferidas da sala 4 para a sala 3.

Na estrutura de vizinhança  $N(s)$  considerada, diz-se que uma solução  $s' \in N(s)$  é um vizinho de  $s$  se ela pode ser acessada a partir desta através de um movimento ou de realocação ou de troca.

### 3.2.3 Função objetivo

Para avaliar uma alocação, os requisitos do problema são primeiramente divididos em duas categorias: (i) requisitos essenciais, que são aqueles que se não forem satisfeitos, gerarão uma alocação inviável; (ii) requisitos não-essenciais, que são aqueles cujo atendimento é desejável mas que, se não satisfeitos, não gerarão alocações inviáveis. No caso em análise, são essenciais os requisitos (a), (b) e (c) descritos na seção 2 e não-essenciais os requisitos (d), (e), (f) e (g).

Desse modo, uma alocação (ou solução)  $s$  pode ser medida com base em duas componentes, uma de inviabilidade ( $g(s)$ ), a qual mede o não atendimento aos requisitos essenciais, e outra de qualidade ( $h(s)$ ), a qual avalia o não atendimento aos requisitos considerados não-essenciais. A função objetivo  $f$  que associa cada solução  $s$  do espaço de soluções a um número real  $f(s)$ , e que deve ser minimizada, pode ser expressada, portanto, na forma:

$$f(s) = g(s) + h(s) \tag{1}$$

A componente  $g(s)$ , que mensura o nível de inviabilidade de uma solução  $s$ , é avaliada com base na expressão:

$$g(s) = \sum_{k=1}^K \alpha_k I_k \quad (2)$$

onde  $K$ : número de medidas de inviabilidade;  
 $I_k$ : valor da  $k$ -ésima medida de inviabilidade.  
 $\alpha_k$ : peso associado à  $k$ -ésima medida de inviabilidade;

Já a componente  $h(s)$ , que mensura a qualidade de uma solução  $s$ , é avaliada por:

$$h(s) = \sum_{l=1}^L \beta_l Q_l \quad (3)$$

onde  $L$ : número de medidas de qualidade;  
 $Q_l$ : valor da  $l$ -ésima medida de qualidade;  
 $\beta_l$ : peso associado à  $l$ -ésima medida de qualidade;

Deve ser observado que uma solução  $s$  é viável se e somente se  $g(s) = 0$ . Nas componentes da função  $f(s)$  os pesos dados às diversas medidas refletem a importância relativa de cada uma delas e, sendo assim, deve-se tomar  $\alpha_k \gg \beta_l \quad \forall k, l$ , de forma a privilegiar a eliminação das soluções inviáveis. Observa-se, também, que algumas das medidas de qualidade são conflitantes com outras como, por exemplo, encontrar uma utilização eficiente de espaço (requisito (f)) e agrupar todas as aulas semanais dos alunos de um mesmo curso e período em uma mesma sala de aula (requisito (e)). Esse conflito aparece porque muitas disciplinas são compartilhadas com vários cursos, enquanto outras são ministradas apenas para um determinado curso. Dessa forma, as turmas das disciplinas são geralmente de tamanhos diferentes. Entretanto, faz-se necessário estabelecer, através de uma relação de pesos, um compromisso adequado de atendimento aos requisitos envolvidos.

### 3.3 Descrição do método utilizado

#### 3.3.3 Geração de uma solução inicial

Uma solução inicial para o problema de alocação de salas é gerada por um procedimento construtivo que segue as idéias da fase de construção do método GRASP (Feo and Resende 1995).

Inicialmente, uma lista de Salas e outra de Turmas são criadas e preenchidas respectivamente com os registros de Salas e Turmas do Banco de Dados do sistema. Após este passo, as salas são ordenadas de acordo com sua capacidade assim com as turmas são ordenadas de acordo com sua demanda. Posteriormente, um inicia-se um *loop* onde a cada iteração toma-se a aula ainda não alocada da turma com maior demanda e escolhe-se uma das salas vagas nos horários da aula, ordenadas pela capacidade, então, a turma é alocada na sala disponível que foi escolhida. Caso não exista uma sala vaga no horário, uma sala virtual é criada e a aula é alocada nesta sala. Desta forma, quando todas as turmas já estiverem alocadas em alguma sala o *loop* é interrompido e a função finalizada.

```

Procedimento ConstruaSolucaoInicial ( )
1. TSala Sala;                               { cria um objeto Sala}
2. TTurma Turma;                             { cria um objeto Turma}
3. preencherLstSalas();                       {Busca Salas do Banco de Dados}
4. preencherLstTurmas();                      {Busca Turmas do Banco de Dados}
5. ordenarLstSalas();                         {ordena Salas por capacidade}
6. ordenarLstTurmas();                       {ordena Turmas por demanda}
7. enquanto (LstTurmas >0) faça              {enquanto existir turmas não alocadas}
8.     Turma = LstTurmas -> Get Item;        { Pegar a Turma de maior demanda}
9.     Sala = salaVazia();                   {achar uma sala disponível para a alocação, ou cria uma sala virtual}
10.    AlocarTurma (Turma, Sala);            { alocar a turma a sala disponível ou a sala virtual criada}
11. fim-enquanto;
12. retorne s;                                {retorna a solução criada}
13. Fim SolucaoInicial;

```

Figura 3.4 - Algoritmo de Construção da Solução Inicial.

### 3.3.4 Algoritmo proposto para a resolução do Problema de Alocação de Salas

O algoritmo proposto é um método de 2 fases. Na primeira fase constrói-se uma solução inicial (seção 3.3.1) se esta for necessária, já que como o sistema disponibiliza a opção do usuário salvar uma solução anteriormente criada é possível carregá-la para nova execução e refinamento desta sem precisar criar um nova solução inicial. Na segunda fase, essa solução é submetida ao Algoritmo *Simulated Annealing* (seção 3.3.4), onde o objetivo é refinar a solução inicial fazendo com que o valor da função objetivo seja o menor possível, ou seja, minimizando a relação de diferença entre a capacidade de uma sala e a demanda das turmas a ela alocadas, representada pelo excesso e folga de alunos nas salas, bem como maximizando o numero de aulas semanais de uma mesma turma alocadas na mesma sala, além de tentar evitar ao máximo o uso de salas virtuais, visando assim atender os requisitos essenciais e não essenciais, fazendo com que o valor da função objetivo seja minimizado de forma bastante satisfatória. o quanto for possível.

O pseudocódigo do algoritmo proposto é apresentado na figura abaixo.

```

Procedimento Executar ( )
1. se (Ação = Iniciar)           {se uma solução inicial precisa ser criada}
2.     s ← ConstruaSolucaoInicial( );
3. s* ← SA(s0);
4. fim Executar;

```

Figura 3.5: Algoritmo proposto: Solução Inicial + *Simulated Annealing*

### 3.3.5 A Arquitetura Cliente/Servidor do PAS

Visto que o sistema desenvolvido possa admitir futuramente o acesso aos dados de que necessita via acesso direto a um servidor da UFOP, foi considerado viável implementar já na primeira versão uma arquitetura cliente servidor, para acesso ao banco de dados, mesmo não

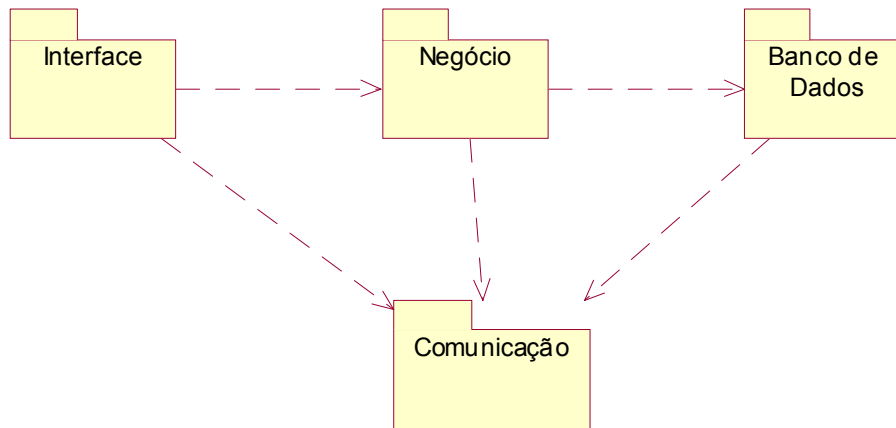
atuando agora como um sistema Cliente/Servidor. Além disso, a escolha dessa arquitetura deve-se aos seguintes motivos:

- Proporcionar uma boa organização do *software*, pois assim o entendimento e manipulação dos componentes de cada camada tornam-se mais fáceis.
- Facilitar a implementação do sistema, pois ao se trabalhar em um determinada camada não é preciso conhecer em detalhes o funcionamento das demais. Basta a princípio conhecer como é feita a interface entre as camadas.
- Tornar futuras manutenções do sistema mais rápidas e fáceis, pois sendo as camadas bem independentes as alterações feitas em uma delas não devem exigir mudanças nas outras.

As decisões de projeto tomadas tiveram como objetivo adotar um padrão orientado à objetos para modularizar e facilitar a manutenção dos sistemas. Com isso, também haverá uma redução da redundância de código através do reaproveitamento de métodos e classes de outros sistemas. Apesar desse modo de desenvolvimento requerer um tempo maior para finalização do projeto, a manutenção (tarefa mais trabalhosa) será facilitada inúmeras vezes.

Para tanto foi montada uma estrutura que se divide em 4 camadas, como é mostrado e exemplificado nas figuras abaixo:

### Arquitetura de Software utilizada pelo PAS para acesso ao Banco de Dados



#### Decisões de Projeto:

No projeto do sistema foram criados 4 pacotes, cada um representando uma camada do sistema:

O pacote Interface contém todas as classes da interface.

O pacote Negócio contém todas as classes da camada de negócio.

Elas contêm apenas métodos a serem chamados pela interface.

O pacote Banco de Dados contém todas as classes da camada de Banco de Dados. Elas contêm os métodos a serem chamados pelo Negócio.

O pacote Comunicação contém as classes que as outras camadas dependem. Estas contêm os atributos que serão utilizados para realizar as ações

do sistema (inserir, alterar, consultar, excluir) relacionadas ao acesso ao Banco de Dados. Esta camada é responsável pela interação entre as demais camadas.

A estrutura citada acima nos garante uma dependência unidirecional, o que nos permite alterar uma camada sem que a camada posterior sofra algum efeito

colateral dessa transformação. Isto garante ao sistema uma grande flexibilidade, bem como uma grande reutilização de código.

Figura 3.6 - Arquitetura de Software utilizada

**INTERFACE:** Na camada Interface estão situadas as classes de interface com os clientes, com os seus respectivos métodos.

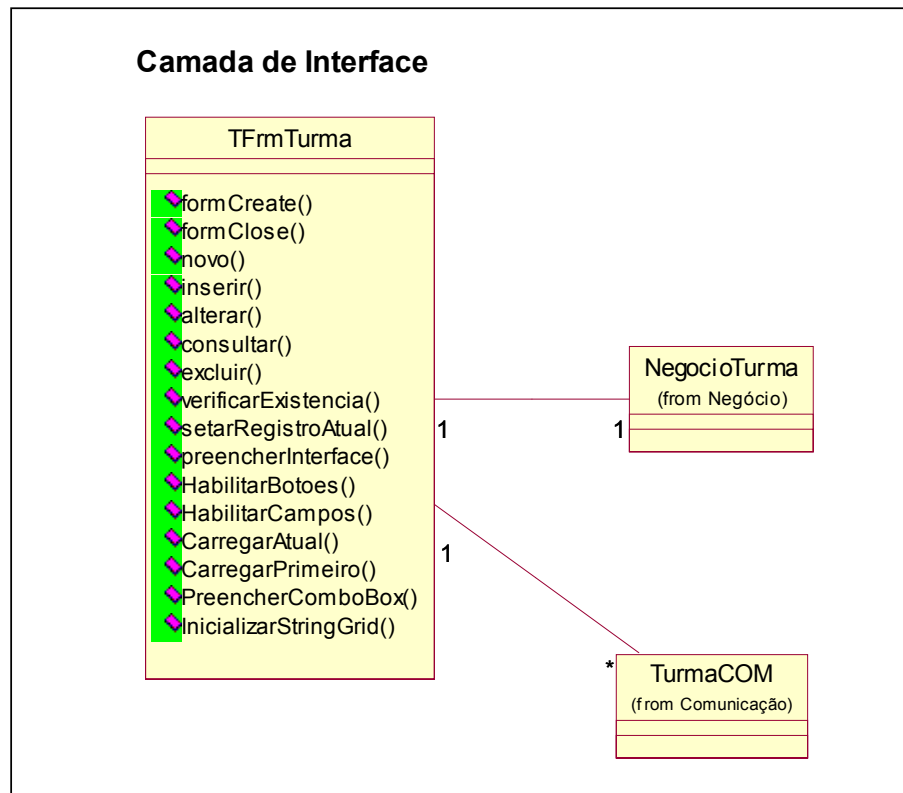


Figura 3.7 - Camada de Interface

**NEGÓCIO:** Na camada Negócio estão situadas as classes ligadas à Interface e ao Banco de Dados que contém os métodos a serem chamados pela interface.

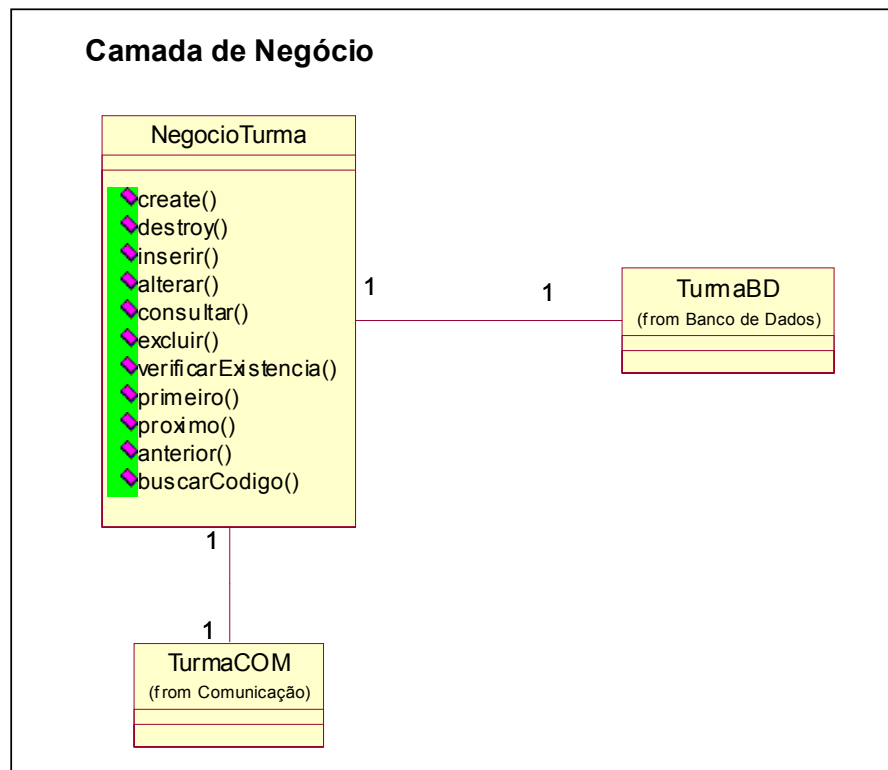


Figura 3.8 Camada de Negócio



**BANCO DE DADOS:** Na camada de Banco de Dados estão os métodos a serem chamados pela camada de Negócio.

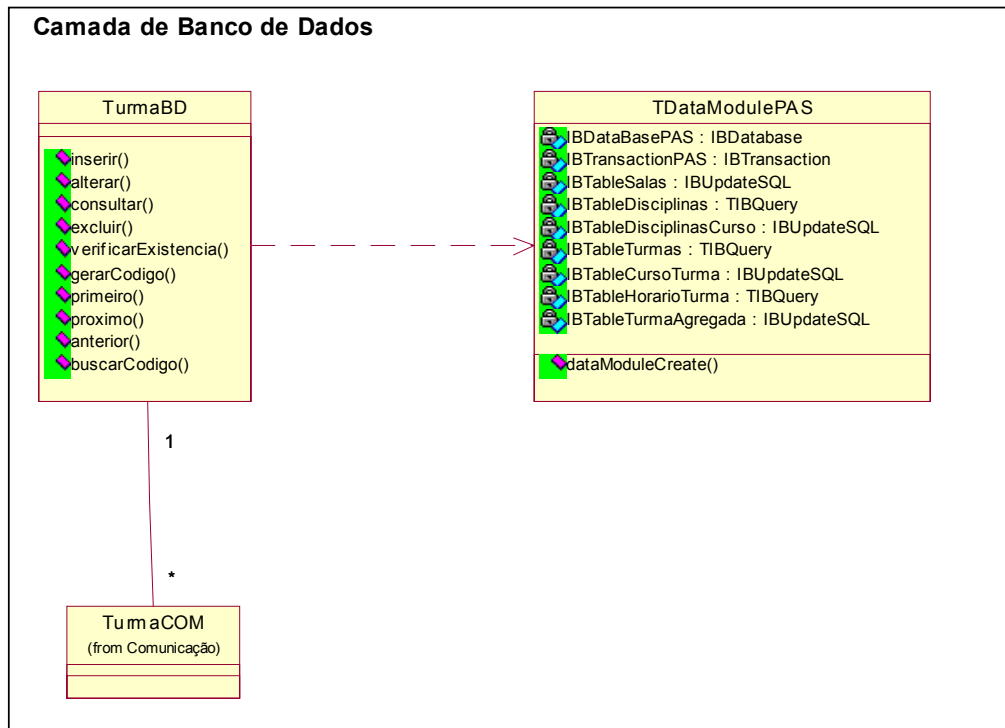


Figura 3.9 - Camada de Banco de Dados

**COMUNICAÇÃO:** Na camada de Comunicação estão as classes que contém os atributos relativos aos campos das tabelas, cada atributo contém seus respectivos métodos Get e Set, como todas as outras camadas dependem destas classes ela possui ligação com todas elas e possibilita a interação entre estas camadas, por isso o nome Comunicação.

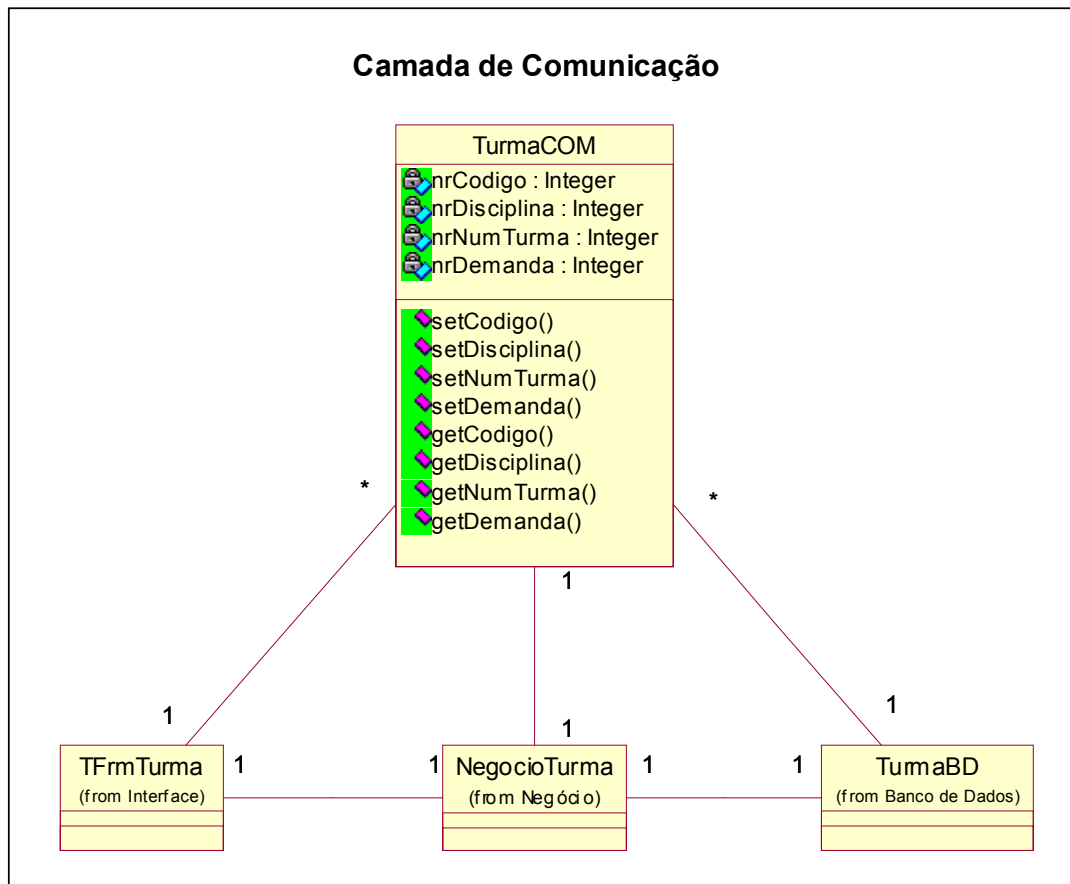


Figura 3.10 - Camada de Comunicação

## 4 Sistema Desenvolvido

### 4.1 Descrição do sistema

O sistema desenvolvido tem o propósito de tentar solucionar o Problema da Alocação de Salas da Universidade Federal de Ouro Preto – UFOP. Ele foi desenvolvido na linguagem C++ com a ferramenta C++ Builder 6.0 e Projeto Mínimo de Banco de Dados InterBase Server. Isso devido ao poder e desempenho da linguagem e a disponibilidade e recursos da ferramenta. O mesmo foi implementado segundo o paradigma de Orientação a Objetos.

O sistema tenta oferecer um ambiente de uso agradável, flexível e aplicável semestralmente pela instituição alvo. Para isso, o sistema foi desenvolvido com uma interface bem parecida com a de “softwares comuns”, de maneira que não necessite de um usuário “especial” para operá-lo. Ele permite ainda que o usuário faça alterações de valores que mudam frequentemente como, por exemplo, permitir a agregação e desagregação de turmas, as quais mudam praticamente todo semestre.

O funcionamento básico do Sistema aplicado ao Problema da Alocação de Salas abordado pode ser mais bem descrito em quatro etapas específicas:

(1) *Entrada dos Dados*: Nessa etapa o usuário deve entrar, utilizando um recurso de carregamento automático dos arquivos em formato de texto, com os dados referentes aos horários das turmas, demandas, disciplinas, cursos e seus respectivos períodos. Feito esse carregamento, o usuário poderá também realizar um controle manual dos dados podendo inclusive efetuar novas entradas, consultas, alterações e exclusões dos dados previamente carregados automaticamente. É importante ressaltar que essa etapa inclui também, a possibilidade de o usuário realizar *AGREGAÇÕES* de turmas, ou seja, o agrupamento e desagrupamento de possíveis turmas (mesmo que possuam nomes diferentes) candidatas a dividirem em um mesmo horário e nos mesmos dias uma mesma sala de aula.

(2) *Especificação dos Parâmetros*: Nessa etapa o usuário deve entrar com os parâmetros relativos ao método de otimização utilizado pelo sistema (SA) tais como número máximo de iterações, taxa de resfriamento, temperatura inicial, temperatura de congelamento, tempo máximo de processamento e critérios de parada do sistema. O sistema possibilita ao usuário optar nessa etapa por especificar os valores padrões do sistema para os parâmetros em questão e caso ignore esta etapa os valores padrões serão assumidos durante a execução do método de otimização.

(3) *Especificação das Penalizações*: Nessa etapa o usuário deve entrar com os pesos para cálculo das penalizações específicas ao problema da alocação de salas abordado tais como excesso de alunos na sala, sala com folga na capacidade e mesma turma alocada em salas diferentes. O sistema possibilita ao usuário optar também nessa etapa por especificar os valores padrões do sistema para os pesos em questão e assim como na especificação dos parâmetros, estes valores serão assumidos durante a execução do método se o usuário ignorar esta etapa..

(4) *Execução do Método de Otimização*: Terminada as etapas anteriores, o usuário poderá realizar a execução do sistema, o qual utilizará o método Simulated Annealing em uma nova solução inicial ou em uma solução criada anteriormente para solucionar o problema da alocação de salas abordado. Durante esse processo o usuário poderá interromper a execução.

Além da execução básica demonstrada é válido ressaltar que após a execução do SA, será exibido na tela principal do sistema o resultado encontrado, no qual o usuário poderá ver a alocação de salas encontrada, fazer trocas manuais de turmas entre salas, salvar a solução e até mesmo executar uma nova solução ou outra anteriormente salva em arquivo. Além disso, sempre que o usuário finaliza o “PAS Solver” é criado automaticamente um arquivo do Excel (no diretório C:\solução.out), que mostra de forma clara a alocação gerada pelo sistema.

## 4.2 Sistema aplicado ao PAS

Visando modelar da melhor forma possível o sistema em relação ao PAS, a implementação do mesmo é fundamentada basicamente em três entidades: Turma, Sala e Solução (Alocação).

- i) **Turma:** Essa entidade tem a finalidade de representar as turmas que necessitam serem alocadas. Essa representação é feita a partir de uma classe chamada *TTurma* cujos atributos representam os valores que compõem uma turma. Essa representação é feita segundo o modelo descrito na Figura 4.1.

TURMA	
Turma	Demanda
Dia	Horário
Agregada	

Figura 4.1 – Representação de uma Turma

*Turma:* código da turma no Banco de Dados

*Demanda:* número de alunos inscritos na turma

*Dia:* O respectivo dia de aula desta turma ( 1 = segunda-feira, 2= terça-feira, ...)

*Horário:* O respectivo horário de aula desta turma ( 1 = 07:30, 2 = 08:20, 3 = 09:30,...)

*Agregada:* indica se a turma esta agregada a outras turmas ( 0 = não agregada; 1 = agregada);

- ii) **Sala:** Essa entidade tem a finalidade de representar as salas que receberão as turmas. Essa representação é feita a partir de uma classe chamada *TSala* cujos atributos representam os valores que compõem uma sala. Essa representação é feita segundo o modelo descrito na Figura 4.2.

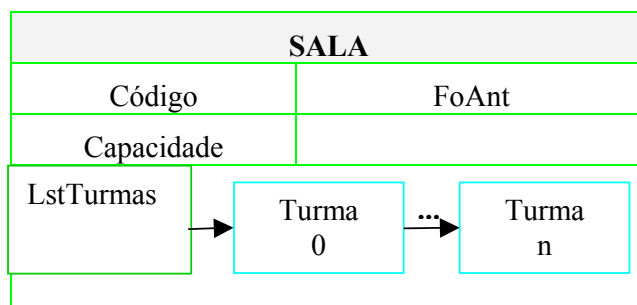


Figura 4.2 – Representação de uma Sala

*Código*: código da sala no Banco de Dados

*Capacidade*: número de vagas que a sala oferece

*FoAnt*: Valor da função objetivo (custo) de alocação entre a respectiva sala e as turmas que esta aloca.

*LstTurmas*: Lista de objetos do tipo Turma (i) que estão alocados na respectiva sala.

iii) **Solução**: Essa entidade tem a finalidade de representar as alocações encontradas em suas respectivas salas e turmas a elas alocadas. Essa representação é feita a partir de uma classe chamada *TSolucao*.

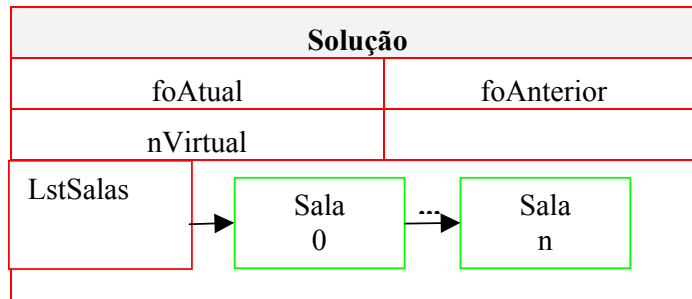


Figura 4.3 – Representação de uma Solução

*foAtual*: Valor da função objetivo atual que avalia a alocação recém criada.

*FoAnterior*: Valor da função objetivo corrente.

*nVirtual*: Número de Salas virtuais criadas na respectiva pela respectiva solução.

*LstSalas*: Lista de objetos do tipo Salas (ii) que contém as Salas e suas respectivas Turmas alocadas.

### 4.3 Interação Sistema X SA

No sistema, o algoritmo do SA sofre algumas modificações em relação ao básico, citado na Figura 3.5. Uma dessas é possibilitar que o algoritmo execute por um tempo determinado pelo usuário, e a principal é a interrupção da execução do algoritmo a qualquer momento.

Assim, o algoritmo utilizado no sistema está descrito na Figura 4.4.

```

Procedimento S.A. ( $f(\cdot)$ ,  $N(\cdot)$ ,  $\alpha$ ,  $S_{Amax}$ ,  $T_0$ ,  $s$ ,  $TMP$ ,  $TC$ )
1.  $s^* \leftarrow s$ ;           {Melhor solução obtida até então}
2.  $IterT \leftarrow 0$ ;       {Número de iterações na temperatura T}
3.  $T \leftarrow T_0$ ;        {Temperatura corrente}
4. inicia a contagem do TempoCorrente
5. enquanto ( $T > TC$ ) ou ( $TMP > TempoCorrente$ ) ou (Parar = true) faça
6.   enquanto ( $IterT < S_{Amax}$ ) faça
7.     se (Parar = true)
8.       então break;
9.      $IterT \leftarrow IterT + 1$ ;
10.    Escolho um movimento
12.    Realizo o movimento e Gera um vizinho qualquer  $s' \in N(s)$ ;
13.     $\Delta = f(s') - f(s)$ ;
14.    se ( $\Delta < 0$ )
15.      então  $s \leftarrow s'$ ;
16.      se ( $f(s') < f(s^*)$ ) então  $s^* \leftarrow s'$ ;
17.      senão Tome  $x \in [0,1]$ ;
18.      se ( $x < e^{-\Delta/T}$ ) então  $s \leftarrow s'$ ;
19.      senão desfazer movimento;
20.    fim-se;
21.  fim-enquanto;
22.   $T \leftarrow \alpha * T$ ;
23.   $IterT \leftarrow 0$ ;
24. fim-enquanto;
25.  $s \leftarrow s^*$ ;
26. Retorne  $s$ ;
27. Fim S.A.;

```

Figura 4.4 - Algoritmo utilizado no sistema

#### Principais alterações:

- *Linha 5*: Verifica se o sistema congelou ( $T > TC$ ) ou se o tempo máximo de processamento já foi atingido ( $TMP > TempoCorrente$ ) ou se o comando Interromper Execução foi acionado.  $T$  é a temperatura corrente e  $TC$  é a temperatura de congelamento,  $TMP$  é o tempo máximo de processamento, ou seja, momento em que o algoritmo pára de executar e  $TempoCorrente$  é o momento corrente do sistema e Parar é a variável booleana que assume o valor *true* quando o usuário deseja interromper a execução;
- *Linha 7*: Verifica se o usuário interrompeu a execução do SA e então sai do *loop* corrente;

## 4.4 Recursos oferecidos pelo sistema

### 4.4.1 Iniciação do Sistema

O Sistema tem início com a apresentação da Tela Inicial (representada pela Figura 4.5). Nesta tela o usuário, além de ter uma visão geral da estrutura de quadro de horários em que as salas serão alocadas, tem também a possibilidade de escolher dentre as opções descritas numa Barra de Menu (localizada na parte superior da Figura 4.5), quais recursos do sistema deseja explorar. Há também a possibilidade de acessar os comandos mais usuais (Nova Solução, Executar, Interromper Execução, Carregar Solução, Salvar Solução e Sair do sistema) através de botões localizados abaixo do quadro de horários.

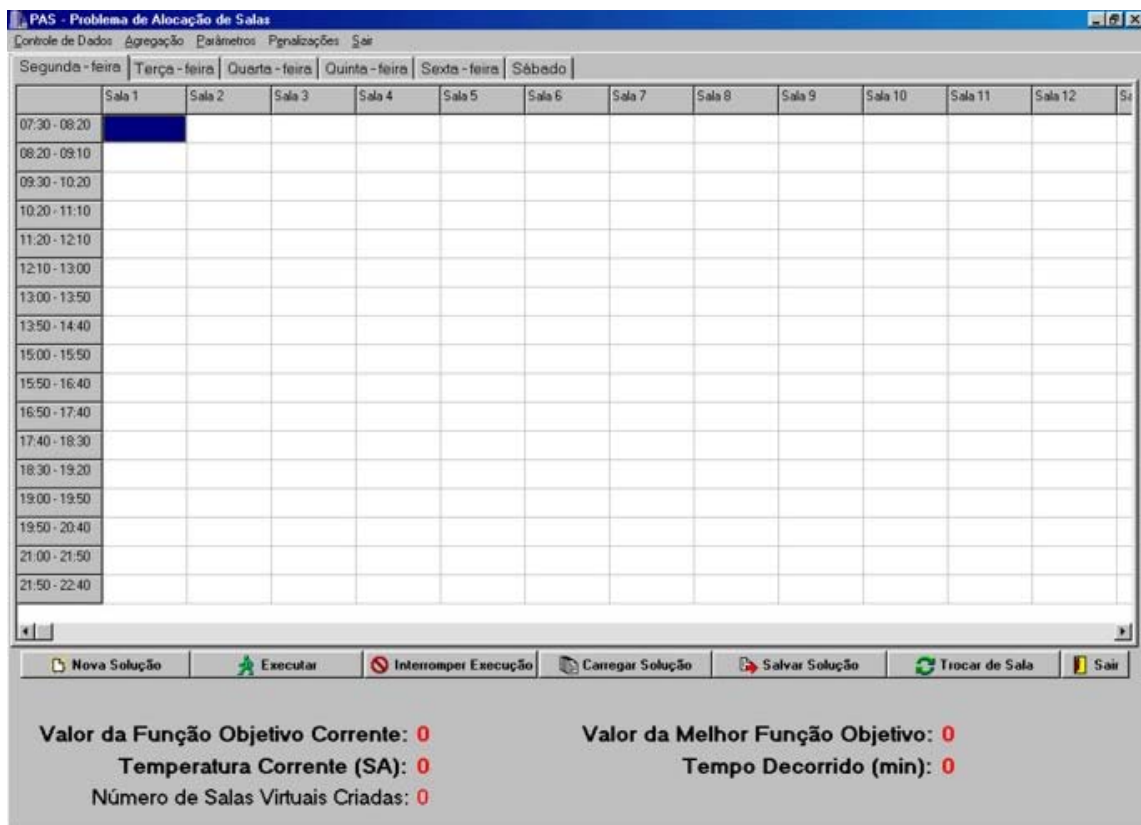


Figura 4.5 - Tela Inicial do Sistema

#### 4.4.2 Entrada Automática de Dados

Para iniciar o processo de entrada dos dados do sistema de forma automática, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Controle de Dados* e em seguida escolher o item *Entrada Automática de Dados*. Feito esse procedimento, o sistema apresentará ao usuário a Tela Entrada Automática de Dados (representada pela Figura 4.6), onde o usuário poderá selecionar o arquivo do tipo texto que desejar carregar automaticamente e clicar no botão *OK*. É importante lembrar que para que esse processo seja realizado com sucesso, os arquivos a serem carregados devem se encontrar localizados na pasta *C:\Iceb\*. Caso o usuário queira voltar a Tela Inicial, deve-se clicar no botão *SAIR*.

Os dados relativos às Salas de Aula constituídos pelos respectivos Números das Salas e suas respectivas Capacidades devem ser entrados de forma manual pelo usuário para posteriormente serem controlados manualmente conforme descreve a seção 4.3.3.

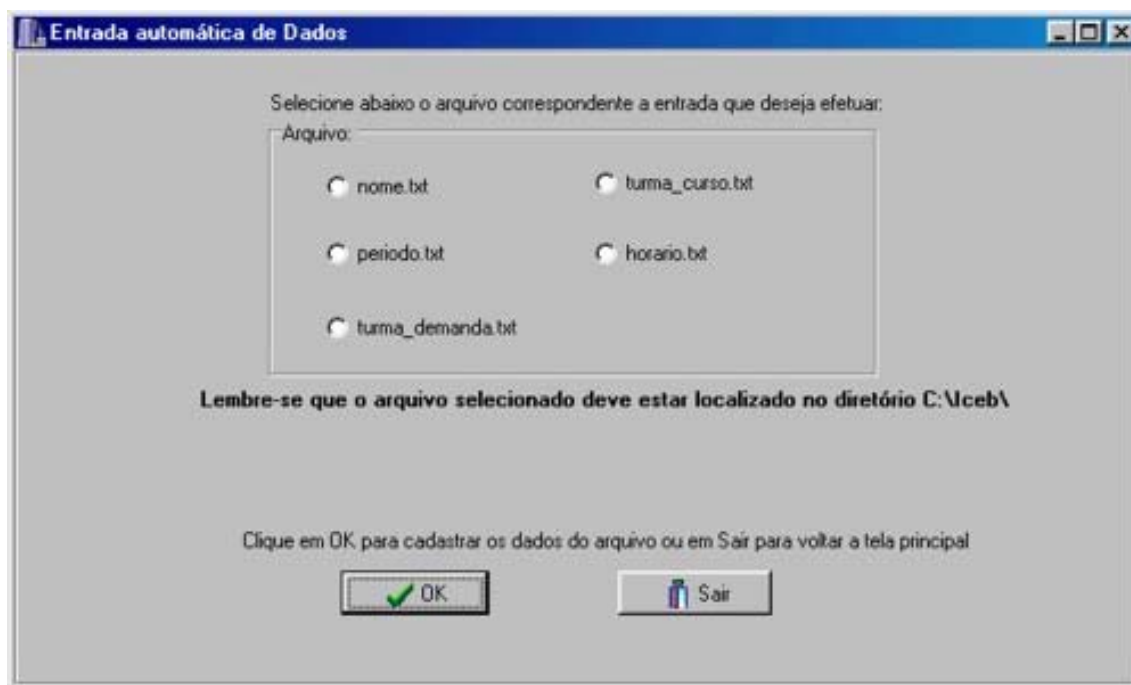


Figura 4.6: Tela Entrada Automática de Dados

Todos os arquivos necessários para esta entrada automática de dados, possuem um formato padrão (especificado pela Prograd da própria UFOP), exemplificado abaixo:

**nome.txt** = Nome da Disciplina | Descrição |

*Exemplo:*

CBI104|BOTAN. APLIC. A FARMACIA D|  
 CBI108|BIOLOGIA DO DESENVOLVIMENTO|  
 CBI110|CITO-HISTOLOGIA C|  
 CBI112|CITO-HISTOLOGIA D|  
 CBI114|ANATOMIA HUMANA|

**período.txt** = Nome da Disciplina | Curso para o qual ela é oferecida | Tipo (1=obrigatória; 2 = facultativa; 3 = eletiva) | Período em que ela é oferecida |

*Exemplo:*

CBI104|FAR|1|4|  
 CBI105|BCB|1|1|  
 CBI105|LCB|1|1|  
 CBI108|BCB|1|2|  
 CBI108|LCB|1|3|

**turma\_demanda.txt** = Nome da Disciplina | Número da Turma | Demanda da Turma |



*Exemplo:*

CBI126|11|44.0|

CIC111|11|30.0|

COM600|22|30.0|

MTM236|11|40.0|

QUI145|54|15.0|

*turma\_curso.txt* = Nome da Disciplina | Número da turma | Curso para o qual ela é oferecida | Demanda de alunos do respectivo curso |

*Exemplo:*

CBI104|31|FAR|17.0|

CBI104|32|FAR|14.0|

CBI104|33|FAR|18.0|

CBI105|41|BCB|16.0|

CBI105|42|BCB|11.0|

*horário.txt* = Nome da Disciplina | Numero da turma | Horário de Início da Aula | Horário de Término da aula

*Exemplo:*

CBI104|31|3|13:50|14:40|

CBI104|31|3|15:00|15:50|

CBI104|31|3|15:50|16:40|

CBI104|32|3|13:50|14:40|

CBI104|32|3|15:00|15:50|

CBI104|32|3|15:50|16:40|

#### **4.4.3 Controle Manual dos Dados**

Terminado o processo de Entrada Automática dos dados descrito na seção 4.3.2, o Sistema permite que o usuário possa exercer um controle manual dos dados das Disciplinas, Turmas e Salas de Aula onde é possível adicionar, alterar, consultar e remover dados relativos àqueles que foram previamente carregados de forma automática.

##### **4.4.3.1 Controle de Disciplinas**

Para iniciar o processo de Controle de Disciplinas, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Controle de Dados* e em seguida escolher o item *Disciplinas*.

Nessa etapa o usuário visualiza os dados relativos às disciplinas que já foram carregados anteriormente e escolhe dentre as opções da Barra de Botões situada na parte inferior da Tela Controle de Disciplinas (representada pela Figura 4.7) a ação que deseja

executar. Dentre as opções oferecidas pelo Sistema ao usuário encontram-se os itens: *Novo* (adicionar nova disciplina), *Consulta* (consultar alguma disciplina existente), *Alterar* (alterar alguma disciplina existente), *Excluir* (excluir alguma disciplina existente); além das opções *Confirmar* (finalizar algum procedimento com sucesso), *Cancelar* (finalizar algum procedimento sem sucesso), e *Sair* (sair do controle de disciplinas).

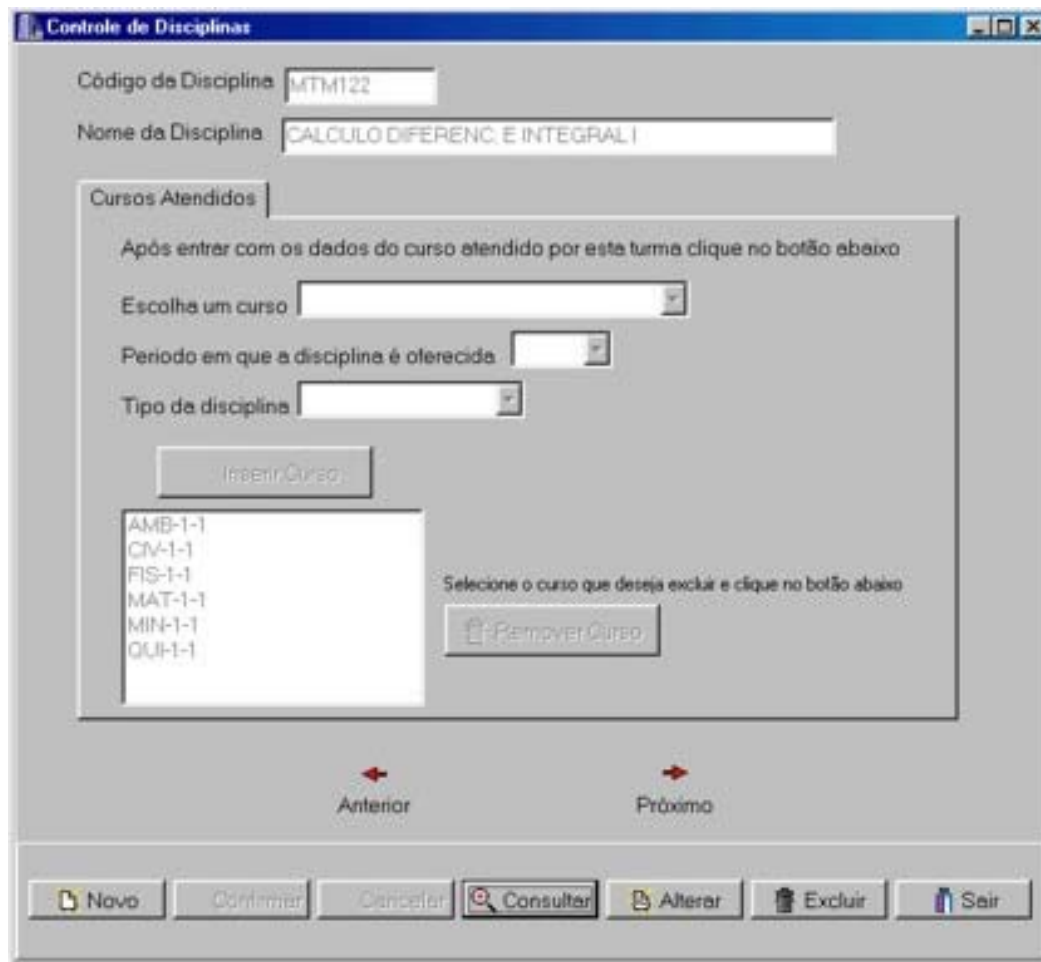


Figura 4.7: Tela Controle de Disciplinas

#### 4.4.3.2 Controle de Turmas

Para iniciar o processo de Controle de Turmas, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Controle de Dados* e em seguida escolher o item *Turmas*.

Nessa etapa o usuário visualiza os dados relativos às turmas que já foram carregados anteriormente e escolhe dentre as opções da Barra de Botões situada na parte inferior da Tela Controle de Turmas (representada pela Figura 4.8) a ação que deseja executar. Dentre as opções oferecidas pelo Sistema ao usuário encontram-se os itens: *Novo* (adicionar nova turma), *Consulta* (consultar alguma turma existente), *Alterar* (alterar alguma turma existente), *Excluir* (excluir alguma turma existente); além das opções *Confirmar* (finalizar algum procedimento com sucesso), *Cancelar* (finalizar algum procedimento sem sucesso), e *Sair* (sair do controle de turmas).

Controle de Turmas

Disciplina: CIC370      Número da Turma: 11      Número de alunos matriculados: 6

Horários | Cursos Atendidos

Clique dentro dos campos correspondentes aos dias e horários em que esta turma possui aulas:

HORÁRIO	SEGUNDA	TERÇA	QUARTA	QUINTA	SEXTA	SÁBADO
07:30 - 08:20						
08:20 - 09:10				X	X	
09:30 - 10:20				X	X	
10:20 - 11:10						
11:20 - 12:10						
12:10 - 13:00						
13:00 - 13:50						
13:50 - 14:40						
15:00 - 15:50						
15:50 - 16:40						
16:50 - 17:40						
17:40 - 18:30						
18:30 - 19:20						
19:00 - 19:50						
19:50 - 20:40						
21:00 - 21:50						
21:50 - 22:40						

← Anterior      Próximo →

Figura 4.8: Tela Controle de Turmas

#### 4.4.3.3 Controle de Salas

Para iniciar o processo de Controle de Salas, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Controle de Dados* e em seguida escolher o item *Salas*.

Nessa etapa o usuário visualiza os dados relativos às salas que já foram carregados anteriormente e escolhe dentre as opções da Barra de Botões situada na parte inferior da Tela Controle de Salas (representada pela Figura 4.9) a ação que deseja executar. Dentre as opções oferecidas pelo Sistema ao usuário encontram-se os itens: *Novo* (adicionar nova sala), *Consulta* (consultar alguma sala existente), *Alterar* (alterar alguma sala existente), *Excluir* (excluir alguma sala existente); além das opções *Confirmar* (finalizar algum procedimento com sucesso), *Cancelar* (finalizar algum procedimento sem sucesso), e *Sair* (sair do controle de salas).

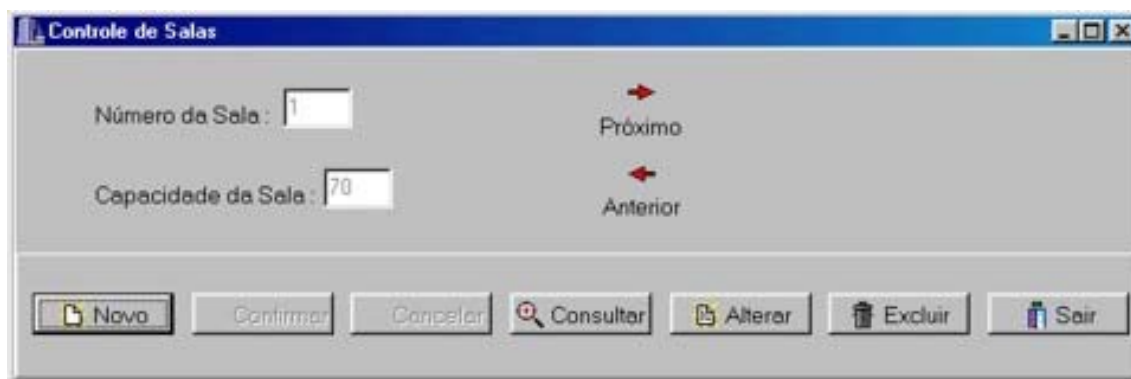


Figura 4.9: Tela Controle de Salas

#### 4.4.4 Limpar todos os dados

Como a cada novo semestre a base de dados deve ser atualizada, já que as turmas passam a ter demandas e horários diferentes, uma maneira mais fácil de fazer essa atualização é limpar toda a base de dados, ou seja, remover todos os registros do banco de dados, para então posteriormente fazer uma nova atualização com os novos registros que provavelmente deverão ser inseridos no sistema através do recurso de “*Entrada Automática de Dados*” (seção 4.4.2).

Desta forma, para que se possa limpar todas as tabelas do banco de dados, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Controle de Dados* e em seguida escolher o item *Limpar Todos os Dados*. Então, o usuário deverá confirmar a limpeza do banco de dados (clicando em *yes*) na caixa de confirmação que irá surgir na tela.

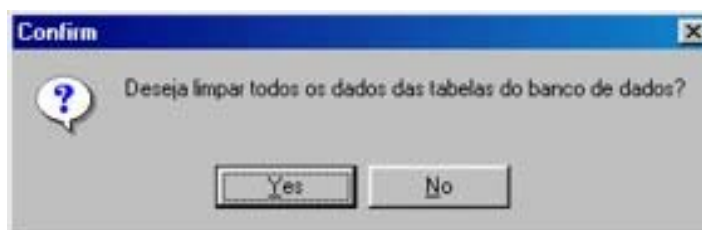


Figura 4.10: Tela de Confirmação de limpeza dos dados do banco de dados

#### 4.4.5 Agregação de Turmas

Para iniciar o processo de agregação de turmas, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Agregação* e em seguida escolher o item *Agregação de Turmas*. Feito esse procedimento, o sistema apresentará ao usuário a Tela Agregação de Turmas (representada pela Figura 4.11), onde o usuário poderá selecionar as turmas possíveis de agregação (clicando sobre elas com a tecla Ctrl pressionada) e clicar no botão *Agregar*. Do mesmo modo, que se existirem turmas já agregadas, será possível realizar a desagregação das turmas selecionando a turma agregada e clicando no botão *Desagregar*.

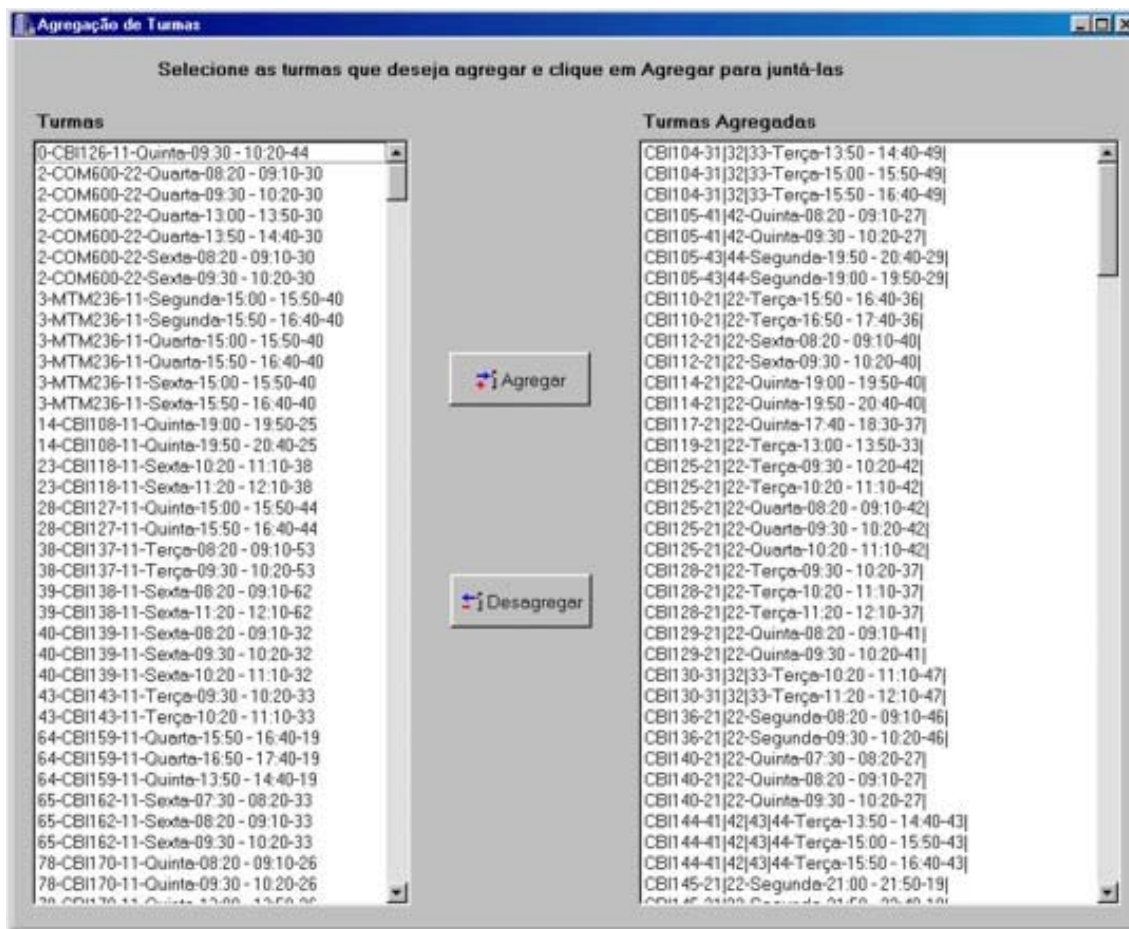


Figura 4.11: Tela Agregação de Turmas

É também importante ressaltar que cada linha do grid (quadro) de turmas da Tela Agregação de Turmas, correspondem a um registro das aulas inseridas no banco de dados do sistema, de forma que todas seguem o mesmo padrão, mostrado abaixo:

*Código da turma no banco de dados – Nome da disciplina – Dia – Horário de início da aula – Horário de término da aula – Demanda*

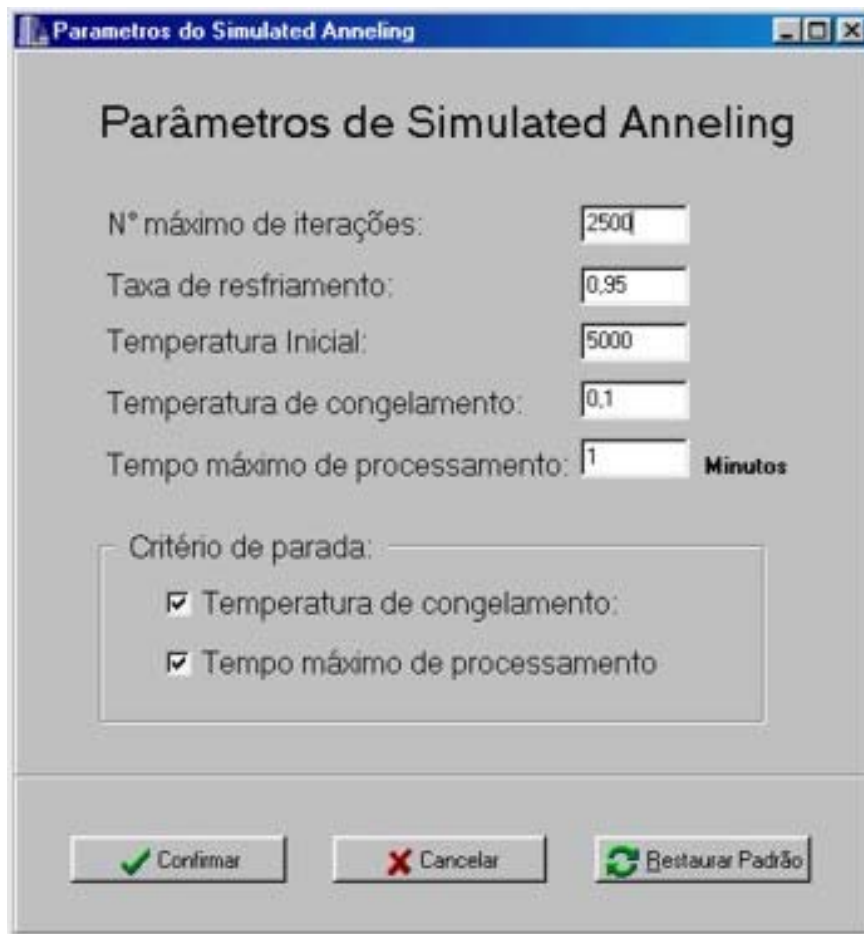
Exemplo: *118-CIC105-3-Quinta-15:00-15:50-20*

Outro fator que deve ser ressaltado é que o sistema não exige que na Agregação o nome das Disciplinas selecionadas seja o mesmo, mas, no entanto, para haja sucesso na execução de uma agregação é obrigatório que o dia e o horário da aula das respectivas turmas selecionadas sejam iguais.

#### 4.4.6 Especificação dos Parâmetros

Para iniciar o processo de especificação dos parâmetros, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Parâmetros* e em seguida escolher o item *Simulated Annealing*. Feito esse procedimento, o sistema apresentará ao usuário a Tela Parâmetros do Simulated Annealing (representada pela Figura 4.12), onde o usuário deverá especificar os parâmetros do sistema descritos de (a) até (f) e clicar no botão *OK*. Ou se preferir, o usuário pode usar os valores padrões do sistema clicando no botão

*Restaurar Padrão.* Caso deseje abortar esse procedimento de especificação dos parâmetros, deve-se clicar no botão *Cancelar*.



Parâmetros do Simulated Annealing

Parâmetros de Simulated Annealing

N° máximo de iterações: 2500

Taxa de resfriamento: 0,95

Temperatura Inicial: 5000

Temperatura de congelamento: 0,1

Tempo máximo de processamento: 1 Minutos

Critério de parada:

- Temperatura de congelamento:
- Tempo máximo de processamento

Confirmar Cancelar Restaurar Padrão

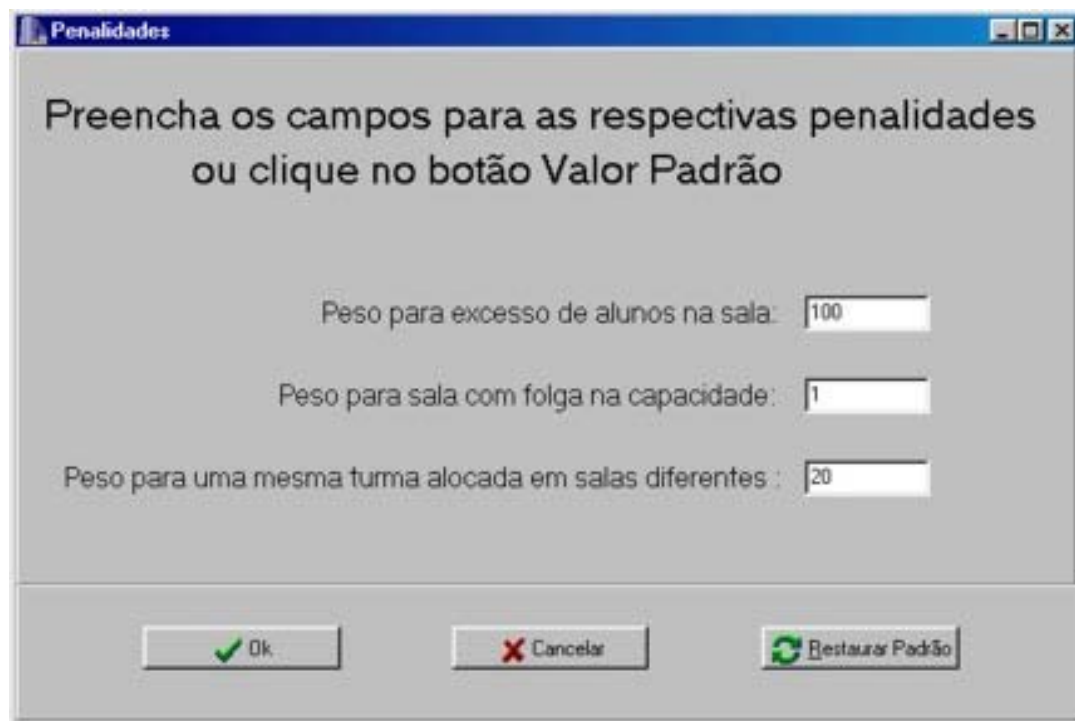
Figura 4.12: Tela Parâmetros do Sistema

- N° máximo de iterações*: número de iterações que o SA vai executar em cada temperatura. Na Figura 4.4 esse número é representado por  $SA_{max}$ . Esse valor é um número inteiro maior do que zero.
- Taxa de resfriamento*: taxa que será aplicada (multiplicada) à temperatura inicial enquanto o sistema não congelar. Na Figura 4.4 essa taxa é representada por  $\alpha$ . Esse valor é um número real variando de zero a um;
- Temperatura inicial*: temperatura na qual o SA vai iniciar a execução. Na Figura 4.4 essa temperatura é representada por  $T_0$ . Esse valor é um número real maior do que zero.
- Temperatura de congelamento*: temperatura limite do SA, ou seja, quando a temperatura corrente for menor do que a temperatura de congelamento significa que o sistema congelou, terminando assim a execução do SA. Na Figura 4.4 essa temperatura é representada por  $TC$ .
- Tempo máximo de processamento*: tempo pelo qual o SA será executado. Na Figura 4.4 esse tempo é representado por  $TMP$ .

- (f) *Critério de parada*: indica qual critério o usuário quer utilizar para determinar o fim da execução do SA. O usuário poderá optar pelos critérios de parada Temperatura de Congelamento e/ou Tempo Máximo de Processamento.

#### 4.4.7 Especificação das Penalizações

Para iniciar o processo de especificação das penalizações, o usuário deve escolher dentre as opções da Barra de Menu da Tela Inicial do Sistema a opção *Penalizações* e em seguida escolher o item *Pesos*. Feito esse procedimento, o sistema apresentará ao usuário a Tela Penalidades (representada pela Figura 4.13), onde o usuário deverá especificar os pesos para cada tipo de penalização descritas de (a) até (c) e clicar no botão *OK*. Ou se preferir, o usuário pode usar os valores padrões do sistema clicando no botão *Restaurar Padrão*. Caso deseje abortar esse procedimento de especificação das penalizações, deve-se clicar no botão *Cancelar*.



A screenshot of a software window titled "Penalidades". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main content area is light gray and contains the following text: "Preencha os campos para as respectivas penalidades ou clique no botão Valor Padrão". Below this text are three input fields, each with a label and a value: "Peso para excesso de alunos na sala: 100", "Peso para sala com folga na capacidade: 1", and "Peso para uma mesma turma alocada em salas diferentes : 20". At the bottom of the window, there are three buttons: "Ok" with a green checkmark icon, "Cancelar" with a red X icon, and "Restaurar Padrão" with a green circular arrow icon.

Figura 4.13: Tela Penalidades do Sistema


(a) *Excesso de Alunos na Sala*: o usuário deve especificar o peso para a penalização por cada aluno além da capacidade da sala. O valor padrão para esse item é 100.

(b) *Sala com Folga na Capacidade*: o usuário deve especificar o peso para a penalização por cada aluno aquém da capacidade da sala. O valor padrão para esse item é 1.

(c) *Mesma Turma Alocada em Salas Diferentes*: o usuário deve especificar o peso para a penalização de cada aula semanal de uma mesma turma que esteja alocada em salas diferentes.


#### 4.4.8 Iniciar uma Nova Solução do SA

Mesmo depois de já ter executado o Simulated Annealing algumas vezes ou mesmo ter carregado alguma solução anteriormente salva, é possível limpar o quadro de alocação da tela principal e gerar uma nova solução para o problema somente a partir das informações


iniciais do banco de dados. Para isso basta clicar no botão  localizado abaixo do quadro de alocação na Tela Principal e depois então clicar no botão “Executar” que também se encontra abaixo do mesmo quadro de alocação. Feito esse procedimento, o sistema dará início ao processo de execução do sistema. Durante esse processo, será possível acompanhar os valores da Função Objetivo, do Tempo de Processamento decorrido durante a execução e de outros dados relevantes ao método SA (função objetivo, temperatura corrente, etc...) na parte inferior direita da Tela Inicial do Sistema.

#### 4.4.9 Interromper Execução

É importante ressaltar que o processo de execução do sistema poderá ser interrompido durante qualquer instante, apresentando assim, a solução melhor encontrada para o problema em questão até o momento da interrupção. Caso o processo de execução não seja interrompido, o sistema executará até que os critérios de paradas sejam satisfeitos e apresentará assim, a solução final encontrada.

Sendo assim, para acionar uma interrupção da execução basta clicar no botão  localizado abaixo do quadro de alocação na Tela Principal.


#### 4.4.10 Continuar Executando o SA

Outro recurso de grande utilidade do sistema é a possibilidade de continuar uma execução recém interrompida ou até mesmo uma solução anteriormente salva e que foi carregada pelo usuário para uma suposta visualização e refinamento desta. Para isso basta clicar sobre o botão  localizado abaixo do quadro de alocação na Tela Principal.

Vale lembrar que este recurso prove ao sistema a opção de reaquecer o Simulated Annealing após o seu resfriamento, ou seja, como é sabido, a medida que o SA é executado sua temperatura vai se resfriando e quanto menor é a sua temperatura, menor também é a probabilidade deste aceitar movimentos de piora para tentar fugir de ótimos locais, desta forma, quando interrompemos a execução do SA e depois damos continuidade a esta, o SA assumirá como novos parâmetros (Temperatura inicial, Numero máximo de iterações e outros) de sua execução os mesmos anteriormente definidos quando no início da execução, desta forma , assim estaremos provendo um reaquecimento da Temperatura do SA.

#### 4.4.11 Salvar Solução

Para salvar uma nova solução gerada ou modificada pelo Pas Solver:


1. Clique no botão  localizado na barra de botões abaixo do quadro de alocações de salas da tela principal.
2. Na caixa **Nome do arquivo**, digite um novo nome para o arquivo.
3. Clique em **Salvar**.

#### 4.4.12 Carregar Solução

Para carregar uma solução anteriormente salva pelo usuário:





1. Clique no botão  localizado na barra de botões abaixo do quadro de alocações de salas da tela principal.
2. Na caixa de diálogo que irá se abrir, navegue entre os diretórios existentes e selecione o arquivo (\*.pas) que deseja carregar.
3. Clique em **Abrir**.

**Obs:** Se uma solução for carregada e sofrer qualquer tipo de modificação como, por exemplo, uma troca manual de salas ou uma nova execução sobre esta, isto não implicará em uma salvamento automático das alterações realizadas na alocação carregada. Para que estas sejam salvas é preciso salvar a solução (seção 4.4.11).

#### 4.4.13 Troca manual de salas

O sistema permite ao usuário trocar turmas (manualmente) entre salas. Para isso, após o sistema apresentar uma solução, basta clicar sobre o botão “Trocar de Sala” localizado abaixo do quadro de alocação na Tela Principal, então na nova tela que se abrirá o usuário deverá selecionar o dia da semana em que se deseja realizar a troca (no Box entre os dois quadros de horários), depois selecionar uma turma em cada quadro clicando sobre elas, e então, clicar sobre o botão “Trocar de Sala”. Se a troca não for possível, será mostrada uma mensagem de erro,

Para inserir uma nova sala (virtual) na solução, basta o usuário clicar no botão “Criar Sala” (entre os dois quadros), assim uma nova sala será criada (sem turmas alocadas).

Seleção a turma (ou horário vago) a serem trocadas clicando sobre suas respectivas células nos dois quadros abaixo:

	Sala 1	Sala 2	Sala 3	Sala 4	Sala 5	Sala 6	Sala 7	Sala 8	Sala 9	Sala 10	Sala 11	Sala 12
07:30 - 08:20	MTM122-83							MTM112-76		MTM124-66		MTM123-
08:20 - 09:10	MTM122-83	CB1137-11	CB1176-2122	COM200-11	COM520-11	FIS620-11		MTM112-76		MTM124-66		MTM123-
09:30 - 10:20	MTM122-83	CB1137-11	CB1176-2122	COM200-11	COM520-11	FIS620-11	MTM125-62	CB1143-11	MTM124-62	MTM112-77	CB1214-11	MTM123-
10:20 - 11:10	MTM150-11	CB1130-31323	CB1164-4142	FIS521-11	COM520-11	QUI200-9878	MTM125-62	CB1143-11	MTM124-62	MTM112-77	CB1214-11	MTM123-
11:20 - 12:10	MTM150-11	CB1130-31323	CB1164-4142	FIS521-11	COM520-11	QUI200-9878					CB1214-11	
12:10 - 13:00												
13:00 - 13:50		MTM112-7273	CB1173-2122	FIS212-22	QUI200-9878	CB1119-2122		MTM123-61	FIS210-32	QUI165-11	COM604-11	QUI173-2
13:50 - 14:40		MTM112-7273	CB1173-2122	FIS212-22	QUI200-9878	CB1144-41424	CB1104-31323	MTM123-61	FIS210-32	QUI165-11	COM604-11	QUI173-2
15:00 - 15:50	MTM151-64	FIS213-34	CIC105-5	FIS210-34	QUI101-2122	CB1144-41424	CB1104-31323	COM201-11	FIS210-33	FIS313-11		FIS213-33

Escolha o dia da semana:  
Terça - feira

**Trocar de Sala**      **Criar Sala**

	Sala 1	Sala 2	Sala 3	Sala 4	Sala 5	Sala 6	Sala 7	Sala 8	Sala 9	Sala 10	Sala 11	Sala 12
07:30 - 08:20	MTM122-83							MTM112-76		MTM124-66		MTM123-
08:20 - 09:10	MTM122-83	CB1137-11	CB1176-2122	COM200-11	COM520-11	FIS620-11		MTM112-76		MTM124-66		MTM123-
09:30 - 10:20	MTM122-83	CB1137-11	CB1176-2122	COM200-11	COM520-11	FIS620-11	MTM125-62	CB1143-11	MTM124-62	MTM112-77	CB1214-11	MTM123-
10:20 - 11:10	MTM150-11	CB1130-31323	CB1164-4142	FIS521-11	COM520-11	QUI200-9878	MTM125-62	CB1143-11	MTM124-62	MTM112-77	CB1214-11	MTM123-
11:20 - 12:10	MTM150-11	CB1130-31323	CB1164-4142	FIS521-11	COM520-11	QUI200-9878					CB1214-11	
12:10 - 13:00												
13:00 - 13:50		MTM112-7273	CB1173-2122	FIS212-22	QUI200-9878	CB1119-2122		MTM123-61	FIS210-32	QUI165-11	COM604-11	QUI173-2
13:50 - 14:40		MTM112-7273	CB1173-2122	FIS212-22	QUI200-9878	CB1144-41424	CB1104-31323	MTM123-61	FIS210-32	QUI165-11	COM604-11	QUI173-2
15:00 - 15:50	MTM151-64	FIS213-34	CIC105-5	FIS210-34	QUI101-2122	CB1144-41424	CB1104-31323	COM201-11	FIS210-33	FIS313-11		FIS213-33

Sair

Figura 4.14: Tela da Troca Manual de Salas

#### 4.4.14 Visualização da solução no Microsoft Excel

Outro recurso automático do “Pas Solver” é a geração de um arquivo que pode ser visualizado no MS Excel e que exibe através das planilhas deste a alocação encontrada pelo SA.

Este arquivo é criado toda vez que o usuário finaliza o Pas Solver, sendo que ele é criado no diretório raiz ( C:\ ) com o nome de “*alocação.xls*”, e representa (salva) a solução visualizada no quadro de alocação de salas da Tela Principal no momento em que o Pas Solver é fechado. Ressaltando que não é necessário salvar a solução para que este arquivo seja criado, no entanto, este arquivo só pode ser aberto pelo Excel, de forma que para que se possa visualizar uma mesma solução no Pas Solver depois de ter finalizado o programa é preciso salvar esta através do recurso “Salvar Solução” (seção 4.4.11).

#### 4.4.15 Resultados da Execução

Terminada o processo de execução do sistema, o usuário terá como resultados do sistema a alocação final das salas na estrutura de quadro de horários da Tela Inicial, além dos valores da Função Objetivo Final, do Tempo Total Decorrido de Processamento em minutos e do Número de Salas Virtuais criadas para que a solução seja possível, além também do valor da função objetivo corrente e da Temperatura Corrente do SA no momento em que a execução foi finalizada. (representados pela Figura 4.15).

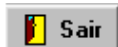
The screenshot shows the 'PAS - Problema de Alocação de Salas' application. The main window displays a grid with columns for days of the week (Segunda - feira to Sábado) and rows for time slots (e.g., 07:30 - 08:20). Each cell in the grid contains a room ID (e.g., MTM122-83, CBI137-11, COM200-11). Below the grid, there are several buttons: 'Nova Solução', 'Executar', 'Interromper Execução', 'Carregar Solução', 'Salvar Solução', 'Trocar de Sala', and 'Sair'. At the bottom, summary statistics are displayed:

- Valor da Função Objetivo Corrente: 63350
- Valor da Melhor Função Objetivo: 41753
- Temperatura Corrente (SA): 347.21420
- Tempo Decorrido (min): 00:01:00
- Número de Salas Virtuais Criadas: 4

Figura 4.15: Tela Resultados da Execução do Sistema

#### 4.4.16 Finalização do Sistema

O Sistema finaliza com a escolha por parte do usuário da opção *Sair* da Barra de Menu da Tela Inicial do Sistema seguida da escolha do item *Sair do PAS*, ou pelo clique no botão



localizado na barra de botões abaixo do quadro de alocações da tela principal.

## 5 Resultados Computacionais

O sistema desenvolvido foi testado em um microcomputador PC AMD Athlon, 1.3MHz, com 128 MB de RAM sob sistema operacional Windows 98, usando dados relativos à distribuição de salas do segundo semestre letivo do ano 2001 (ICEB2001/2), acrescido de dois outros problemas teste, um com 17 salas (Teste17) e outro com 22 salas (Teste22). Esses dois últimos problemas foram gerados aleatoriamente, mantendo-se uma proporção entre a quantidade de horários de aulas e horários disponíveis em salas semelhante à existente no problema real. Algumas das características desses problemas encontram-se explicitadas na Tabela 5.1.

<b>Instância</b>	<b>Número de salas</b>	<b>Número de turmas</b>	<b>Número de horas-aula</b>
<b>Teste17</b>	17	214	713
<b>ICEB2001/2</b>	20	233	763
<b>Teste22</b>	22	281	938

Tabela 5.1: Características das instâncias consideradas

A Tabela 5.2 mostra, para cada teste realizado, o melhor valor da função objetivo, bem como o tempo de CPU gasto pelo método até que o sistema atingisse a temperatura de congelamento, o número máximo de iterações consideradas em uma dada temperatura, a taxa de resfriamento, temperatura inicial e a temperatura de congelamento consideradas.

<b>Instância</b>	<b>Algoritmo</b>	<b>Melhor Solução</b>	<b>Desvio (%)</b>	<b>Tempo gasto para encontrar a melhor solução (segundos)</b>
<b>Teste17</b>	SA	7320	4.93	4767
<b>ICEB2001/2</b>	SA	10600	16.39	5358
<b>Teste22</b>	SA	26329	10.87	2835

Tabela 5.2: Resultados computacionais

Destaca-se que o método SA não é fortemente influenciado pela solução inicial, isto é, soluções iniciais de baixa qualidade não interferem significativamente na qualidade da solução final.

## 6 Conclusões e trabalhos futuros

Neste projeto foi apresentada uma solução para o problema de alocação de salas utilizando uma técnica de metaheurística. Uma solução inicial é gerada por um procedimento construtivo parcialmente guloso e submetida à heurística *Simulated Annealing*.

Esse procedimento mostrou ser eficaz, conseguindo produzir soluções finais com boa qualidade.

Em vista do desempenho satisfatório da técnica proposta, o ICEB adotou o sistema desenvolvido, o qual será utilizado a partir do início do primeiro semestre letivo de 2003.

Propõe-se, como trabalho futuro, a inclusão de novos requisitos no modelo e de algumas facilidades de interface para o sistema. No entanto, a grande proposta para trabalhos futuros diz respeito à implementação e incorporação ao sistema de uma técnica híbrida SA + BT (Simulated Annealing + Busca Tabu), a qual combina as características mais apropriadas de SA e BT de forma a obter um procedimento mais eficaz. Esta técnica já foi testada em Souza et. al. (2002) e se mostrou superior às técnicas SA e BT tomadas isoladamente.

## 7 Referências Bibliográficas

- Abramson, D. (1991). "Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms", *Management Science*, 37:98-113.
- Bardadym, V. A (1996) "Computer-Aided School and University Timetabling: The New Wave", *Lecture Notes in Computer Science*, 1153:22-45.
- Burke, E.K., Cowling, P., Landa Silva, J.D. and McCollum, B. (2001) "Three Methods to Automate the Space Allocation Process in UK Universities", *Lecture Notes in Computer Science*, 2079: 254-276.
- Carter, M.V. and Tovey, C.A. (1992). "When Is the Classroom Assignment Problem Hard?", *Operations Research*, 40:S28-S39.
- Costa, D. (1994). "A tabu search algorithm for computing an operational timetable". *European Journal of Operational Research*, 76:98-110.
- de Werra, D. (1995) "An introduction to timetabling", *European Journal of Operational Research*, 19:151-162.
- Dowland, K.A. (1993) "Simulated Annealing", In Reeves, C.R. (ed), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, 20-69.
- Dowland, K.A. (1998). "Off-the-Peg or Made-to-Measure? Timetabling and Scheduling with SA and TS", *Lecture Notes in Computer Science*, 1408:37-52.
- Erben, W. and Keppler, J. (1996). "A Genetic Algorithm Solving a Weekly Course-Timetabling Problem", *Lecture Notes in Computer Science*, 1153:198-211.
- Even, S., Itai, A. and Shamir, A. (1976) "On the complexity of timetabling and multicommodity flow problems", *SIAM Journal of Computation*, 5:691-703.
- Feo, T.A. and Resende, M.G.C. (1995) "Greedy randomized adaptive search procedures", *Journal of Global Optimization*, 6:109-133.
- Glover, F. (1986) "Future Paths for Integer Programming and Links to Artificial Intelligence", *Computers and Operations Research*, 5: 553-549.
- Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer academic Publishers, Boston.
- Hertz, A. (1992) "Tabu search for large scale timetabling problems", *European Journal of Operational Research*, 54:39-47.
- Rich, D.C. (1996) "A Smart Genetic Algorithm for University Timetabling", *Lecture Notes in Computer Science*, 1153: 181-197.
- Santos, A.M., Marques, E. and Ochi, L.S. (1997). "Design and implementation of a timetable system using genetic algorithm". Second International Conference on Practice and Theory of Automated Timetabling, Toronto, Canada.
- Schaefer, A. (1999) "A survey of automated timetabling", *Artificial Intelligence Review*, 13:87-127.
- Ueda, H., Ouchi, D., Takahashi, K. and Miyahara, T. (2001) "A Co-evolving Timeslot/Room Assignment Genetic Algorithm Technique for Universities Timetabling", *Lecture Notes in Computer Science*, 2079: 48-63.