ABORDAGEM HEURÍSTICA PARA O PROBLEMA DE PROGRAMAÇÃO DE HORÁRIOS DE CURSOS

RODRIGO DE CARVALHO

ABORDAGEM HEURÍSTICA PARA O PROBLEMA DE PROGRAMAÇÃO DE HORÁRIOS DE CURSOS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Engenharia Elétrica.

ORIENTADOR: RODNEY REZENDE SALDANHA
CO-ORIENTADOR: MARCONE JAMILSON FREITAS SOUZA

Belo Horizonte
Agosto de 2011

Agradecimentos

Em primeiro lugar agradeço a DEUS pelas graças recebidas, por me amparar nos momentos mais difíceis e por ter colocado pessoas em minha vida que contribuíram para que esse sonho se tornasse realidade.

Aos meus pais, Antônio e Maria, pelo carinho, dedicação e apoio.

Aos colegas do laboratório GOPAC, pela companheirismo e disponibilidade nos momentos difíceis. Em especial aos amigos Luciana e Bruno, pelos momentos de descontração e amizade.

Ao meu orientador Professor Rodney Rezende Saldanha, pelos dois anos de orientação e amizade. Agradeço também pela confiança em direcionar a mim alguns desafios que enriqueceram a minha formação como pesquisador.

Ao meu co-orientador Professor Marcone Jamilson Freitas Souza, pelas idéias e conselhos valiosos.

Ao Professor Alexandre Xavier Martins, pela confiança, amizade, idéias e conselhos ao longo desses 5 anos de convivência.

Ao Professor Ricardo Saraiva de Camargo, pela disponibilidade e atenção.

E à todos aqueles que de alguma forma contribuiram para realização desse trabalho.

“"Não mostre ao seu Deus o tamanho do seu problema,
mostre ao seu problema o tamanho do seu Deus."”
(Autor desconhecido)

Resumo

O presente trabalho tem como foco o problema de programação de cursos baseados em currículos. O problema consiste na definição do horário de cursos e na alocação de salas de aula respeitando-se um conjunto de restrições, tais como a capacidade das salas, indisponibilidades dos professores etc. Para resolvê-lo, foram desenvolvidos métodos heurísticos baseados em *Iterated Local Search* (ILS) e *Variable Neighborhood Descent* (VND). Também foi desenvolvido um procedimento de construção que é capaz de gerar boas soluções iniciais com tempos computacionais relativamente pequenos. Devido às particularidades do problema, visto que cada instituição de ensino normalmente considera objetivos e restrições muito diferentes, adotamos para a validação do trabalho instâncias do *International Timetabling Competition* 2007 (ITC). Os resultados obtidos apontam um desempenho satisfatório para o método que utiliza o VND como procedimento de busca local do ILS. Além disso, foi desenvolvido um *software* para construir e gerenciar um quadro de horário.

Abstract

This work focus on the Curriculum based Course Timetabling. This problem consists in defining the schedule and classroom allocation, respecting a set of constraints, such as classroom capacity, unavailability of teachers, and so on. In order to solve the problem, heuristic methods based on Iterated Local Search (ILS) and Variable Neighborhood Descent (VND) have been developed. A construction procedure was also developed, which is capable of generating good initial solutions with relatively small computational times. Due to the particularities of the problem, since each institution normally considers very different aims and constraints, the validation of this work was done adopting instances from the International Timetabling Competition 2007 (ITC). The results obtained indicate a satisfactory performance for the method that uses VND as local search procedure for ILS. Furthermore, a software was developed to build and manage a timetable.

Lista de Figuras

Lista de Tabelas

Sumário

Acrônimos

CSS *Cascading Style Sheets*
DE *Differential Evolution*
DMED Desvio Médio
Fies Fundo de Financiamento ao Estudante do Ensino Superior
FO Função Objetivo
FOM Função Objetivo Média
GD *Great Deluge*
GRASP *Greedy Randomized Adaptative Search Procedure*
GUI *Graphical User Interface*
HC *Hill Climbing*
IE Instituição de Ensino
IES Instituição de Ensino Superior
IFS *Iterative Forward Search*
ILS *Iterated Local Search*
ITC *International Timetabling Competition*
LRC Lista Restrita de Candidatos
MEC Ministério da Educação
NP *Non-Deterministic Polynomial Time*
OMG *Object Management Group*
PHP *Hypertext Preprocessor*
PSR Problema de Satisfação de Restrição
Reuni Programa de Apoio a Planos de Reestruturação e Expansão das Universidades Federais
ProUni Programa Universidade para Todos
SA *Simulated Annealing*
SAT *Boolean Satisfiability*
SGQH Sistema Gerenciador de Quadro de Horário
UML *Unified Modelling Language*
VND *Variable Neighborhood Descent*
VNS *Variable Neighborhood Search*

Capítulo 1

Introdução

1.1 Contexto Geral

Ao longo dos últimos 50 anos, a comunidade académica vem se esforçando para buscar soluções automatizadas para o processo de alocação de recursos sob restrições. Um dos problemas típicos dessa natureza é o problema de programação de quadro de horário, também conhecido como *timetabling problem*, que pode ser aplicado em diversos contextos, como por exemplo: eventos esportivos [1, 2], transporte [3], escalonamento de enfermeiras em hospitais [4] e horários educacionais [5, 6, 7, 8].

Quando se trata de instituições de ensino, a programação de horários é algo muito importante, pois influencia diretamente na vida de funcionários, alunos e professores. Uma vez implementado um quadro de horários, esse normalmente é utilizado durante todo o período letivo, fazendo com que em várias situações alunos e professores tenham que se adaptar a ele, pois as soluções geralmente não atendem à todos os interesses e disponibilidade dos envolvidos.

Além disso, questões financeiras estão ligadas diretamente a esse problema, pois um quadro de horário mal feito pode implicar em aumento de custos. Por exemplo, a má atribuição de cursos $\times$ salas, pode gerar a necessidade de novas salas, ou períodos extras para suprir uma demanda, com consequente aumento de custos para a Instituição de Ensino (IE).

Ainda, segundo Lara [9], quando se trata de Instituições de Ensino Superior (IES) brasileiras, as quais estão sujeitas a um mecanismo de avaliação oficializado pelo Ministério da Educação (MEC), uma solução de um quadro de horários pode implicar em arranjos melhores ou piores quanto aos conceitos obtidos nas Avaliações Institucionais. Isso porque o plano de disciplinas a serem ofertadas e consequentemente o quadro de horários, estão na base estrutural da organização de uma IES.

Alguns fatores dificultam a resolução manual desse problema, tal como o número de alunos, número de cursos, tipo de instituição e as restrições. Por exemplo, tradicionalmente os

professores não possuem todos os horários disponíveis para as disciplinas, seja por que trabalham em outros cursos, na mesma ou em outra instituição ou porque assumem outras atividades, etc. Essa restrição de horários dos professores aumenta a complexidade na busca de uma solução viável [10].

Deste modo, a procura de soluções de forma manual pode ser uma tarefa árdua, em alguns casos impraticável, pois demandaria serviço de diversas pessoas durante vários dias e, mesmo assim, ao fim do processo a solução encontrada poderia não ser satisfatória, ou mesmo viável. Por exemplo, um professor poderia ficar descontente com várias janelas (aulas fragmentadas) na sua programação semanal.

Ainda levando em consideração a expansão do ensino superior no país por meio do Reuni (Programa de Apoio a Planos de Reestruturação e Expansão das Universidades Federais), Fies (Fundo de Financiamento ao Estudante do Ensino Superior) e ProUni(Programa Universidade para Todos), onde o número de cursos, vagas e professores cresceram substancialmente, torna-se necessário, na maioria dos casos, um auxílio computacional na geração, ou adaptação de quadros de horários devido a alta complexidade.

Nesse contexto, métodos computacionais que auxiliem na boa administração dos interesses de professores, alunos e IES, expressos em um quadro de horários adequado, passa a ser um problema de grande interesse prático, ao lado de sua importância teórica.

De forma geral, esse problema criou uma atenção especial no meio acadêmico com o objetivo de gerar um quadro de horário de forma automática e, ao mesmo tempo, de melhor qualidade em relação aos gerados manualmente. Desde a década de 60 métodos computacionais tem sido estudados com esses objetivos, como pode ser visto em [11, 12]. Atualmente conferências e competições [13, 14, 15, 16] têm sido organizadas com a intenção de discutir o tema no meio científico.

Entre os métodos computacionais, os de otimização têm sido amplamente empregados para resolver esse problema. Tais métodos têm como objetivo procurar uma solução de qualidade que atenda as restrições impostas pelo problema. É importante para um bom entendimento dos métodos de otimização, o conhecimento de alguns conceitos e definições utilizados na literatura. Abaixo são listados alguns desses termos:

- Variáveis: São entidades que se alteram durante o processo de otimização, podendo ser contínuas (reais), inteiras ou discretas. Por exemplo, no problema de programação de quadro de horários onde uma solução pode ser representada por uma matriz $c \times h$, sendo c o numéro de cursos e h o número de períodos disponíveis (Dia da Semana $\times$ Horário), cada posição x_{ch} é uma variável, que assume valores discretos referentes a qual sala o curso c no período h foi alocada.

- Restrições: São funções de igualdade ou desigualdade sobre as variáveis do problema que descrevem situações consideradas não desejáveis. Por exemplo, podemos representar a condição de não atribuição de mais de uma aula em uma mesma sala em um mesmo horário por meio de uma restrição.

- Espaço de busca: É o conjunto, espaço ou região que compreende as soluções possíveis ou viáveis sobre as variáveis do problema a ser otimizado, sendo delimitado pelas funções de restrição.

- Função Objetivo: É a função que define a qualidade de uma solução em relação as variáveis do problema que se quer otimizar.

- Ótimo: É o ponto formado pelas variáveis que extremizam a função objetivo e satisfazem as restrições.

Entre os métodos de otimização existem algumas possíveis abordagens para resolução desse problema, que podem ser resumidas em duas principais classes: métodos exato e heurístico. A primeira, refere-se ao procedimento que encontra a solução ótima para o problema, porém o uso desses métodos para problemas com tal complexidade pode ser inviável. Já o segundo é um procedimento que não garante a solução ótima do problema, mas tem se mostrado bastante eficiente levando-se em consideração os fatores de qualidade de solução e tempo computacional.

A complexidade de um problema, geralmente, é representada pelo tempo de processamento requisitado para a resolução do mesmo. Os problemas podem pertencer a duas classes de problemas: P e NP. A classe P possui todos os problemas que podem ser resolvidos por algoritmos determinísticos em tempo polinomial; e a classe NP, *Non-deterministic Polynomial time*, é o conjunto de problemas que podem ser resolvidos por algoritmos não-determinísticos em tempo polinomial processando em uma máquina não determinística de Turing. Um estudo mais aprofundado sobre essa última classe pode ser encontrado no trabalho apresentado por Dune [17].

O problema de quadro de horários na maioria dos casos reais (aplicavél à uma Instituição de Ensino), está inserido na classe de problemas NP [18]. Por esse motivo, o uso de métodos de otimização exatos pode se tornar inviável em dimensões de instâncias mais elevadas em tempo computacional viável. Por exemplo: suponha que um algoritmo O(2^n) possa resolver um problema com 40 cursos em 30 horas em uma computador que realiza 10^7 operações desse

algoritmo por segundo. Então, levaria cerca de 60 horas para 41 cursos, e quase 2 anos para 49 cursos. Ou seja, pequenas alterações no tamanho da entrada fazem com que o tempo de resposta aumente exponencialmente. Por esse motivo métodos heurísticos são geralmente utilizados nesse tipo de problema.

1.2 Objetivo

O objetivo dessa dissertação é estudar e propor uma metodologia com auxilio computacional para resolver o problema de programação de quadro de horário de cursos. Mais especificamente, será estudado o problema conhecido como programação de quadro de horários de cursos baseado em currículo, seguindo as normas e restrições estabelecidas no *International Timetabling Competition* (ITC 2007). Neste problema o quadro de horário é gerado baseado no currículo oficial da IES.

Para isso será proposta uma abordagem heurística, tendo como base o método *Iterated Local Search* [19] visando a obtenção de soluções de boa qualidade em um tempo computacional competitivo em relação a outros métodos propostos na literatura. Além disso é proposto um software para gerenciar o quadro de horário criado. Como objetivos específicos desse trabalho podemos destacar:

- Revisão da Literatura sobre problema de programação de quadro de horários educacional;
- Construção de uma heurística para o problema;
- Testes computacionais da heurística proposta para instâncias conhecidas na literatura;
- Comparação dos resultados obtidos com os da literatura;
- Desenvolvimento do *software* de gerenciamento de quadro de horários;

1.3 Contribuições

A principal contribuição desta dissertação é a proposta de um método que solucione o problema de programação de quadro de horário de forma eficiente. Ou seja, esse trabalho propõe um método capaz de fazer a programação de um quadro de horário viável, melhor e mais rápido em comparação aos gerados manualmente, além de obter resultados satisfatórios em relação aos métodos existentes na literatura.

Assim, inicialmente é proposto um método heurístico para gerar soluções de forma eficiente. Em uma segunda etapa são usados movimentos direcionados por metaheurísticas propostas na litetura que são capazes de melhorar significativamente as soluções geradas.

Como poderá ser visto nos próximos capítulos, o método proposto em comparações com as abordagens existentes da literatura, é competitivo.

1.4 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma: O capítulo 2 apresenta o exame da literatura e descreve as diferentes abordagens para problemas de quadro de horário educacional, assim como métodos para resolução dos mesmos. No capítulo 3 é descrito o problema especifico abordado neste trabalho. No capítulo 4 o método proposto é descrito. O capítulo 5 apresenta os testes computacionais. O capítulo 6 apresenta o software desenvolvido para criar e gerenciar um quadro de horários. As conclusões e principais contribuições alcançadas são apresentadas no capítulo 7.

Capítulo 2

Problema de Programação de Quadro de Horário Escolar

Neste capítulo é apresentada uma revisão bibliográfica para o problema de Quadro de horário educacional. Primeiramente é feita uma breve descrição sobre otimização combinatória, classe que o problema de programação de horário educacional está inserido. Posteriormente são mostrados problemas pertencentes à família de problemas de programação de horário educacional e suas principais características, assim como as formulações matemáticas. A seguir é feita uma breve contextualização histórica de metodologias para resolução desse problema. Por último, é feita uma breve descrição das melhores metodologias propostas para o problema especifico tratado neste trabalho, programação de quadro de horários de curso baseado em currículo.

2.1 Otimização Combinatória

Na otimização combinatória são estudados os problemas que têm a característica de possuir um número finito de soluções possíveis e embora, a princípio, uma solução ótima possa ser obtida através de uma simples enumeração, na prática, frequentemente isto se torna inviável, devido ao número extremamente alto de soluções possíveis. Nesse caso, mesmo que se utilize um supercomputador, o tempo de processamento poderia ser de várias semanas ou até anos.

A equação (2.1) é proposta por Frangouli [20] para determinar o espaço de pesquisa máximo para problemas de quadro de horário de cursos universitários. Sendo B o tamanho do espaço de pesquisa, b_d o número de dias letivos na semana, b_a o número de aulas para programação das aulas, b_s o número de salas disponíveis, b_m e b_c como sendo o número médio de aulas para cada curso, e o número de cursos respectivamente, temos:

$$B = (b_d \times b_a \times b_s)^{b_m \times b_c} \tag{2.1}$$

A fim de obter uma ideia sobre o fator combinatório do problema de programação de horários, considere um semestre em uma universidade, onde aulas de 30 cursos diferentes têm de ser escalonados, em média cada curso com 4 aulas semanais. Vamos supor também que há 5 dias úteis em uma semana, e cada dia com 10 períodos (horários), e que há 4 salas para que uma aula seja atribuída. Neste caso, que é bastante típico desse problema, o espaço de busca a ser examinado para a construção de um calendário semanal é igual a $(5 \times 10 \times 4)^{4\times30}$, ou seja, 1,32 $\times$ 10^{276}.

De forma geral, podemos dividir os problemas de otimização combinatória em: Decisão, Busca e Otimização.

- Problema de Decisão: Consiste em procurar uma resposta sim ou não para um determinado problema. Por exemplo: Existe um quadro de horário S que atenda R restrições? A resposta para esse problema seria se existe ou não.

- Problema de Busca: Consiste em procurar uma estrutura S que atenda R restrições. Por exemplo: Encontre um quadro de horário S que atenda R. A resposta para esse problema seria o quadro de horário.

- Problema de Otimização: Nesse problema, o objetivo é achar a melhor solução S que atenda R restrições, segundo algum critério C. Por exemplo: Encontrar o melhor quadro de horário S, atendendo R, de tal forma que o número de aulas isoladas seja o mínimo. A resposta seria uma solução S, tal que não exista nenhuma outra melhor para o problema.

Na literatura esses problemas são classificados como P, NP, NP-Completo e NP-Difícil. Os problemas pertencentes à classe P são aqueles que são possíveis de mostrar que existe um algoritmo, que no pior caso seja de ordem polinomial, que o resolva. Por exemplo, o problema de ordenação de um vetor, onde dada uma lista contendo n elementos, o objetivo é ordenar esses elementos segundo algum critério, é facilmente resolvido por algoritmos tais como o método Bolha, que tem a sua complexidade O(n^2) no pior caso, e o algoritmo *HeapSort*, O($n.log\ n$), no pior caso. Embora $log\ n$ não seja um polinómio, seu valor é menor que n, ou seja, não é pior que um polinômio, por isso pode ser encaixado entre os polinomiais. Ainda vale ressaltar que, na prática, os algoritmos polinomiais tendem a ter grau 2 ou 3 no máximo, e não possuem coeficientes grandes.

Os problemas pertencentes à classe NP (Não-determinístico Polinomial) compreendem todos os problemas de decisão φ, tais que existe uma justificativa à resposta SIM para φ, cujo passo de reconhecimento pode ser realizado por um algoritmo polinomial no tamanho da entrada de φ. Por exemplo: considere um problema de decisão referente ao problema de quadro

de horário onde se procura uma estrutura S que atenda R restrições. Uma justificativa para uma resposta SIM seria apresentando uma estrutura S. O passo de reconhecimento seria verificar se a estrutura S é válida em tempo polinomial. Vale ressaltar que na definição da classe NP não se exige uma solução polinomial para φ, é apenas necessário existir um algoritmo polinomial para verificar a resposta SIM.

É possível verificar se um problema pertence à classe P ou NP a partir de uma transformação polinomial. Imagine que um problema P1 tem que ser revolvido, e já é conhecido um algoritmo que resolva P2. A transformação polinomial pode ser realizada pelos seguintes passos:

- Transforme os dados de entrada $E1$ do problema $P1$ em dados de entrada E2 para o problema $P2$.

- Execute o algoritmo conhecido para resolver $P2$

- Transforme a solução $S2$ de $P2$ obtida com os dados de entrada E2 em uma solução S1 de P1.

Pode se dizer que um problema é polinomialmente transformável quando as transformações da entrada ($E1 \rightarrow E2$) e de saída ($S2 \rightarrow S1$) são feitas em tempo polinomial, e se $P2$ admitir uma resposta SIM se e somente se $P1$ tiver uma resposta SIM. Deste modo, podemos dizer se $P2 \in P$ então $P1 \in P$; de forma análoga, se $P2 \in NP$ então $P1 \in NP$.

É possível utilizar tais relações para dividir NP em classes de problemas equivalentes. Deste modo podemos inferir que os problemas $P \in NP$, porém são considerados os mais fáceis dessa classe.

A classe NP-Completo consiste na classe de problemas mais difíceis entre os problemas de NP. Para que um problema se enquadre nessa classe, é necessário: 1) Mostrar que o problema está em NP; 2) Mostrar que é possível fazer uma transformação polinomial com algum problema conhecido como NP-Completo ao problema estudado. Essa classe é dita a mais difícil no meio cientifico, devido ao estudo realizado por Cook [21]. A relação entre os problemas dessa classe permite dizer que se caso um problema NP-Completo seja resolvido em tempo polinomial, todos os outros poderão ser resolvidos.

Por fim, temos os problemas NP-Difíceis, os quais são compostos por problemas que atendam ao item 2 descrito anteriormente. Deste modo, todo problema NP-Completo $\subset$ NP-Difícil. Porém problemas NP-Difíceis podem não ser um problema de decisão, ou seja, logo nem todos os problemas NP-Difíceis são NP-Completos.

Ainda não foi descoberto nenhum algoritmo de tempo polinomial para resolver um problema NP-Completo; por esse motivo, são utilizados métodos aproximados ou heurísticos. Gomes [22] ressalta que a utilização de métodos de otimização para resolução de problemas combinatórios é restringido de forma considerável. Por exemplo, a utilização de algoritmos determinísticos voltados para otimização não-linear de espaços contínuos é impraticável, pois os mesmos dependem do cálculo da derivada da função que, por definição, não existe no espaço discreto onde tal problema é definido.

2.2 Problema de quadro de horários educacional

O problema de programação de quadro de horário educacional pertence à grande família de problemas de geração de quadro de horários(*timetabling*) e agendamento (*scheduling*), onde o objetivo é atribuir um conjunto de entidades (como tarefa, eventos públicos, pessoas entre outros) a um número limitado de recursos ao longo do tempo de tal forma que satisfaça um conjunto de requisitos de calendário.

Quando se trata de problemas de programação de horários em uma instituição de ensino, o objetivo é programar uma série de encontros, geralmente no período de uma semana, entre professores e alunos satisfazendo um conjunto de restrições que envolvem recursos (professores, salas de aula, salas especializadas), aspectos pedagógicos e satisfabilidade/indisponibilidade dos envolvidos.

Tais problemas são característicos, objetivos e restrições podem variar de instituição a instituição, mas de modo geral é possível dividir os problemas de quadro de horário escolar em três classes. Esta divisão é baseada no tipo de entidade envolvida (Universidade ou Escola) e os tipos de restrições associadas ao problema: Problema de programação de horários em Escolas, problema de programação de horários de Cursos, problema de programação de horários de Exames. Essa classificação foi proposta por Schaerf [23], sendo a mais utilizada na literatura, entretanto uma classificação mais abrangente e independente do tipo de instituição é encontrado no trabalho apresentado por Bardadym[24].

Em Souza [10], o problema de programação de quadro de horários em Escolas é definido da seguinte maneira: Seja um conjunto de turmas, um conjunto de professores e um conjunto de horários reservado para realização das aulas. As turmas são consideradas conjuntos disjuntos de alunos que têm o mesmo currículo. Para cada turma existe um conjunto de matérias, com suas respectivas cargas horárias que devem ser cursadas. Para cada professor é definido quais disciplinas e turmas ele deve lecionar. O objetivo básico é fazer um quadro de horário, geralmente semanal, de tal forma que: 1) as cargas horárias de todas disciplinas sejam cumpridas; 2) que cada turma não tenha aula com mais de um professor ao mesmo tempo; 3) que um professor

não seja alocado em mais de uma turma ao mesmo tempo. Nesse tipo de problema as aulas são realizadas geralmente em um único turno, além disso, as aulas normalmente possuem uma mesma duração. Outra característica desse problema é que, como as turmas são conjuntos de alunos disjuntos, elas recebem suas aulas em uma mesma sala, exceto para aulas de matérias que exijam salas especializadas, assim é o professor que desloca entre as salas.

Já o problema de programação de exames é definido como um problema relacionado a instituições com características de uma universidade típica. É composto por um conjunto de estudantes previamente matriculados em cursos, um conjunto de exames para cada estudante e um conjunto de horários disponibilizados para realizar exames. O objetivo primário nesse problema é alocar os exames a horários de modo que nenhum estudante tenha que fazer dois exames simultaneamente. Algumas restrições são típicas desse tipo de problema, tais como: $a)$ estudantes não podem fazer mais que um número *n* de exames por dia. $b)$ alguns exames de certos cursos não podem preceder a exames de outros. $c)$ alguns exames têm que ser alocados em um mesmo horário.

O problema de programação de cursos também é considerado um problema de uma instituição com características de uma universidade típica. Basicamente existe um conjunto de cursos (Programação, Cálculo etc), e para cada curso, um número de aulas. Existe um conjunto de currículos (Sistemas de Informação, Engenharia Elétrica etc), e para cada currículo se tem um conjunto de cursos. Estudantes se matriculam nos cursos disponíveis. Uma turma de um curso pode ter estudantes de vários currículos diferentes. Ainda existe um conjunto de horários disponivéis para alocação das aulas, e para cada horário um número limitado de salas. O problema consiste em alocar as aulas aos horários disponíveis, e atribuir cada aula a uma sala, respeitando disponibilidade e capacidade dessas, de forma que nenhum estudante tenha aulas simultâneas. Uma das características de problemas dessa classe é que ao contrário do problemas de horários em escola, existe uma maior flexibilidade em relação aos horários, pois a princípio um curso pode ser escalonado em todo o período de funcionamento da instituição (manhã, tarde e noite). Ainda outra diferença é que como o conceito de turmas nesse problema é diferente em relação ao problema em escolas secundárias, aqui os alunos deslocam-se para terem suas aulas, pois pode ocorrer em uma mesma aula vários alunos de currículos diferentes.

As restrições impostas a esses problemas podem ser divididas em duas classes: rígidas e flexíveis. As restrições rígidas são as que viabilizam uma solução, ou seja, quando não atendidas, a solução passa ser infactível, enquanto as restrições chamadas de flexíveis medem o grau de satisfabilidade de uma dada solução factível, essas variam geralmente de acordo com o problema a ser resolvido. Abaixo são listadas algumas restrições básicas para problemas dessa natureza e sua classificação:

- Sobreposição de professores: Um professor não pode lecionar mais de uma aula em um

dado período. (Rígida)

- Sobreposição de turma: Uma turma não pode ter mais de uma aula em um mesmo horário (período). (Rígida)
- Sobreposição de aulas especializadas: Em um dado horário (período) o número de salas especializadas requeridas, tais como laboratórios, não pode ser maior que o número de salas disponíveis.(Rígida)
- Restrições geográficas: É necessário que haja tempo suficiente para deslocar de um prédio a outro. Desse modo, aulas em dois locais distantes seja para professor ou aluno não podem ser alocados em horários seguidos. (Rígida)
- Preferencial: alocar professores a aulas de acordo com a sua preferência, tanto em relação a qual disciplina lecionar quanto aos dias da semana que sera alocado para lecionar uma disciplina. (Flexível)
- Compacidade: alocar aulas de um mesmo currículo seguidas, evitando aulas isoladas, ou janelas entre aulas. (Flexível)

Outras restrições podem ser vistas em [25, 26, 27].

Para se obter uma solução para o problema de quadro de horários podemos tratá-lo a partir de duas abordagens. A primeira é geralmente chamada de problema de viabilidade, ou busca, onde o objetivo é obter uma solução viável, ou seja, uma solução que atenda as restrições. Esse problema é demonstrado ser NP-Completo [18]. A seguinte é chamada de problema de otimização, onde o objetivo é encontrar entre todas as soluções viáveis, aquela(s) chamada(s) de ótima(s), nesse caso o problema pertence a classe dos NP-Difícil [28].

Grande parte das formulações referentes ao problema de quadro de horário pertence à classe dos problemas NP-Completos. A prova disso pode ser encontrado em diversos trabalhos, como por exemplo, em [18, 23, 29, 28, 30]. Isso implica que até o momento não existe nenhum algoritmo determinístico limitado polinomialmente que resolva tal problema.

Cooper e Kingston [18] provaram a NP-completude para diferentes interpretações desse problema em situações reais, a partir da transformação polinomial do problema em problemas conhecidos na literatura como NP-Completos, tais como coloração de grafos e *bin packing*.

No trabalho apresentado por Even *et. al* [29] foi mostrado que o problema de quadro de horários escolar é NP-Completo, a partir da redução polinomial de uma versão simplificada desse problema ao problema de Satisfabilidade. O problema de Satisfabilidade, também conhecido como SAT, consiste em dada uma expressão booleana na forma normal conjuntiva, verificar se ela pode ser satisfeita de modo que ao atribuir valores as variáveis a expressão seja verdadeira [31].

Existem formulações que podem ser resolvidas em tempo polinomial. Um exemplo é mostrado em Souza [10], porém não são considerados alguns requisitos importantes na construção de um quadro de horário, e quase sempre não atendem especificações de problemas reais, tais como requisitos pedagógicos, organizacionais e pessoais.

Os requisitos pessoais referem-se às preferências do corpo docente, tais como preferência de período para lecionar, quantidade de aulas geminadas, entre outros. Os organizacionais referem-se aos requisitos das instituições de ensino, como por exemplo, evitar a alocação isolada de professores (apenas uma aula no dia), geralmente procura-se concentrar as aulas em determinados dias, para que nos demais o professor possa se dedicar à pesquisa, planejamento de aulas e atendimento a alunos. Os pedagógicos são os requisitos que, se atendidos, podem proporcionar um melhor aproveitamento das aulas, como por exemplo evitar disciplinas com um alto grau de dificuldade sejam alocados em um mesmo dia para um determinado curso.

Lara [9] cita em seu trabalho que a função objetivo é uma das grandes dificuldades para se definir nesse problema. Geralmente as formulações medem o quanto uma solução está infringindo as restrições (especificações do problema), para cada uma é definido um coeficiente (penalidade). Deste modo o problema passa a ser como definir esses coeficientes. Daskalaki *et al.* [32] relatam que se todos os coeficientes forem definidos com o mesmo valor, todas as soluções viáveis poderiam em tese ser "ótimas" (degenerado), dificultando a convergência de alguns métodos.

2.3 Abordagens

As principais técnicas propostas na literatura para se resolver o problema de quadro de horário escolar se baseiam principalmente na simulação da forma humana de se resolver o problema e no problema de coloração de grafo [33].

As técnicas baseadas na forma humana de resolver o problema são chamadas de heurísticas construtivas, consistem no preenchimento gradual do quadro de horário, onde cada aula é alocada uma a uma até que todas as aulas sejam alocadas, ou até que um conflito apareça. Tal abordagem pode ser vista nos trabalhos [34, 35, 36], entre outros.

Técnicas baseadas em coloração de grafos são muito utilizadas devido aos bons resultados obtidos e a forma simples de se converter esse problema ao problema de quadro de horário, como pode ser visto em [37, 38, 39, 40, 30]. Por exemplo: Dado um grafo G não direcionado, composto por um conjunto de n vértices $V = v_1, ..., v_n$ e um conjunto E de arestas ligando vários pares distintos de vértices, o problema de coloração de grafos consiste em procurar atribuir cores a cada vértice pertencente a V dado que: (a) nenhum par de vértice com arestas em comum pode ter uma mesma cor atribuida, e (b) o número de cores usadas deve ser o mínimo.

Para representar o problema de quadro de horário como um problema de coloração de grafos, basta considerar cada evento como um vértice e adicionar uma aresta entre qualquer par de vértices que correspondem a um par de evento que não pode ser escalonado/atribuido a um mesmo período *(timeslot)*. Cada período disponível corresponde a uma cor, o objetivo é achar uma solução que não use mais cores que as disponíveis. Pré-alocações podem ser tratadas prefixando cores aos vértices correspondentes. Restrições de disponibilidade de professores e turmas são manipuladas restringindo-se cores que podem ser usadas nos vértices correspondentes. Um exemplo retirado do trabalho apresentado em [41] é mostrado na Figura 2.1.

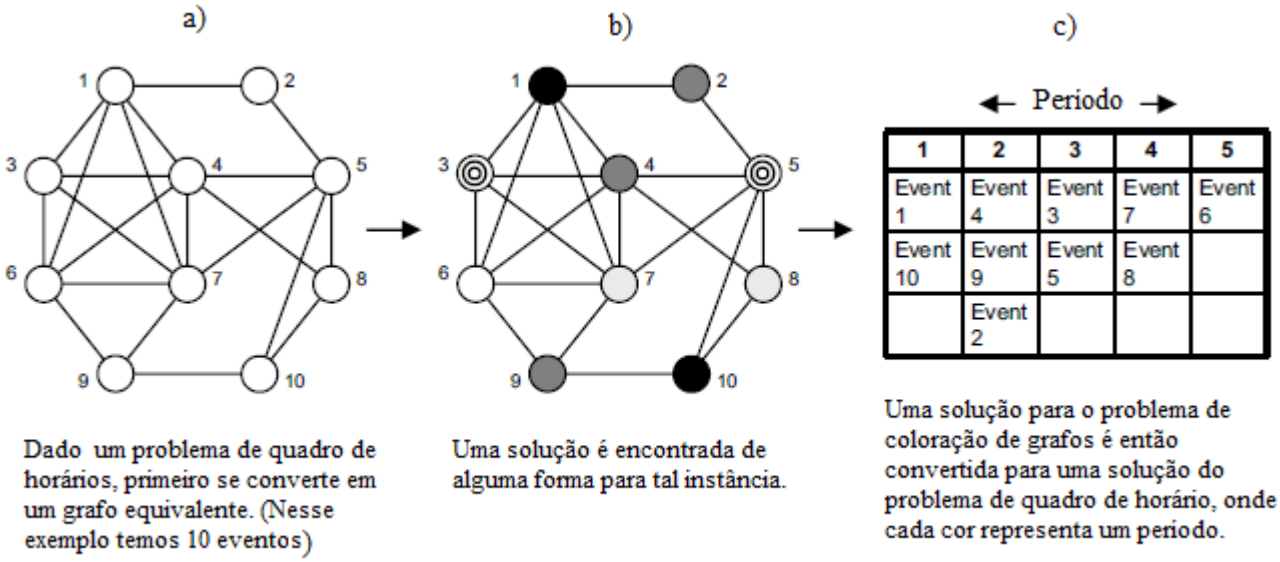

Figura 2.1. Transformação de um problema Timetabling em Coloração de grafos, e Coloração de grafos em timetabling

A parte *(a)* da Figura 2.1 representa um problema de quadro de horário representado em forma de grafo. A parte*(b)* representa o problema resolvido e a parte *(c)* o problema em forma de quadro de horário.

2.4 Formulação matemática para o problema de quadro de horário escolar

2.4.1 Problema de programação de horários em Escolas

Essa formulação foi proposta por de Werra [42]. Sejam: $T = \{t_1, t_2, ..., t_t\}$ o conjunto de turmas, $P = \{p_1, p_2, ..., p_p\}$ o conjunto de professores e $H = \{\ h_1, h2, ..., h_h\ \}$ o conjunto de horários, R

uma matriz de inteiros não negativos, em que $r_{ij} \in R$ é a carga horária do professor j na turma i.

O objetivo é encontrar

$$x_{ijk} \quad \forall i \in T,\ \ j \in P,\ \ k \in H \tag{2.2}$$

s.a

$$\sum_{k=1}^{|H|} x_{ijk} = r_{ij} \quad \forall i \in T,\ \ j \in P \tag{2.3}$$

$$\sum_{j=1}^{|P|} x_{ijk} \leq 1 \quad \forall i \in T,\ \ k \in H \tag{2.4}$$

$$\sum_{i=1}^{|T|} x_{ijk} \leq 1 \quad \forall j \in P,\ \ k \in H \tag{2.5}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in T,\ \ j \in P,\ \ k \in H \tag{2.6}$$

Nesta formulação, $x_{ijk} = 1$ caso o professor j tenha aula na turma i no período k, e $xijk = 0$ caso contrário. As restrições definidas pela equação 2.3 garantem o número de aulas correto entre o professor j e a turma i. A eq. 2.4 garante que não exista sobreposição de turmas e a eq. 2.5 que não ocorra sobreposição de professor.

Esse modelo não considera restrições tais como disponibilidade de professores e pré-alocações do tipo: professor p_p leciona na turma t_t em um horário k. Junginger [34] propôs uma formulação abordando tais restrições.

Encontrar

$$x_{ijk} \quad \forall i \in T,\ \ j \in P,\ \ k \in H \tag{2.7}$$

s.a

$$\sum_{k=1}^{|H|} x_{ijk} = r_{ij} \quad \forall i \in T,\ \ j \in P \tag{2.8}$$

$$\sum_{j=1}^{|P|} x_{ijk} \leq dt_{ik} \quad \forall i \in T,\ \ k \in H \tag{2.9}$$

$$\sum_{i=1}^{|T|} x_{ijk} \leq dp_{jk} \quad \forall j \in P, \ k \in H \tag{2.10}$$

$$x_{ijk} \geq A_{ijk} \quad \forall i \in T, \ j \in P, \ k \in H \tag{2.11}$$

$$x_{ijk} \in \{0,1\} \quad \forall i \in T, \ j \in P, \ k \in H \tag{2.12}$$

Nesta última formulação, $dt_{ik} = 1$ caso a turma i esteja disponível no horário k, ou $dt_{ik} = 0$ caso contrário; $dp_{jk} = 1$ se o professor j está disponível no horário k, ou $dp_{jk} = 0$ caso contrário; $A_{ijk} = 1$ caso exista uma pré-alocação professor j deve lecionar na turma i no período k, ou $A_{ijk} = 0$ caso contrário.

As restrições definidas pela eq. 2.9 garantem que uma turma não terá uma aula em um horário não disponível. Já a eq. 2.10 garante que um professor não será alocado em um horário não disponível para ele. A eq. 2.11, por sua vez, garante que uma pré-alocação será respeitada.

Nos modelos apresentados, o objetivo era apenas determinar uma solução viável (problema de viabilidade), porém na maioria dos problemas a necessidade é achar um boa solução, ou seja, otimizar uma função objetivo. Junginger [34] propôs uma função (eq. 2.13) com o objetivo de transformar o problema de viabilidade em um problema de otimização.

$$\min \sum_{i=1}^{|T|} \sum_{j=1}^{|P|} \sum_{k=1}^{|H|} d_{ijk} x_{ijk} \tag{2.13}$$

Esta função objetivo atua como um mecanismo de penalização, onde d_{ijk} é um valor associado à satisfabilidade de o professor i ser alocado à turma j no horário k. Outras formulações para esse problema podem ser vistas nos trabalho apresentados por Yoshihawa *et al.* [43] e Santos [44].

2.4.2 Problema de programação de horários de Cursos

Uma formulação básica para esse problema, extraída do trabalho apresentado por Werra [42], é apresentado a seguir. Sejam n cursos $C = \{c_1, ..., c_n\}$, onde o curso c_n consiste em l_n aulas. Seja q o q-ésimo currículo $CR = \{Cr_1, ..., Cr_q\}$ com grupos de cursos que possuem estudantes em comum. Isso significa que todos os cursos em Cr_q devem ser agendados em tempos diferentes. Ainda, sendo $lmax_k$ o número máximo de aulas que podem ser agendadas no período k (ou seja, o número de salas disponíveis no período k). O modelo é dado pelas

equações (2.14) a (2.18).

$$max \sum_{i=1}^{c} \sum_{k=1}^{p} d_{ik} y_{ik} \tag{2.14}$$

$$\sum_{k=1}^{|H|} y_{ik} = l_i \quad \forall i \in C \tag{2.15}$$

$$\sum_{i=1}^{n} y_{ik} \leq lmax_k \quad \forall k \in H \tag{2.16}$$

$$\sum_{i \in Cr_l} y_{ik} \leq 1 \quad \forall l \in CR, \;\; k \in H \tag{2.17}$$

$$y_{ik} = \{0, 1\} \quad \forall i \in C, \;\; k \in H \tag{2.18}$$

Neste modelo, $y_{ik} = 1$ se a aula do curso *i* é agendada no período k, e $y_{ik} = 0$ caso contrário. O parâmetro d_{ik} é um valor associado com a prioridade de alocar uma aula do curso i ao período k.

As restrições representadas pela equação 2.15 atribuem a cada curso um número correto de aulas. Pela equação 2.16 garante-se que não seja agendado um número maior de aulas do que o número de salas disponíveis para cada período. Já as restrições representadas pela equação 2.17 previnem o conflito de aulas agendadas em um mesmo período.

2.4.3 Problema de programação de horários de Exames

Similar ao problema de programação de horários para cursos, a formulação desse problema é apresentada a seguir: Sejam n cursos C $= \{c_1, ..., c_n\}$, e para cada curso, l_n^e exames. Seja e o e-ésimo grupo de exames $S = \{s_1, ..., s_e\}$ com grupos de estudantes em comum. Isso significa que todos os exames em S_e devem ser agendados em tempos diferentes. Ainda, sendo $k \in |H|$ definindo um período, $lmax_k^e$ o número máximo de exames que podem ser agendados no período k e $y_{ik} = 1$ se um exame do curso $i \in C$ é agendado no período k, e $y_{ik} = 0$ caso contrário.

As restrições representadas pela equação 2.20 atribuem a cada curso um número correto de exames. Pela equação 2.21 limita-se o número de exames por período. Já as restrições representadas pela equação 2.22 previnem o conflito de exames agendados em um mesmo período para cada conjunto de curso. A função objetivo nessa formulação considera que o desejável para o problema de quadro de horários de exames é evitar que um mesmo estudante faça dois

exames consecutivos [42].

$$min \sum_{k=1}^{|H|-1} \sum_{z=1}^{e} \sum_{i,j \in S_z : i \neq j} y_{ik} \;\; y_{j,k+1} \tag{2.19}$$

$$\sum_{k=1}^{|H|} y_{ik} = l_i^e \quad \forall i \in C \tag{2.20}$$

$$\sum_{i=1}^{n} y_{ik} \leq lmax_k^e \quad \forall k \in H \tag{2.21}$$

$$\sum_{i \in S_s} y_{ik} \leq 1 \forall \quad s \in S, \;\; k \in H \tag{2.22}$$

$$y_{ik} = \{0, 1\} \quad \forall i \in C, \;\; k \in H \tag{2.23}$$

2.5 Histórico

Segundo Lobo [45], uma das primeiras referências para esse problema é apresentada em [46] (1960). Nesse trabalho os autores apresentaram técnicas para construção de uma solução por intermédio de computadores, sendo feitas comparações entre problemas de quadro de horários e outros problemas de agendamento típicos da época.

Posteriormente Csima e Gotlieb [11] também propuseram um método utilizando computador baseado em operações envolvendo matrizes binárias. Pequenos testes mostraram que o método proposto obteve um bom desempenho para várias instâncias-teste.

Gotlieb [12] apresentou a primeira formulação completa para o problema de quadro de horários para cursos *(Timetabling Problem-Course Scheduling)*. Essa abordagem consistiu fixar em um determinado período de tempo, um conjunto de aulas de forma que atendesse as exigências acadêmicas estabelecidas por um determinado currículo de estudos para um certo grupo de disciplinas. Algumas restrições não foram contempladas para esse problema com o objetivo de simplificação, tal como disponibilidade de professor.

Broder [47] foi pioneiro ao trabalhar no problema de programação de quadro de horários para exames. Esse trabalho teve como objetivo minimizar os horários conflitantes para ocorrência de exames (avaliações) finais. Foi utilizado o algoritmo *Largest Degree First*, baseado em coloração de grafos, e o algoritmo de Monte Carlo para criar soluções. Nesse mesmo ano, Cole [48] deu sua grande contribuição ao propor um método que permitia inserir a esse problema algumas restrições não abordadas antes, tais como: estabelecimento de ordem entre exames, restrição de capacidade de número de alunos em sala de aula, entre outros.

O método húngaro[49] foi aplicado por Lions [50] ao problema de quadro de horários. Essa metodologia foi utilizada devido a similaridade encontrada por esse autor entre a matriz incidência de um problema de quadro de horários com o problema de transporte.

Ainda na década de 60 outras formulações e métodos foram propostos para revolver problemas dessa natureza, como pode ser visto em [51, 52, 53, 54, 55].

Garey e Johnson [28] demonstraram que o problema de coloração de grafos e *bin packing* são casos particulares do problema de programação de horários. Vários trabalhos referentes ao problema de quadro de horário foram modelados na forma de um problema aplicado a grafos [56, 57, 58, 59].

Schmit [60] em 1980 em sua revisão bibliográfica sobre problemas de quadro de horários, mencionou que embora não existisse uma terminologia comum no meio cientifico para descrever o problema, esse já era citado em livros de teoria de grafos, como pode ser visto em [61]. Foi apresentado por Manvel [62] uma abordagem para coloração de grandes grafos como forma de trabalhar com instâncias mais complicadas do problema de programação de quadro de horários.

No trabalho apresentado por Gans[36] foi proposto um procedimento heurístico construtivo não determinista na escolha de aulas e horários para escalonamento. Mata [63] também propôs uma heurística construtiva onde uma restrição de aulas geminadas para determinadas disciplinas foi inserida.

Um procedimento baseado em fluxo em grafos foi proposto por Ostermann e Werra [64]. Essa abordagem consistiu em dividir o problema de quadro de horários a uma sequência de problemas de fluxo em grafos, onde o objetivo era achar o maior emparelhamento de peso máximo, dado que para cada aula é criado um grafo e existe um nó professor ligado a um nó turma caso exista um encontro pré-determinado. Nessa abordagem o custo de cada arco define a urgência da aula.

Cangalovic e Schreuder [38] apresentaram um método exato para tratar problemas de horários em escolas onde o tempo de duração das aulas são diferentes. A partir de um abordagem por coloração de grafos, onde cada aula foi representada por um vértice cujo seu peso era proporcional a duração da aula, foi proposto um algoritmo de enumeração implícita baseado no princípio *branch-and-bound*[65].

Fahrion e Dollansky [66] trataram o problema de programação de horários como um problema de satisfação de restrição (PSR). Em um PSR procura-se encontrar o valor de cada variável discreta que satisfaça um conjunto de restrições. Para resolver um PSR é utilizada alguma linguagem baseada em restrições, permitindo expressar restrições de diversos tipos, as quais são verificadas durante os cálculos [10]

Caldeira [67] propôs o uso de algumas versões de algoritmos genéticos para resolver pequenas instâncias do problema de quadro de horário escolar, o objetivo desse trabalho foi

comparar as diferentes abordagens.

Uma heurística baseada em caminhos mínimos em grafo para melhorar um quadro de horário é proposto em Souza [10]. Também nesse trabalho uma modelagem matemática para o problema é apresentado.

Vários outros trabalhos propuseram outras abordagens para revolver esse problema. Os citados anteriormente são apenas parte dos trabalhos importantes na área de estudo, e que contribuíram na evolução desse trabalho. Na próxima seção são descritos os principais métodos heurísticos utilizados nos dias atuais para resolução do problema de quadro de horários para cursos.

2.6 Heurísticas

Muitos métodos heurísticos têm sido utilizados na produção automatizada de quadro de horários. Entre as heurísticas se destacam as metaheurísticas baseadas em busca local: Busca Tabu e *Simulated Annealing*, que têm se mostrado as melhores em relação à qualidade de solução.

Além disso, outras metaheurísticas tais como Algoritmo Genético, GRASP [68], ILS(Iterated Local Search) [19], VNS (Variable Neighborhood Search) [69] e híbridos têm sido aplicadas a esse problema. Nesta seção serão apresentadas as principais metaheurísticas utilizadas na literatura para resolver o problema tratado e correlatos.

2.6.1 Busca Tabu

A Busca Tabu é um método de busca local que consiste em explorar o espaço de soluções movendo-se de uma solução para outra que seja seu melhor vizinho. Esta estratégia, juntamente com uma estrutura de memória para armazenar as soluções geradas (ou características destas) permite que a busca não fique presa em um ótimo local.

Esse método consiste em dada uma solução inicial v, a cada iteração explorar um subconjunto V da vizinhança de v. O melhor vizinho $v' \in V$ é escolhido para continuar o processo mesmo que não seja melhor que a solução v. Tal estratégia é utilizada para escapar de ótimos locais, porém pode fazer com que o algoritmo cicle, ou seja, fique alternando entre um conjunto de soluções durante todo o processo de otimização.

Para evitar que isso ocorra, é criada um lista denominada Lista Tabu. Essa lista contém todos os movimentos feitos anteriormente no processo de otimização, não permitindo que sejam feitos durante um certo tempo t, evitando que ao algoritmo faça buscas em lugares repetidos (ciclagem). Porém, essa lista também pode impedir que o algoritmo alcance soluções ainda não visitadas; por isso, um critério de aspiração dos elementos da lista é criado; um exemplo seria após um número t de iterações, o elemento é retirado da lista.

O método é interrompido quando algum critério de parada é satisfeito, tais como número máximo de iterações, obtenção da solução alvo, tempo de processamento, entre outros.

Hertz [37] propôs métodos de Busca Tabu para resolver o problema de quadro de horário universitário, que se baseiam em um método para o problema de coloração de grafos apresentado em Hertz et al.[70]. Nessa abordagem o autor dá prioridade significativa às restrições do tipo rígidas *(Hard)*. Verificou-se em testes para duas instituições de ensino que os métodos propostos obtiveram melhores resultados quando comparadas as soluções geradas manualmente.

Um algoritmo de Busca Tabu Adaptativo composto por 3 fases, inicialização, intensificação e diversificação foi proposto por Lu et al. [71]. A fase de inicialização constrói um quadro de horário factível de forma gulosa. Posteriormente as fases de intensificação e diversificação são utilizadas para reduzir o número de restrições flexíveis violadas mantendo a solução factível. Na fase de intensificação é usado um algoritmo de Busca Tabu, enquanto na fase de diversificação é usado o método *Iterated Local Search* [19]. Um operador/movimento denominado *kempeswap* é proposto nesse trabalho. *Kempeswap* é um procedimento definido pela troca de horários entre as aulas de duas cadeias *Kempe*. Se focarmos somente nas disciplinas e conflitos de currículo, cada solução para o problema pode ser representada por um grafo G onde os vértices são as aulas e as arestas interligam aulas de um mesmo currículo ou ministradas por um mesmo professor. Em um quadro de horário viável, uma cadeia *Kempe* é o conjunto de vértices que formam um componente ligado no subgrafo de *G* induzido por os vértices que pertencem a dois períodos. Resultados mostraram que o algoritmo foi eficiente quando comparados com outros da literatura.

Outros métodos baseados em Busta Tabu para problemas de quadro de horários podem ser vistos em [30, 40, 72, 73]

2.6.2 Simulated Annealing

A metaheurística *Simulated Annealing* (SA) consiste em uma técnica de busca local probabilística, proposta originalmente por Kirkpatrik *et al.* [74] que se fundamenta em uma analogia com a termodinâmica, ao simular o resfriamento de um conjunto de átomos aquecidos, operação conhecida como recozimento. Esta técnica começa sua busca a partir de uma solução inicial qualquer. O procedimento principal consiste em um laço que gera aleatoriamente, em cada iteração, um único vizinho v' da solução corrente v.

Considerando-se um problema de maximização, seja $\triangle$ a variação de valor da função objetivo ao mover-se para uma solução vizinha candidata, isto é, $\triangle$ = $f(v')$ - $f(v)$. O método aceita o movimento e a solução vizinha passa a ser a nova solução corrente se $\triangle > 0$. Caso $\triangle < 0$ a solução vizinha candidata também poderá ser aceita, mas neste caso, com uma probabi-

lidade $e^{-\triangle/temp}$, sendo $temp$ um parâmetro do método, chamado de temperatura, que regula a probabilidade de se aceitar soluções de pior custo.

A temperatura $temp$ assume inicialmente um valor elevado, $temp_0$, permitindo com maior probabilidade que o método aceite uma solução vizinha pior ou igual a corrente para uma eventual busca. Após um numero de iterações pré-estabelecido o valor de $temp$ diminuirá gradativamente através de um parâmetro chamado razão de resfriamento, rt, que é responsável em reduzir a temperatura e, assim, a probabilidade de aceitar soluções piores ou iguais à corrente.

O método é interrompido quando a temperatura $temp$ está próxima de zero e nenhuma solução de piora é aceita, isso é, quando o sistema se torna estável.

No trabalho apresentado por Saleh e Coddington [75] foi investigado uma série de abordagens baseadas em *simulated annealing* para resolver o problema de quadro de horário universitário (curso). Foram verificados meios de resfriamento diferentes. Uma das abordagens, denominada *Simulated Annealing* com resfriamento adaptativo e reaquecimento, se mostrou muito robusta, obtendo um quadro de horário válido para um problema de grande escala. Outros trabalhos baseados em nesta metaheurística para resolução do problema de quadro de horários podem ser visto em [76, 77]

2.6.3 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure*) é um método iterativo, proposto por Feo e Resende[68], que consiste de duas fases: uma fase de construção, na qual uma solução é gerada, elemento a elemento; e de uma fase de busca local, na qual um ótimo local na vizinhança da solução construída é pesquisado.

Na fase de construção para a seleção de um elemento para solução parcial, a princípio todos os candidatos são considerados; contudo, o número de candidatos pode em muitos casos ser extremamente elevado. Por isso, normalmente, é considerada apenas uma lista restrita de candidatos (LRC). A LRC é composta por elementos p que atendam a equação 2.24:

$$LRC \leq \min + \alpha(\max - \min) \tag{2.24}$$

onde α é um parâmetro de entrada, que define o tamanho da lista restrita de candidatos, podendo variar de 0 a 1. Quanto maior este parâmetro, mais candidatos são aceitos para compor a solução parcial. A partir de uma LRC, seleciona-se um elemento deste conjunto aleatoriamente e não necessariamente o melhor. Essa escolha aleatória permite que este procedimento possa ser usado várias vezes para obter soluções distintas. Pela forma como são construídas as soluções na etapa inicial, essas não representam na maioria dos casos um ótimo local, por esse motivo é fortemente recomendada uma etapa de busca local. As iterações do GRASP são totalmente independentes, esse é o motivo de ser também conhecida como uma heurística do tipo multistart.

A melhor solução encontrada ao longo de todas as iterações GRASP realizadas é retornada como resultado.

Barbosa et al. [78] propôs uma metaheurística utilizando a fase de construção GRASP seguida de buscas locais e um procedimento de perturbação baseado na metaheurística ILS (*Iterated Local Search*) [19]. Em cada execução foi gerada apenas uma solução inicial devido ao alto custo computacional. Resultados mostraram que o algoritmo foi eficiente, ficando entre os 5 melhores da *The second international timetabling competition* (ITC 2007). Outro trabalho utilizando tal metodologia pode ser visto em [79].

2.6.4 Algoritmo Genético

Algoritmos Genéticos foram propostos por Holland [80] como sendo algoritmos de busca com um propósito geral, tendo características estocásticas, busca de múltiplos pontos inspirados na evolução. Inicialmente o objetivo do seu trabalho era explicar os processos adaptativos em sistemas naturais para desenvolver sistemas artificiais baseados nestes processos. Posteriormente tal metodologia foi empregada na resolução de problemas de otimização.

O comportamento desses algoritmos correspondem a analogia de uma seleção natural e da genética. Basicamente incorporam uma solução potencial para um problema específico numa estrutura semelhante a de um cromossomo e aplicam operadores de seleção, mutação e "cross-over" a essas estruturas de forma a preservar informações críticas relativas à solução do problema. Durante o processo de busca esses cromossomos passam por situações tais como a teoria da evolução, onde disputam recursos, e o mais fortes ou adeptos ao meio sobrevivem durante as gerações, fazendo com que os outros deixem de existir.

Ueda et al.[81] apresentou um algoritmo genético de duas fases para resolver o problema de quadro de horários. Esse trabalho se baseou na criação de duas diferentes populações, na primeira indivíduos relativos à programação de classes e na segunda a alocação de salas. Uma combinação entre os indivíduos de ambas as populações foi utilizada para o calculo do desempenho *fitness* em cada geração. Foi possível mostrar que essa abordagem era mais eficiente que um algoritmo genético comum.

Outros trabalhos, utilizando algoritmos genéticos para o problema de horários, podem ser encontrados em [82, 83, 84, 85, 86].

2.6.5 Outros Algoritmos

Um estudo sobre um *framework* baseado em hiperheurística para problemas de quadro de horários universitário é feito por Qu et al. [87]. Uma hiperheurística é uma heurística que pode ser utilizada para lidar com qualquer problema de otimização, desde que a ela sejam fornecidos

alguns parâmetros, como estruturas e abstrações, relacionados ao problema considerado. Resultados competitivos foram obtidos quando comparados aos melhores resultados da literatura, tanto para o problema de escalonamento de horário quanto a exames. Foi possível mostrar sua eficiência e generalidade em domínios diferentes de problemas.

Algoritmos híbridos compostos por mais de uma metaheurísticas têm sido utilizados com sucesso para resolver problemas de otimização combinatória. Em Bellio et al. [88] foi proposto um algoritmo de busca local híbrido composto pelas metaheurísticas *Simulated Annealing* e Busca Tabu. Foi definido nesse trabalho um espaço de busca consistindo apenas em algumas restrições do tipo rígida, as demais foram adicionadas a função objetivo, penalizando em grande escalas quando violadas. A principal razão de abordar esse problema dessa maneira é que algumas restrições envolvem muitas variáveis, dessa maneira é mais usual gerenciar tais restrições por uma função custo. Foi usada uma estratégia de controle de pesquisa chamada *token-ring* para hibridizar os componentes da busca local. Essa estratégia consiste em dado um estado inicial e um conjunto de algoritmos, fazer circular uma busca para cada algoritmo a partir da melhor solução encontrada anteriormente. Esse algoritmo se mostrou eficiente, conseguindo 13 melhores resultados dos 21 conhecidos na literatura, para as mesmas instâncias que serão utilizadas no presente trabalho.

Em Fonseca et al. [89] o problema de quadro de horário universitário é reduzido ao problema da Satisfabilidade (SAT) para geração de uma solução a partir de um algoritmo resolvedor SAT[90] e posteriormente uma metaheurística Busca Tabu é utilizada para otimização da solução obtida. A principal contribuição desse trabalho foi a união de tais metodologias, ainda pouco utilizadas na literatura. Um trabalho relevante foi apresentado por Achá e Nieuwenhuis [8], que reduziu o problema abordado ao problema *Weighted Partial Max SAT*, abordagem voltada para otimização do problema de satisfabilidade, obtendo 10 das 32 melhores soluções conhecidas para um conjunto de instâncias.

No trabalho apresentado por Geiger [91] é proposta uma metaheurística baseada na idéia do método *Threshold Accepting*. De forma geral, esse método se baseou apenas na remoção e reatribuições de eventos pertencentes ao quadro de horário. Resultados obtidos para o conjunto de instâncias do ITC 2007 mostraram que o método proposto foi eficiente, ficando entre os cinco primeiros colocados nessa competição.

Muller [92] apresentou um algoritmo de busca consistindo em 3 fases. Na primeira fase uma solução é criada utilizando o algoritmo *Iterative Forward Search* (IFS) [25]. Esse método consiste em gerar uma solução iterativamente baseado na não violação de restrições. A cada passo é escolhida uma variável que ainda não faz parte da solução. Caso seja atribuído a ela um valor que resulte em um conflito, as variáveis conflitantes passam a não fazer parte da solução corrente. O método termina quando todas as variáveis estiverem atribuídas à solução. Na fase seguinte é usado o algoritmo *Hill Climbing* (HC) [93] para procurar um ótimo local.

Esse método consiste em busca local simples que é executada até que soluções melhores que a corrente não possam ser mais encontradas. Por fim, os métodos *Great Deluge* [94] e *Simulated Annealing* são usados na busca de melhores soluções.

Great Deluge (GD) é um algoritmo que utiliza um limite *B* que é imposto sobre o valor da solução corrente. No caso de um problema de minimização, uma nova solução é aceita caso seu valor seja menor que esse limite. O valor inicial de *B* é calculado de acordo com a equação 2.25:

$$B = GD_{sup} \times S_{best} \tag{2.25}$$

Nesta equação, S_{best} é a melhor solução encontrada para o problema até o momento e GD_{sup} é um parâmetro denominado coeficiente de limite superior. A cada iteração, B é diminuído por uma taxa de resfriamento GD_{dc}, como é mostrado na equação 2.26.

$$B = B \times GD_{dc} \tag{2.26}$$

Essa busca continua até que B chegue ao valor de um limite inferior calculado por:

$$B \leq S_{best} \times GD_{inf}^{at} \tag{2.27}$$

Quando a condição expressa pela equação 2.27 é satisfeita, B é reiniciado de acordo com a equação 2.28. O parâmetro at é iniciado com 1, e a cada vez que o limite inferior é alcançado, seu valor é aumentado em uma unidade. Caso seja encontrada uma solução melhor que a global, o valor de at volta a receber 1.

$$B \doteq S_{best} \times GD_{ub}^{at} \tag{2.28}$$

O método Simulated Annealing é usado quando o limite inferior é alcançado pelo método GD. Posteriormente, quando o SA acaba sua execução, é passado o controle para o método HC. Esse laço é executado até que um critério de parada seja alcançado.

Esse método foi o finalista em três sessões da competição ITC2007, sendo ganhador de 2 delas. Tal metodologia se mostrou eficiente para uma ampla classe de problemas de *timetabling* com apenas pequenos ajustes, fazendo assim um dos algoritmos mais referenciados nessa linha de pesquisa atualmente.

Capítulo 3

Problema de programação de Quadro de Horário de Cursos baseado em Currículo

Neste capitulo é descrito o problema abordado em relação à formulação, instâncias e algumas regras utilizadas no ITC 2007. Tais regras serão respeitadas para que se possa fazer uma comparação utilizando os mesmos critérios em relação as metodologias finalistas do ITC 2007.

3.1 Descrição do Problema

O problema de quadro de horários baseado em currículo (*Curriculum Based Course Timetabling*) consiste em escalonar l aulas de vários cursos, dado um número de salas e períodos (dias $\times$ horário), em que conflitos entre os cursos são definidos de acordo com os currículos publicados pela IES. A formulação utilizada neste trabalho é baseada na proposta por Di Gaspero *et al.* [95], sendo composta pelas seguintes entidades:

- Períodos: Um período é um par composto por um dia e um horário. Cada dia é dividido em um número fixo de intervalos de tempo (horários), que é igual para todos os dias. O número total de períodos é o produto dos intervalos de tempo e dias.
- Cursos e Professores: Cada curso é composto por um número fixo de palestras a serem agendadas em períodos distintos, um número de alunos, e um professor. Por exemplo, o curso Programação é lecionado pelo professor Prof1, e possui 50 alunos.
- Salas: Cada sala possui uma capacidade expressa em termos de números de mesas (assentos) disponíveis.

- Currículo: Consiste em um grupo de cursos, tal que qualquer par de cursos do grupo tem estudantes em comum. Baseado no currículo se tem os conflitos entre os cursos e outras restrições flexíveis.

A solução desse problema é atribuir aos períodos e salas, todas as aulas de cada curso, violando o mínimo de restrições flexíveis e nenhuma do tipo rígida.

Para fins didáticos, são mostradas a seguir as notações utilizadas na formulação matemática deste trabalho.

Parâmetros:

Notação	Descrição
C	conjunto de cursos, $C = \{ c_1, c_2, ..., c_n \}$
SL	conjunto de salas, $SL = \{sl_1, sl_2, ..., sl_{sl} \}$
H	conjunto de períodos, $H = \{ h_1, h_2, ..., h_h \}$
CR	conjunto de currículo, $CR = \{ Cr_1, Cr_2, Cr_{cr}\}$ onde $Cr_z = \{c_1, c_n\}$
D	conjunto de períodos pertencentes a um dia,
	$D = \{ d_1 = (h_1, h_2, ..., h_5), ..., d_d = (h_6, ..., h_h)\}$
l_i	número de aulas para o curso $i \in C$
cap_i	capacidade da sala $i \in SL$ em termos de assentos
dem_i	número de alunos matriculados no curso $i \in C$
md_i	número mínimo de dias que uma disciplina
	$i \in C$ deve ser atribuída ao quadro de horário.

$$t_{ij} = \begin{cases} 0, & \text{se o curso } i \in C \text{ possui uma indisponibilidade no período } j \in H \\ 1, & \text{caso contrário} \end{cases}$$

Variáveis:

Notação	Descrição
cs_{ik}	violação da capacidade de uma sala $k \in R$ em relação demanda
	de um curso $i \in C$
mdt_i	violação do número mínimo de dias trabalhados de um curso $i \in C$
es_i	violação de estabilidade de sala para disciplina $i \in C$
as_{ik}	referente à alocação da sala $k \in SL$ para o curso $i \in C$
cmp_{ij}	violação de compacidade do currículo $i \in CR$ no dia $j \in D$

$$x_{ijk} = \begin{cases} 1, & \text{se o curso } i \in C \text{ foi alocado no período } j \in H \text{ na sala } k \in SL \\ 0, & \text{caso contrário} \end{cases}$$

3.1.1 Restrições Rígidas

- Aulas (H_1): Todas as palestras de um curso devem ser agendadas, e devem ser atribuídas a períodos distintos. Uma violação ocorre quando uma palestra não é agendada. As equações 3.1 e 3.2 definem matematicamente essa restrição:

$$\sum_{j \in H} \sum_{k \in SL} x_{ijk} = l_i \ \ \forall i \in C \tag{3.1}$$

$$\sum_{k \in SL} x_{ijk} \leq 1 \ \ \forall i \in C, j \in H \tag{3.2}$$

- Ocupação de Salas(H_2): Duas aulas não podem ser atribuídas a uma mesma sala em um mesmo período (Eq. 3.3).

$$\sum_{i \in C} x_{ijk} \leq 1 \ \ \forall j \in H, k \in SL \tag{3.3}$$

- Conflitos(H_3): Aulas de um mesmo currículo ou lecionadas por um mesmo professor devem ser atribuídas a diferentes períodos. Duas aulas conflitantes no mesmo período representam uma violação (Eq. 3.4).

$$\sum_{i \in Cr_{cr}} \sum_{k \in SL} x_{ijk} \leq 1 \ \ \forall j \in H, \ cr \in CR \tag{3.4}$$

  É importante ressaltar que currículos auxiliares são criados para representar cursos lecionados por um mesmo professor. Por exemplo, se dois cursos, c_1 e c_2 são lecionados por um mesmo professor, então um currículo auxiliar aux é criado contendo esses dois cursos: $cr_{aux} = \{c_1, c_2\}$. Assim essa restrição atende a imposição que aulas lecionados por um mesmo professor não podem ser escalonadas em um mesmo horário.

- Disponibilidade(H_4): Se o professor do curso não estiver disponível para ensinar em um dado período, então as aulas não podem ser atribuídas a esses períodos. Uma violação ocorre quando uma aula for atribuída a um período indisponível (Eq. 3.5).

$$\sum_{k \in SL} x_{ijk} \leq t_{ij} \ \ \forall j \in H, i \in C \tag{3.5}$$

3.1.2 Restrições Flexiveis

- Capacidade da Sala (S_1): Para cada palestra, o número de estudantes deve ser menor ou igual ao número de mesas (assentos). Cada estudante a mais conta como uma violação, as equações 3.6 e 3.7 representam matematicamente essa restrição.

$$cs_{ik} \geq 0 \tag{3.6}$$

$$cs_{ik} \geq x_{ijk} \times (dem_i - cap_k) \ \ \forall j \in H, i \in C, k \in SL \tag{3.7}$$

- Mínimo de Dias Trabalhados (S_2): As palestras de cada curso devem ser distribuídas entre um número mínimo de dias. Uma violação ocorre quando as aulas são escalonas em menos dias que o mínimo determinado. Matematicamente são utilizadas três equações para definir essa restrição. A equação 3.8 verifica se existe aula do curso i no dia w. Já a função 3.9 calcula se a quantidade mínima de dias é respeitada. A equação 3.10 é utilizada pois caso o curso esteja escalonado em mais dias que o mínimo desejado, a equação 3.9 retornará um valor negativo, nesse modelo nos interessa os valores positivos em relação à violação de uma restrição.

$$m_{iw} = \begin{cases} 1, & \text{Se } \sum_{j \in D_w} \sum_{k \in SL} x_{ijk} \geq 0; \\ 0, & \text{caso contrário.} \end{cases} \quad w \in D \ , \ \forall i \in C \tag{3.8}$$

$$rd_i \geq md_i - \sum_{w \in D} m_{iw} \ \ \forall i \in C \tag{3.9}$$

$$rd_i \geq 0 \ \ \forall i \in C \tag{3.10}$$

- Compacidade de Currículo (S_3): Aulas pertencente a um currículo devem ser adjacentes (períodos consecutivos). Para um dado currículo é contabilizada uma violação quando uma palestra não está adjacente a outra palestra em um mesmo dia. Matematicamente essa restrição é representada pelas equações 3.11 e 3.12.

$$\sum_{i \in Cr_g} \sum_{k \in SL} (x_{i,j-1,k} + x_{i,j+1,k}) + (1 - \sum_{i \in Cr_g} \sum_{k \in SL} x_{i,j,k}) \leq 2 + cmp_{gh} \tag{3.11}$$

$$\forall g \in CR, \ h \in D, \ j \in d_h : 2 \leq j \leq |H| - 1$$

$$(2 - \sum_{i \in Cr_g} \sum_{k \in SL} (x_{i,j-1,k} + x_{i,j+1,k})) + \sum_{i \in Cr_g} \sum_{k \in SL} x_{i,j,k} \leq 2 + cmp_{gh} \tag{3.12}$$
$$\forall g \in CR,\ h \in D,\ j \in d_h : 2 \leq j \leq |H| - 1$$

- Estabilidade de Sala (S_4): Todas as aulas de um curso devem ser dadas em uma mesma sala. Uma violação ocorre quando existem aulas de um mesmo curso em salas diferentes (Eq. 3.13 e Eq. 3.14).

$$as_{ik} = \begin{cases} 1, & \text{Se } \sum_{j \in H} x_{ijk} \geq 0,\ \ \forall i \in C\ , k \in SL; \\ 0, & \text{caso contrário.} \end{cases} \tag{3.13}$$

$$es_i \geq \sum_{k \in SL} as_{ik} - 1,\ \ \forall i \in C \tag{3.14}$$

3.1.3 Função Objetivo

A função objetivo (FO) busca minimizar o número de violações em relação às restrições flexíveis, sendo definida pela eq. 3.15.

$$\alpha_1 \sum_{i \in C} \sum_{k \in SL} cs_{ik} + \alpha_2 \sum_{i \in C} rd_i + \alpha_3 \sum_{q \in CR} \sum_{w \in D} cmp_{qw} + \alpha_4 \sum_{i \in C} es_i \tag{3.15}$$

Nesta equação, α é um coeficiente de penalidade, sendo seus valores definidos de acordo com as regras do ITC: $\alpha_1 = 1, \alpha_2 = 5, \alpha_3 = 2, \alpha_4 = 1$.

3.2 Instâncias

Os métodos propostos neste trabalho serão testados utilizando 21 instâncias reais da Universidade de Udine (Itália), disponíveis no site tabu.diegm.uniud.it/ctt. Esse site fornece ferramentas interessantes para que métodos possam ser testados, tais como: validador de soluções, gerador de instâncias, software para calcular tempo máximo de processamento na máquina. Além disso, é um repositório que muitos pesquisadores têm utilizado para reportar soluções, facilitando possíveis comparações com a literatura.

Cada instância é formada por somente um arquivo. Cada arquivo contém um cabeçalho e quatro seções, cursos, salas, currículos e restrições. O cabeçalho fornece todos os valores escalares e cada seção fornece as matrizes para o problema. Um exemplo do formato é mostrado a seguir:

Name: ToyExample
Courses: 4
Rooms: 2
Days: 5
Periods-per-day: 4
Curricula: 2
Constraints: 8

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

ROOMS:
A 32
B 50

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec

UNAVAILABILITY-CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3

A seção cursos (COURSES) está disposta da seguinte maneira: nome do curso, professor, quantidade de aulas, mínimo de dias que a disciplina deve ser escalonada e número de alunos, separados por espaço. Já a seção salas (ROOMS) define o nome da sala e a capacidade em termos de assentos. A seção currículo (CURRICULA), define o nome do currículo, a quantidade de cursos pertencentes ao currículo, e o nome de cada curso pertencente ao currículo. Por fim, a seção indisponibilidades (UNAVAILABILITY-CONSTRAINTS), caracteriza uma indisponibilidade de um curso, em um período de um determinado dia.

Dias e períodos começam em 0, ou seja, segunda-feira é representado pelo número 0. Por exemplo, a restrição (UNAVAILABILITY-CONSTRAINTS) TecCos 3 2 indica que o curso TecCos não pode ser atribuído na quinta feira no terceiro período.

A Tabela 3.1 apresenta as características das instâncias utilizadas, e está disposta da seguinte maneira: nome da instância, número de cursos, número de aulas (por semana), número de períodos, número de currículos e quantidade de restrições relativas a indisponibilidade.

Todas as instâncias são reais, pertencendo à Universidade de Udine, Itália. Cada instância possui uma solução viável, ou seja, não existem restrições rígidas violadas. Além disso, 4 instâncias testes (Tabela 3.2) foram utilizadas para calibrar o método proposto.

Para que seja possível validar os resultados deve-se fornecer um arquivo de saída de tal forma que cada linha represente uma atribuição de sala, em um determinado período de um dia, para uma determinada aula. O *layout* deve seguir a seguinte definição: Curso Sala Dia Período

Um exemplo é mostrado a seguir:

TecCos B 3 0

Nesse exemplo uma aula do curso TecCos acontece na quinta-feira no primeiro período na sala B.

3.3 Regras e Objetivo

As regras estabelecidas em [26] (ITC 2007) serão obedecidas para que seja possível comparar os resultados obtidos pelos métodos implementados neste trabalho e os resultados conhecidos para um conjunto de instâncias da literatura.

Obrigatoriamente o algoritmo deve rodar em uma máquina de um único processador, e poderá ser implementada em qualquer linguagem.

O tempo total de execução do método não deve ultrapassar o tempo estipulado por uma programa fornecido em http://tabu.diegm.uniud.it/ctt/. Esse programa tenta nivelar entre todos os métodos um tempo parecido, de acordo com a capacidade (Memória, Processador etc) do computador. Ou seja, teoricamente um computador com menos capacidade de processamento terá um limite maior de tempo em relação a um computador com mais capacidade de processamento.

O algoritmo deve ser capaz de receber um arquivo de entrada como descrito nas seções anteriores e fornecer um arquivo de saída equivalente no tempo estipulado.

Tabela 3.1. Características das instâncias .

Nome	Cursos	Aulas	Salas	Períodos	Currículos	indisponibilidades
comp1	30	160	6	30	14	53
comp2	82	283	16	25	70	513
comp3	72	251	16	25	68	382
comp4	79	286	18	25	57	396
comp5	54	152	9	36	139	771
comp6	108	361	18	25	70	632
comp7	131	434	20	25	77	667
comp8	86	324	18	25	61	478
comp9	76	279	18	25	75	405
comp10	115	370	18	25	67	694
comp11	30	162	25	40	13	84
comp12	88	218	11	36	150	1368
comp13	82	308	19	25	66	468
comp14	85	275	17	25	60	486
comp15	72	251	16	25	68	382
comp16	108	366	20	25	71	518
comp17	99	339	17	25	70	548
comp18	47	138	9	36	52	594
comp19	74	277	16	25	66	475
comp20	121	390	19	25	78	691
comp21	94	327	18	25	78	463

Tabela 3.2. Instâncias-teste

Nome	Cursos	Aulas	Salas	Períodos	Currículos	indisponibilidades
test1	46	207	12	20	26	0
test2	52	223	12	20	30	0
test3	56	252	13	20	55	0
test4	55	250	10	25	55	0

Nenhuma informação adicional sobre a instância deve ser utilizada, por exemplo, resultados de execuções anteriores. A mesma versão e parâmetros fixos do algoritmo devem ser utilizados para todas as instâncias.

O algoritmo pode ser determinístico ou estocástico. No caso de um algoritmo estocástico, o resultado deve ser reproduzível.

O principal objetivo é gerar um quadro de horário viável onde seja possível satisfazer todas as restrições rígidas e o maior número de restrições flexíveis, em um intervalo de tempo predeterminado.

Capítulo 4

Heurística Proposta

Algumas características particulares dos problemas combinatórios fazem que a utilização de alguns métodos de otimização não sejam possíveis de serem empregados. Por exemplo, a utilização de algoritmos determinísticos que necessitam de cálculo de derivada voltados para otimização não linear é impraticável, pois a derivada não existe no espaço discreto onde tais problemas são definidos. Técnicas exatas garantem que a solução encontrada ao fim do processo de otimização seja ótima, porém podem se tornar inviáveis na resolução de problemas de maior porte.

Devido a essas razões, métodos heurísticos são uma forma eficiente de se resolver esses problemas, pois não dependem de premissas matemáticas fortes tais como linearidade, diferenciabilidade, unimodalidade ou convexidade. Além disso, como comentado anteriormente, são eficientes na busca de boas soluções em problemas de tal natureza.

O método escolhido neste trabalho é composto por ideias de três metaheurísticas conhecidas na literatura, *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e *Greedy Randomized Adaptive Search Procedure* (GRASP). A escolha dessa metodologia é embasada nos bons resultados já alcançados por esses métodos quando aplicados a problemas de dificuldade semelhante, como pode ser visto em vários trabalhos [96, 97, 78, 98, 99, 100, 101], entre outros. Entretanto, tal metodologia não é muito utilizada em problemas de geração de quadro de horários, o que torna interessante a investigação de seu desempenho nesta classe de problemas.

O presente capítulo apresenta a heurística proposta para resolução do problema de programação de horários de cursos baseado em currículo. O modo de representação de uma solução, uma estratégia eficiente de construção, heurísticas utilizadas e alguns movimentos são também descritos.

4.1 Representação de uma solução

Uma solução x é representada por uma matriz $n \times d$, onde n e d são, respectivamente, a quantidade de cursos e períodos do problema. A célula x_{ij} dessa matriz define a qual sala o curso i no período j foi escalonado. Um exemplo é mostrado na Figura 2.

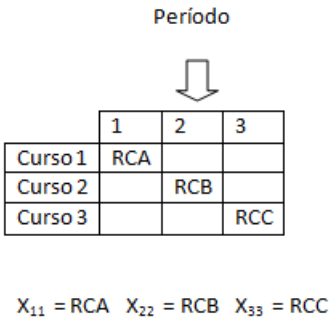

Figura 4.1. Representação computacional do problema de quadro de horário

Neste exemplo, é designada uma aula no período 1 na sala RCA para o curso 1, uma aula no período 2 na sala RCB para o curso 2 e uma aula no período 3 na sala RCC para o curso 3.

Uma das vantagens em se utilizar esse tipo de codificação é que em algumas restrições, tais como, sobreposição de aulas de um mesmo curso e indisponibilidade, podem ser facilmente contornadas, impedindo que se gere soluções inviáveis em relação a violação dessas restrições. Por exemplo, é impossível representar aulas de um mesmo curso, em um mesmo período, em salas distintas. Além disso, a indisponibilidade pode ser representada com a inserção de valores -1 no quadro de horários, indicando assim a indisponibilidade do curso. Outro fator importante é que a partir dessa codificação, o procedimento de avaliação de um quadro de horário pode ser facilmente codificada.

4.2 Movimentos

Seja x uma solução para o problema de programação de horário. É necessário definir uma estrutura de vizinhança, ou seja, uma função M, na qual podemos chegar a x' por meio de uma modificação, geralmente chamada de movimento. Neste trabalho são executados 6 tipos de movimentos, todos baseados no trabalho de Muller [92], ganhador do ITC 2007. Cada um deles é descrito a seguir:

- M_1: Mudar uma aula de período. A Figura 4.2 ilustra esse movimento. Nesta Figura, a aula do curso 1 realizada na sala RCA no período 1 passa a ser realizada no período 2.

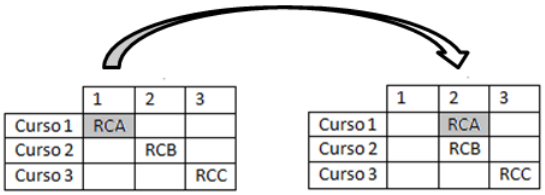

Figura 4.2. Representação do movimento M_1

- M_2: Mudar uma aula de sala. A Figura 4.3 exemplifica esse movimento. Nela, mostra-se que a aula do curso 1 realizada na sala RCA no período 1 passa, agora, a ser realizada na sala RCB.

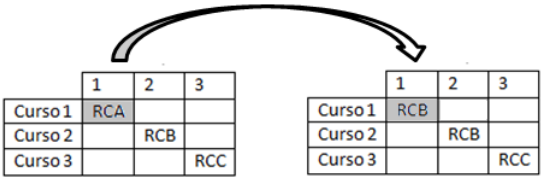

Figura 4.3. Representação do movimento M_2

- M_3: Selecionar uma aula e retirá-la do quadro de horário. Um novo período e uma sala são escolhidos a partir de uma busca local, que consiste em testar todas as salas e períodos disponíveis para a disciplina referente a essa aula. A Figura 4.4 ilustra esse movimento. Nesta Figura, a aula do curso 1 realizada no primeiro período é escolhida (parte *a* da Figura). Posteriormente, são mostradas todas as combinações possíveis de realocação dessa aula no quadro de horário (parte *b* da Figura).

- M_4: Escolher um curso em que a restrição mínimo de dias trabalhados não é atendida. Uma aula é alocada para um dia onde o curso selecionado não está presente, e uma aula é retirada onde existem duas ou mais aulas programadas em um mesmo dia. A Figura 4.5 ilustra esse movimento. É considerado que um dia possui 3 aulas, e que o curso 1 tem que ser escalonado em pelo menos 2 dias. Nesta Figura, uma aula alocada no primeiro dia (parte *a* da Figura) é movida para um dia onde não existe aula desse curso (parte *b* da Figura)

- M_5: Selecionar um currículo e uma aula isolada, caso exista. Fazer a tentativa de realocar essa aula junto a outras aulas que pertençam ao mesmo currículo. A Figura 4.6, que ilustra esse movimento, considera que os cursos 1 e 2 pertencem a um mesmo currículo,

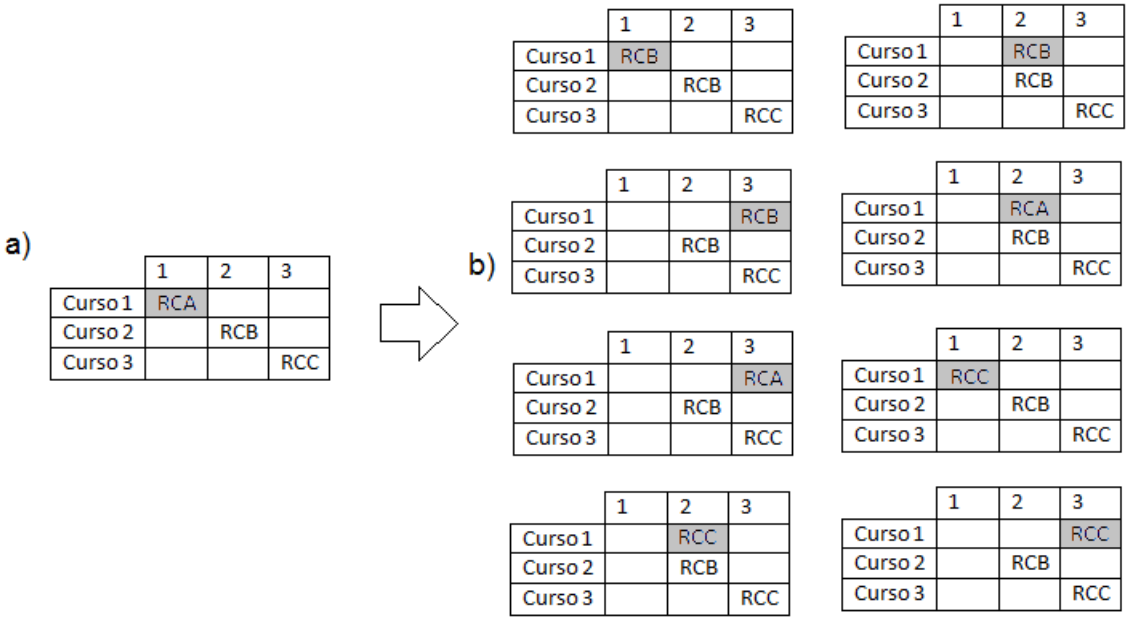

Figura 4.4. Representação do movimento M_3

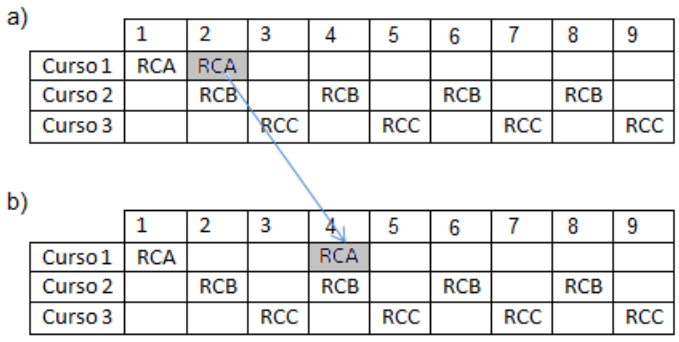

Figura 4.5. Representação do movimento M_4

e que um dia possui 3 períodos. Assim, a parte *a* da Figura mostra uma aula alocada de forma isolada no período 8 (terceiro dia), e a parte *b* mostra essa aula passando para um dia onde existe(m) aula(s) escalonada(s) para esse currículo.

- M_6: Escolher um curso e uma sala onde esse curso já foi atribuído. Fazer, a seguir, uma tentativa de atribuir todas as aulas desse curso para a sala escolhida. Se já existir uma aula para outro curso na sala escolhida, é feita a alocação da aula do curso conflitante para uma sala disponível. A Figura 4.7 exemplifica esse movimento. A parte *a* da Figura mostra a sala e o curso escolhidos. A parte *b*, mostra a atribuição de todas as aulas desse curso para a sala escolhida. Por fim, a parte *c* da Figura, mostra a aula que deve ser realocada.

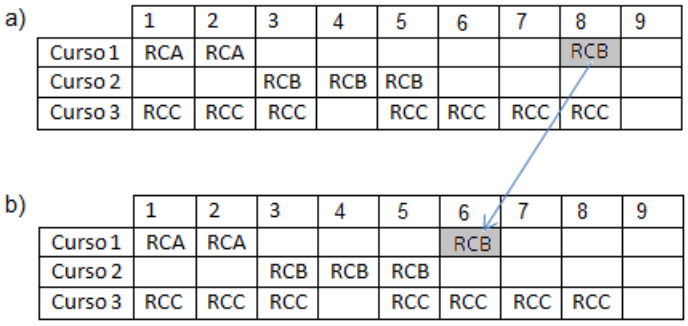

Figura 4.6. Representação do movimento M_5

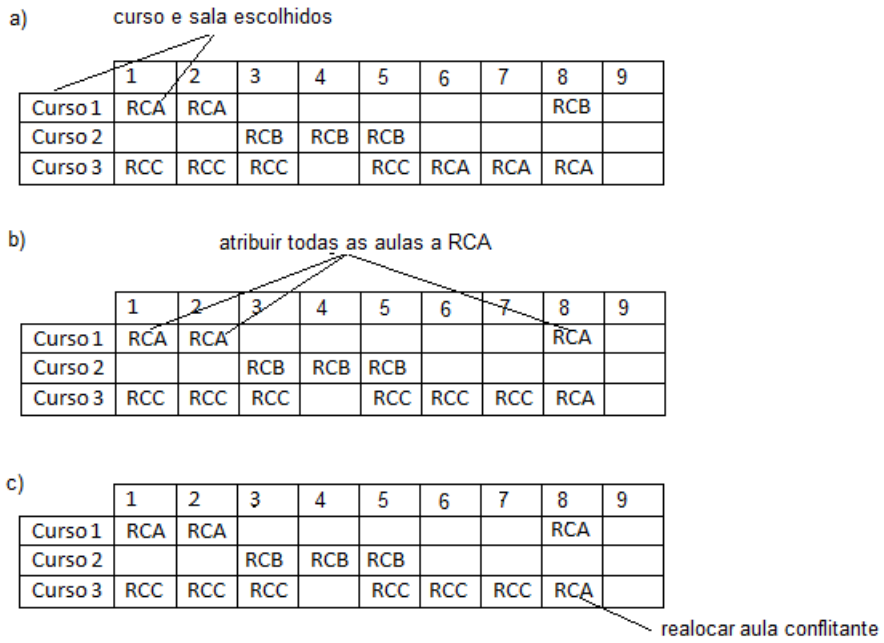

Figura 4.7. Representação do movimento M_5

4.3 Método de Construção

Foram testados dois métodos de construção, ambos baseados no método de construção GRASP. O primeiro é baseado na implementação proposta por Feo e Resende [68] e o segundo, é uma extensão do primeiro, e inclui parte das ideias do trabalho apresentado por Geiger [91].

4.3.1 Construção CT1

O primeiro passo no processo de construção de uma solução é definir uma lista de cursos C. Para cada curso $c_i \in C$ são escalonadas l aulas, uma a uma, da seguinte maneira: Para cada aula do curso c_i são avaliadas as combinações de salas (sl) e períodos disponíveis (h), criando, assim,

um conjunto denominado $Lista$ de soluções. Posteriormente é criada uma lista de soluções candidatas ($LRC \in Lista$) que atendem ao critério de seleção $min+\alpha(max-min)$ (Eq. 2.24). É escolhida uma solução parcial que pertença à lista LRC de forma aleatória para compor a solução corrente s_c. O processo continua até que todas as aulas sejam alocadas. O pseudocódigo desse método é apresentado no Algoritmo 1.

Algoritmo 1: Método de Construção CT1.

```
Entrada:
Saída: Solução (s)
início
    para i = 1 até |C| faça
        para j = 1 até l_i faça
            para h = 1 até |H| faça
                para sl = 1 até |SL| faça
                    Lista ← Insira(s_c,j,h,sl); Retire(s_c,j,h,sl);
                fim para
            fim para
            LRC ∈ Lista | ∀s ∈ LRC, f(s) < min + α(max − min)
            s_c ← s ∈ LRC
        fim para
    fim para
fim
```

4.3.2 Construção CT2

O método de construção CT2 é baseado no método de construção CT1. Porém, foram feitas modificações com o objetivo de melhorar seu desempenho, tais como mudança no critério do tamanho da lista LRC, definição de prioridade para escalonar aulas dos cursos, *backtracking*, entre outros.

Três observações foram essenciais para chegar até o método proposto: $a)$ a ordem em que as disciplinas são escalonadas influenciam diretamente na qualidade da solução inicial; $b)$ a fórmula para criar a LRC, na sua forma original, favorece que soluções infactíveis sejam geradas; $c)$ Os pesos atribuídos à equação (4.1) influenciam diretamente no tempo de construção de uma solução. Para cada observação foi criada uma estratégia para que boas soluções fossem criadas em um tempo razoável.

4.3.3 Prioridade

Para a observação a foram criadas duas estratégias, uma para definir a prioridade dos cursos para serem escalonados e outra para redefinir essa prioridade. A primeira, chamada de "defina

prioridade", consiste em calcular a prioridade de cada curso $c_i \in C$ baseada em informações do problema e fatores de ponderamento (w). Quanto maior o valor obtido a partir da equação (4.1), maior a prioridade de escalonamento para o curso c_i.

$$prioridade_i = cn_i w_1 + id_i w_2 + dem_i w_3, \ \ \forall i \in C \tag{4.1}$$

Nesta equação, (cn_i) é o número de cursos conflitantes com o curso i, (id_i) é o número de indisponibilidades de horários e (dem_i) indica a quantidade de estudantes para o curso i.

A segunda estratégia, nomeada “redefina prioridade”, consiste em realocar c_i na lista de prioridade. Deste modo, caso um curso i durante a fase de construção viole alguma das restrições rígidas, sua prioridade é aumentada. As equações (4.2) e (4.3) mostram como esse processo é feito.

$$prob_i += \begin{cases} 5, & \text{se } prob_i = 0; \\ prob_i * 3, & \text{caso contrario.} \end{cases} \tag{4.2}$$

$$prioridade_i = prioridade_i + prob_i; \tag{4.3}$$

Na eq. (4.2), $prob_i > 0$ indica que o curso i já foi realocado e $prob_i = 0$ indica que não. Assim quanto maior o número de vezes que a disciplina i for realocada, maior é o aumento de prioridade dado a ela. Além disso, após o procedimento de realocação, todos os cursos de menor prioridade em relação ao curso i que compõem a solução parcial são desalocados para posteriormente serem alocados novamente. Um exemplo é mostrado a seguir:

Seja uma lista de prioridade obtida de acordo com a estratégia “defina prioridade”:

Curso	Prioridade
c_1	20
c_2	17
c_3	15
c_4	13

Imagine que ao escalonar aulas para os cursos da lista anterior, ao chegar no curso c_4 é criada uma violação de uma restrição rígida. O primeiro passo é calcular sua nova prioridade. Considerando que $prob_4 = 0$, a nova prioridade pode ser calculada por 13 + 5 = 18. Desse modo, temos uma nova lista de prioridade:

Curso	Prioridade
c_1	20
c_4	18
c_2	17
c_3	15

Todas as aulas dos cursos já escalonadas entre a prioridade antiga e a nova são desalocadas, no caso, as aulas dos cursos c_2, c_3 e c_4. Dados tais procedimentos, o método de construção descrito detalhadamente na seção 4.3.6 reinicia novamente a partir da nova posição do curso c_4.

4.3.4 LRC

Para a observação b foi proposta uma equação para definir os elementos integrantes da LRC. Não foi utilizada a equação proposta por Feo e Resende [68] pois ela favorecia a criação de soluções não factíveis. Este fato pode ser observado tendo em vista que o *gap* entre os valores $\min$ e $\max$ pode ser muito grande devido à modelagem (formulação) considerada no método heurístico, que permite a criação de soluções não factíveis com uma penalização alta. Um exemplo é mostrado a seguir. Aplicando a equação 2.24 proposta em [68], seja α = 0,1, $\max$ = 10000 e $\min$ = 101. Temos, então: 10 + 0,1(10000-100) = 1091. Assim, a lista seria composta por soluções com valor até 1091, e provavelmente, ela conteria soluções não factíveis.

Em vista deste inconveniente, foi criada a LRC composta por soluções que atendem a regra definida pela equação 4.4. Deste modo, o número de possíveis soluções infactíveis fazendo parte da LCR é consideravelmente menor. Considerando os valores dados no exemplo anterior, a lista seria composta apenas por soluções com valor até 111.

$$LRC \leq min + (\alpha \times min); \tag{4.4}$$

4.3.5 Calibragem de pesos

Já para atender a observação c, e calibrar os pesos w_1, w_2 e w_3 que definem a prioridade, foi utilizado o método de otimização Evolução Diferencial (DE)[102]. Ele consiste em um método de otimização simples e eficiente que vem sendo muito utilizado na otimização não-linear com variáveis continuas.

Por seguir uma linha de evolução de soluções ao longo do tempo e ser um algoritmo populacional, esse método é classificado como um algoritmo evolutivo, embora não tenha qualquer inspiração no processo evolutivo natural. Seus operadores utilizam conceitos matemáticos e estratégias heurísticas.

O DE básico é composto por dois operadores, Mutação Diferencial e *crossover*. A Mutação Diferencial consiste em escolher três indivíduos (vetores) da população e criar um novo a partir da adição ponderada da diferença entre dois desses indivíduos, assim como mostrado na equação 4.5.

$$u_r = u_a + \mu(u_c - u_b) \tag{4.5}$$

Nesta equação, u_r é o novo individuo gerado, e u_a, u_b e u_c são três indivíduos (vetores) da população.

O *crossover* é composto por um indivíduo denominado alvo (u_j) e o novo indivíduo criado na mutação diferencial (u_r). Considerando u_c a probabilidade de cruzamento, z um vetor de números aleatórios variando entre 0 e 1 e l referenciando uma variável pertencente ao individuo u_j, esse operador consiste em criar um novo indivíduo u'_j de acordo com a regra descrita na equação 4.6.

$$u'_{jk} = \begin{cases} u_{rk}, & \text{se } rand_z < a_{prob} \text{ ou } k = l; \\ u_{jk}, & \text{caso contrário} \end{cases} \quad k = 1, ..., |p_j| \tag{4.6}$$

Cada indivíduo da população é usado uma vez como indivíduo alvo. O pseudocódigo desse método é representado pelo Algoritmo 2.

A função $f(x)$ utilizada no método DE consiste no valor médio retornado após 5 execuções do método de construção, onde x é um vetor composto por w_1, w_2, w_3 e α. São consideradas apenas soluções criadas com a média de tempo menor que 30 segundos. Caso o tempo seja maior, a solução é penalizada em 10 vezes, ou seja, $10f(x)$. Esse procedimento foi feito com a expectativa de se gerar boas soluções de forma rápida.

O método de calibração foi utilizado para 4 instâncias-teste (test1,test2,test3 e test4), sendo o valor retornado para o conjunto de pesos w e α utilizados nas demais instâncias.

4.3.6 Construção

O primeiro passo no processo de construção de uma solução é definir uma lista de prioridade LPC de cursos $c_i \in C$, como mostrado na seção 4.3.3. Para cada curso c_i pertencente à LPC, seguindo a ordem de prioridade, são escalonadas l aulas, uma a uma, da seguinte maneira: Para cada aula do curso c_i são avaliadas as combinações de salas e períodos disponíveis criando, assim, um conjunto S de soluções. Posteriormente é criada uma lista de soluções candidatas ($LRC \in S$) que atende ao critério definido na seção 4.3.4. É escolhida uma solução parcial que pertença à lista LRC de forma aleatória. A cada escolha é testado se a solução parcial é factível; caso não seja, o método "redefine prioridade", descrito na seção 4.3.3, é utilizado. Para evitar um laço infinito, a redefinição de prioridade só acontece no primeiro minuto. Caso uma

Algoritmo 2: Evolução Diferencial

Entrada: Parâmetro de ponderação (μ), Total de iterações (n), Probabilidade de Cruzamento (a_{prob})
Saída: Solução ($u*$)
, sendo U a população inicial, u_{jk} um indivíduo dessa população e k a k-ésima variável do indivíduo j.

início
  para $i = 1$ *até* n **faça**
    para $j = 1$ *até* $|U|$ **faça**
      [u_a, u_b, u_c] = Escolha 3 indivíduos $\in U$ aleatoriamente;
      u_r = $u_a + \mu(u_c - u_b)$;
      $a_{target} \in u_j$
      $rand_k \in [0, 1], k = 1, ..., |u_a|$
      u'_{jk} = $\begin{cases} u_{rk}, & \text{se } rand_k < a_{prob} \text{ ou } k = a_{target}; \\ u_{jk}, & \text{caso contrário} \end{cases}$ $k = 1, ..., |u_j|$
      se $f(u_j) > f(u'_j)$ **então**
        $u_j = u'_j$
      fim se
    fim para
  fim para
  retorne $u*$
fim

solução factível não seja encontrada nesse tempo, o método de redefinir prioridade não é mais utilizado. O processo continua até que todas as aulas sejam alocadas. Chamaremos tal processo de alocação de cada aula de "Alocação". O Algoritmo 2 mostra seu pseudocódigo.

No Algoritmo **??**, o método "define prioridade" recebe a lista de cursos C, e a ordena de acordo com o critério estabelecido anteriormente. Assim, cada curso $i \in C$ é visitado na ordem estabelecida. O método "redefina prioridade" redefine a prioridade do curso i na lista C.

4.4 Heurísticas

4.4.1 ILS

Esse método foi baseado na heurística ILS (*Iterated Local Search*) [19], que explora o espaço de soluções por meio de uma sequência de perturbações a uma solução, seguidas de buscas locais.

Um método ILS é composto por um gerador de solução inicial, um procedimento de busca local, um procedimento de perturbação e um critério de aceitação. O gerador de solução inicial desenvolvido é descrito na seção 4.3. Já o procedimento de perturbação consiste em retirar ν aulas do quadro de horários. O nível de perturbação é aumentado à medida que as soluções não

Algoritmo 3: Método de Construção CT2.

Entrada:
Saída: Solução (s)
início
  Defina prioridade(C);
  para $i = 1$ *até* $|C|$ **faça**
    para $j = 1$ *até* l_i **faça**
      para $h = 1$ *até* $|H|$ **faça**
        para $sl = 1$ *até* $|SL|$ **faça**
          Lista $\leftarrow$ Insira(s_c,j,h,sl);
          Retire(s_c,j,h,sl);
        fim para
      fim para
      $LRC \in Lista \mid \forall s \in LRC,\ f(s) < min + (\alpha \times min)$
      $s_c \leftarrow s \in LRC$
      se s_c *for inviável e tempo* < 60 **então**
        Redefina prioridade(i);
      fim se
    fim para
  fim para
fim

melhoram durante um certo número de iterações; nesse caso, ν é incrementado até um valor máximo max. O método de busca local consiste em alocar as ν aulas, uma a uma, retiradas do quadro de horário, testando todas as possibilidades em relação a salas e horários disponíveis para cada curso. A ordem em que essas aulas são novamente atribuídas ao quadro de horários respeita a LPC definida no método de construção. Desta forma, aulas de cursos com maior prioridade são escalonadas primeiro. A solução gerada é aceita caso seja melhor que a melhor solução encontrada até o momento. O método é executado até que o tempo limite se esgote. Seu pseudocódigo é mostrado no Algoritmo 4.

4.4.2 VND

VND é um método de refinamento que consiste em explorar o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança, aceitando somente soluções de melhora da solução corrente e retornando à primeira estrutura quando uma solução melhor é encontrada. Dependendo do problema abordado, a busca pelo melhor vizinho pode ser custosa computacionalmente; deste modo, é comum fazer a busca pela primeira solução de melhora. Outra alternativa é considerar a exploração apenas em um certo percentual da vizinhança.

Primeiramente utilizamosPrimeiramente utilizamos o método VND com o objetivo de

Algoritmo 4: ILS

Entrada: Perturbação máxima max, Pertubação inicial ν_0
Saída: Solução (x)
início
  $x \leftarrow gerasolucao()$
  enquanto *critério de parada não satisfeito* **faça**
    $x' \leftarrow$ Perturbação (ν, x)
    $x' \leftarrow$ Busca local (x)
    $x \leftarrow$ Critério Aceitação (x, x')
    $n = \begin{cases} n+1, & \text{se } \nu < max \text{ e } f(x') > f(x); \\ n_0, & f(x') < f(x) \text{ ou } \nu = max \end{cases}$
  fim enquanto
fim

verificar o seu desempenho isoladamente. Posteriormente, ele foi incorporado ao método ILS com o objetivo de fazer o refinamento das soluções. Em ambos os casos apenas uma parte da vizinhança foi explorada, de forma aleatória. Além disso, o critério de aceitação foi relaxado, ou seja, soluções com o mesmo valor de função objetivo foram aceitas. O critério de aceitar soluções com o mesmo valor de função objetivo permite que o método caminhe por movimentos "movimentos laterais", explorando, assim, uma fatia maior do espaço de busca. Os pseudocódigos dos métodos VND, e ILS+VND são apresentados nos algoritmos 5 e 6 respectivamente.

Algoritmo 5: VND

Entrada: Solução parcial (x), M (Conjunto de Movimentos)
Saída: Solução (x)
início
  enquanto *critério de parada não satisfeito* **faça**
    para i*=1 até* $|M|$ **faça**
      $x' =$Explorar Vizinhança $M_i(x)$
      se $f(x) \geq f(x')$ **então**
        $x = x'$
        $i = 1$
      fim se
    fim para
  fim enquanto
fim

Algoritmo 6: ILS + VND

Entrada: Solução parcial (x),M (Conjunto de Movimentos),n_0 número de perturbações inicial, max número máximo de perturbações
Saída: Solução (x)
início
  enquanto *critério de parada não satisfeito* **faça**
    $x' \leftarrow$ Perturbação (n, x)
    $x' \leftarrow$ Busca local (x)
    $x \leftarrow$ Critério de Aceitação (x, x')
    $n =$ $\begin{cases} n+1, & \text{se } n < max \text{ e } f(x') > f(x); \\ n_0, & f(x') < f(x) \end{cases}$
    para i=*1 ate* $|M|$ **faça**
      $x' =$Explorar Vizinhança $M_i(x)$
      se $f(x) \geq f(x')$ **então**
        $x = x'$
        $i = 1$
      fim se
    fim para
  fim enquanto
fim

Capítulo 5

Resultados

Este capítulo apresenta um conjunto de experimentos que foram realizados com o objetivo de verificar o desempenho dos algoritmos implementados. Além disso, é mostrado o resultado obtido pelo procedimento de calibração.

Com o objetivo de escolher as melhores combinações dentre os procedimentos desenvolvidos (CT1, CT2, ILS, VND), foram testados nas instâncias-teste (Tabela 3.2) as seguintes combinações: CT1 + VND, CT2 + VND, CT1 + ILS, CT2 + ILS, CT1 + ILS+VND e CT2 + ILS+VND. Destas, foram apresentados apenas os resultados das duas melhores combinações, a saber, CT2 + ILS e CT2 + ILS+VND.

Deste modo, inicialmente são mostrados os resultados obtidos pelos algoritmos ILS e ILS+VND tendo o CT2 como procedimento de construção. Posteriormente é feita uma comparação do algoritmo que obteve melhor desempenho neste trabalho com os dos finalistas do ITC 2007.

Os algoritmos foram desenvolvidos na linguagem Java e testados em um computador Pentium 2.2 GHz com 2 GB de RAM no sistema operacional Windows 7. Para cada instância testada, foi dado um tempo de 500 segundos na busca de soluções. Esse tempo foi definido utilizando um algoritmo fornecido pelos idealizadores do ITC, que utiliza informações da máquina de teste para equiparar todos os algoritmos da literatura em relação ao processamento. Todos os métodos foram executados 20 vezes para cada instância.

As tabelas de resultados para os algoritmos ILS e ILS + VND são apresentadas com os seguintes dados: quantidade de violações das restrições flexíveis S_1, S_2, S_3 e S_4 para a melhor solução encontrada, o melhor valor de função objetivo (FO) obtido pelo método em questão, média da função objetivo (FOM) após 20 execuções do método, desvio padrão, o melhor valor de função objetivo alcançado entre os participantes do ITC 2007 ($BEST$). À Tabela 5.1 também é adicionada uma coluna (Média tempo) contendo o tempo médio em segundos para se obter uma solução. Ainda, as tabelas 5.1 e 5.2 apresentam o desvio médio ($DMED$) dos

resultados obtidos pelos algoritmos implementados nesse trabalho em relação ao melhor valor de função objetivo alcançado no ITC 2007. A tabela que compara os melhores algoritmos do ITC 2007, o método proposto por Barbosa *et al.* [78] e o melhor algoritmo implementado neste trabalho, contém os valores médios de função objetivo. A comparação com o método de Barbosa *et al.* é feita devido ao fato de a proposta desses autores ser parecida com a abordada no presente trabalho e obedecer as regras do ITC.

O desvio médio é calculado a partir da equação (5.1), na qual $best$ é a melhor média de função objetivo obtida na competição ITC 2007 para uma instância alvo, $replicacao_i$ o valor da função objetivo em uma replicação i para a instância-alvo e ntr o número total de replicações.

$$\frac{\sum_i^{ntr} |replicacao_i - best|}{ntr} \tag{5.1}$$

Os testes foram executados com o parâmetro ν do algoritmo ILS variando entre 5 e 20 para todas as instâncias, tal como especificado no algoritmo 4. Tais valores foram obtidos a partir de testes empíricos.

5.1 Calibração de parâmetros

Os resultados obtidos pelo método de calibração para os parâmetros w_1, w_2, w_3 e α foram respectivamente 5, 1, 3 e 0,01.

Os parâmetros μ e a_{prob} do método *Differential Evolution* (DE) foram fixados em 0,5 e 0,9, respectivamente. Tais valores foram utilizados em vista do emprego bem sucedido deste método em funções *benchmark* encontradas na literatura, tais como Função de Bohachevsky, Função Foxholes, Função de Rastrigin [103].

5.2 Método de Construção CT2

Todos as execuções obtiveram soluções viáveis, ou seja, nenhuma solução do tipo rígida foi violada. O método de construção gastou, no pior caso, cerca de 4% do tempo total de processamento. A Tabela 5.1 apresenta os resultados obtidos.

Apesar de a solução inicial não possuir uma boa qualidade em relação ao número de restrições flexíveis violadas, de certa forma o método se mostrou adequado, uma vez que a obtenção de soluções factiveis para tal problema pode ser em muitos casos uma tarefa complicada.

Tabela 5.1. Resultados obtidos pelo método de construção

Nome	S1	S2	S3	S4	Melhor FO	FOM	DP FO	Média tempo	BEST
comp1	7	5	5	18	60	93,1	22,5	0,6	5
comp2	0	42	87	142	527	568,9	25	10,9	60,5
comp3	2	35	66	125	434	487	30,3	7,8	84,5
comp4	1	28	50	145	386	412,3	14,8	7,1	39,5
comp5	0	43	245	39	744	823,6	78,2	3,3	325,2
comp6	4	34	91	175	531	596,6	23,3	11,4	56,8
comp7	28	36	98	226	632	689,8	42,1	20,6	33,9
comp8	12	34	41	175	439	467,6	23,5	9,1	46
comp9	0	46	54	158	496	530,7	21,6	8,5	113,1
comp10	3	32	78	200	519	550,7	29,5	12,3	21,3
comp11	10	9	13	8	89	114,3	17,6	0,9	0
comp12	17	61	176	93	767	868,6	78,8	8,3	351,6
comp13	6	30	61	157	435	462,6	15,6	9,4	73,9
comp14	0	36	68	143	459	502,5	21,9	6,6	62,3
comp15	0	30	72	130	424	472	26,1	7,7	84,5
comp16	13	35	69	203	529	570,8	27,2	14,8	41,2
comp17	0	46	69	173	541	587,3	18,9	12,1	86,6
comp18	0	34	41	57	309	351,5	17,8	1,5	73,3
comp19	6	46	50	145	481	522,3	26,2	19,8	68,8
comp20	36	40	77	180	570	669	32,8	14,5	31,3
comp21	0	37	109	178	581	606,1	18	17,4	108

5.3 CT2+ILS

Embora o método ILS tenha melhorado efetivamente a solução inicial, seus resultados foram bem piores em relação ao melhor resultado do ITC 2007, como pode ser visto na Tabela 5.2.

A Figura 5.2 mostra graficamente a convergência do algoritmo, apresentando o valor da função objetivo em relação ao tempo de processamento para duas instâncias (comp2 e comp5).

Durante o processo de otimização é possível perceber que nos primeiros 200 segundos o método converge de forma eficiente, porém posteriormente sua velocidade de convergência diminui consideravelmente, e, em alguns casos, o método fica estagnado em algumas soluções.

Um fator que provavelmente contribuiu para que esse fato acontecesse foi o tipo de movimento adotado. Durante os testes, percebemos que após um tempo de convergência o algoritmo encontra uma grande dificuldade em melhorar a solução corrente, sendo possível observar esse comportamento na Figura 5.2.

Em decorrência dessa dificuldade, tivemos a ideia de inserir o método VND a essa metodologia. Entre tantas outras, essa foi escolhida devido ao sucesso quando empregado a problemas de dificuldades semelhantes, como citado anteriormente.

Tabela 5.2. Resultado obtido pelo método ILS

Nome	S1	S2	S3	S4	Melhor FO	FOM	DP FO	BEST	DMED
comp1	4	0	0	2	6	7,6	1,1	5	1,14
comp2	0	70	14	27	237	257,6	12,7	60,5	31,1
comp3	0	15	48	27	198	219	15,5	84,5	21,9
comp4	0	3	31	29	106	125,8	12,3	39,2	17,4
comp5	5	44	146	10	527	595	37,6	325,2	55,3
comp6	0	12	41	66	208	265,1	27,6	56,8	36,3
comp7	9	17	47	89	277	297	12,8	33,9	42,8
comp8	0	2	43	41	137	158,5	13,1	46	16,4
comp9	0	19	38	33	204	225,1	16,1	113,1	24,1
comp10	0	16	32	64	208	231,8	16,6	21,3	36,2
comp11	0	0	0	0	0	2,25	2,3	0	0,96
comp12	0	59	88	16	487	574	42,6	351,6	36,21
comp13	0	8	44	51	179	192	8,7	73,9	19,7
comp14	0	11	36	37	164	179,4	14,9	62,3	19,2
comp15	0	14	43	30	186	204,6	13,9	84,5	22
comp16	0	18	44	53	200	226,1	18,6	41,2	34,6
comp17	0	18	54	56	254	275	13,9	86,6	33,4
comp18	0	13	26	4	121	135,5	6,1	73,3	11,1
comp19	0	17	44	32	205	221,1	9,2	68,8	27,1
comp20	6	19	53	79	286	297,3	5,2	34,3	44,6
comp21	0	18	66	44	266	290,1	13,7	108	32,7

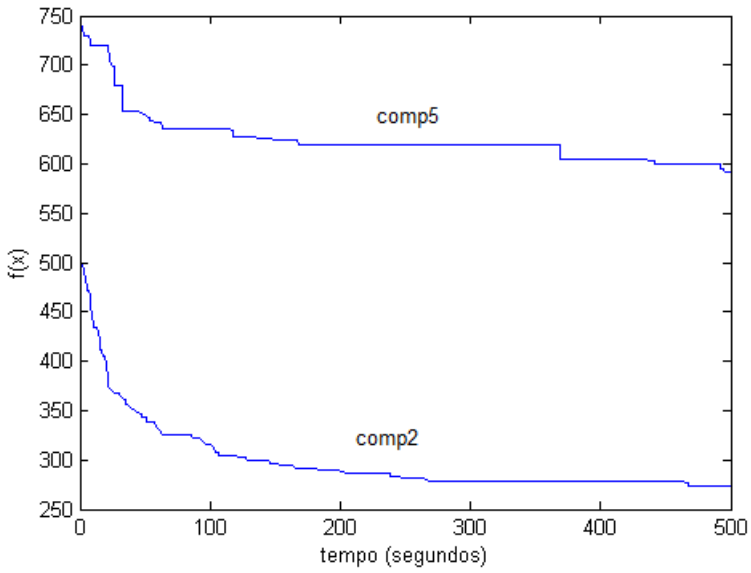

Figura 5.1. Evolução tempo x função objetivo para o método de ILS para as instâncias comp2 e comp5

5.4 CT2 + ILS + VND

Os resultados após a inserção do método VND foram em média 28% melhores, como pode ser visto na Tabela 5.3. A união desses métodos permitiu que o algoritmo proposto convergisse mais efetivamente, e criou mecanismos para explorar mais efetivamente o espaço de soluções. A Figura 5.2 mostra o desvio médio dos resultados obtidos pelos algoritmos propostos em relação ao melhor valor de função objetivo obtido o ITC 2007 para cada instância.

Tabela 5.3. Resultados obtidos pelo método ILS + VND

Nome	S1	S2	S3	S4	Melhor FO	FOM	DP FO	BEST	DMED
comp1	4	0	0	1	5	5.8	1,4	5	1,1
comp2	0	8	54	27	175	192,2	10,4	60,5	22,8
comp3	0	7	34	28	131	149,1	12,4	84,5	15,1
comp4	0	3	26	29	96	113,1	12,6	39,2	17,9
comp5	0	36	110	15	415	432,6	12,8	325,2	19,9
comp6	0	10	29	66	174	199,6	15,5	56,8	26,8
comp7	0	6	28	24	110	132,2	14,1	33,9	21,5
comp8	0	1	27	38	97	122,6	13,9	46	15,5
comp9	0	12	23	28	134	150,1	10,2	113,1	9,7
comp10	0	1	28	34	95	120,1	17,2	21,3	21,4
comp11	0	0	0	0	0	0	0	0	0
comp12	0	41	91	27	416	447,8	26,9	351,6	19,9
comp13	0	5	39	18	121	143,2	14,1	73,9	14,2
comp14	0	8	30	13	113	139,4	13,8	62,3	16,6
comp15	0	10	32	20	134	151,1	13,9	84,5	14,4
comp16	0	14	30	41	171	199,4	13,3	41,2	29,5
comp17	0	12	44	1	149	175,5	22,9	86,6	22,4
comp18	0	10	24	0	98	130,4	17,8	73,3	19,0
comp19	0	7	36	32	139	157,4	9,1	68,8	15,8
comp20	6	12	44	57	211	232	13,6	34,3	34,2
comp21	0	11	39	38	171	200,2	16,7	108	21,2

A Figura 5.3 mostra graficamente a convergência do algoritmo CT2 + ILS e CT2 + ILS + VND para as duas instâncias escolhidas na seção 5.3.

Nota-se que os métodos tiveram dificuldade em um mesmo momento, provavelmente pelo motivo de terem encontrado um ótimo local; porém, o uso do método VND forneceu recurso necessário para que dentro de um intervalo de tempo t essa estagnação fosse vencida. Isso pode ser explicado devido à exploração de vizinhanças diferentes, onde um ótimo local para uma vizinhança M não implica que também é um ótimo local para outra vizinhança M'.

Para comprovar a melhor eficiência do algoritmo CT2 + ILS + VND foi aplicada uma técnica para verificar o desempenho em relação a tempo de processamento e convergência. Aqui o objetivo é medir a distribuição de probabilidade acumulada de um algoritmo para alcançar

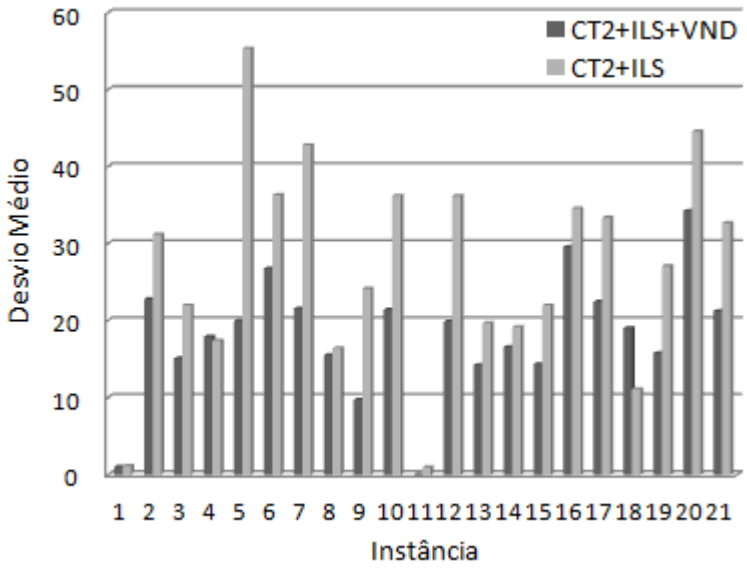

Figura 5.2. Desvio médio dos algoritmos propostos em relação ao melhor resultado obtido no ITC 2007

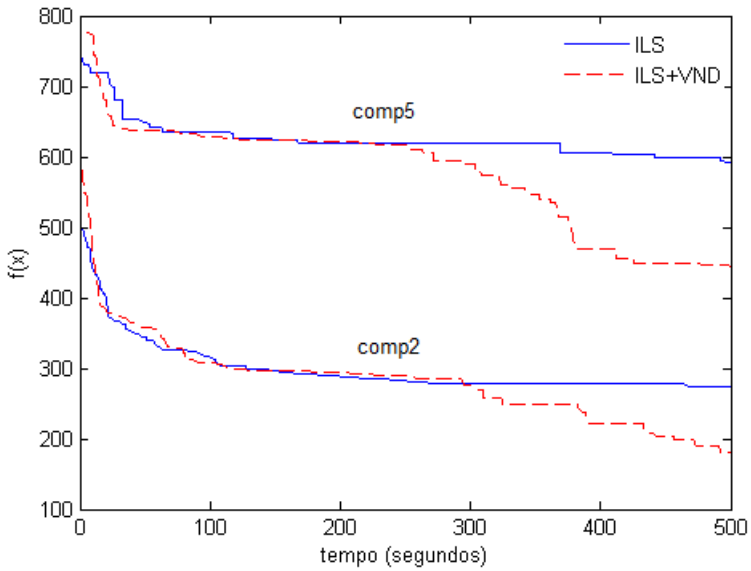

Figura 5.3. Evolução tempo x função objetivo para os métodos de CT2 + ILS e CT2 + ILS + VND para as instâncias comp2 e comp5

uma determinada solução alvo. Para isso, uma instância é escolhida e uma solução alvo é determinada. Após 200 execuções do método, obtém-se a probabilidade acumulada de o mesmo alcançar o alvo dado um intervalo de tempo. Escolhemos a instância comp11, sendo o alvo o seu valor ótimo. Essa instância foi escolhida uma vez que ambos os métodos conseguem chegar ao valor ótimo. A Figura 5.4 mostra a evolução tempo x função objetivo para essa instância,

retirada aleatoriamente de uma das execuções (linha contínua representa o Método CT2 + ILS, e a pontilhada CT2 + ILS + VND), e a Figura 5.5 a probabilidade acumulada.

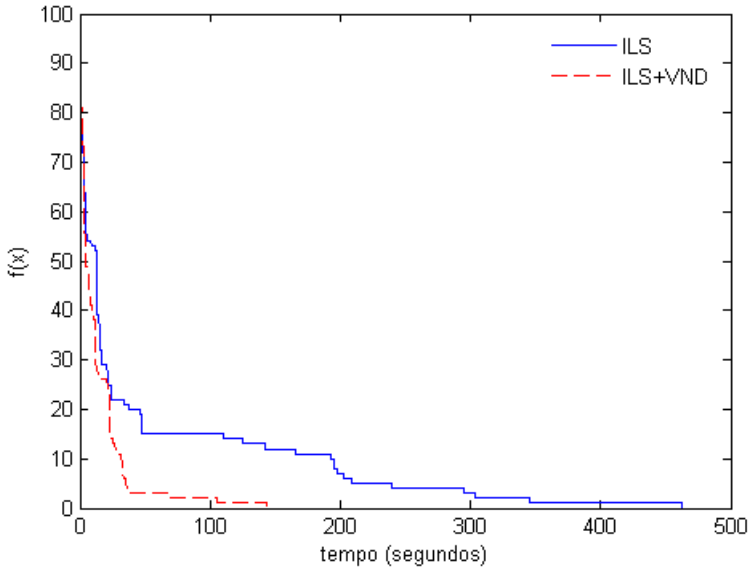

Figura 5.4. Evolução tempo x função objetivo para os algoritmos CT2 + ILS e CT2+ILS+VND para as instância comp11

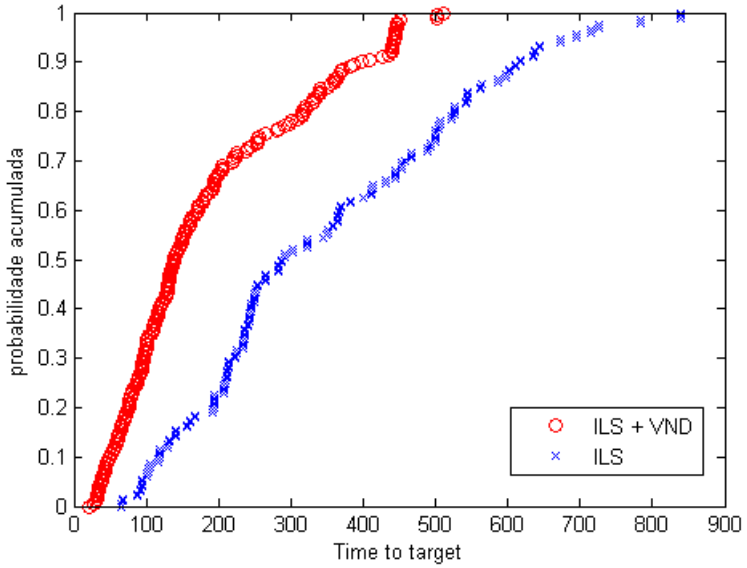

Figura 5.5. Gráfico de probabilidade acumulada dos algoritmos CT2+ILS e CT2+ILS+VND para a instância comp11

Como pode ser visto na Figura 5.5, o algoritmo CT2+ILS+VND é capaz de alcançar o alvo em um menor tempo de processamento. Em média, esse algoritmo apresenta 30 pontos

percentuais de chance a mais de encontrar o alvo em um tempo t.

Uma análise para verificar os movimentos mais efetivos nesse algoritmo foi feita. Essa análise consistiu em contabilizar quantas vezes um movimento melhorou a função objetivo em uma execução para cada instância. Assim obtivemos a seguinte relação: ILS, M_4, M_5, M_6, M_3, M_1 e M_2, com o primeiro tendo a maior importância e o último, a menor. Ao analisar o impacto da retirada de alguns movimentos no algoritmo proposto, foi percebido que os movimentos M_1 e M_2 não influenciam o resultado final.

5.5 Testes adicionais

O algoritmo CT2+ILS+VND foi executado com o tempo limite de duas horas, cinco vezes para cada instância. Os resultados obtidos foram em média 5% melhores em relação ao mesmo algoritmo sendo executado com o tempo limite de 500 segundos.

Embora não apresentado aqui, outros testes foram executados com a inserção de diferentes métodos ao método CT2+ILS+VND, tais como o critério de Metrópolis utilizado no *Simulated Annealing* [74], o operador *KempSwap* apresentado por Lu e Hao [71] e a técnica de reconexão por caminhos [104]. A utilização desses métodos não contribuiu para nenhuma melhora no algoritmo, tornando-as sem utilidade.

5.6 ITC 2007

A Tabela 5.4 mostra os melhores resultados alcançados por cada um dos algoritmos finalistas do ITC 2007, assim como aqueles obtidos pelo algoritmo de Barbosa *et al.* [78] e o melhor algoritmo desenvolvido neste trabalho. A seguir são mostradas as relações de autores e métodos base utilizados nessa competição.

- Atsuta *et al.* [105]: Metodologia baseada em um problema de satisfação de restrição (CSP) e heurísticas, tais como Busca Tabu e ILS.
- Clark *et al.*[106]: Metodologia baseada em um resolvedor de problemas de satisfabilidade (SAT) e abordagens heurísticas, incluindo Busca Tabu.
- Geiger [91]: Variante deterministica do método SA denominada “Threshold Accepting”.
- Lu e Hao [71]: Busca Tabu adaptativa.
- Muller [92]: Metodologia baseada em CSP, busca local, *Great Deluge* e *Simulated Annealing*.

- Barbosa *et al.*[78]: Método baseado no GRASP e ILS.

Tabela 5.4. Comparação entre diferentes métodos da literatura

Nome	Atsuta [105]	Clark [106]	Geiger [91]	Lu e Hao [71]	Muller [92]	Barbosa [78]	VND+ILS
comp1	5,1	27	5	6,7	5	5	5,8
comp2	65,6	131,1	142,7	61,2	60,5	124,4	192,2
comp3	89,1	138,4	160,3	84,5	94,8	127,2	149,1
comp4	39,2	90,2	82	46,9	42,2	53,4	113,1
comp5	334,5	843	538,2	325,2	343,5	978,5	432,6
comp6	74,1	149,3	110,8	69,4	56,8	114,7	199,6
comp7	49,8	153,5	76,6	41,5	33,9	95,5	132,2
comp8	46	96,5	81,7	53,6	46,5	59,5	122,6
comp9	113,3	148,9	162,5	116,5	113,1	124,6	150,1
comp10	38	101,3	81,3	34,8	21,3	77,67	120,1
comp11	0	5,7	0,3	0	0	0	0
comp12	361,6	445,3	485,1	360,1	351,6	434,6	447,8
comp13	76,1	122,9	110,4	79,2	73,9	93,8	143,2
comp14	62,3	105,9	99	65,9	61,8	82,9	139,4
comp15	89,1	138	160,3	84,5	94,8	-	151,1
comp16	50,2	107,3	82,6	49,1	41,2	-	199,4
comp17	107,3	166,6	143,4	100,7	86,6	-	175,5
comp18	73,3	126,8	129,4	80,7	91,7	-	130,4
comp19	79,6	125,4	132,8	69,5	68,8	-	157,4
comp20	65	179,3	97,5	60,9	34,3	-	232
comp21	138,1	185,8	185,3	124,7	108	-	200,2

Em relação aos resultados obtidos por Barbosa *et al.* [78], não foi possível fazer uma comparação em relação a todas as instâncias porque alguns resultados obtidos por esse autor não foram disponibilizados. Foi feita a implementação do método desses autor, porém os resultados obtidos não coincidiram com aqueles divulgados, inviabilizando assim a tentativa de mensurar a qualidade das soluções das instâncias comp15 a comp21.

Apesar de as metodologias serem parecidas, os resultados alcançados por Barbosa *et al.* [78] foram melhores em praticamente todas as instâncias (comp1 a comp14), sendo inferior apenas na instância comp5. Essa instância apresenta maior proporção de conflitos entre os cursos em comparação com as demais e nela o algoritmo proposto neste trabalho teve desempenho muito melhor.

As médias dos valores de função objetivo obtidos por todos os métodos são mostradas em duas tabelas. A tabela 5.5 apresenta o resultado para as 14 primeiras instâncias, onde é considerado o trabalho apresentado por Barbosa *et al.* [78]. Posteriormente, na Tabela 5.6, para

21 instâncias, onde esse trabalho não será considerado, pois não foram publicados os resultados de todas as instâncias.

Observa-se que os resultados obtidos pelo método CT2+ILS+VND são satisfatórios. Como pode ser visto nas Tabelas 5.4 e 5.6, o algoritmo alcança resultados próximos àqueles obtidos pelo quinto colocado do ITC 2007 (Clark), sendo que em algumas instâncias ele consegue ser superior em relação à qualidade da solução. Ainda, na Tabela 5.5, o algoritmo obteve a quarta melhor média em relação aos seis algoritmos comparados.

Tabela 5.5. Resultado dos valores médios da função objetivo para 14 instâncias

Muller [92]	93,2
Lu e Hao [71]	96,1
Atzuna [105]	96,7
Geiger [91]	152,5
VND+ILS	167,7
Barbosa [78]	169,4
Clark [106]	182,7

Tabela 5.6. Resultado dos valores médios da função objetivo para 21 instâncias

Muller [92]	87,1
LuHao [71]	91,2
Atzuna [105]	93,2
Geiger [91]	146
Clark [106]	170,8
VND+ILS	171,1

Analisando os resultados obtidos, pode-se observar que o algoritmo CT2+ILS+VND produz resultados satisfatórios em relação aos métodos finalistas do ITC 2007. O refinamento das soluções foi mais eficiente quando se utilizou o método VND junto ao ILS, porém o algoritmo ainda se manteve em alguns casos presos em ótimos locais durante um intervalo de tempo, o que provavelmente prejudicou sua convergência durante os 500 segundos.

Outra observação relevante é que o algoritmo proposto foi comparado com os cinco melhores algoritmos da competição, o que o valoriza enquanto resolvedor do problema abordado.

Capítulo 6

Software

Esse capítulo descreve o sistema de informação, denominado SGQH (Sistema Gerenciador de Quadro de Horário), desenvolvido para solucionar o problema de geração de quadro de horários de cursos de acordo com a modelagem do problema apresentado no capítulo 3.

A linguagem HTLM, juntamente com recursos javascript e css foram utilizadas para fazer a interface com o usuário. Além disso, as regras de negócio do sistema foram programadas em PHP utilizando o banco de dados MYSQL.

O objetivo foi tentar fornecer um ambiente amigável para se utilizar o método (*solver*) desenvolvido neste trabalho, ou qualquer outro que possa ser acoplado ao sistema desde que obedeça a interface de comunicação adotada.

Além disso, segundo Souza [10], alguns autores acreditam que os problemas de horários não podem ser totalmente automatizados. Existem duas justificativas para isso, primeiro existem razões que não podem ser facilmente expressas em um sistema automatizado, de forma que torne um quadro melhor que o outro. Outra razão é que uma vez que o espaço de soluções é amplo, a intervenção humana pode conduzir a busca para direções mais promissoras, nas quais alguns métodos dificilmente teriam condições de chegar em um tempo hábil.

6.1 Sistemas de Informação

Segundo Laudon *et al.* [107], um sistema de informação pode ser definido como um conjunto de componentes inter-relacionados que propiciam a coleta, recuperação, processamento, armazenamento e distribuição de informação utilizadas como apoio para as funções organizacionais como coordenação, controle e processo de tomada de decisão de uma certa organização. A coleta de dados brutos em uma organização ou ambiente externo exige um processamento para que se tornem informações significativas para as pessoas ou atividades em que serão empregadas.

A Figura 6.1 mostra as três atividades básicas para um sistema de informação. A entrada pode ser definida como todo processo que envolve captação ou coleta de fonte de dados dentro de uma organização ou no ambiente externo, podem ser citados como exemplos formulários, banco de dados, registros, etc. Já a atividade processar consiste em toda operação que torne uma entrada bruta em uma forma útil e apropriada para o usuário, como por exemplo o processo de geração de quadro de horários feito neste trabalho. A saída é o modo de apresentar ao usuário o resultado do processamento, tal como gráficos, planilhas, etc. Por fim, temos o *feedback*, que é a analise da saída que permite ao usuário corrigir dados de entrada ou processamento com o objetivo de adequar o sistema às suas reais necessidades.

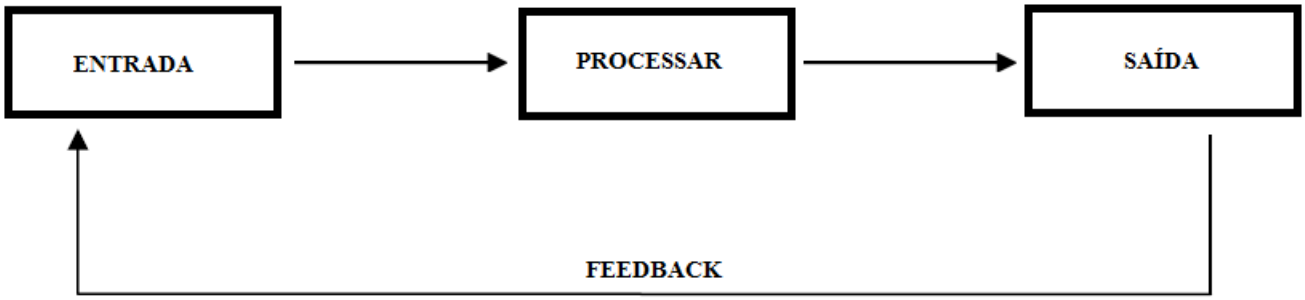

Figura 6.1. Atividades básicas de um sistema de informação

Neste trabalho os dados de entrada referem-se a toda informação necessária para que seja possível criar um quadro de horários a partir do *solver* desenvolvido. Esses dados são inseridos no sistema a partir de uma interface desenvolvido em HTML + CSS + java script. Abaixo são descritas as suas entradas.

- Dados das salas de aula (capacidade, nome)
- Dados do curso (nome, quantidade de alunos, quantidade de aulas, quantidade mínima de dias, professor)
- Dados do professor (login, senha, indisponibilidade)
- Dados de currículo (nome, cursos)
- Quantidade de períodos e dias disponíveis
- Dados de usuário (login, senha, permissão)

A fase de processamento consiste em armazenar os dados em um banco de dados e posteriormente processá-los com o objetivo de criar quadros de horários factíveis em tempo hábil. Para isso foram utilizadas as linguagens PHP e Java.

A saída do sistema consiste em gerar um quadro de horário, na forma de uma tabela, em diferentes perspectivas. O *feedback* pode ser considerado a análise do quadro de horário gerado, permitindo ao usuário que o adéque de acordo com a sua necessidade.

6.2 Arquitetura Cliente-Servidor

Segundo Battisti [108], cliente-servidor é uma arquitetura em que o processamento da informação é dividido em módulos ou processos distintos. Um processo é responsável pela manutenção da informação (Servidor), enquanto que outro é responsável pela obtenção dos dados (Cliente).

Mais especificamente, cliente é a parte responsável pela tarefa de requisição de pedidos ao servidor e também por toda a parte relativa à interação com o usuário final. Normalmente os sistemas cliente abstraem do usuário todas as funções de rede e do servidor, fazendo parecer que todos os processos estão rodando em um mesmo local.

Já o lado servidor, é a parte responsável de um sistema cliente/servidor que tem a função de receber dos clientes as requisições, processá-las e devolvê-las os resultados. A grande vantagem desse sistema é que, por ser totalmente reativo, só é disparado quando recebe alguma requisição do cliente. Isso faz com que o servidor não procure interagir com outros servidores durante um pedido de requisição, o que torna o processo de ativação uma tarefa a ser desempenhada apenas pelo cliente que o solicitou. A figura 6.2 mostra um exemplo desse tipo de arquitetura.

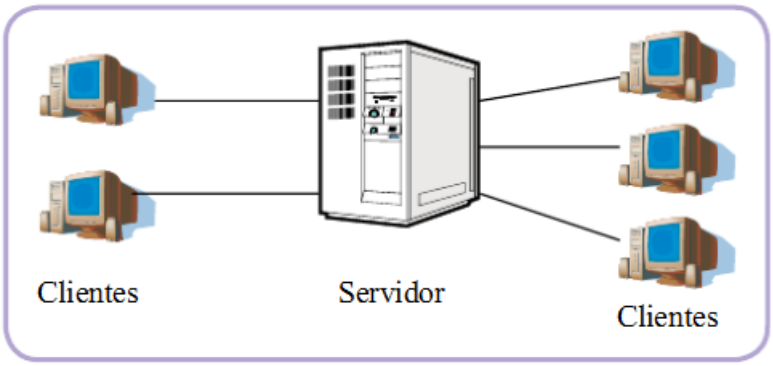

Figura 6.2. Arquitetura cliente servidor

Esse tipo de arquitetura fornece algumas vantagens, como por exemplo, maior facilidade de manutenção, onde é possível substituir, reparar, ou atualizar um servidor sem que seus clientes percebam, ou fiquem sem um determinado serviço. Além disso, todos os dados são armazenados nos servidores, que geralmente possuem controles de segurança muito maior do que a maioria dos clientes. Servidores podem controlar melhor o acesso e recursos, para garantir

que apenas os clientes com as permissões adequadas possam acessar e alterar dados, possuem mecanismos de *backup*, entre outros.

É nesse paradigma que foi elaborado o sistema desenvolvido neste trabalho. O lado cliente é composto pela camada de apresentação do sistema, sendo apresentado ao usuário em *HTML* e java script. Esses fornecem recursos de entrada e saída do sistema, tais como formulários e relatórios.

O lado servidor é composto pelo aplicativo desenvolvido na linguagem PHP, que controla toda a entrada e saída de dados feita no sistema e pelo método *ILS+VND* (Java) responsável em gerar o quadro de horários e pelo banco de dados. A Figura 6.3 detalha o funcionamento do sistema.

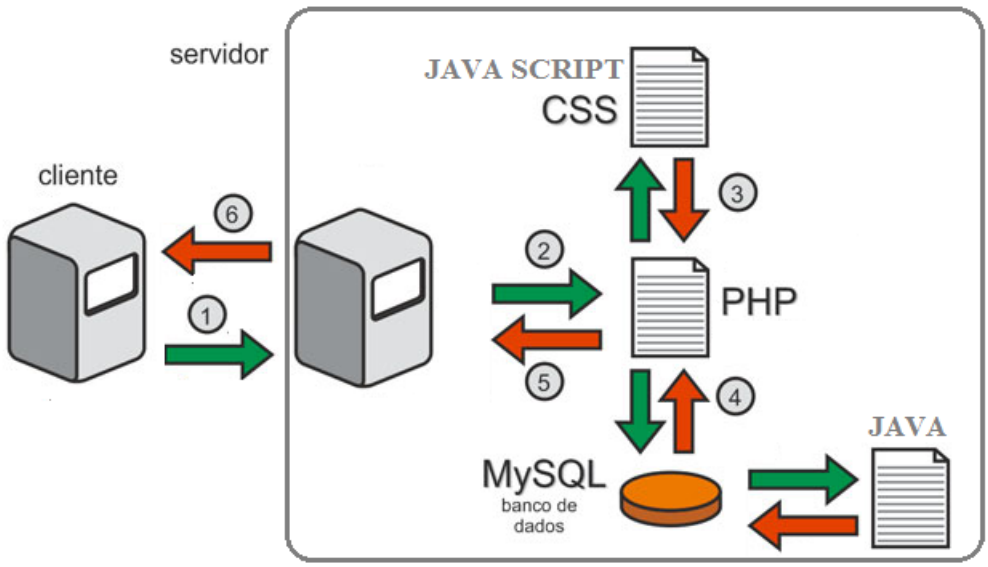

Figura 6.3. Arquitetura do SGQH

O processo se inicia com uma requisição do cliente ao servidor (1), posteriormente o servidor encaminha ao *PHP* a solicitação (2); assim é feito o processamento necessário entre os recursos visuais, e de banco de dados (3 e 4) e retornada uma página *HTML* ao cliente (5 e 6). O programa gerador de quadro de horário (*VND+ILS*) comunica com o aplicativo via banco de dados. Desta forma, quando um *flag* é ativado, ele procura otimizar durante um tempo t o quadro de horários, e caso ele seja melhor que a solução já estabelecida, a solução é salva.

6.3 Tecnologia

Nos últimos anos várias tecnologias para desenvolvimento de software têm surgido para atender diversos tipos de demandas de programadores, cada uma com seus problemas e vantagens.

Escolher uma tecnologia apropriada para um desenvolvimento de um sistema é um fator determinante para o sucesso do projeto.

Nesta seção são caracterizadas as tecnologias, no contexto de linguagem de programação e banco de dados.

6.3.1 Java

Java é uma linguagem relativamente simples, de fácil aprendizado ou migração, pois possui um reduzido número de construções. A diminuição das construções mais suscetíveis a erros de programação, tais como ponteiros e gerenciamento de memória via código de programação também faz com que a programação em Java seja mais fácil de se manipular. Contém um conjunto de bibliotecas que fornecem grande parte da funcionalidade básica da linguagem, incluindo rotinas de acesso à rede, criação de interface gráfica, etc. Baseada no paradigma da Orientação a Objetos, encapsulamento de dados (variáveis) e métodos de manipulação desses dados, a linguagem permite a modularização das aplicações, reuso e manutenção simples do código já implementado [109].

Além disso, a tecnologia Java e seus IDE´s são *free*, gerando nenhum custo de desenvolvimento, a não ser o de desenvolvimento. Outro fator interessante é sua portabilidade. Um mesmo código Java pode executar em diversas plataformas sem a necessidade de alteração de código, assim as aplicações podem ser facilmente migradas entre servidores.

6.3.2 PHP

O PHP (*Hypertext Preprocessor*) é uma linguagem que permite criar aplicações web dinâmicas, possibilitando uma interação com o usuário através de formulários, parâmetros de entrada, entre outras características. A diferença do PHP em relação a linguagens semelhantes, como o Javascript, é que o código PHP é executado no servidor, sendo enviado para o cliente apenas o código HTML. Desta maneira, é possível interagir com bancos de dados e aplicações existentes nos servidores. O fato de ser executada no servidor permiti que o PHP seja executada em computadores com poucos recursos de processamento, bastando basicamente um *browser* internet.

Como as aplicações PHP ficam hospedadas somente no servidor, o *deployment* de aplicações se torna muito mais simples, eliminando uma das complexidades dos sistemas cliente-servidor, o controle de versão de software nas estações. Outra característica importante dessa linguagem é que a escrita de uma aplicação web que faça interação com um banco de dados é extremamente simples.

Essa linguagem foi escolhida para fazer a interface do sistema entre banco de dados $\times$ usuario $\times$ *solver* devido às suas qualidades e o conhecimento prévio do autor deste trabalho nessa linguagem, tornando a produção do sistema mais rápida.

6.3.3 MYSQL

Um banco de dados é uma coleção de dados estruturados. Ele pode ser qualquer coisa desde uma simples lista de compras a uma galeria de imagens ou a grande quantidade de informação de uma universidade. Para adicionar, acessar, e processar dados armazenados em um banco de dados de um computador, é necessário um sistema de gerenciamento de bancos de dados como o Servidor MySQL. Como os computadores são muito bons em lidar com grandes quantidades de dados, o gerenciamento de bancos de dados funciona como a engrenagem central na computação, seja como utilitários independentes ou como partes de outras aplicações.

O Servidor MySQL foi desenvolvido originalmente para lidar com bancos de dados muito grandes de maneira muito mais rápida que as soluções existentes e tem sido usado em ambientes de produção de alta demanda por diversos anos de maneira bem sucedida. Apesar de estar em constante desenvolvimento, o Servidor MySQL oferece hoje um rico e proveitoso conjunto de funções. A conectividade, velocidade, e segurança fazem com que o MySQL seja altamente eficiente para acessar bancos de dados em sistemas baseados em internet.

Outras características importantes tais como portabilidade, pois suporta a maioria das plataformas existentes no mercado; compatibilidade, já que existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação; sua pouco exigência em relação a *hardware* e sua facilidade de uso tornou esse gerenciador um dos mais utilizados no mundo.

Além disso é um software livre e suporta a maioria dos recursos necessários de um bom gerenciador de banco de dados (Triggers, Store Procedures, Functions, etc). Em vista de suas qualidade, esse servidor de banco de dados foi escolhido para integrar o sistema desenvolvido neste trabalho.

6.3.4 Java Script

Javascript é uma linguagem de programação utilizada para criar pequenos programas encarregados de realizar ações dentro do âmbito de uma página web.

Trata-se de uma linguagem de programação do lado do cliente, porque é o navegador que suporta a carga de processamento. Graças a sua compatibilidade com a maioria dos navegadores modernos, é a linguagem de programação do lado do cliente mais utilizada.

Com Javascript é possível criar efeitos especiais nas páginas e defini-las de um modo interativo com o usuário. O navegador do cliente é o encarregado de interpretar as instruções Javascript e executá-las para realizar estes efeitos.

A necessidade de criar páginas interativas no sistema desenvolvido fez com que o uso dessa linguagem fosse indispensável.

6.3.5 CSS

Cascading Style Sheets, ou simplesmente CSS, é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como por exemplo HTML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

Em vez de colocar a formatação dentro do documento, o desenvolvedor cria um *link* (ligação) para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um sistema. Quando quiser alterar a aparência do sistema basta portanto modificar apenas um arquivo.

Em outras palavras, podemos definir CSS com um conjunto de propriedades visuais para elementos de um sistema, de forma centralizada.

6.4 UML

A UML (*Unified Modelling Language*) é uma linguagem/notação de diagramas para especificar, visualizar e documentar modelos de sistemas programados no paradigma de programação orientada a objetos, essa metodologia é controlada pelo *Object Management Group* (OMG) e é a norma da indústria para documentar graficamente um sistema.

Com essa linguagem é possível expressar várias visões necessárias para desenvolvimento e implantação de um sistema. Uma visão é uma abstração do sistema, formada por um conjunto de diagramas. Abaixo são descritos os tipos de visões encontrados na UML [110, 111].

- Visão “use-case”: É uma visão que descreve a funcionalidade do sistema desempenhada pelos atores externos do sistema (usuários). A visão use-case é central, já que seu conteúdo é base do desenvolvimento das outras visões do sistema. Essa visão é montada sobre os diagramas de use-case e eventualmente diagramas de atividade.

- Visão Lógica: Descreve como a funcionalidade do sistema será implementada. É feita principalmente pelos analistas e desenvolvedores. Em contraste com a visão use-case, a visão lógica observa e estuda o sistema internamente. Ela descreve e especifica a estrutura estática do sistema (classes, objetos, e relacionamentos) e as colaborações dinâmicas

quando os objetos enviam mensagens uns para os outros para realizarem as funções do sistema. Propriedades como persistência e concorrência são definidas nesta fase, bem como as interfaces e as estruturas de classes. A estrutura estática é descrita pelos diagramas de classes e objetos. O modelamento dinâmico é descrito pelos diagramas de estado, sequência, colaboração e atividade.

- Visão de Componentes: É uma descrição da implementação dos módulos e suas dependências. É principalmente executado por desenvolvedores, e consiste nos componentes dos diagramas.

- Visão de concorrência: Trata a divisão do sistema em processos e processadores. Este aspecto, que é uma propriedade não funcional do sistema, permite uma melhor utilização do ambiente onde o sistema se encontrará, se o mesmo possui execuções paralelas, e se existe dentro do sistema um gerenciamento de eventos assíncronos. Uma vez dividido o sistema em linhas de execução de processos concorrentes (*threads*), esta visão de concorrência deverá mostrar como se dá a comunicação e a concorrência destas *threads*. A visão de concorrência é suportada pelos diagramas dinâmicos, que são os diagramas de estado, sequencia, colaboração e atividade, e pelos diagramas de implementação, que são os diagramas de componente e execução.

- Visão de Organização: A visão de organização mostra a organização física do sistema, os computadores, os periféricos e como eles se conectam entre si. Esta visão será executada pelos desenvolvedores, integradores e testadores, e será representada pelo diagrama de execução.

Neste trabalho apresentaremos a visão denominada *use-case* e a visão lógica (diagrama de classes). A visão *use-case* permite que inicialmente os requisitos necessários do software sejam levantados, além de informar como os usuários devem proceder em relação ao sistema. Já a visão lógica vai permitir ao leitor ter uma idéia de toda estrutura estática do sistema, em particular as entidades existentes. O diagrama de classe não foi totalmente detalhado para não tornar muito complexo o seu entendimento e seu desenho.

6.4.1 Caso de Uso

A seguir são descritos os principais atores do sistema, os quais representam papéis desempenhados por pessoas, dispositivos ou outros quando interagem com o sistema. Um ator cujo identificador seja ALUNO não representa um aluno específico, mais sim um aluno qualquer, uma pessoa que esteja interagindo com o sistema nessa qualidade. A Figura 6.4 mostra o di-

agrama de caso de uso do sistema SGQH. No restante desta seção são detalhados os casos de uso pertencente ao SGQH.

- Administrador: Usuário responsável em gerir o sistema de quadro de horário, que tem permissão total no sistema para executar todas as operações.

- Professor: Esse usuário tem a permissão de ver os relatórios do sistema, além de cadastrar sua disponibilidade no sistema (recurso)

- Aluno: Usuário que tem somente a permissão de ver o quadro de horário

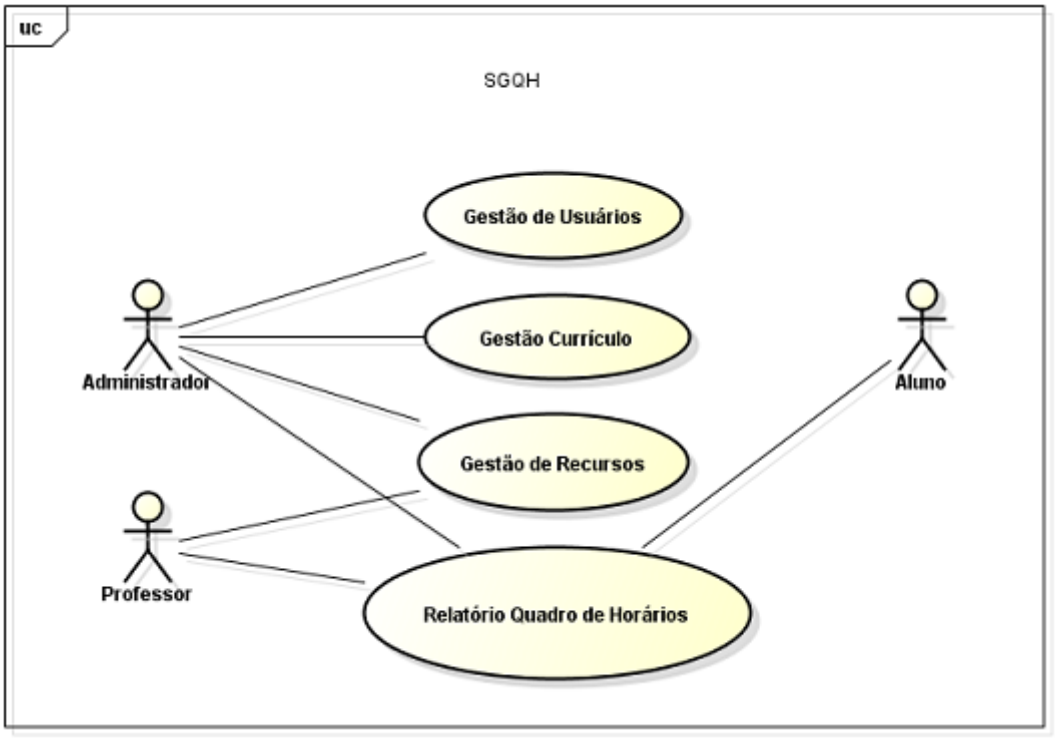

Figura 6.4. Diagrama de caso de uso sistema SGQH

6.4.2 Caso de uso Gestão de Usuários

Nesta seção são mostrados o diagrama de caso de uso para Gestão de Usuários (Figura 6.5) e os casos de uso em forma textual.

Precondição: Estar logado no sistema como Administrador.

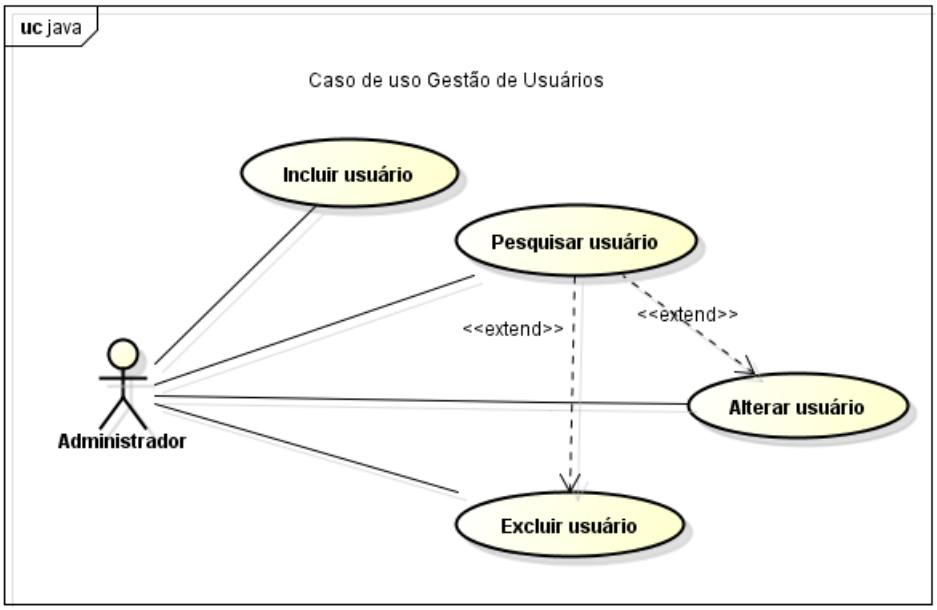

Figura 6.5. Diagrama de caso de uso Gestão de Usuários

Fluxo Principal:

1	O Administrador aciona o comando usuários.
2	O SGQH exibe tela com informações de alguns usuários cadastrados.

Fluxo Alternativo Pesquisar usuário:

1	O Administrador informa o login/nome do usuário.
2	O Administrador aciona o comando procurar.
3	O SGQH exibe os dados do usuário na tela.

Fluxo Alternativo Incluir usuário:

1	O Administrador aciona o comando incluir.
2	O Administrador preenche todos os dados de usuario.
3	O Administrador define o tipo de usuario (Administrador, Professor ou Aluno).
4	O Administrador aciona o comando salvar.
5	O SGQH cadastra o usuário.

Fluxo Alternativo Alterar usuário

1	Fluxo Alternativo Pesquisar Usuário
2	O Administrador aciona o comando alterar
3	O Administrador altera dados do usuário
4	O Administrador aciona o comando salvar.
5	O SGQH altera dados do usuário.

Fluxo Alternativo Excluir usuário

1	Fluxo Alternativo Pesquisar Usuário
2	O Administrador aciona o comando excluir
3	O SGQH exclui dados do usuário.

6.4.3 Caso de uso Gestão Currículo

Nesta seção são mostrados o diagrama de caso de uso para Gestão de Currículo (Figura 6.6) e os casos de uso em forma textual.

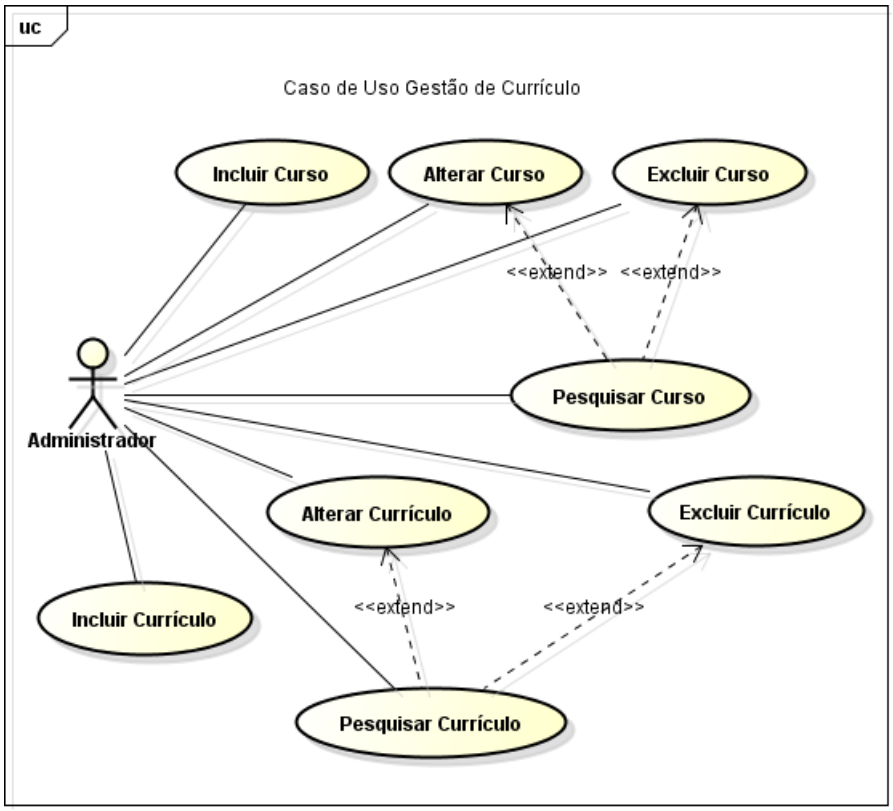

Figura 6.6. Diagrama de caso de uso Gestão Currículo

Precondição: Estar logado no sistema como Administrador.

Fluxo Principal curso:

1	O administrar aciona o comando curso
2	O SGQH exibe tela com algumas informações dos cursos cadastrados.

Fluxo Alternativo Pesquisar curso:

1	O Administrador informa o nome do curso.
2	O Administrador aciona o comando procurar.
3	O SGQH exibe os dados do curso na tela.

Fluxo Alternativo Incluir curso:

1	O Administrador aciona o comando incluir curso.
2	O Administrador preenche todos os dados para o curso.
3	O Administrador seleciona professor que será responsável pelo curso.
4	O Administrador aciona o comando salvar.
5	O SGQH cadastra o curso.

Fluxo Alternativo Alterar curso

1	Fluxo Alternativo Pesquisar curso.
2	O Administrador aciona o comando alterar.
3	O Administrador altera dados do curso.
4	O Administrador aciona o comando salvar.
5	O SGQH altera dados do curso.

Fluxo Alternativo Excluir curso

1	Fluxo Alternativo Pesquisar curso
2	O Administrador aciona o comando excluir
3	O SGQH exclui dados do curso caso esse não pertença a nenhum currículo.

Fluxo Principal currículo:

1	O administrar aciona o comando currículo
2	O SGQH exibe tela com algumas informações dos currículos cadastrados.

Fluxo Alternativo Pesquisar currículo:

1	O Administrador informa o nome do currículo.
2	O Administrador aciona o comando procurar.
3	O SGQH exibe os dados do curriculo na tela.

Fluxo Alternativo Incluir currículo:

1	O Administrador aciona o comando incluir currículo.
2	O Administrador preenche todos os dados para o currículo.
3	O Administrador seleciona os curso que faram parte do currículo.
4	O Administrador aciona o comando salvar.
5	O SGQH cadastra o currículo.

Fluxo Alternativo Alterar currículo

1	Fluxo Alternativo Pesquisar currículo.
2	O Administrador aciona o comando alterar.
3	O Administrador altera dados do currículo.
4	O Administrador aciona o comando salvar.
5	O SGQH altera dados do currículo.

Fluxo Alternativo Excluir currículo

1	Fluxo Alternativo Pesquisar currículo
2	O Administrador aciona o comando excluir
3	O SGQH exclui dados do currículo.

6.4.4 Caso de uso Gestão de Recursos

Nessa seção são mostrados o diagrama de caso de uso para Gestão de Recursos (Figura 6.7) e os casos de uso em forma textual.

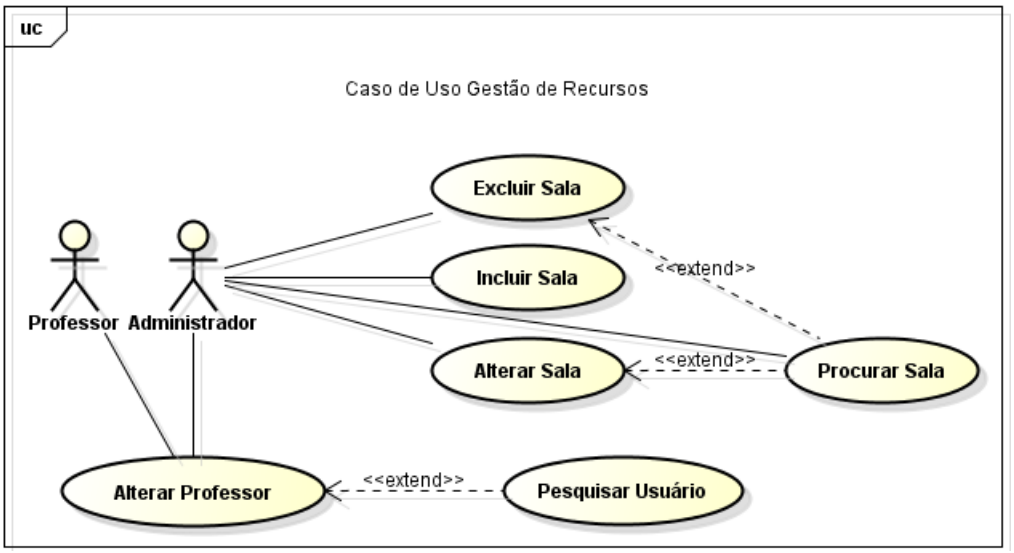

Figura 6.7. Diagrama de caso de uso Gestão de Recursos

Precondição: Estar logado no sistema como Administrador.

Fluxo Alternativo Pesquisar Sala:

1	O Administrador informa o identificador da sala.
2	O Administrador aciona o comando procurar.
3	O SGQH exibe os dados da sala na tela.

Fluxo Alternativo Incluir Sala:

1	O Administrador aciona o comando incluir.
2	O Administrador preenche todos os dados da sala.
3	O Administrador aciona o comando salvar.
4	O SGQH cadastra a sala.

Fluxo Alternativo Alterar Sala

1	Fluxo Alternativo Pesquisar Sala
2	O Administrador aciona o comando alterar
3	O Administrador altera dados da sala
4	O Administrador aciona o comando salvar.
5	O SGQH altera dados da sala.

Fluxo Alternativo Excluir sala

1	Fluxo Alternativo Pesquisar Sala
2	O Administrador aciona o comando excluir
3	O SGQH exclui dados da Sala, caso exista aula atribuída a essa sala o Quadro de horário é indisponibilizado.

Fluxo Alternativo Alterar Professor

1	Fluxo Alternativo Pesquisar Usuário (Professor)
2	O Administrador aciona o comando alterar
3	O Administrador altera dados relativo a disponibilidade do Professor (sem restrições)
4	O Administrador aciona o comando salvar.
5	O SGQH altera dados relativo a disponibilidade do professor.

Precondição: Estar logado no sistema como Professor.

Fluxo Alternativo Alterar Professor

1	O Professor aciona o comando alterar dados
2	O Professor altera dados relativo a disponibilidade (com restrições em relação de períodos indisponíveis)
3	O Professor aciona o comando salvar.
4	O SGQH altera dados relativo a disponibilidade do professor.

6.4.5 Caso de uso Relatório Quadro de Horários

Nesta seção são mostrados o diagrama de caso de uso para Relatório Quadro de Horários (Figura 6.8) e os casos de uso em forma textual.

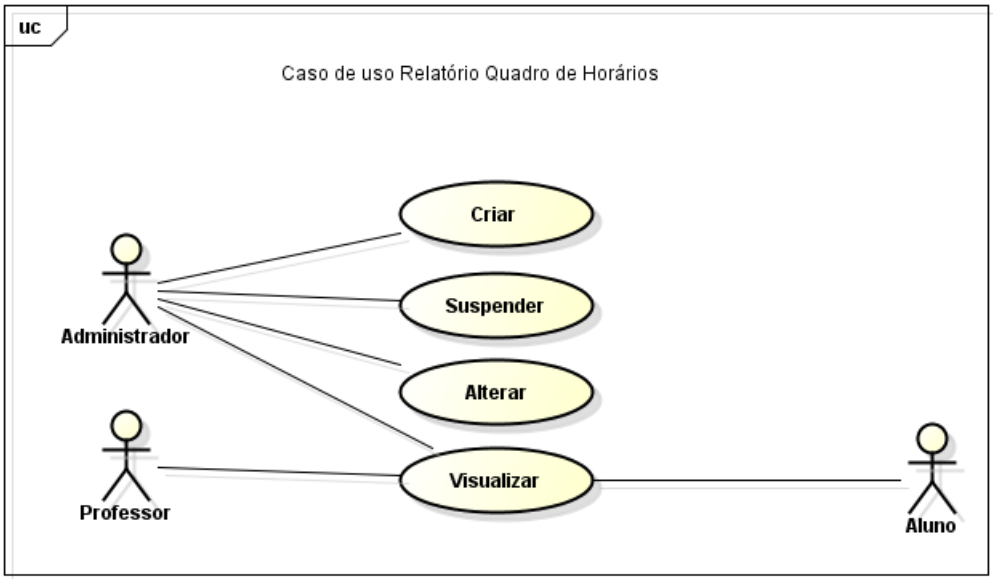

Figura 6.8. Diagrama de caso de uso Relatório Quadro de Horário

Precondição: Estar logado no sistema como Administrador.

Fluxo Principal Relatório Quadro de Horário:

1	O Administrador aciona o comando Timetable
2	O SGQH exibe tela com algumas opções de gerenciamento.

Fluxo Alternativo Criar Quadro de Horário:

1	O Administrador aciona o comando Criar
2	Se existir um quadro de horário já criado o SGQH pergunta se deve começar a gerar o quadro de horário a partir dessa solução
2	O SGQH exibe tela com sistema indisponível até que o processo de geração termine.
3	O SGQH exibe quadro de horário criado

Fluxo Alternativo Suspender Quadro de Horário:

1	O Administrador aciona o comando Suspender
2	O SGQH suspende o quadro de horário corrente.

Fluxo Alternativo Suspender Quadro de Horário:

1	O Administrador aciona o comando Suspender
2	O SGQH suspende o quadro de horário corrente.

Fluxo Alternativo Alterar Quadro de Horário:

1	O Administrador aciona o comando Alterar
2	O Administrador escolhe o currículo
3	O SGQH entra no modo de alteração de quadro de horário
4	O Administrador altera o quadro de aciona o comando salvar
5	O SGQH salva o quadro de horário

Fluxo Alternativo Visualizar Quadro de Horário:

1	O Administrador aciona o comando Visualizar
2	O Administrador escolhe a perspectiva para visualizar o quadro de horário
3	O SGQH mostra o quadro de horário

6.4.6 Diagrama de Classes

Um diagrama de classes descreve os tipos de objetos (instância de uma classe) no sistema e os vários tipos de relacionamentos estáticos que existem entre eles [110]. As classes, no diagrama de classes, podem ser representadas por um retângulo com o seu nome, e seus relacionamentos geralmente são representados por meio de linhas/setas. Alguns desses relacionamentos são mostrados na Figura 6.9.

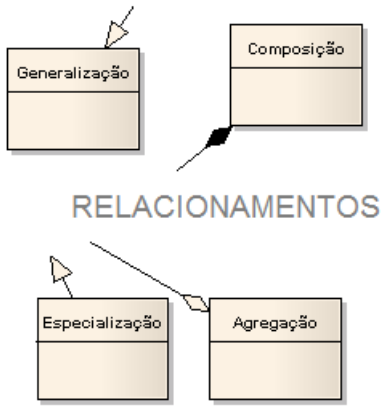

Figura 6.9. Tipos de relacionamento em Diagrama de Classes

Na agregação a alocação do objeto agregado não ocorre de forma estática no interior da classe do objeto que agrega. A alocação do objeto ocorre de forma estática fora do objeto que

agrega ou de forma dinâmica em seu interior. O objeto que agrega guarda apenas o identificador ou endereço do objeto agregado. Por consequência, uma agregação por associação não é rigorosamente uma agregação.

O tipo de relacionamento Composição é uma agregação de fato, significa que é feito uma criação ou alocação de um objeto dentro de um outro objeto de forma estática. A seguir é exemplificado por um trecho de código escrito na linguagem Java esse relacionamento, sendo que Pedido e ItemPedido caracterizam duas classes distintas.

```
Class Pedido{
int numero;
ItemPedido item;
}
```

A generalização ou especialização é um relacionamento que ocorre efetivamente em termos estruturais entre duas classes. Das duas classes envolvidas, uma será considerada uma classe base (ou superclasse) e a outra será considerada uma classe derivada (ou subclasse). O significado deste relacionamento depende do sentido de leitura da relação. Lendo-se a relação no sentido da classe base para a classe derivada entende-se a relação como uma especialização, ou seja, a classe derivada seria uma classe especial definida a partir da classe base. Lendo-se a relação no sentido da classe derivada para a classe base, entende-se a relação como uma generalização, ou seja, a classe base representa um caso geral da classe derivada.

A Figura 6.10 apresenta o diagrama de classe de forma simplificada do método heurístico. A classe Arquivo e BancoDados são responsáveis em fornecer os dados necessários para o método heurístico e gravar os resultados obtidos por intermédio da classe Main (classe principal). A classe Problema é onde foi implementado o método heurístico utilizado nesse trabalho. As classes Curso, Sala e Currículo são as entidades envolvidas no domínio da aplicação.

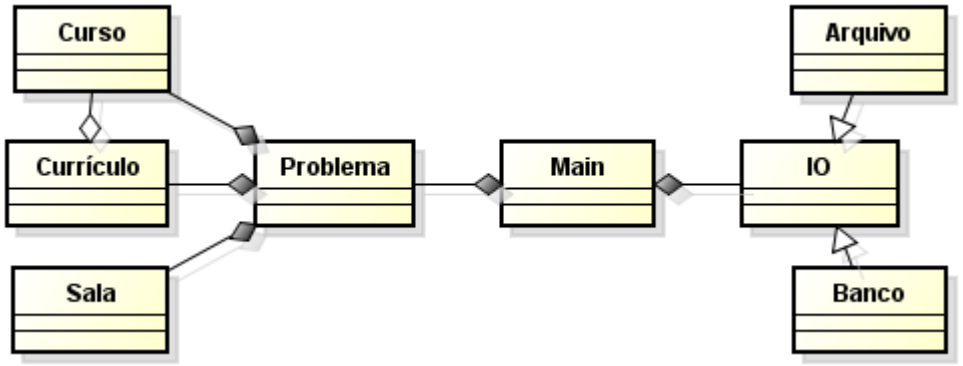

Figura 6.10. Diagrama de Classe do método heurístico

A Figura 6.11 apresenta o diagrama de classe na sua forma simplificada para o sistema desenvolvido. O tipo de usuário que acessa o sistema é definido pela classe Usuario, que se especializa nas classes Professor, Administrador e Aluno. Cada classe possui seus próprios atributos. A classe Controlador é reponsável em gerenciar todo o sistema, da interface grafica (GUI) ao acesso ao banco de dados (Banco). As demais classes são entidades envolvidas no domínio da aplicação.

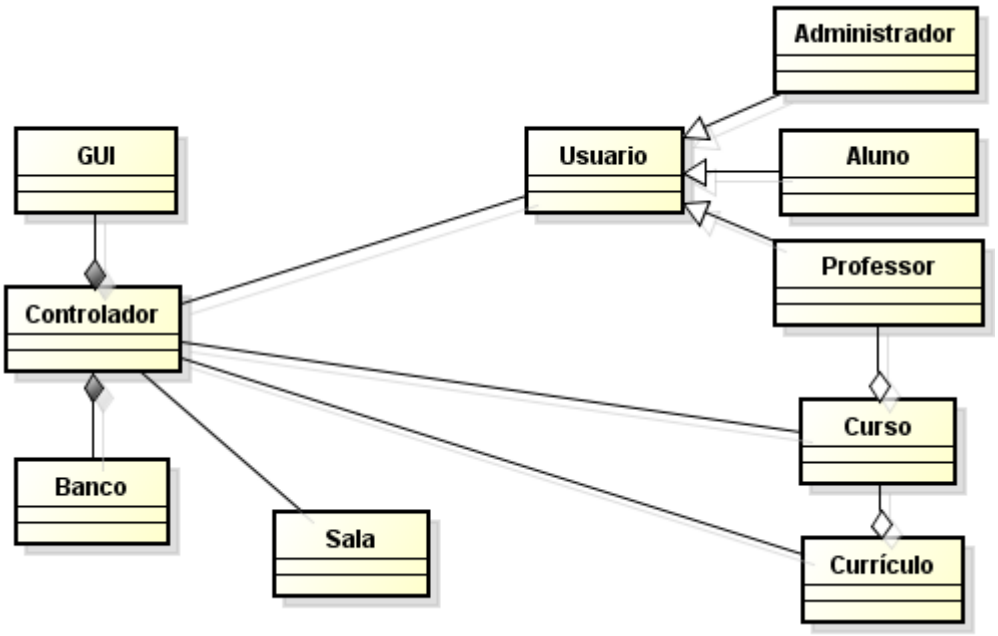

Figura 6.11. Diagrama de Classe do SGQH

6.5 Interface Gráfica

Nesta seção são mostradas as interfaces gráficas utilizadas no sistema (Figuras 6.11-6.26).

Figura 6.12. Tela de acesso ao sistema

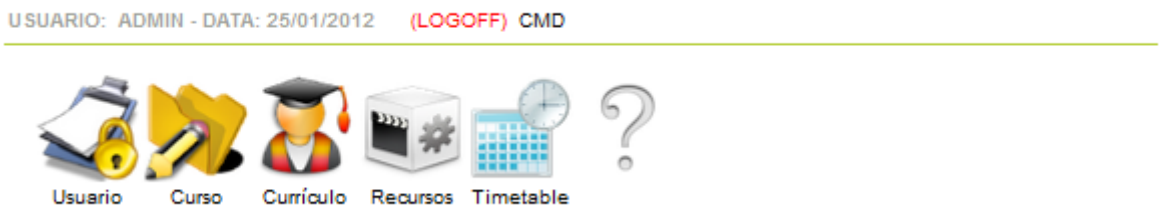

Figura 6.13. Tela principal

Figura 6.14. Tela principal Usuários

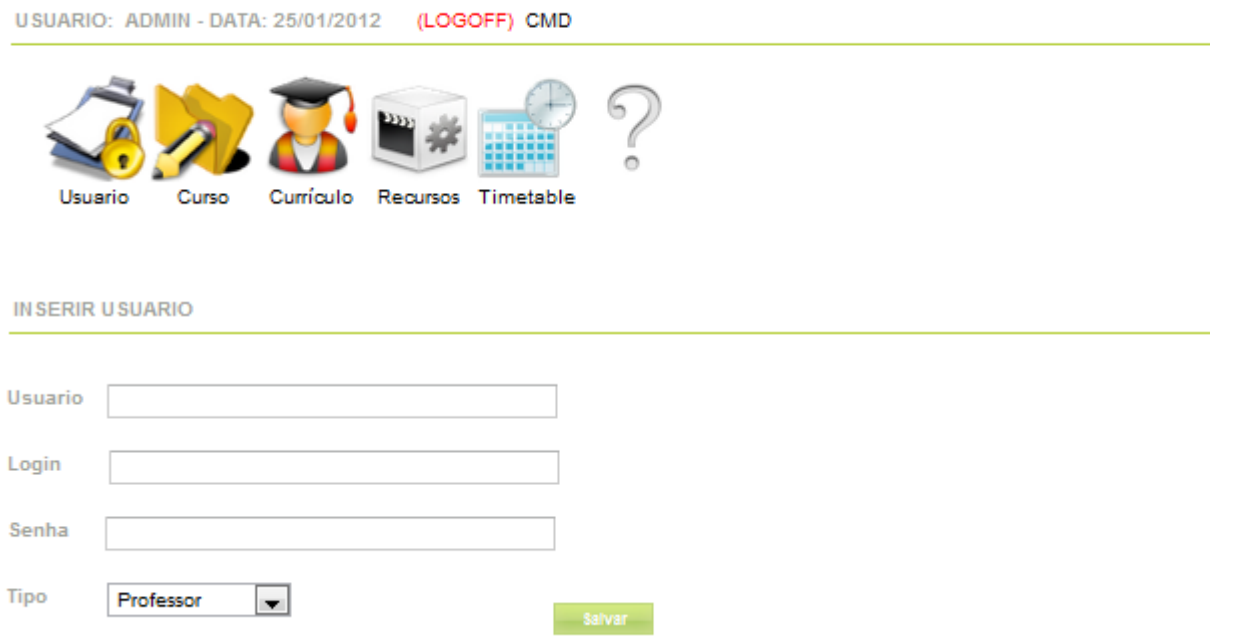

Figura 6.15. Tela Cadastro/Alteração Usuários

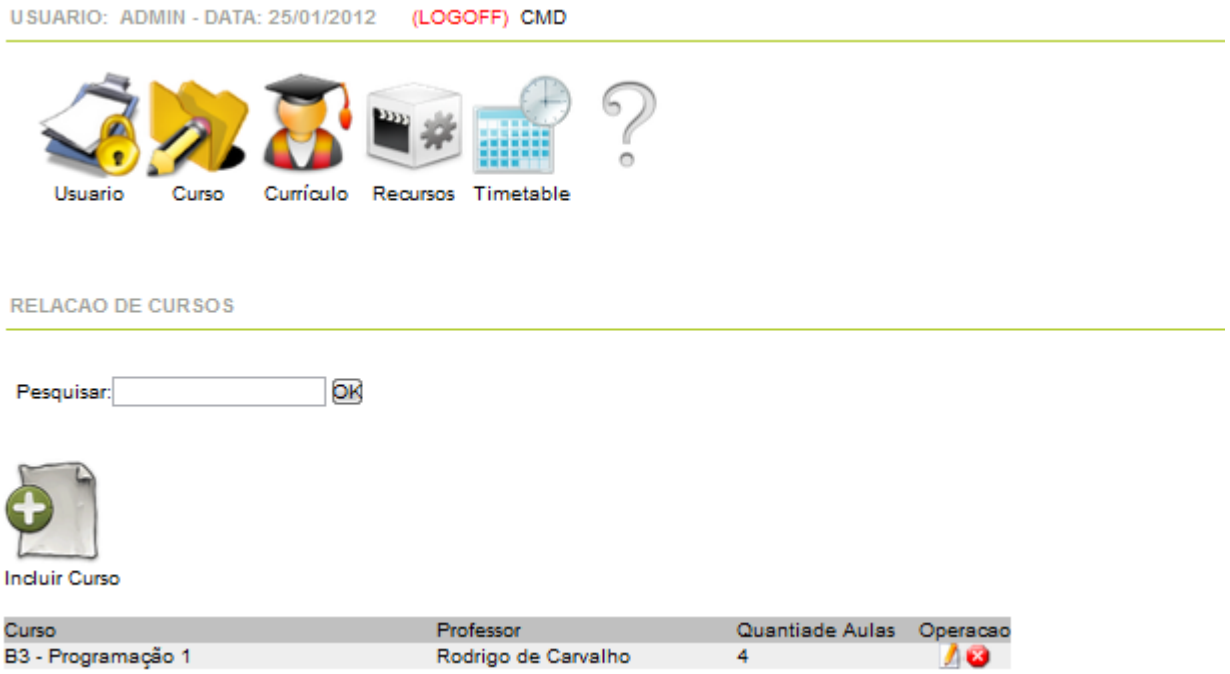

Figura 6.16. Tela Principal Curso

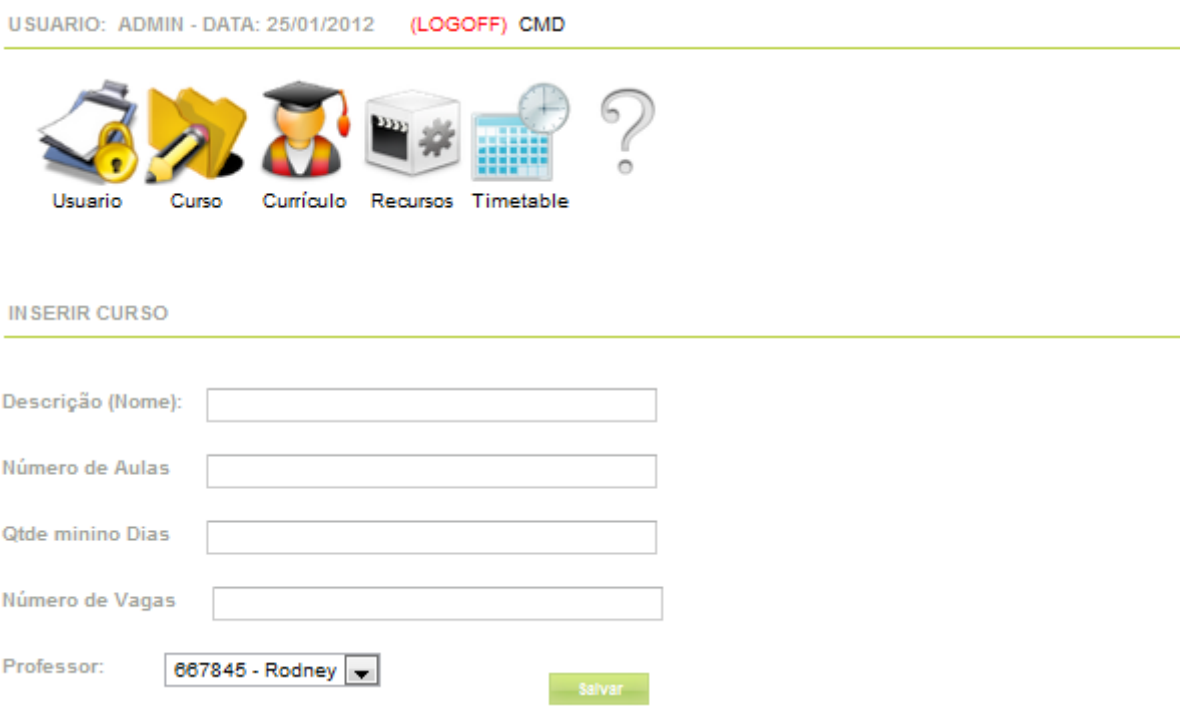

Figura 6.17. Tela Cadastro/Alteração Curso

Figura 6.18. Tela Principal Sala

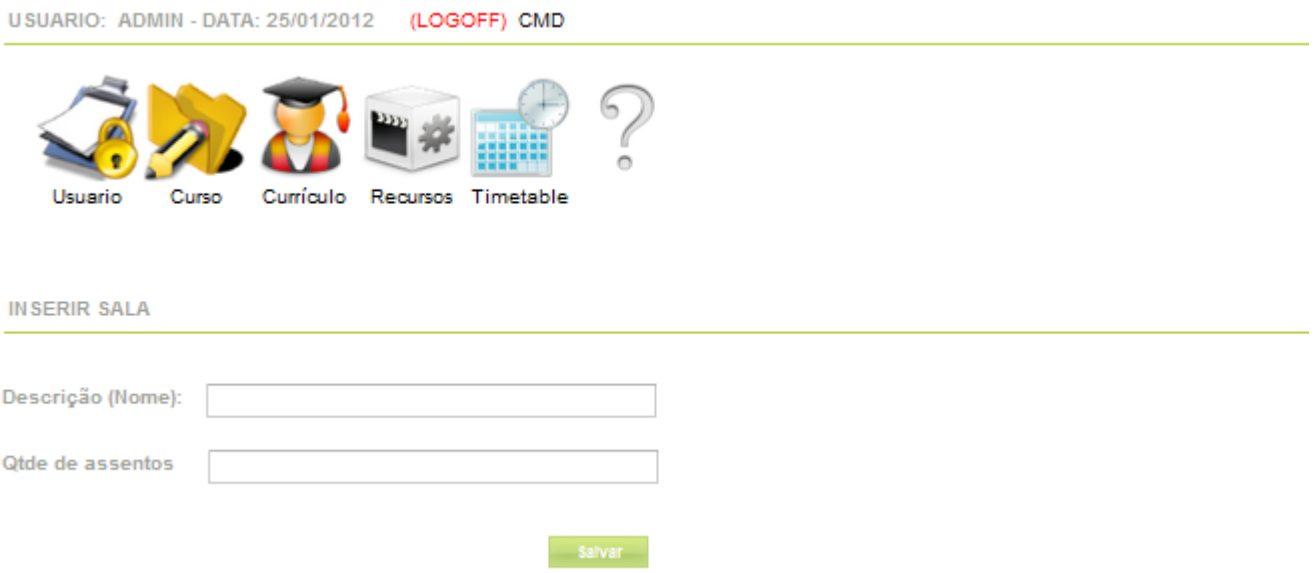

Figura 6.19. Tela Cadastro/Alteração Sala

Figura 6.20. Tela Principal Currículo

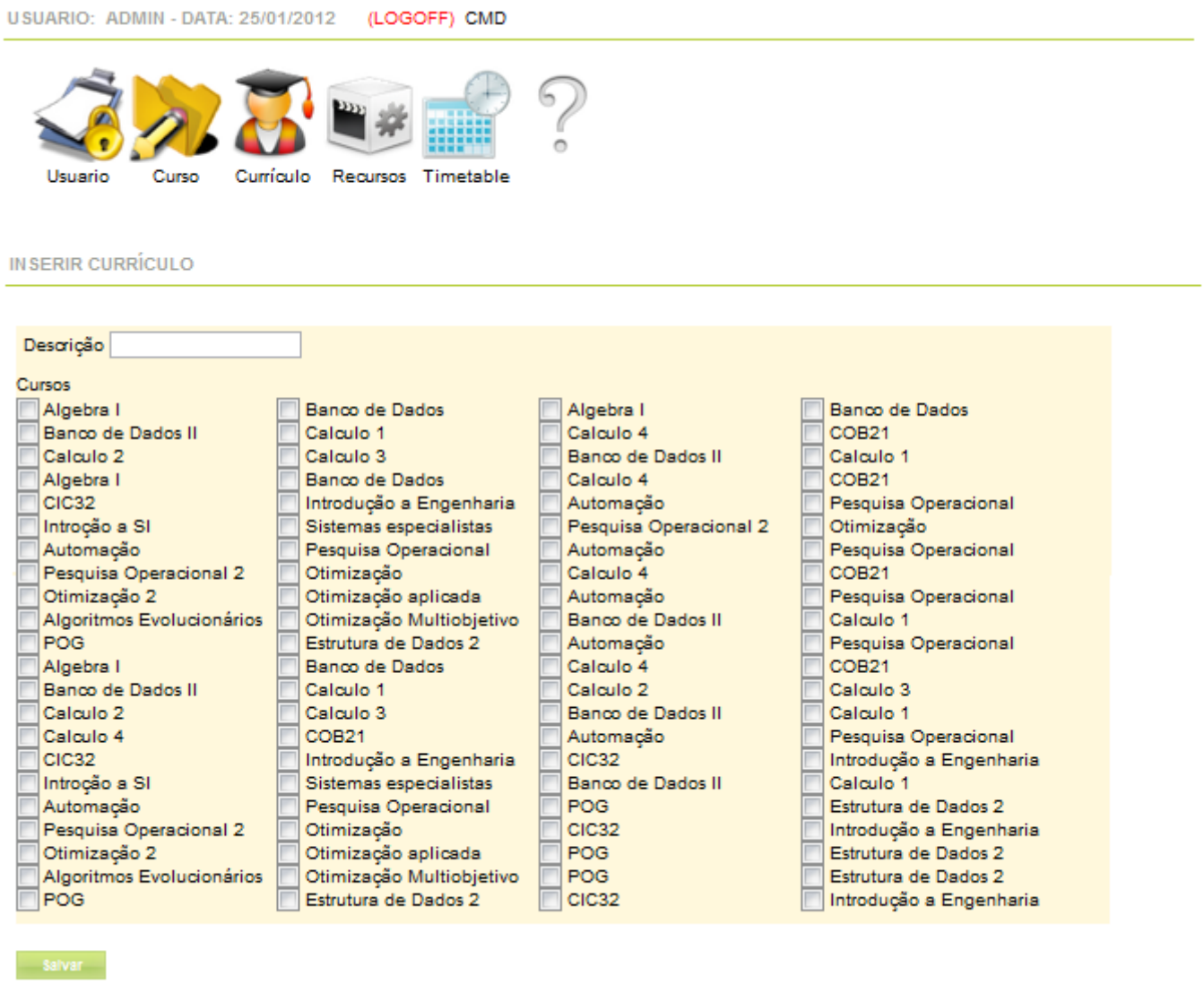

Figura 6.21. Tela Cadastro/Alteração Currículo

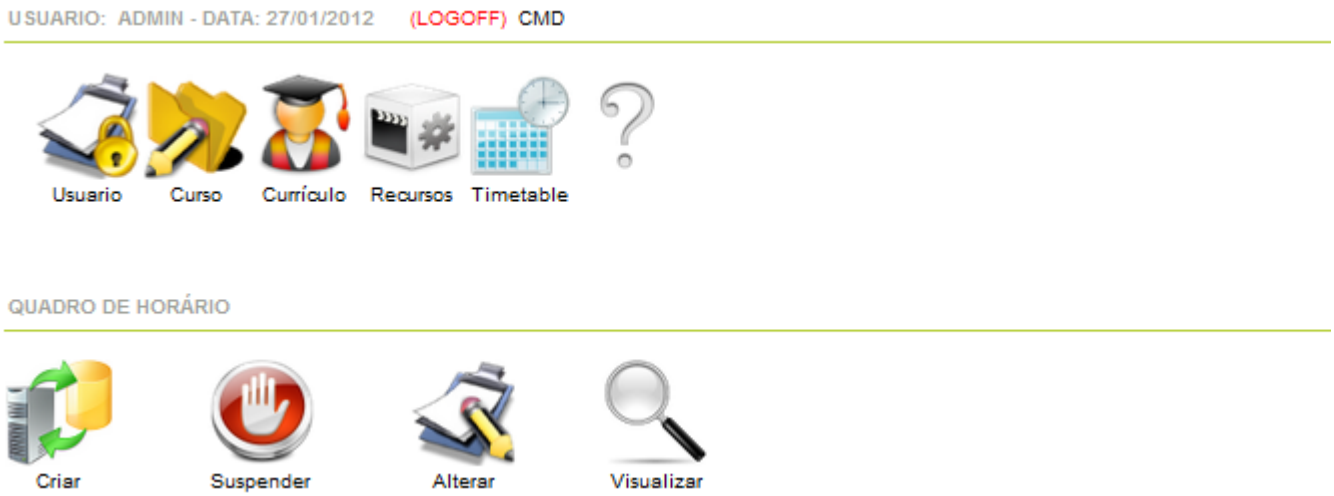

Figura 6.22. Tela Principal Quadro de Horários

Segunda	Terça	Quarta	Quinta	Sexta	Sabado
Pesquisa Operacional			Otimização	Otimização	
Pesquisa Operacional			Otimização	Otimização	
Programação 1			Programação 1	Estrutura de Dados 1	
Programação 1			Programação 1	Estrutura de Dados 1	
Filosofia			Filosofia	Trabalho Final	

Figura 6.23. Tela Quadro de Horários - Currículo (Interface de Alteração)

QUADRO DE HORÁRIOS - SALA 2206

Segunda	Terça	Quarta	Quinta	Sexta	Sabado
Pesquisa Operacional		POG	Otimização	Alg. Evolucionários	
Redes		POG	Sistemas Distribuidos	Otimização	
Programação 1	Estrutura de Dados 1		Programação 1	Estrutura de Dados 1	
Programação 1	Estrutura de Dados 1	POG 2	Programação 1		
Filosofia		POG 2	Filosofia	Trabalho Final	

Figura 6.24. Tela Quadro de Horários - Sala

PROFESSOR RODNEY R. SALDANHA

Segunda	Terça	Quarta	Quinta	Sexta	Sabado
SALA 2206					
SALA 2206	SALA 1200				
SALA 2206	SALA 1200		SALA 1501	SALA 1501	
	SALA 1200		SALA 1501	SALA 1501	
	SALA 1200				

Figura 6.25. Tela Quadro de Horários - Professor

USUARIO: ADMIN - DATA: 28/01/2012 (LOGOFF) CMD

ENGENHARIA ELETRICA

Segunda	Terça	Quarta	Quinta	Sexta	Sabado
Programação 1	Calculo	Otimização			
Programação 1	Calculo	Otimização			
Programação 1	Calculo			Otimização	
		Calculo		Otimização	
		Calculo		Otimização	

SISTEMAS DE INFORMAÇÃO

Segunda	Terça	Quarta	Quinta	Sexta	Sabado
Programação 1	Sist. Operacionais	Sist. Distribuidos			
Programação 1	Sist. Operacionais	Sist. Distribuidos			
Programação 1	Sist. Operacionais	Sist. Distribuidos		Otimização	
		Calculo		Otimização	
		Calculo		Otimização	

ADMINISTRAÇÃO

Segunda	Terça	Quarta	Quinta	Sexta	Sabado
Teoria Admin.	Teoria Admin.	Pesquisa Operacional	TGA	TGA	
Teoria Admin.	Teoria Admin.	Pesquisa Operacional		TGA	
Calculo	Calculo	TGA		Contabilidade	
		Calculo		Contabilidade	
		Calculo		Contabilidade	

Figura 6.26. Tela Quadro de Horários - Visão Geral

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho apresentou um estudo sobre o problema de geração de quadro de horários de cursos universitários baseado em currículos.

No Capítulo 1, foi mostrada a importância deste problema no contexto teórico e no prático. Também foi feita uma breve discussão sobre o grau de dificuldade em resolvê-lo, além de citar os principais objetivos e contribuições do trabalho.

O Capítulo 2 apresentou as principais variantes do problema de geração de quadro de horários por meio de descrições e suas formulações matemáticas. Ainda, uma breve contextualização histórica em relação aos métodos da literatura para resolver o problema é feita. O problema especifico tratado neste trabalho foi descrito no Capítulo 3, sendo mostrados os objetivos e restrições, além das limitações impostas pelo ITC 2007.

O Capítulo 4 apresentou a metodologia proposta para resolver o problema. Foram mostrados os métodos utilizados para obtenção de uma solução, desde o método de geração de solução até o método final de refinamento utilizado.

Os resultados computacionais foram apresentados no Capítulo 5 com o objetivo de mensurar o desempenho do método proposto. Inicialmente foram mostrados os resultados obtidos pelo método de construção, que obteve em todos os testes uma solução viável em tempo hábil, menos de 4%, em média, do tempo total de processamento. Posteriormente foram mostrados os resultados obtidos pelo método ILS, e a combinação do método ILS+VND. Os resultados alcançados por essa combinação foram, em média, 30% melhores em relação ao método ILS. Quando comparado com os métodos finalistas do ITC 2007, os resultados foram satisfatórios, ficando próximo ao quinto colocado dessa competição.

O sistema de informação (*software*) desenvolvido foi descrito no Capítulo 6. Foram mostradas as tecnologias, linguagens de programação e banco de dados utilizados. Além disso, alguns recursos UML foram utilizados para descrever e documentar as funcionalidades do sistema.

Neste trabalho foi possível contribuir tanto em caráter teórico, por meio do estudo do problema, e da proposta dos métodos de resolução, quanto em caráter prático, pela resolução de instâncias realísticas e do *software* de gerenciamento proposto.

Em vista dos resultados promissores do algoritmo proposto, de sua flexibilidade e do sistema desenvolvido, propõe-se incluir no mesmo um número maior de restrições, além daquelas impostas pelo ITC. O objetivo é tornar o sistema desenvolvido um produto a ser utilizado nas mais diversas Instituições de Ensino Universitário brasileiras.

Como trabalhos futuros, apontamos:

- Conceber outros mecanismos de perturbação para ser acrescentados ao ILS, de forma a evitar a estagnação das soluções.
- Combinar o método ILS com outros métodos de refinamento de soluções.
- Implementar outras restrições que apareçam em Instituições de Ensino Universitário.
- Expandir o método implementado para as diferentes variantes do problema.
- Utilizar essa metodologia na competição ITC 2011, que acontece entre outubro de 2011 a maio de 2012.

Referências Bibliográficas

[1] J. Schõnberger, D. C. Mattfeld, H. Kopfer, Algorithm timetabling for sport leagues, .European Journal of Operation Research 153 (2004) 102–116.

[2] M. T. Mine, M. S. A. Silva, M. J. F. Souza, S. G. P., L. S. Ochi, Iterated local search aplicado à programação de jogos de competições esportivas realizadas em dois turnos completos e espelhados: um estudo de caso, XIII Congreso Latino-Iberoamericano de Investigación Operativa, Montevideo. Anales del XIII Congreso Latino-Iberoamericano de Investigación Operativa. Montevideo : Universidad de La Republica 1 (2006) 1–6.

[3] M. Schoenauer, S. Y., An efficient memetic permutation based evolutionay algorithm for real-world train timetabling, Edinburgh, UK. CEC05: Congresso n Evolutionary Computation, IEEE Press 1 (2005) 2752–2759.

[4] F. D. Croce, S. F., A variable neighborhood search based matheuristic for nurse rostering problems, in: PATAT 2010, 8th International Conference on the Practice and Theory of Automated Timetabling, 2010.

[5] J. Aubin, J. A. Ferland, A large scale timetabling problem, Computers and Operations Research 16 (1989) 67–77.

[6] S. Petrovic, E. Burke, University timetabling, Leung J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis 45 (2004) .

[7] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper-heuristic for educacional timetabling problems., European Journal of Operation Research 176 (2007) 177–192.

[8] R. A. Achá, R. Nieuwenhuis, Curriculum-based course timetabling with sat and maxsat, in: PATAT 2010, 8th International Conference on the Practice and Theory of Automated Timetabling, 2010.

[9] B. A. S. Lara, Quadro de horários acadêmico: Uma abordagem com foco na avaliação institucional e na gestão de custos de insituições de esino superior privadas brasileiras, Dissertação de Mestrado. Pós Graduação em Ciencia da Computação UFMG, 2006.

[10] M. J. F. Souza, Programação de horários em escolas: uma aproximação por metaheurísticas, Tese (Doutorado em Engenharia de Sistemas e Computacao) UFRJ, Rio de Janeiro, 2000.

[11] J. Csima, C. C. Gotlieb, Tests on a computer method for construction of school timetables, Communications of the ACM 7 (1961) 160–163.

[12] C. C. Gotlieb, The construction of class-teacher timetables, Proceedings of IFIP Congress (1963) 73–77.

[13] E. Burke, P. Ross, Practice and theory of automated timetabling, in: Lecture Notes in Computer Science, 1996.

[14] E. Burke, M. Carter, Practice and theory of automated timetable 2, in: Lecture Notes in Computer Science, 1997.

[15] E. Burke, W. Erben, Practice and theory of automated timetabling 3, in: Lecture Notes in Computer Science, 2000.

[16] B. McCollum, E. Burke, G. White, Pratice and theory automated timetabling (patat 2010), in: Conference Proceedings, 2010.

[17] P. E. Dune, An annotated list of selected np-complete problems, Disponível na Internet em http://www.csc.liv.ac.uk/ ped/teachadmin/COMP202/annotatede-np.html.

[18] T. B. Cooper, J. H. Kingston, The complexity of timetable construction problems., Selected papers from the First International Conference on Practice and Theory of Automateed Timetabling, London, Uk. Springer-Verlag. (1996) 283–195.

[19] H. R. Lourenco, O. Martin, T. Stutzle, Iterated Local Search, Handbook of Metaheuristics, 2003.

[20] H. Frangouli, V. Harmandas, P. Stamatopoulos, Construction of optimum timebles for university course - a clp approach, Proc. of the Trird International Conference and Exhibition on Practical Applications of Prolog , (1995) 393–408.

[21] S. A. Cook, The complexity of theorem-proving procedures, in: STOC'71 Proceeding of the third annual ACM sysmposium on Theory of computing, 1971.

[22] B. N. Gomes, Uma abordagem evolucionÁria para o projeto de redes eixo-raio com alocaÇÃo simples, Master's thesis, Universidade Federal de Minas Gerais, Programa de Pós Graduação em Eng. Elétrica (2011).

[23] A. Schaerf, A survey of automated timetabling, Artificial Intelligence Review 13 (1999) 87–127.

[24] V. A. Bardadym, Computer-aided school and university timetabling: The new wave, Practice and theory of automated timetabling 1153 (1996) 22–45.

[25] T. Muller, Constraint-based timetabling, Ph.D. thesis, Charles University in Prague, Faculty of Mathematics and Pysics (2005).

[26] L. Gaspero, B. McCollum, A. Schaerf, The second international timetabling competition (itc - 2007): Curriculum-based course timetabling (track 3), Tech. rep. (2007).

[27] M. Carter, G. Laporte, Recent developments in practical course timetabling, (Burke, and Carter, 1998) (1998) 3–19.

[28] M. R. Garey, D. S. Johnson, A guide to npcompleteness, Computers and Intractability First ed. San Francisco: W. H. Freeman and Company, 1979.

[29] S. Even, A. Itai, A. Shamir, On the complexity of timetabling and multicommodity flow problems., SIAM Journal of Computation 5 (1976) 691–703.

[30] R. Lewis, Metaheuristics for university course timetabling, Doctoral Thesis,Napier University, Edinburgh, Scotland, 2006. (Disponível em: http: //www.cardiff.ac.uk/carbs/quant/rhyd/rhyd.html).

[31] J. Gu, P. W. Purdom, J. Franco, B. W. Wah, Algorithms for the satisfiability (sat) problem: a survey. satisfiability problem: Theory and applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (1996) 18–152.

[32] S. Daskalaki, T. Birbas, E. Housos, An integer programming formulation for a case study in university tiimetabling, European Journal of Operation Research 153 (2004) .

[33] M. Kubale, Graph Colorings, American Mathematical Society, ISBN 0-8218-3458-4, 2004.

[34] W. Junginger, Timetabling in germany - a survey, Interfaces 16 (1986) 66–74.

[35] D. B. Papoulias, The assignment -to-days problem in a school tiimetable, a heuristic approach, European Journal of Operation Research 4 (1980) 31–41.

[36] O. B. de Gans, A computer timetabling system for secondary schools in the netherlands, European Journal of Operation Research 7 (1981) 175–182.

[37] A. Hertz, Tabu search for large scale timetable problems., European Journal of Operation Research 54 (1991) 39–47.

[38] M. Cangalovic, A. M. a. Schreuder, Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of differents lengths, European Journal of Operation Research 51 (1991) 248–258.

[39] M. Carter, G. Laporte, S. Y. Lee, Examination timetabling: Algorithmic strategies and applications., Journal of the Operational Research Society 74 (1996) 373–383.

[40] L. Di Gaspero, A. Schaerf, Tabu search techniques for examination timetabling, in: E. Burke, W. Erben (Eds.), Practice and Theory of Automated Timetabling III, Vol. 2079 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2001, pp. 104–117.

[41] R. Lewis, A survey of metaheuristics-based technique for university timetabling problems, OR Spectrum 30 (2008) 167–190.

[42] D. Werra, Na introduction to timetabling, European Journal of Operational Research 19 (1985) 151–162.

[43] M. Yoshihawa, K. Kaneko, Y. Nomura, A constraint based approach to high school tiimetabling problems: a case study, Proceedings of the 12th National Conference on Artificial Intelligence (1994) 1111–1116.

[44] H. G. Santos, Formulações e algoritmos para o problema de programação de horários em escolas, Ph.D. thesis, Programade Pos-Graduação em Computação da Universidade Federal Fluminense (2007).

[45] E. L. M. Lobo, Uma solução do problema de horário escolar via algoritmo genético paralelo., Master's thesis, Centro Federal Tecnologico de Minas Gerais (2005).

[46] J. S. Appleby, D. V. Black, E. A. Newman, Technique for producting school timetables on a computer and their application to other scheduling problems, The Computer Journal 3 (1960) 237–245.

[47] S. Broder, Final examination scheduling, Comm. A. C. M. 7 (1964) 494–498.

[48] A. J. Cole, The preparation of examination timetables using a small-store computer, Comput. Journal 7 (1964) 117–121.

[49] W. K. Harold, The hungarian method for the assignment problem, Naval Research Logistic Quarterly 2 (1955) 83–97.

[50] J. Lions, Matrix reduction using hungarian method for the generation of scholl timebles, Comm. A. C. M. 9 (1966) 319–354.

[51] G. R. Sherman, A combinatorial problem arising from scheduling of university classes, Journal of the Tennessee Academy of Science 38 (1963) 115.

[52] W. Bossert, J. B. Harmon, Student sectioning on the ibm 7090, Tech. rep., IBM Corp., Cambridge, Mass (1963).

[53] E. D. Barraclough, The aplication of a digital computer to the construction of timebling, The Computer Journal 8 (1965) 136–146.

[54] G. Mihoc, E. Balas, The problem of optimal timetables, Roumaine Math 141 (1965) 381–388.

[55] M. Almond, An algorithm for constructing university tiimetable, The Computer Journal 8(4) (1966) 331–340.

[56] C. J. H. McDiarmid, The solution of a timetabling problem, J. Inst. Math 9 (1972) 23–34.

[57] D. Werra, Equitable colorations graphs, Fracaise Informat 239 (1971) 30.

[58] D. Werra, Construction of school timebles by flow methods, INFOR - Cand. J. Operational Res. and Information Processing 228 (1971) 499.

[59] T. B. Scott, Graph colouring with preassignment and unavailability constraints, Ars Combinatoria 2 (1976) 25–32.

[60] G. Schmidt, T. Strohlein, Timetablie construction - an annotated bibliography, The Computer Journal 23 (1980) 307–316.

[61] J. A. Bondy, U. S. R. Murty, Graph Theory with Application, London: Macmillan, 1976.

[62] B. Manvel, Colouring large graphs, in: Proceedings of the 1981 Southeast Conference on Graph Theory, Combinatorial and Computer Science, 1981.

[63] S. S. Mata, O problema de horario na escola de segundo grau, modelagem e implementação., Master's thesis, Dissertação de mestrado, Progam de Engenharia de Sistema e Computação, COPEE/UFRJ, Janiero (1989).

[64] R. Ostermann, D. Werra, Some experiments with a timetabling system, OR Spektrum 3 (1983) 199–204.

[65] A. H. Land, A. G. Doig, An automatic method of solving discrete programming problems, Econometrica 28 (1960) 497–520.

[66] R. Fahrion, G. Dollansky, Construction of university faculty timetabling using logic programming techiniques, Discrete Applied Mathematics 35 (1992) 221–236.

[67] J. C. Caldeira, A. C. Rosa, Scholl timebling using genetic search, Annals of PATAT (1997) 115–122.

[68] T. A. Feo, M. G. C. Resende, Greedy randomized adaptative search procedures, J. of Global Optimization 6 (1995) 109–133.

[69] N. Mladenovic, A variable neighborhood algorithm - a new metaheuristics for optimization combinatorial, Abstracts of Papers at Optimization Days , (1995) .

[70] A. Hertz, D. Werra, Using tabu search techniques for graph coloring., Computing 39 (1987) 345–351.

[71] Z. Lu, J. Hao, Adaptive tabu search for course timetabling, European Journal of Operational Research 200 (2010) 235–244.

[72] D. Costa, A tabu search algorithm for computing an operational tiimetable, European Journal of Operation Research 79 (1994) 98–110.

[73] A. Schaerf, Tabu search techniques for large high-school timetabling problems, Proceedings of the Thirteenth National Conference on Artificial Intelligence AAA1Press/ MIT Press (1996) 363–368.

[74] S. Kirkpatric, C. D. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[75] M. A. Saleh, P. Coddington, A comparison of annealing techiniques for academic course scheduling, Lecture Notes in Computer Science 1408 (1988) 92–114.

[76] M. Tuga, R. Berrettta, M. A., A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem, Computer and Information Science (2007) 400–405.

[77] T. Duong, K. Lam, Combining constraint programing and simulated annealing on university exam timetabling, in: Proceedings of RIVF Conference, Hanoi, 2004.

[78] H. S. C. Barbosa, S. de Souza, M. J. F. Souza, Programação de cursos universitários baseada em currículos via uma meta-heurística híbrida grasp-ils utilizando o procedimento kempeswap, Simpósio de Pesquisa Operacional & Logistica da Marinha (SPOLM 2011) 2011.

[79] M. J. F. Souza, L. S. Ochi, N. A. Maculan, Grasp-tabu search algorithmfor solving school timetabling problems., Metaheuristics: Computer Decision-Making, Kluwer Academic Publisher (2003) 659–672.

[80] J. H. Holland, Adaption in natural and artificial systems, University of Michigan Press , 1975.

[81] H. Ueda, D. Ouchi, k. Takahashi, T. Miyahara, A co-evolving timeslot/room assignment genetic algorithm technique for university timetabling. computer science, Practice and theory of automated timetabling III 2079 (2001) 48–63.

[82] E. Burke, D. Elliman, R. Weare, A genetic algorithm based university timetabling system, in: East-West Conference on Computer Technologies in Education, Crimea, Ukraine, 1994, pp. 35–40.

[83] D. Corne, P. Ross, H. Fang, Evolving timetables, The Pratical Handbook of Genetic Algorithms 1 (1995) 219–276.

[84] P. Ross, E. Hart, D. Corne, Genetic algorithms and timetabling, Springer-Verlag New York, Inc., New York, NY, USA, 2003, pp. 755–771.

[85] O. Ulker, E. Ozcan, E. E. Korkmaz, Linear linkage encoding in grouping problems: applications on graph coloring and timetabling, Practice and theory of automated timetabling III 3867 (2007) 347–363.

[86] R. Nabeel, Hybrid genetic algorithms with great deluge for course timetabling, International Journal of Computer Science and Network Security 10 (2010) 283–288.

[87] R. Qu, E. K. Burke, Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems, Journal of the Operational Research Society 60(9) (2009) 1273–1285.

[88] R. Bellio, G. L. Di, A. Schaerf, Design and statistical analysis of a hybrid local search algorithm for course timetabling, Journal of Scheduling (2011) 1–13.

[89] G. H. G. Fonceca, R. G. Ribeiro, F. V. C. Martins, Uma abordagem hÍbrida de sat e busca tabu para o problema da programaÇÃo de horÁrios escolares, in: XLIII Simpósio Brasileiro de Pesquisa Operacional, Ubatuba, São Paulo, 2011.

[90] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malk, Chaft: Engineering an efficient sat solver, Proc. 38th Design Automation Conference (DAC'01) , (2001) .

[91] M. J. Geiger, Applying the threshold accepting metaheuristic to curriculum based course timetabling. a contribution to the second international timetabling competition itc 2007, ANNALS OF OPERATIONS RESEARCH (2010) 1–14.

[92] T. Muller, Itc2007 solver description: a hybrid approach, Annals of Operations Research 127 (2009) 429–446.

[93] Z. Michalewicz, D. B. Fogel, How To Solve IT: Modern Heuristics, Springer, 2000.

[94] G. Dueck, New optimization heuristics: The great deluge algorithm and record to record travel, Journal of Computational Physics 105 (1993) 86–92.

[95] L. Di Gaspero, B. McCollum, A. Schaerf, Curriculum-based couse timetabling, Tech. rep. (2007).

[96] A. X. Martins, C. Duhamel, P. Mahey, R. Saldanha, M. C. Souza, ariable neighborhood descent with iterated local search for routing and wavelength assignment., Computers & Operations Research 39 (2012) 2133–2141.

[97] A. X. Martins, C. Duhamel, M. C. Souza, R. Saldanha, P. Mahey, A vnd-ils heuristic to solve the rwa problem, Lecture Notes in Computer Science 6701 (2012) 577–582.

[98] A. J. S. Neto, D. Vianna, M. F. D. Vianna, Heuristica grasp e ils para o planejamento de estocagem de contêineres em navios porta-contêineres, Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional . (2011) ,.

[99] I. M. COELHO, S. RIBAS, M. J. F. . SOUZA, Um método híbrido baseado em grasp, ils e vnd para o planejamento operacional de lavra em minas a céu aberto., in: Anais do XVI Seminário de Iniciação Científica da UFOP, 2008.

[100] M. M. Silva, A. Subramanian, L. S. Ochi, Uma heurística baseada em grasp e ils para o problema da mínima latência, in: Anais do XLIII Simpósio de Pesquisa Operacional, 2011.

[101] P. L. A. Munhoz, M. H. P. Perché, M. J. F. Souza, Um algoritmo baseado em grasp, iterated local search, descida em vizinhança variável e reconexão por caminhos para o problema de sequenciamento em uma máquina com penalidades por antecipação e atraso, in: Anais do XI EMC. Volta Redonda (RJ) : Escola de Engenharia Industrial Metalúrgica de Volta Redonda/UFF,, 2008.

[102] R. Storn, K. Price, Diferential evolution - a simple e efficient adaptative schema global optimzation over continuos spaces, Technical Report TR-95-012, ICSI , (1995) .

[103] A. Qing, Differential Evolution Fundamentals and Applications in Electrical Engineering, wiley, 2009.

[104] F. Glover, Tabu search and adaptive memoru programing advances, applications and challenges, Interfaces in Computer Science and Operations Research (1996) 1–75.

[105] M. Atsuta, K. Nonobe, T. Ibaraki, An approach using general csp solver., Disponível em: http://www.cs.qub.ac.uk/itc2007.

[106] M. Clark, M. Henz, B. Love, Quikfix a repair-based timetable solver, The 7th International Conference on the Practice and Theory of Automated Timetabling , (2007) .

[107] K. C. Laudon, J. P. Laudon, Sistemas de Informação Gerenciais., 2007.

[108] J. Battisti, SQL Server 2000. Administração e Desenvolvimento., Axcell Books, 2001.

[109] L. S. Indrusiak, Linguagem java, Tech. rep., UFPE (1996).

[110] M. Fowler, C. Kobryn, Uml Essencial, Bookman, 2005.

[111] A. M. R. da Silva, C. A. E. Videira, UML Metodologias e Ferramentas Case, Centro Atlantico, 2001.