UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

GERALDO SIMONETTI BELLO

ABORDAGEM DO PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA SUJEITO A RESTRIÇÕES UTILIZANDO COLORAÇÃO DE GRAFOS

VITÓRIA
2007

GERALDO SIMONETTI BELLO

ABORDAGEM DO PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA SUJEITO A RESTRIÇÕES UTILIZANDO COLORAÇÃO DE GRAFOS

Dissertação apresentada ao Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Mestre.
Orientadoras:
Profª Drª Maria Claudia Silva Boeres
Profª Drª Maria Cristina Rangel

VITÓRIA
2007

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Bello, Geraldo Simonetti, 1963-

B417a Abordagem do problema de programação de grade horária sujeito a restrições utilizando coloração de grafos / Geraldo Simonetti Bello. – 2007.

136 f. : il.

Orientadora: Maria Claudia Silva Boeres.

Co-orientadora: Maria Cristina Rangel.

Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Otimização combinatória. 2. Programação de grade horária. 3. Coloração de grafos. 4. Busca Tabu. I. Boeres, Maria Claudia Silva. II. Rangel, Maria Cristina. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

Abordagem do Problema de Programação de Grade Horária sujeito a Restrições utilizando Coloração de Grafos

Geraldo Simonetti Bello

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 12/11/07 por:

Profª. Drª. Maria Cláudia Silva Boeres - DI/UFES

Profª. Drª. Maria Cristina Rangel - DI/UFES

Prof. Dr. Elias Silva de Oliveira - DI/UFES

Prof. Dr. Luiz Satoru Ochi - IC/UFF

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória, Novembro de 2007

Agradeço a Deus pelo dom da existência e por me guiar na busca do progresso intelectual e moral.

Agradeço aos meus pais Leovigildo e Glória e aos meus saudosos avós Floriano, Nadyr, Talma e Celina, pelo amor incondicional e pelos exemplos com que me oriento nos desafios da vida.

Agradeço à minha esposa Carla e à minha filha Bianca pelo carinho, compreensão, encorajamento, paciência e apoio em todos os momentos.

Agradeço às professoras Dra. Maria Claudia Silva Boeres e Dra. Maria Cristina Rangel, pela orientação, acompanhamento, incentivo e confiança na realização deste trabalho.

“Nunca ande pelo caminho traçado, pois ele conduz somente até aonde os outros foram.” (Alexander Graham Bell)

“O futuro pertence àqueles que acreditam na beleza de seus sonhos.” (Anna Eleanor Roosevelt)

RESUMO

Este trabalho aborda os problemas de Programação de Grade Horária Escolar e de Coloração de Grafos, apresenta suas características, descreve as principais abordagens propostas na literatura para solução destes dois problemas, incluindo a que reformula a versão básica do problema de Programação de Grade Horária Escolar como um problema de Coloração de Grafos e utiliza os algoritmos correspondentes a esta abordagem para a sua resolução. Um problema de Programação de Grade Horária Escolar com restrições adicionais àquelas encontradas na versão básica é escolhido na literatura e é desenvolvido um modelo que estende a correspondência entre os dois problemas, contemplando também restrições adicionais. São propostas adaptações em um algoritmo Busca Tabu para Coloração de Grafos descrito na literatura e em uma implementação existente deste algoritmo em linguagem C, de forma a permitir a resolução do problema com restrições adicionais considerado. O problema escolhido é reformulado como problema de Coloração de Grafos e o algoritmo modificado é utilizado para a sua resolução. São apresentados os resultados computacionais para um conjunto de instâncias associado ao problema e é verificado o efeito desta reformulação nos resultados, comparando-os com os existentes na literatura, que não empregam esta reformulação.

Palavras-chave: Otimização Combinatória, Programação de Grade Horária, Coloração de Grafos, Busca Tabu

ABSTRACT

This work addresses the Class/Teacher Timetabling Problem and Graph Coloring Problem, presents features, describes the main approaches proposed in the literature for the solution of these two problems, including the reformulation of the basic version of the Class/Teacher Timetabling Problem as a Graph Coloring Problem, and uses the algorithms on this approach for their resolution. A Class/Teacher Timetabling Problem with additional constraints to those found in the basic version is chosen in the literature and a model that extends the correspondence between the two problems is developed, also contemplating additional constraints. Adaptations in a Tabu Search algorithm for Graph Coloring described in the literature and in the existing implementation in C language are proposed, in order to allow the resolution of the problem with the additional constraints considered. The chosen problem is reformulated as a Graph Coloring Problem and the modified algorithm is used for its resolution. The computational results for a set of instances related to the problem are presented and the effect of this reformulation in the results is verified, comparing them with those in the literature, which does not use this reformulation.

Keywords: Combinatorial Optimization, Timetabling, Graph Coloring, Tabu Search

LISTA DE FIGURAS

Figura 1 - Ilustração elaborada por Euler sobre o problema das pontes de Königsberg..... 23

Figura 2 - Exemplo de grafo com 7 vértices e 10 arestas..... 24

Figura 3 - Exemplo de grafo 3-partido..... 25

Figura 4 - Exemplo de digrafo..... 26

Figura 5 - Multigrafo representando o problema das pontes de Königsberg..... 27

Figura 6 - Grafo desconexo..... 28

Figura 7 - Uma rede..... 29

Figura 8 - Coloração de vértices de um grafo G com c (G) = 3..... 31

Figura 9 - 4-coloração de arestas de um grafo *G*..... 31

Figura 10 - Principais Problemas de Programação de Grade Horária..... 33

Figura 11 - Estrutura Geral de uma Hiper-Heurística..... 39

Figura 12 - Principais abordagens para modelagem e resolução do problema de Programação de Grade Horária Educacional encontrados na literatura..... 40

Figura 13 - Exemplo da aplicação do algoritmo genético ao problema de Programação de Grade Horária Educacional..... 49

Figura 14 - Pesquisa efetuada pelos algoritmos meméticos sobre o subespaço de ótimos locais, através da utilização de métodos de busca local..... 51

Figura 15 - (a) Formigas exploram os caminhos até o alimento - (b) A maior parte das formigas escolhe o caminho mais -curto..... 52

Figura 16 - Exemplo de Formação em V de um bando de pássaros..... 54

Figura 17 - Algoritmo de Otimização por Nuvem de Partículas..... 55

Figura 18 - Movimento de uma partícula no espaço de soluções..... 56

Figura 19 - Programação por Restrições e Problema de Satisfação de Restrições..... 60

Figura 20 - Programação em Lógica..... 62

Figura 21 - Programação em Lógica com Restrições..... 63

Figura 22 - (a) Neurônio - (b) Rede neural..... 65

Figura 23 - Representação simplificada de uma rede neural..... 66

Figura 24 - Função sigmóide $Y_i = f_i(X_i) = \frac{1}{2}\left(1 + \tanh\left(\frac{X_i}{T}\right)\right)$, para diferentes valores do parâmetro T..... 68

Figura 25 - Modelo esquemático de um neurônio artificial..... 68

Figura 26 - Mapa colorido com 4 cores - Mapa de Divisões Territoriais do Brasil..... 71

Figura 27 - Uma 4-coloração do grafo planar correspondente ao mapa da Figura 26..... 72

Figura 28 - (a) Grafo de 27 vértices e 51 arestas – (b) Listagem de arquivo contendo a representação deste grafo no formato *Dimacs*..... 76

Figura 29 - Algoritmo Guloso Simples..... 78

Figura 30 - Trecho do algoritmo *RLF*..... 83

Figura 31 - O algoritmo *Dsatur*..... 83

Figura 32 - O algoritmo Busca Tabu..... 87

Figura 33 - O algoritmo *Tabucol*..... 91

Figura 34 - Versão do algoritmo *Tabucol* implementada por Joseph Culberson..... 93

Figura 35 - Exemplo de grade horária e matriz Q correspondente..... 104

Figura 36 - (a) Grafo adjunto G - (b) Grafo $G^{"}$ correspondente..... 111

Figura 37 - Processo proposto para resolução do PPGHE..... 115

LISTA DE QUADROS

Quadro 1 – Variações de Algoritmos Gulosos..... 81

Quadro 2 – Quadro comparativo de duas versões do algoritmo *Tabucol*..... 94

Quadro 3 – Exemplos de instâncias de problemas de Programação de Grade Horária Educacional publicamente disponíveis..... 101

Quadro 4 – Descrição das instâncias utilizadas..... 117

Quadro 5 – Subconjuntos de valores dos parâmetros do algoritmo *Tabucol* modificado..... 120

LISTA DE SIGLAS E ACRÔNIMOS

ACO	*Ant Colony Optimization*
AS	*Ant System*
BFS	*Breadth-First Search*
BSC	*Backtracking Sequential Coloring*
CBR	*Case-Based Reasoning*
CLP	*Constraint Logic Programming*
Comut	Programa de Comutação Bibliográfica
CP	*Constraint Programming*
CSP	*Constraint Satisfaction Problem*
Dimacs	*Center for Discrete Mathematics and Theoretical Computer Science*
Dsatur	*Saturation Degree*
GA	*Genetic Algorithms*
CentOS	*Community Enterprise Operating System*
GCC	*GNU Compiler Collection*
GCP	*Graph Coloring Problem*
GRASP	*Greedy Randomized Adaptive Search Procedure*
GTS	*GRASP-Tabu Search*
ILS	*Iterated Local Search*
LBFS	*Lexicographic Breadth-First Search*
LF	*Largest First*
LP	*Logic Programming*
MMAS	*MAX-MIN Ant System*
NP	*Non-Deterministic Polynomial*

PPGHE	Problema de Programação de Grade Horária Escolar
Prolog	*Programmation en Logique*
PSO	*Particle Swarm Optimization*
RAM	*Random Access Memory*
RLF	*Recursive Largest First*
SA	*Simulated Annealing*
Sch	*School*
SF	*Smallest First*
Tabucol	*Tabu Search for Graph Coloring*
TS	*Tabu Search*
TSP	*Traveling Salesman Problem*
XRLF	*Extended Recursive Largest First*

SUMÁRIO

1 **INTRODUÇÃO**.....16

2 **CONCEITOS BÁSICOS – COMPLEXIDADE COMPUTACIONAL E TEORIA DOS GRAFOS E REDES**.....20
2.1 TEORIA DA COMPLEXIDADE COMPUTACIONAL.....20
2.1.1 **Complexidade de Algoritmos**.....20
2.1.2 **Complexidade de Problemas**.....20
2.2 TEORIA DOS GRAFOS.....22
2.2.1 **Histórico**.....22
2.2.2 **Conceitos de Grafos**.....23
2.2.3 **Conceitos de Fluxo em Redes**.....29
2.2.4 **Conceitos de Coloração de Grafos**.....30

3 **PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA**.....33
3.1 DEFINIÇÃO E CLASSIFICAÇÃO.....33
3.2 PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA EDUCACIONAL (*EDUCATIONAL TIMETABLING*).....34
3.3 ABORDAGENS PARA RESOLUÇÃO DO PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA EDUCACIONAL.....37
3.3.1 **Conceitos..**.....37
3.3.2 **Programação Linear Inteira**.....41
3.3.3 **Fluxo em Redes (*Network Flow*)**.....42
3.3.4 **Coloração de Grafos**.....43
3.3.5 **Subida de Encosta (*Hill Climbing*)**.....44
3.3.6 **Recozimento Simulado ou Têmpera Simulada (*Simulated Annealing - SA*)**.....44
3.3.7 ***Great Deluge Algorithm***.....45
3.3.8 **Busca Tabu (*Tabu Search-TS*)**.....46
3.3.9 **Procedimento de Busca Adaptativa, Aleatória e Gulosa (*Greedy Randomized Adaptive Search Procedure - GRASP*)**.....46
3.3.10 **Algoritmos Genéticos (*Genetic Algorithms-GA*)**.....47
3.3.11 **Algoritmos Meméticos**.....50
3.3.12 **Otimização por Colônia de Formigas (*Ant Colony Optimization-ACO*)....**51
3.3.13 **Otimização por Nuvem de Partículas (*Particle Swarm Optimization-PSO*)**54
3.3.14 ***Opinion Algorithm***.....58
3.3.15 **Problema de Satisfação de Restrições (*Constraint Satisfaction Problem - CSP*)**.....59
3.3.16 **Programação em Lógica (*Logic Programming-LP*)**.....61
3.3.17 **Programação em Lógica com Restrições (*Constraint Logic Programming - CLP*)**.....62
3.3.18 **Raciocínio Baseado em Casos (*Case-Based Reasoning – CBR*)**.....63
3.3.19 **Lógica Difusa (*Fuzzy Logic*)**.....64
3.3.20 **Redes Neurais**.....65
3.3.21 **Hiper-Heurísticas**.....69

4 **COLORAÇÃO DE GRAFOS**.....71
4.1 HISTÓRICO, FORMULAÇÕES E APLICAÇÕES.....71
4.2 O FORMATO *Dimacs* PARA ARQUIVOS CONTENDO GRAFOS..... 73
4.3 ABORDAGENS PARA RESOLUÇÃO DO PROBLEMA DE COLORAÇÃO DE GRAFOS.....77
4.3.1 **Coloração Seqüencial – Algoritmo Guloso (*Greedy Algorithm*)**.....77
4.3.2 **Algoritmo Guloso Iterativo (*Iterated Greedy Algorithm*)**.....82
4.3.3 **Algoritmo Maior Primeiro Recursivo (*Recursive Largest First-RLF*)**.....82
4.3.4 **Algoritmo *Dsatur (Saturation Degree)***..... 83
4.3.5 **Algoritmo *Dsatur* com Retrocesso (*Backtrack Dsatur*)**.....84
4.3.6 **Busca Local Iterativa (*Iterated Local Search-ILS*)**..... 84
4.3.7 **Outras abordagens para coloração de grafos**.....84
4.4 RESOLUÇÃO DO PROBLEMA DE COLORAÇÃO DE GRAFOS UTILIZANDO BUSCA TABU.....86

5 **REFORMULAÇÃO DE UM PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA ESCOLAR COM RESTRIÇÕES RÍGIDAS E FLEXÍVEIS**.....95
5.1 O PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA ESCOLAR COMO PROBLEMA DE COLORAÇÃO DE GRAFOS..... 95
5.2 REFORMULAÇÃO DE UM PPGHE COM RESTRIÇÕES RÍGIDAS E FLEXÍVEIS A PARTIR DO PROBLEMA DE COLORAÇÃO DE GRAFOS..... 100
5.2.1 **Escolha e Descrição do PPGHE**.....101
5.2.2 **Reformulação do PPGHE como um problema de Coloração de Grafos**.....108
5.2.3 **Adaptação do algoritmo *Tabucol* às restrições adicionais do PPGHE**..... 111

6 **RESULTADOS COMPUTACIONAIS**.....117

7 **CONCLUSÕES E TRABALHOS FUTUROS**.....123

8 **REFERÊNCIAS**.....126

1 INTRODUÇÃO

Coloração de grafos, conhecido problema da literatura de teoria dos grafos, é utilizado na modelagem de diversas aplicações. A forma geral destas aplicações envolve a construção de um grafo onde os vértices representam os itens de interesse e cada aresta conecta dois itens incompatíveis. O problema de otimização corresponde a buscar o conjunto mínimo de cores, de forma que cada item receba uma cor e itens incompatíveis possuam cores diferentes.

Este problema é classificado como NP-Difícil (*NP-Hard*) (GAREY; JOHNSON, 1979) para grafos de ordem genérica. Portanto, é improvável que seja encontrado um algoritmo que resolva estes problemas em tempo polinomial. Por isso, existem diversos algoritmos que usam heurísticas para coloração de grafos, sem entretanto, garantir uma solução ótima no caso geral.

Uma das aplicações na qual a abordagem da coloração de grafos pode ser empregada é em problemas de Programação de Grade Horária Escolar (*Class/Teacher Timetabling Problem*). Esta classe de problemas consiste em alocar recursos em intervalos de tempo, procurando obter a configuração mais otimizada, porém satisfazendo determinadas restrições. Por exemplo, ao alocar professores de disciplinas às respectivas turmas em uma instituição de ensino, duas disciplinas ministradas por um mesmo professor não podem ser programadas para o mesmo período de tempo. Da mesma forma, duas disciplinas requeridas pela mesma turma de alunos também não podem se sobrepor na grade horária (*timetable*), entre outras restrições

O problema de Programação de Grade Horária Escolar é de grande importância para as instituições de ensino, uma vez que o processo ocorre periodicamente e interfere no cotidiano de todos os professores e alunos da instituição.

O problema de otimização correspondente é classificado como NP-Difícil (EVEN; ITAI; SHAMIR, 1976), na maior parte de suas versões e por isso é geralmente tratado por meio de métodos heurísticos.

Neufeld e Tartar (1974) abordaram o problema de Programação de Grade Horária Escolar utilizando Coloração de Grafos, contemplando a versão básica desse problema, cujas únicas restrições são: a não-ocorrência de conflitos de horários (nenhum aluno ou professor pode ser alocado para duas aulas no mesmo horário), eventos em horários pré-fixados e indisponibilidade de determinados horários. Entretanto, instituições de ensino podem demandar outros tipos de restrições, como limites diários de aulas, alocação de duas aulas em períodos consecutivos, concentração das aulas no menor número possível de dias e evitando a ocorrência de lacunas (gaps), isto é, horários sem atividade entre dois horários de aula de um mesmo dia (Souza, 2000). Restrições adicionais deste tipo geralmente não são contempladas na abordagem tradicional do problema através da Coloração de Grafos.

O objetivo principal desta dissertação é reformular o problema de Programação de Grade Horária Escolar com restrições rígidas (devem ser satisfeitas) e flexíveis (se satisfeitas, melhoram a solução) como um problema de Coloração de Grafos e comparar os resultados apresentados nas soluções de um mesmo conjunto de instâncias do problema de Programação de Grade Horária Escolar, utilizado por Souza (2000), Souza, Maculan e Ochi (2003) e Santos, Ochi e Souza (2007), primeiro sem formulá-lo como coloração de grafos e depois considerando esta formulação e utilizando algoritmos que empregam heurísticas para coloração de grafos.

A dissertação também apresenta um panorama das principais abordagens propostas para solução dos problemas de Programação de Grade Horária Educacional e de Coloração de Grafos e das principais instâncias disponíveis.

Para o desenvolvimento desta dissertação, foram realizadas duas linhas de pesquisa bibliográfica, uma sobre os problemas de Programação de Grade Horária e outra sobre o problema de Coloração de Grafos, com o propósito de aprofundar conceitos, caracterizar os problemas, identificar diferenças entre os vários tipos e conhecer as abordagens propostas para solução e as instâncias disponíveis.

Na definição de alguns termos e respectivas siglas ao longo da dissertação, foi acrescentado o correspondente em língua inglesa entre parênteses, com o propósito de facilitar a consulta a outros trabalhos relacionados ao tema, escritos neste idioma. Além disso, determinados termos e siglas são mencionados na dissertação preferencialmente neste idioma, por serem casos consagrados na literatura.

Outro aspecto com relação à escrita, que representa uma contribuição desta dissertação, refere-se à grafia da palavra meta-heurística. Durante a realização das pesquisas bibliográficas citadas, foi observado que não existe um consenso quanto à forma de escrever este vocábulo composto, sendo identificadas na literatura pelo menos três grafias diferentes: meta-heurística, metaheurística e metaeurística. Com o propósito de esclarecer a questão, foi efetuada consulta à Academia Brasileira de Letras (ACADEMIA, 2007). A recomendação, em resumo, é que seja utilizada a grafia meta-heurística, a fim de que a compreensão prevaleça sobre as convenções gráficas, ainda que o prefixo *meta-* não conste na lista relativa ao uso do hífen que figura no Formulário Ortográfico de 1943.

O trabalho é dividido em sete capítulos, descritos a seguir:

- O Capítulo 1 consiste nesta introdução.

- O Capítulo 2 apresenta uma visão geral dos conceitos da Teoria de Complexidade Computacional, abrangendo algoritmos e problemas, e da Teoria dos Grafos e Redes, necessários ao desenvolvimento do trabalho.

- O Capítulo 3 aborda os problemas de Programação de Grade Horária e, mais detalhadamente, os relativos à Grade Horária Educacional, para os quais apresenta definições, classificações (incluindo o tipo Escolar, utilizado nessa dissertação), conceitos e principais abordagens para resolução obtidas na pesquisa bibliográfica.

- O Capítulo 4 aborda o problema de Coloração de Grafos, descrevendo sua origem no estudo do problema das quatro cores para mapas planares, apresentando suas formulações como problema de decisão e problema de

otimização, e como problema de atribuição e problema de partição de conjuntos, relacionando algumas aplicações, apresentando as principais abordagens para resolução obtidas na pesquisa bibliográfica e, mais detalhadamente, a resolução do problema utilizando a meta-heurística Busca Tabu, bem como a versão original do algoritmo que foi modificado para utilização nesta dissertação.

- O Capítulo 5 aborda a reformulação do problema de Programação de Grade Horária Escolar com restrições rígidas e flexíveis para o problema de Coloração de Grafos. É apresentada a resolução do problema reformulado utilizando Busca Tabu. Além disso, são apresentadas as alterações efetuadas no algoritmo Busca Tabu descrito no Capítulo 4.

- O Capítulo 6 descreve as instâncias utilizadas e apresenta os resultados computacionais obtidos com o algoritmo modificado, empregando a meta-heurística Busca Tabu para coloração de grafos, comparado com os resultados utilizando Busca Tabu sem Coloração de Grafos, obtidos por Souza, Maculan e Ochi (2003) e Santos, Ochi e Souza (2007), para as mesmas instâncias.

- O Capítulo 7 apresenta as conclusões sobre o trabalho desenvolvido e indica perspectivas de trabalhos futuros.

2 CONCEITOS BÁSICOS – COMPLEXIDADE COMPUTACIONAL E TEORIA DOS GRAFOS E REDES

2.1 TEORIA DA COMPLEXIDADE COMPUTACIONAL

Os conceitos a seguir foram baseados em Hastad (1998).

A teoria da complexidade computacional se divide em complexidade de algoritmos e complexidade de problemas.

2.1.1 Complexidade de Algoritmos

O tamanho de uma instância de entrada de um problema é definido como o número de símbolos necessários para representar a instância.

A complexidade de um algoritmo define, para cada tamanho n de instância de entrada, a quantidade máxima de iterações necessárias para resolver a instância. Entretanto, ao invés de expressar a função exata do número de iterações necessárias *f(n)*, é utilizada a sua ordem de grandeza *g(n)*, tal que $\lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = k$, onde k é uma constante. Ou seja, considera-se a menor função limitante superior *g(n)* para caracterizar a ordem de grandeza da complexidade do algoritmo, e diz-se que a complexidade do algoritmo é *O(g(n))*.

Quando um algoritmo encontra a solução de um problema em até $k.n^c$ iterações, a partir de uma instância de entrada do problema de tamanho n, onde k e c são constantes, diz-se que a complexidade do algoritmo é $O(n^c)$.

2.1.2 Complexidade de Problemas

Problemas de decisão são problemas cuja solução é uma resposta “sim” ou “não”.

Problemas de otimização consistem em obter uma solução viável que otimize (maximize ou minimize) uma função objetivo.

A teoria da complexidade é baseada nos problemas de decisão. Entretanto, os problemas de otimização também podem ser formulados como problemas de decisão.

Quando um algoritmo encontra a solução de um problema em até $k.n^c$ iterações, a partir de uma instância de entrada do problema de tamanho n, onde k e c são constantes, diz-se que o problema é resolvido em tempo polinomial.

A classe P (*Polynomial*) abrange a classe de problemas de decisão que podem ser resolvidos por algoritmos determinísticos em tempo polinomial.

A classe NP (*Non-deterministic Polynomial*) abrange todos os problemas de decisão para os quais uma proposta de solução para uma instância, correspondendo à resposta "sim", pode ser verificada em tempo polinomial por um algoritmo, ou seja, se existe um algoritmo capaz de verificar em tempo polinomial se uma possível solução é realmente uma solução.

A classe P é um subconjunto da classe NP ($P \subset NP$). Até o momento, não é conhecido se NP = P, ou seja, estabelecer se um problema de decisão, cuja resposta afirmativa pode ser verificada em tempo polinomial, pode ou não também ser resolvido em tempo polinomial é uma questão ainda não solucionada.

Uma redução de um problema S_1 em um problema S_2 é um mapeamento das instâncias i_1 do problema S_1 para instâncias i_2 do problema S_2, de maneira que a solução do problema S_2 para a instância i_2 seja também a solução do problema S_1 para a instância i_1.

Um problema de decisão pertencente à classe NP é NP-completo se todos os outros problemas da classe NP podem ser reduzidos a este problema em tempo polinomial. Portanto, os problemas da classe NP-completo são os problemas de decisão de maior complexidade da classe NP.

Muitos problemas de otimização combinatória têm um problema de decisão correspondente que é NP-Completo. Estes problemas de otimização são denominados NP-difíceis (NP-*hard*).

Na Seção 4.1 o problema de Coloração de Grafos é definido como problema de decisão e como problema de otimização.

2.2 TEORIA DOS GRAFOS

2.2.1 Histórico

Conforme Alexanderson (2006), o surgimento da teoria dos grafos é atribuído à Leonhard Euler, através da resolução (EULER, 1735, apud ALEXANDERSON, 2006) do problema das pontes de Königsberg (*Königsberg Bridge Problem*), cidade européia atravessada por sete pontes, interligando duas ilhas entre si e com as margens de um rio (Figura 1).

O problema consiste em identificar se é possível ou não determinar um percurso através da cidade, atravessando todas as sete pontes exatamente uma vez e retornando ao ponto de partida. Euler demonstrou ser impossível realizar este percurso para o exemplo tratado e estabeleceu as condições necessárias para a determinação de percursos fechados, onde todas as conexões ligando os pontos sejam utilizadas exatamente uma vez (denominados mais tarde ciclos Eulerianos). As noções topológicas utilizadas por Euler no tratamento dos pontos da cidade conectados pelas pontes, constituíram as bases que fundamentaram o desenvolvimento da teoria dos grafos.

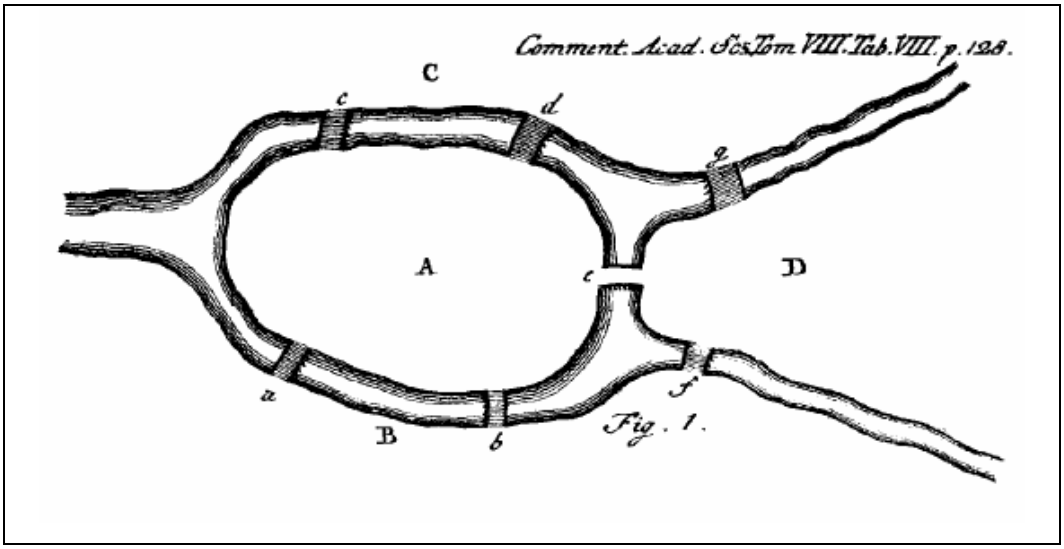


Figura 1 – Ilustração elaborada por Euler sobre o problema das pontes de Königsberg [EULER, 1735, apud ALEXANDERSON, 2006]

Outro problema estudado por Euler (1766, apud ALEXANDERSON, 2006), o Percurso do Cavalo (Knight's Tour), consiste em percorrer o tabuleiro do jogo de xadrez aplicando o movimento da peça correspondente ao cavalo, de forma que cada quadrado do tabuleiro seja visitado exatamente uma vez. Este problema foi precursor dos ciclos Hamiltonianos, outro importante conceito da teoria dos grafos, que foi tratado posteriormente por William Rowan Hamilton em 1856, ao estudar caminhos ligando locais de forma que todos os locais sejam visitados exatamente uma vez, e que ainda é objeto de pesquisas nos dias atuais.

2.2.2 Conceitos de Grafos

Os conceitos a seguir são baseados em Papadimitriou e Steiglitz (1998), Diestel (2005) e Boaventura Netto (2003).

Um grafo é definido como um par $G = (V,E)$ de conjuntos disjuntos, onde V é um conjunto finito e não-vazio, cujos elementos são denominados vértices ou nós e E é um conjunto não-vazio cujos elementos, denominados arestas ou linhas, são subconjuntos de V com cardinalidade dois.

Para a representação de um grafo, normalmente é desenhada uma circunferência para cada vértice, além de uma linha ligando duas destas circunferências, caso os dois vértices correspondentes formem uma aresta.

A título de exemplo, a Figura 2 representa o grafo G = ({1, 2, 3, 4, 5, 6, 7}, {{1,5}, {2,3}, {2,5}, {2,6}, {3,4}, {3,5}, {4,6}, {4,7}, {5,6}, {6,7}).

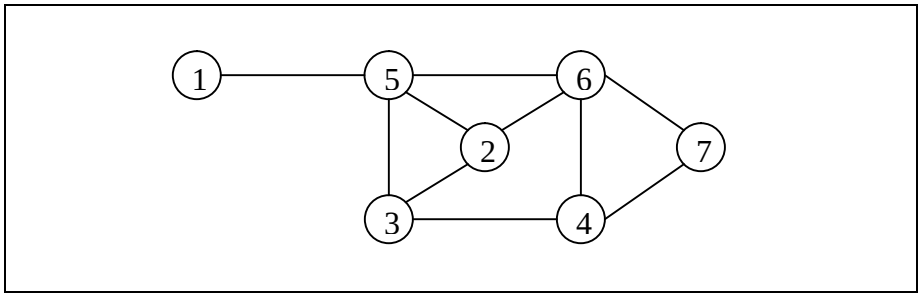


Figura 2 – Exemplo de grafo com 7 vértices e 10 arestas

O número de vértices de um grafo G é denominado ordem e representado por $|V|$. O grafo da Figura 2 possui ordem 7 ($|V| = 7$).

Dois vértices ligados por uma aresta são denominados vértices adjacentes. A aresta é dita incidente a ambos os vértices e estes são denominados terminais da aresta. Por exemplo, no grafo G da Figura 2, os vértices 1 e 5 são adjacentes e a aresta {1,5} é incidente aos vértices 1 e 5, os quais são terminais desta aresta.

Um grafo G é denominado completo se todos os vértices de G são adjacentes entre si, ou seja, cada vértice é adjacente a cada um dos demais vértices.

Seja $V = \{v_1,...,v_n\}$ um conjunto de n elementos. O conjunto de conjuntos $\{Y_1,...,Y_k\}$ define uma k-partição de V se os conjuntos Y_i, $i = 1,...,k$, são não-vazios, disjuntos e sua união é V, ou seja: $Y_i \neq f$, $i = 1,...,k$; $\bigcap_{i=1}^{k} Y_i = f$; $\bigcup_{i=1}^{k} Y_i = V$.

Um conjunto de vértices $Y \subset V$ de um grafo $G = (V, E)$ é denominado conjunto independente (*independent set*) ou conjunto estável (*stable set*) se nenhum par de seus elementos é adjacente, ou seja, $\forall u, v \in Y, \{u, v\} \notin E$.

Um grafo *G* é denominado *k*-partido se existir uma partição do seu conjunto de vértices em k subconjuntos Y_i, $i = 1,...,k$, tal que não existam arestas ligando vértices de um mesmo Y_i, ou seja, todas as arestas de *G* são da forma { *p*, *q* } tais que $p \in Y_i$ e $q \in Y_j$, $i \neq j$. Portanto, em um grafo k-partido todos os subconjuntos Y_i, $i = 1,...,k$, são conjuntos independentes. Quando *k*=2 o grafo *G* é denominado bipartido. A Figura 3 mostra um exemplo de grafo 3-partido, destacando os 3 conjuntos independentes através de linhas pontilhadas.

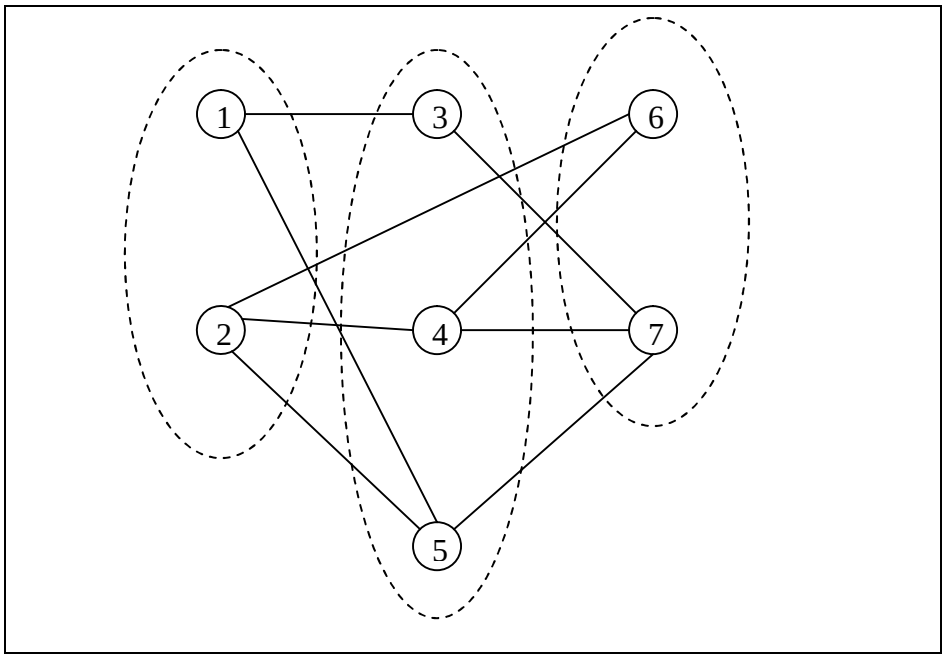


Figura 3 – Exemplo de grafo 3-partido

Um grafo *G* é planar caso possa ser representado em um plano de forma que não haja interseção entre duas arestas quaisquer, exceto em um vértice comum. O grafo mostrado na Figura 3 é planar. Qualquer grafo completo com mais de quatro vértices não é planar.

Um subgrafo *G' = (V',E')* de um grafo G é um grafo tal que *V'*$\subseteq$ *V* e *E'*$\subseteq$ *E*.

Por exemplo, considerando o grafo *G* da Figura 2, se for definido o grafo *G'* tal que *V'*={2,5,6} e *E'*={{2,5},{2,6},{5,6}}, então *G'* é um subgrafo de *G*.

Uma clique é um subgrafo completo de um grafo *G*. O subgrafo *G'* do exemplo anterior é um grafo completo. Logo, *G'* é uma clique.

Um hipergrafo é definido como um par *G = (V,E)* de conjuntos disjuntos, onde *V* é um conjunto finito e não-vazio e *E* é um conjunto cujos elementos são subconjuntos não-vazios de *V* com qualquer cardinalidade. Logo, grafos são casos especiais dos hipergrafos.

Um digrafo, (ou grafo direcionado ou grafo orientado) é definido como um par *G = (V,A)* de conjuntos disjuntos, tal que $A \subseteq V^2$. V é um conjunto finito e não-vazio, cujos elementos são denominados nós ou vértices e A é um conjunto cujos elementos, denominados arcos, são subconjuntos não-vazios de pares ordenados dos elementos de *V*.

A Figura 4 apresenta um exemplo de digrafo.

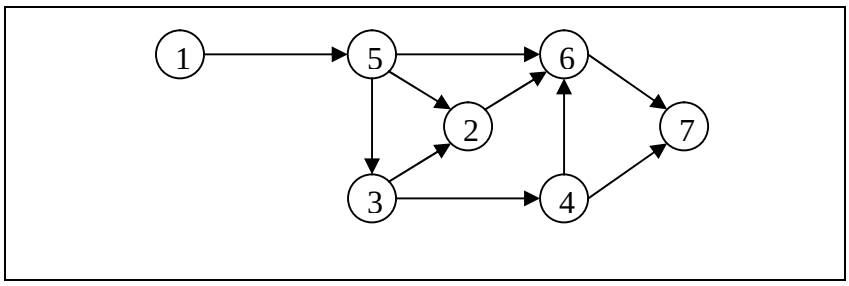


Figura 4 – Exemplo de digrafo

Um laço (*loop*) é uma aresta de *E* cujas extremidades são incidentes a um mesmo vértice.

Duas ou mais arestas incidentes a um mesmo par de vértices são denominadas arestas múltiplas ou paralelas.

Multigrafo é a denominação dada ao grafo que possui arestas múltiplas (PAPADIMITRIOU; STEIGLITZ, 1998). Alguns autores, tais como Diestel (2005), definem que multigrafos também podem ter laços.

Na Figura 5 é apresentado o multigrafo correspondente ao problema das pontes de Königsberg, tratado na Subseção 2.2.1.

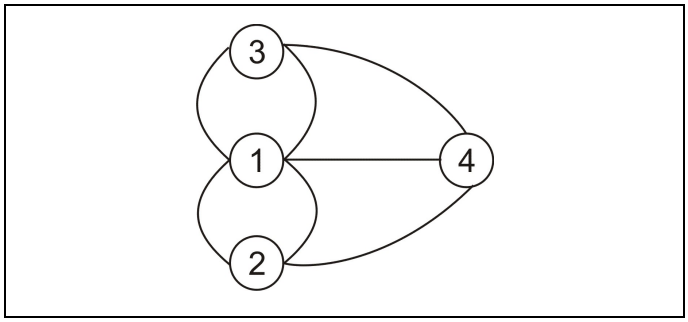


Figura 5 - Multigrafo representando o problema das pontes de Königsberg

Conforme Diestel (2005), alguns autores utilizam o multigrafo denominando-o de grafo e definem como grafo simples o grafo que não possui laços e não possui arestas múltiplas.

No decorrer deste trabalho, sempre que o termo grafo for utilizado, refere-se ao grafo simples e não orientado.

O grau de um vértice é o número de arestas incidentes àquele vértice, que é igual ao número de vértices a ele adjacentes.

Por exemplo, no grafo da Figura 2, o grau do vértice 1 é igual a 1, o grau do vértice 7 é igual a 2, os graus dos vértices 2, 3 e 4 são iguais a 3 e os graus dos vértices 5 e 6 são iguais a 4.

Em um digrafo, o grau de entrada de um vértice (*indegree*) é o número de arcos com a extremidade final neste vértice e o grau de saída (*outdegree*) é o número de arcos com a extremidade inicial no vértice.

Por exemplo, no digrafo da Figura 4, o grau de entrada do vértice 6 é igual a 3 e o grau de saída deste vértice é igual a 1.

Um percurso (*walk*) em um grafo *G = (V,E)* é uma seqüência de vértices $v_1, v_2, ..., v_k$, k > 1, tal que $\{v_i, v_{i+1}\} \in E$, $1 \leq i \leq n-1$. Se $v_1 = v_k$ o percurso é denominado fechado (*closed walk*).

Um caminho (*path*) é um percurso em que todos os v_i são distintos, ou seja, não há vértices repetidos.

Um ciclo (*cycle*) é um percurso fechado em que todos os v_i são distintos, exceto o primeiro e o último.

Percursos, percursos fechados, caminhos e ciclos são denominados Eulerianos se utilizam todas as arestas do grafo exatamente uma vez.

Percursos, percursos fechados, caminhos e ciclos são denominados Hamiltonianos se utilizam todos os vértices do grafo exatamente uma vez.

Um grafo *G = (V,E)* é denominado conexo (*connected*) se existe um caminho ligando quaisquer dois vértices de V. Caso contrário, o grafo é denominado desconexo (*disconnected*). A Figura 6 mostra um exemplo de grafo desconexo.

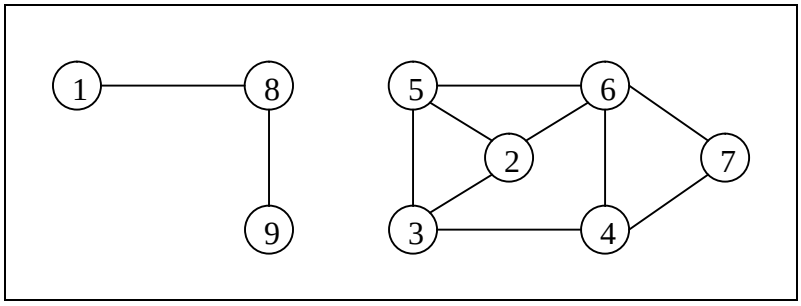


Figura 6 – Grafo desconexo

Um grafo completo é também denominado totalmente conexo (*completely connected*).

Uma árvore (*tree*) é a denominação dada a um grafo conexo e sem ciclos.

Para solucionar o problema das pontes de Königsberg, apresentado na Subseção 2.2.1, Euler formulou o primeiro teorema da Teoria dos Grafos:

Teorema 4.1 (Euler): Um multigrafo conexo e não orientado possui um ciclo Euleriano se, e somente se, todos os seus vértices possuírem grau par.

Formular um teorema equivalente ao apresentado acima para o caso dos ciclos Hamiltonianos é um dos grandes desafios da Teoria dos Grafos, uma vez que determinar se um grafo possui um ciclo Hamiltoniano é um problema NP-Completo.

2.2.3 Conceitos de Fluxo em Redes

Os conceitos a seguir são baseados em Diestel (2005) e Boaventura Netto (2003).

Uma rede $R = (V, A, s, t, c)$ é um digrafo (V, A) associado a um nó fixo denominado fonte (*source*) $s \in V$ com grau de entrada nulo, um nó fixo denominado sumidouro (*sink*) ou terminal $t \in V$ com grau de saída nulo e um mapeamento $c : A \rightarrow \mathrm{N}$ que associa um valor numérico denominado capacidade $c(u,v) \in \mathrm{N}$ a cada arco $(u,v) \in A$, conforme representado no exemplo da Figura 7.

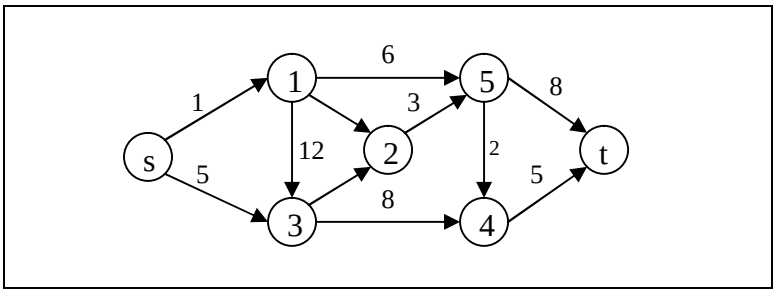


Figura 7 – Uma rede

Um fluxo em uma rede $R = (V, A, s, t, c)$ é uma função $f : A \rightarrow \mathrm{N}$ que associa um valor numérico $f(u,v) \in N$ a cada arco $(u,v) \in A$, tal que:

- $0 \leq f(u,v) \leq c(u,v), \forall (u,v) \in A$
- $\sum_{(u,v) \in A} f(u,v) = \sum_{(v,w) \in A} f(v,w), \forall v \in V - \{s,t\}$

A capacidade de um arco representa a quantidade máxima de fluxo que pode passar através do arco. Assim, as condições acima estabelecem que o fluxo associado a cada arco não pode ultrapassar a capacidade do arco e, em cada nó da rede, o somatório

dos fluxos que entram deve ser igual ao somatório dos fluxos que saem do nó, com exceção dos nós fonte e sumidouro.

Fluxo máximo em uma rede $R = (V, A, s, t, c)$ é o maior valor de fluxo que pode ser originado no nó fonte e terminado no nó sumidouro, respeitando a capacidade de cada arco.

2.2.4 Conceitos de Coloração de Grafos

Os conceitos a seguir são baseados em Diestel (2005) e Boaventura Netto (2003).

Uma coloração própria de vértices de um grafo $G=(V,E)$ é um mapeamento $c:V \rightarrow \{1, 2, \ldots, k\}$ tal que se $\{u,v\} \in$ E então $c(u) \neq c(v)$.

Os elementos do conjunto $\{1, 2, \ldots, k\}$ associados aos vértices do grafo G são denominados cores.

Denomina-se conflito à situação onde uma mesma cor está atribuída a dois vértices do grafo ligados por uma aresta.

Uma aresta é denominada aresta conflitante (*conflicting edge*) se os seus vértices terminais possuem a mesma cor.

Quando não existem conflitos, a coloração é viável ou factível. Caso contrário, a coloração é inviável ou infactível.

Uma coloração de vértices do grafo G que utiliza k cores é denominada k-coloração.

O menor número inteiro k tal que exista uma k-coloração de um grafo G é denominado número cromático de G e representado por $c(G)$.

Se $c(G) \leq k$ o grafo G é chamado de k-colorível.

Se $c(G) = k$ o grafo G é denominado k-cromático.

A Figura 8 mostra uma 3-coloração de um grafo *G* 3-cromático.

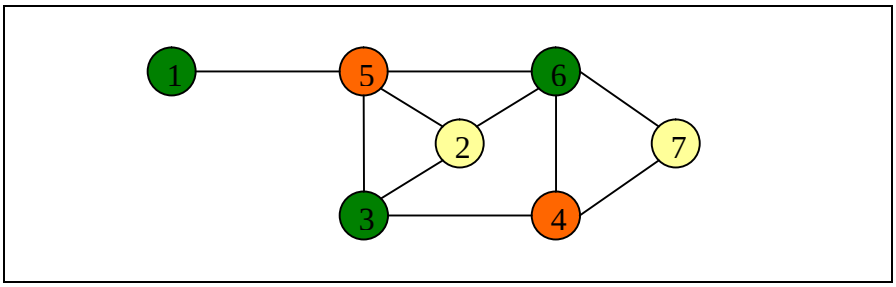


Figura 8 – Coloração de vértices de um grafo G com c (G) = 3

Uma *k*-coloração é uma partição dos vértices do grafo *G* em *k* conjuntos de vértices não-adjacentes (conjuntos independentes). Cada um destes conjuntos independentes de vértices é denominado uma classe de cores. Portanto, um grafo é *k*-colorível se, e somente se, é *k*-partido.

Uma coloração de arestas de um grafo $G=(V,E)$ é um mapeamento $c:E \rightarrow \{1, 2, \ldots, k\}$ tal que arestas adjacentes recebam cores diferentes.

Uma coloração de arestas do grafo *G* que utiliza *k* cores é denominada *k*-coloração de arestas.

O menor número inteiro *k* tal que exista uma k-coloração de arestas de um grafo *G* é denominado índice cromático de *G* e representado por $c'(G)$.

A Figura 9 mostra uma 4-coloração de arestas de um grafo *G*.

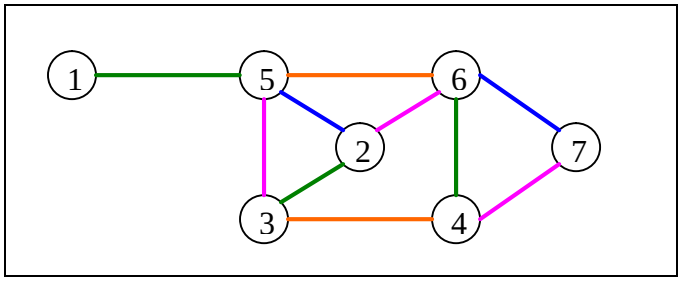


Figura 9 – 4-coloração de arestas de um grafo *G*

A terminologia de coloração de grafos apresentada neste capítulo deriva do problema das quatro cores formulado para coloração de mapas. No Capítulo 4 este problema é abordado, juntamente com aplicações e métodos utilizados para coloração de grafos.

No decorrer deste trabalho, sempre que o termo coloração de grafos for utilizado, refere-se à coloração de vértices do grafo.

3 PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA

3.1 DEFINIÇÃO E CLASSIFICAÇÃO

O Problema de Programação de Grade Horária (*Timetabling*) deriva do Problema de Escalonamento (*Scheduling*) e é definido por Wren (1996, apud BURKE et al., 2000) como a alocação, submetida a restrições, de determinados recursos a eventos colocados em um número limitado de períodos de tempo e locais, de forma a satisfazer tão aproximadamente quanto possível um conjunto de objetivos estabelecidos.

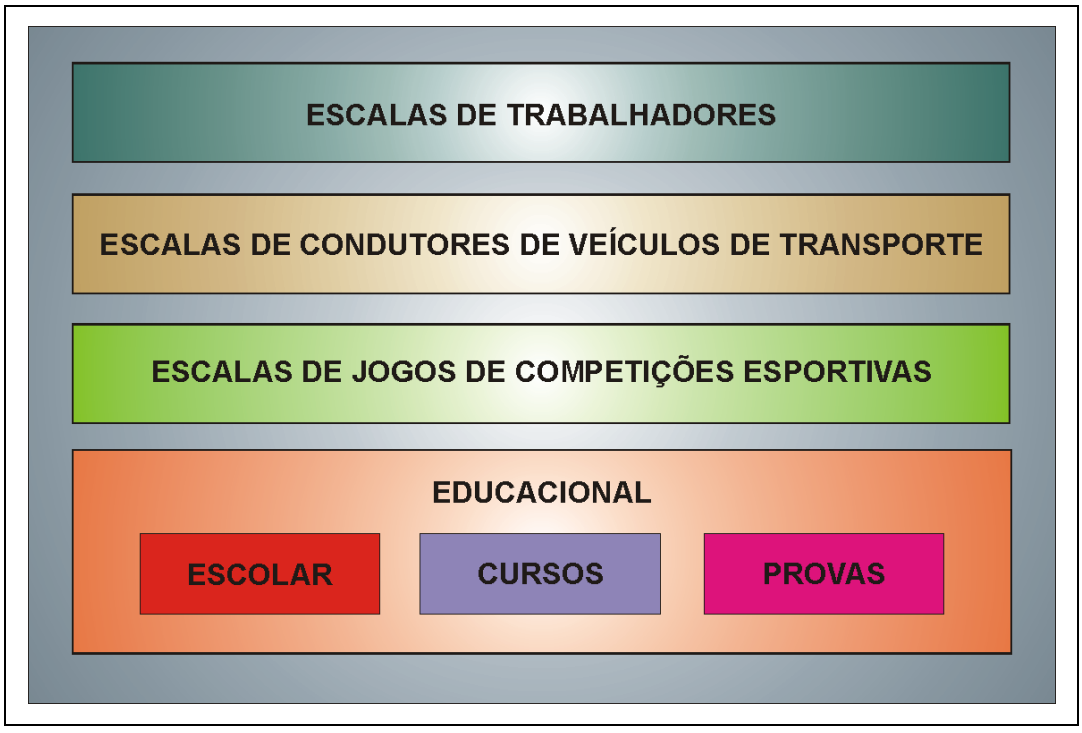


Figura 10 – Principais Problemas de Programação de Grade Horária

Conforme mostrado na Figura 10, o Problema de Programação de Grade Horária pode ser aplicado a diversos tipos de problemas, entre os quais inclui-se:

- Escalas de Trabalhadores (*Employee Timetabling*): trata da alocação de trabalhadores de uma empresa às suas tarefas em um conjunto de turnos de trabalho dentro de um período fixo de tempo, geralmente semanal, respeitando determinadas regras (restrições) pessoais e do trabalho. Em

Chiarandini, Schaerf e Tiozzo, (2000) e Drezet, Chesnes e Bellenguez-Morineau (2006) são apresentadas diferentes abordagens deste problema;

- Escalas de Condutores de Veículos de Transporte (*Transportation Timetabling*): Segundo Kwan (2004, apud LAPLAGNE; KWAN; KWAN, 2006) este problema determina a composição de um conjunto de turnos de condutores de veículos para a operação de transporte diário, assegurando a cobertura requerida e minimizando os custos operacionais;

- Escalas de Jogos de Competições Esportivas (*Sports Timetabling*): consiste na programação de uma tabela de jogos de competições esportivas, satisfazendo aos requisitos da competição e minimizando os custos. Trick (2000), Biajoli e outros (2003) apresentam exemplos de abordagens deste problema;

- Educacional (*Educational Timetabling*): aborda a geração de grade horária para instituições de ensino, que é o foco do estudo deste trabalho. Dos diversos tipos de Problemas de Programação de Grade Horária, o educacional é um dos mais amplamente estudados. Este problema é analisado mais detalhadamente a seguir.

3.2 PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA EDUCACIONAL (*EDUCATIONAL TIMETABLING*)

Em um problema de Programação de Grade Horária Educacional, a grade horária de uma instituição de ensino é elaborada através da alocação de um determinado número de eventos (normalmente aulas ou provas), em uma grade de horários previamente fixados (normalmente uma semana), utilizando determinados recursos (salas, professores, turmas) e sujeitos a determinadas restrições (pedagógicas, dos professores e da instituição).

As restrições geralmente são classificadas em dois tipos: Restrições Rígidas (*Hard Constraints*) e Restrições Flexíveis (*Soft Constraints*):

- Restrições Rígidas: Não podem ser violadas, ou seja, devem sempre ser satisfeitas, caso contrário, a solução não será viável.

- Restrições Flexíveis: Devem ser satisfeitas o máximo possível, mas quando violadas não invalidam a solução.

Normalmente uma função objetivo é utilizada para avaliar a satisfação de restrições flexíveis na solução, podendo utilizar diferentes pesos para cada restrição, de forma a provocar uma maior penalidade na violação de determinadas restrições que sejam mais importantes que outras. Procura-se obter um valor mínimo de penalidade, ainda que algumas restrições flexíveis sejam violadas. Algumas formulações do problema incluem também as restrições rígidas na função objetivo, com pesos muito maiores que os das restrições flexíveis.

Existem diversas variações para o Problema de Programação de Grade Horária Educacional, dividindo-o em classes. Estas formulações não são rigorosas, existindo problemas com características específicas que não se enquadram totalmente em uma única classe.

Uma classificação freqüentemente encontrada, proposta por Schaerf (1999a), leva em conta o tipo de instituição de ensino – universidade ou escola – bem como o tipo de restrições aplicadas, dividindo o Problema de Programação de Grade Horária Educacional em três categorias:

- Problema de Programação de Grade Horária Escolar (*School Timetabling Problem* ou *Class/Teacher Timetabling Problem*): programação semanal das aulas de uma escola, de forma que professores e alunos não tenham mais de uma aula em um mesmo horário e a carga horária seja cumprida. Restrições

adicionais podem ser impostas. A programação é baseada em grupos de disciplinas fixas para cada turma. As aulas (SOUZA, 2000) geralmente ocorrem em um mesmo turno, têm igual duração e são realizadas nas mesmas salas;

A programação de grade horária de universidades foi dividida em duas outras categorias, cursos e provas, descritas a seguir:

- Problema de Programação de Grade Horária de Cursos (*Course Timetabling Problem* ou *University Timetabling Problem*): programação semanal das aulas de um conjunto de cursos de uma universidade, minimizando a sobreposição de aulas dos cursos que possuam estudantes em comum. Não existe um currículo fixo, cada aluno escolhe quais disciplinas (obrigatórias e optativas) que irá cursar em cada período, respeitando pré-requisitos. As aulas (SOUZA, 2000) podem ocorrer em qualquer turno, podem ter durações diferentes e são alocadas a um número limitado de salas, cujas capacidades devem ser respeitadas. Uma mesma disciplina pode ser ministrada em diversos cursos;

- Problema de Programação de Grade Horária de Provas (*Exam Timetabling Problem* ou *Examination Timetabling Problem*): programação de provas de um conjunto de cursos em uma universidade, evitando sobreposição de provas de cursos com estudantes em comum e distribuindo as provas pelos alunos o máximo possível. Diferencia-se (SOUZA, 2000) do problema de Programação de Grade Horária de Cursos pela natureza das restrições envolvidas: determinadas provas não podem ocorrer antes de outras, outras provas precisam ser realizadas em horários consecutivos, alunos têm um limite diário de provas, e assim por diante.

3.3 ABORDAGENS PARA RESOLUÇÃO DO PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA EDUCACIONAL

A maioria dos métodos desenvolvidos para a resolução do problema de Programação de Grade Horária Educacional tem suas raízes na Pesquisa Operacional ou na Inteligência Artificial.

3.3.1 Conceitos

Com o objetivo de facilitar a descrição, alguns conceitos prévios serão abordados a seguir:

- **Métodos Construtivos e Métodos de Busca Local:** Métodos construtivos produzem uma solução do início ao fim, geralmente sem partir de uma solução inicial. Um novo elemento é escolhido a cada iteração para ser incorporado à solução parcial. O processo de escolha utilizado mais freqüentemente é o guloso (*greedy*), no qual os elementos são avaliados através de uma heurística para tomada de decisão, que fornece a melhor opção possível a cada iteração, proporcionando o maior ganho, daí a razão do nome guloso. Os métodos de Busca Local utilizam uma solução inicial como ponto de partida e a definição de uma vizinhança associada à cada solução, de onde são pesquisadas as novas soluções. A transformação pela qual se obtém uma solução vizinha a partir de uma solução inicial é denominada movimento. Através do movimento de uma solução para outra existente na vizinhança, a solução inicial vai sendo refinada de forma iterativa. (MICHALEWICZ; FOGEL, 2002)

- **Métodos Evolutivos e Métodos Híbridos**: Os métodos construtivos e de busca local, apresentados anteriormente, utilizam uma única solução como base para as explorações seguintes a cada iteração. Os métodos evolutivos, ao contrário, se baseiam na evolução a partir da melhoria simultânea de um conjunto de soluções potenciais para um problema, denominado população,

através da competição e da interação entre seus elementos, favorecendo a sobrevivência das melhores soluções e reproduzindo no grupo características destas soluções. Métodos híbridos combinam características de mais de um método. (MICHALEWICZ; FOGEL, 2002)

- **Heurísticas, Meta-heurísticas e Hiper-heurísticas:** Heurísticas são algoritmos que buscam uma solução de boa qualidade para um problema de otimização, em um tempo computacional aceitável, mas sem garantia que a solução encontrada é ou está próxima da solução ótima, ou seja, sem uma garantia de desempenho (*performance guarantee*) formal. As heurísticas normalmente são utilizadas em problemas cuja complexidade da solução, através de algoritmos exatos, cresce exponencialmente em função de algum parâmetro, tornando o seu emprego inadequado ou inviável. Meta-heurísticas são abordagens heurísticas de caráter mais genérico, possibilitando sua aplicação na resolução de diferentes tipos de problemas de otimização, com poucas adaptações. Hiper-heurísticas são algoritmos (em um nível mais elevado) que selecionam heurísticas apropriadas (em um nível mais baixo) para serem aplicadas na solução de problemas de otimização. Uma hiper-heurística ocupa-se com a exploração do espaço de busca de heurísticas de baixo nível, ao invés de tratar diretamente das soluções do problema. A Figura 11 mostra a estrutura geral de uma hiper-heurística (BURKE et al., 2007), (BAI et al., 2006), (BURKE; KENDALL; SOUBEIGA, 2003), (PAPADIMITRIOU; STEIGLITZ, 1998).

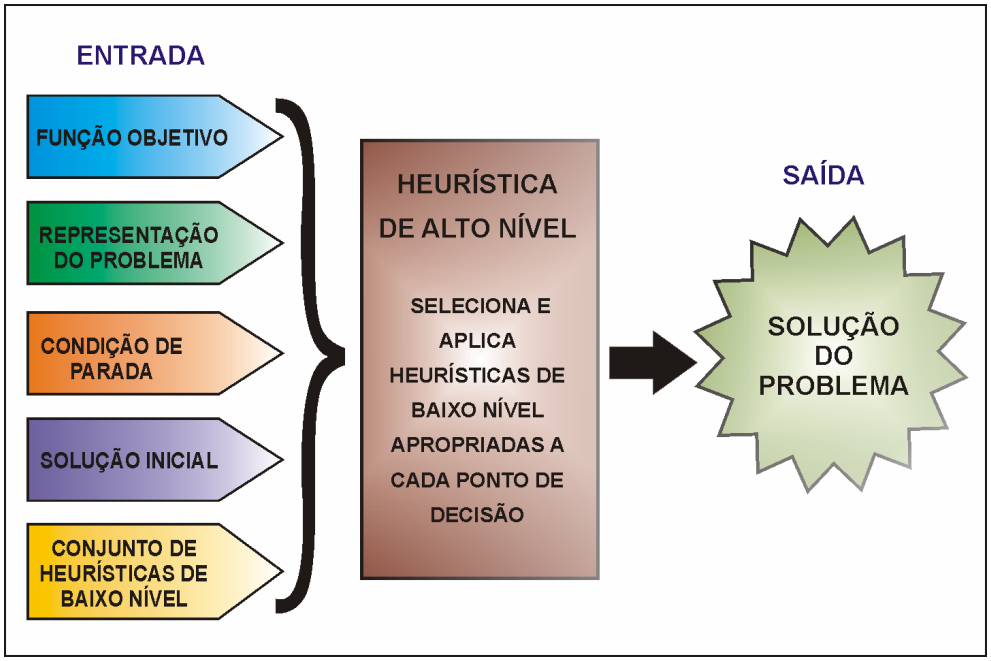


Figura 11 – Estrutura Geral de uma Hiper-Heurística. Adaptado de (BAI et al., 2006) e (BURKE; KENDALL; SOUBEIGA, 2003)

- ***Swarm Intelligence***: (Dorigo et al., 2004) O termo *Swarm Intelligence* foi proposto inicialmente por Beni and Wang (1989, apud Dorigo et al., 2004), aplicado a sistemas robóticos. Posteriormente, Bonabeau, Dorigo e Theraulaz (1999, apud Dorigo et al., 2004) estenderam este conceito, formulando a seguinte definição mais geral: "*Swarm Intelligence* é qualquer tentativa de projetar algoritmos ou dispositivos distribuídos de solução de problemas inspirados no comportamento coletivo das colônias de insetos sociais e de outras sociedades animais". Desta forma, foi estabelecido um novo paradigma computacional para resolução de uma grande variedade de problemas, baseados na observação de comportamentos coletivos encontrados na natureza.

A seguir são descritas algumas das abordagens mais encontradas na literatura (Figura 12) para o problema de Programação de Grade Horária Educacional. Algumas se aplicam a todos os tipos vistos anteriormente, enquanto outras são voltadas para uma categoria específica.

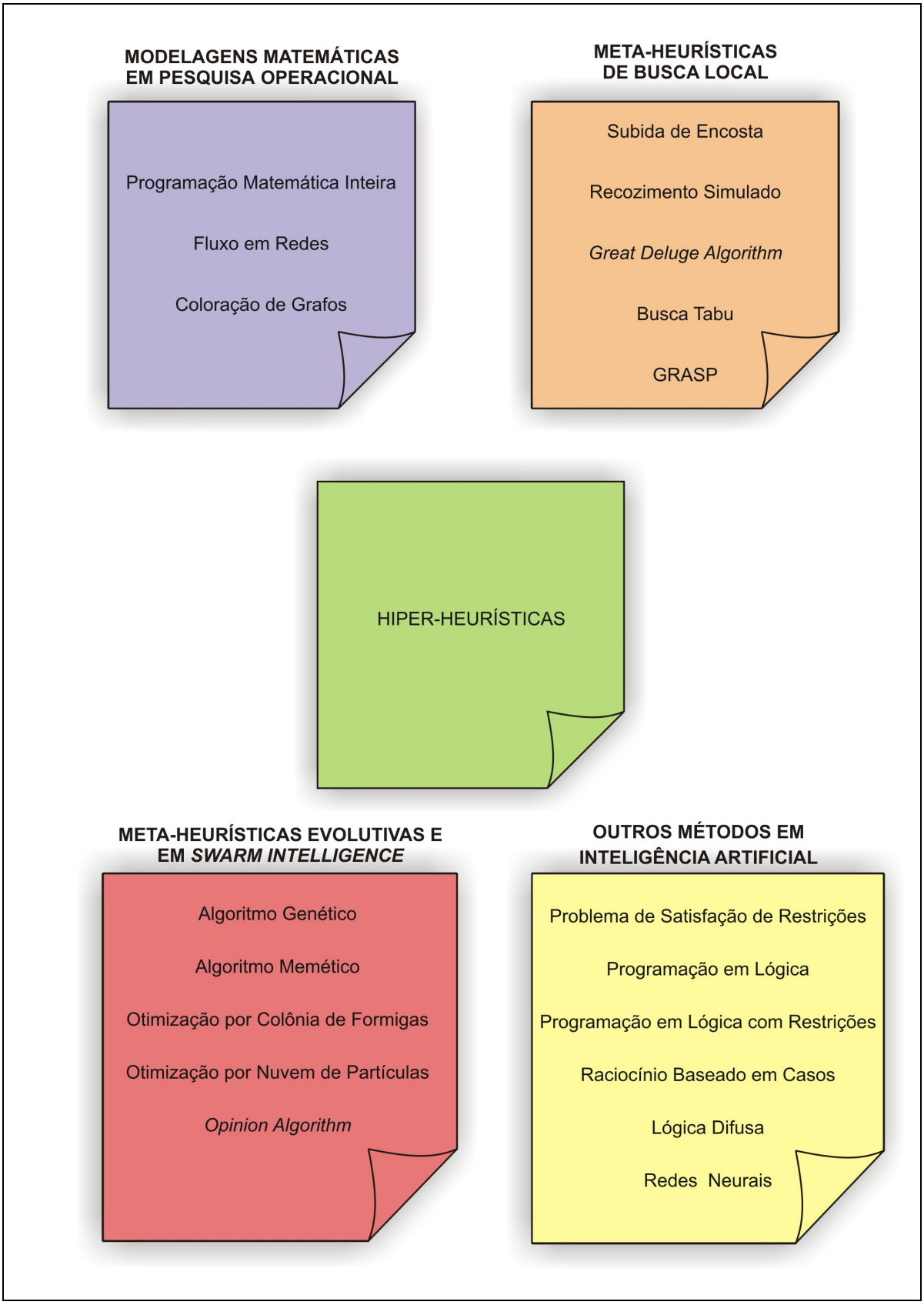


Figura 12 - Principais abordagens para modelagem e resolução do problema de Programação de Grade Horária Educacional encontrados na literatura

3.3.2 **Programação Linear Inteira**:

Nesta abordagem o problema é formulado como um problema de programação linear inteira. Como exemplo, será utilizada uma forma do problema de Programação de Grade Horária Escolar, onde as restrições rígidas são que professores e alunos não tenham mais de uma aula em um mesmo horário, a carga horária seja cumprida e as indisponibilidades de professores e turmas sejam respeitadas. As restrições flexíveis são consideradas através de pesos que compõem uma função objetivo a ser minimizada. (SCHAERF, 1999a), (DE WERRA, 1985) e (JUNGINGER, 1986)

Seja:

- Um conjunto de professores $T = \{t_i\}, i = 1,...,a$;
- Um conjunto de turmas $C = \{c_j\}, j = 1,...,b$;
- Um conjunto de horários $H = \{h_k\}, k = 1,...s$;
- Uma matriz de requerimentos $a \times b \quad R = [r_{ij}]$, onde r_{ij} representa a quantidade de aulas do professor t_i para a turma c_j;
- Uma matriz binária $a \times s \quad T = [t_{ik}]$, tal que $t_{ik} = 1$ se o professor t_i está disponível no horário h_k e $t_{ik} = 0$ caso contrário;
- Uma matriz binária $b \times s \quad C = [c_{jk}]$, tal que $c_{jk} = 1$ se a turma c_j está disponível no horário h_k e $c_{jk} = 0$ caso contrário;
- Um conjunto de pesos $D = \{d_{ijk}\}$, onde quanto maior o valor do peso d_{ijk} menos desejável é a realização da aula do professor t_i para a turma c_j no horário h_k.

Uma formulação matemática do problema acima como problema de programação inteira é apresentada a seguir:

$$Min \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{s} d_{ijk} x_{ijk} \qquad (1)$$

Sujeito a:

$$\sum_{k=1}^{s} x_{ijk} = r_{ij} \qquad (i = 1,...,a;\ j = 1,...,b) \qquad (2)$$

$$\sum_{j=1}^{b} x_{ijk} \leq t_{ik} \qquad (i = 1,...,a;\ k = 1,...,s) \qquad (3)$$

$$\sum_{i=1}^{a} x_{ijk} \leq c_{jk} \qquad (j = 1,...,b;\ k = 1,...,s) \qquad (4)$$

$$x_{ijk} = 0 \text{ ou } 1 \quad (i = 1,...,a;\ j = 1,...,b;\ k = 1,...,s)$$

onde $x_{ijk} = 1$ se o professor t_i aplica uma aula para a turma c_j no horário h_k e $x_{ijk} = 0$ caso contrário.

- A minimização da função objetivo (1) tende a evitar as alocações menos desejáveis, uma vez que estas estão associadas aos maiores pesos d_{ijk}.
- As restrições (2) asseguram que a carga horária de cada professor seja cumprida.
- As restrições (3) asseguram que cada professor ministre no máximo uma aula em cada horário, e somente se o professor estiver disponível naquele horário.
- As restrições (4) asseguram que cada turma tenha no máximo uma aula em cada horário, e somente se a turma estiver disponível naquele horário.

3.3.3 Fluxo em Redes (*Network Flow*)

O problema de programação de grade horária educacional é modelado como um problema de fluxo em redes, tratado na Subseção 2.2.3. Grafos orientados (um para

cada horário k) são associados ao problema, nos quais os nós representam turmas e professores. Cada aula do professor t_i para a turma c_j é representada pelo arco (t_i, c_j). Um nó fonte *s* é conectado a todos os professores através de arcos (s, t_i) e um nó sumidouro *t* é conectado a todas as turmas através de arcos (c_j, t). Limites superiores (capacidades) e inferiores são associados aos arcos. Nas *k* redes formadas, cada caminho de *s* para *t* representa uma aula e a grade horária educacional completa é obtida a partir de uma solução para todos os *k* problemas de fluxo máximo gerados. (OSTERMANN e DE WERRA, 1983), (DE WERRA, 1985)

3.3.4 Coloração de Grafos

Conforme Neufeld e Tartar (1974, 1975), um problema de Programação de Grade Horária Escolar pode ser transformado em um problema de coloração de grafos, apresentado na Subseção 2.2.4, da seguinte forma:

- cada vértice do grafo representa uma aula;
- uma aresta interligando dois vértices indica que as respectivas aulas associadas não podem ser programadas para ocorrerem em um mesmo horário;
- cada cor representa um horário da grade horária correspondente.

O grafo associado ao problema de Programação de Grade Horária Escolar é denominado grafo adjunto. Este processo será detalhado na Seção 5.1.

Problemas de Programação de Grade Horária de Cursos e Provas também foram abordados como problemas de Coloração de Grafos (BURKE; ELLIMAN; WEARE, 1994), (DE WERRA, 1985).

3.3.5 Subida de Encosta (*Hill Climbing*)

Método simples de busca local que foi empregado na resolução do problema de Programação de Grade Horária Educacional por Appleby, Blake e Newman (1960 apud BURKE et al, 2003).

Nesta abordagem, partindo da solução inicial a vizinhança da solução existente é pesquisada e uma nova solução é aceita em substituição à existente somente se apresentar resultado melhor ou igual à solução atual.

O nome se deve à analogia, para o problema de maximização, com uma pessoa escalando uma montanha, buscando alcançar o cume, acima de todos os vizinhos.

Posteriormente, outras abordagens derivadas de *Hill Climbing* foram desenvolvidas, as quais permitem, sob determinadas condições, que sejam aceitas soluções piores que a atual, como forma de se alcançar depois soluções melhores no espaço de soluções. Serão apresentadas a seguir três destas técnicas: Recozimento Simulado, *Great Deluge Algorithm* e Busca Tabu.

3.3.6 Recozimento Simulado ou Têmpera Simulada (*Simulated Annealing - SA*)

Esta meta-heurística de busca local proposta por Kirkpatrick e outros (1983 apud SCHAERF, 1999a) também pode ser aplicada ao problema de Programação de Grade Horária Educacional com função objetivo para medir os conflitos por violação de restrições. A partir da solução inicial, uma nova solução é obtida na vizinhança pela escolha aleatória de uma aula em um período e o movimento desta aula para um outro período também aleatório. É calculada a diferença d entre os valores da função objetivo para as soluções nova e existente. A nova solução é aceita automaticamente em substituição à solução atual se apresentar custo menor ($d < 0$). Caso contrário ($d \geq 0$) a nova solução poderá ser aceita com probabilidade $P = e^{-dT}$, onde o

parâmetro *T* representa a temperatura, por analogia com a termodinâmica na simulação do resfriamento de átomos aquecidos, nos processos de têmpera de metais ou recozimento de vidros ou cerâmicas, onde os materiais são aquecidos a uma temperatura elevada e depois resfriados gradualmente. Este parâmetro é suavemente modificado a cada iteração, alterando a probabilidade de aceitação da solução.

Elmohamed, Coddington e Fox (1998) utilizam abordagens baseadas em *Simulated Annealing* para resolução do problema de Programação de Grade Horária Educacional.

3.3.7 *Great Deluge Algorithm*

Foi introduzido por Dueck (1993). Como na abordagem de *Simulated Annealing*, este algoritmo também poderá aceitar soluções piores que a solução corrente, porém sob outras condições: a nova solução será aceita se o valor da função objetivo para esta solução for menor ou igual a um limite superior *L*, denominado nível. Os nomes do algoritmo e do limite *L* se referem à analogia, para o problema de maximização, com alguém tentando se refugiar em lugares elevados para se proteger de um grande dilúvio, procurando se mover por caminhos escolhidos de forma a não ser alcançado, à medida que o nível da água vai aumentando. No começo do processo o nível *L* é igual ao valor da função objetivo da solução inicial e, durante a pesquisa, vai sendo diminuído deste valor um fator fixo definido pelo usuário, que na analogia citada representa a elevação progressiva do nível da água.

Burke e outros (2003) e, Burke e Bykov (2006), descrevem extensões e variantes do *Great Deluge Algorithm* aplicadas a problemas de Programação de Grade Horária Educacional.

3.3.8 Busca Tabu (*Tabu Search-TS*)

Outra meta-heurística de busca local que foi aplicada ao problema de Programação de Grade Horária Educacional (HERTZ, 1991), (HERTZ, 1992), (COSTA, 1994), (SCHAERF,1999a), (SCHAERF,1999b), (DI GASPERO; SCHAERF, 2001).

Partindo de uma solução inicial viável, o algoritmo explora um subconjunto de soluções na vizinhança da solução corrente. O membro com menor valor da função objetivo deste subconjunto substitui a solução corrente, independentemente se este valor é melhor ou pior que o valor da função objetivo para a solução atual.

Para procurar evitar a ocorrência de ciclos (retornos a uma solução já visitada pelo algoritmo), movimentos que levem novamente a soluções já visitadas recentemente devem ser impedidos durante certo tempo. Para este fim é utilizada uma lista de movimentos proibidos, denominada lista tabu, composta dos movimentos reversos aos movimentos que foram realizados nas iterações mais recentes. Estes movimentos são liberados após um determinado número de iterações, fixo ou variável, denominado período tabu (tabu tenure). Para não restringir demais a composição do subconjunto de vizinhos, existe ainda um mecanismo denominado critério de aspiração, que pode cancelar a condição tabu de um movimento e executá-lo, por exemplo, se o valor da função objetivo da solução resultante for melhor que o valor correspondente à melhor solução encontrada até o momento.

Essa meta-heurística será detalhada na Seção 4.4.

3.3.9 Procedimento de Busca Adaptativa, Aleatória e Gulosa (*Greedy Randomized Adaptive Search Procedure - GRASP*)

Meta-heurística híbrida iterativa composta de duas fases: uma construtiva e outra de busca local. (FEO; RESENDE, 1995, apud RESENDE e RIBEIRO, 2003)

A fase construtiva gera uma solução viável. Em cada passo desta fase, todos os elementos que podem ser incorporados à solução construída (sem que ela deixe de ser factível) são avaliados por uma função **gulosa** e é gerada uma lista com os elementos que proporcionam os melhores custos. Um elemento desta lista é escolhido **aleatoriamente** e incorporado à solução parcial. Então a lista é atualizada e os custos reavaliados, fazendo assim com que o processo seja **adaptativo**. A fase de busca local investiga a vizinhança da solução factível gerada na fase construtiva até que um mínimo local seja encontrado. As iterações são repetidas até que um critério de parada seja alcançado. O resultado será a melhor solução obtida em todo o processo.

Souza, Guimarães e Costa (2002) utilizaram uma técnica híbrida para abordar o problema de Programação de Grade Horária Educacional, baseada nas meta-heurísticas GRASP, Algoritmos Genéticos e Busca Tabu.

Em Souza, Ochi e Maculan (2003), o problema é tratado através de uma meta-heurística híbrida GRASP-Busca Tabu com recursos adicionais.

3.3.10 Algoritmos Genéticos (*Genetic Algorithms-GA*)

Meta-heurística evolutiva (HOLLAND, 1975, 1992) (GOLDBERG, 1989) utilizada na resolução de problemas de otimização, inspirada na teoria de evolução das espécies de Charles Darwin, através da seleção natural dos mais aptos (SCHAERF, 1999a).

Cada iteração do algoritmo é denominada uma geração. Um conjunto de soluções viáveis gerado é denominado população e cada uma das soluções é chamada de indivíduo da população. Cada indivíduo é representado como um cromossomo, ou seja, uma seqüência de símbolos, escolhidos dentro de um conjunto finito, denominado alfabeto. Cada elemento da seqüência é denominado gene e cada elemento do alfabeto é denominado alelo, representando assim um valor possível para um gene. Uma função de adaptação é utilizada para avaliar a qualidade de cada

solução. Este valor atribuído a cada indivíduo pela função de adaptação é denominado aptidão (*fitness*). Esta função normalmente é definida como a função objetivo do problema a ser resolvido.

Na fase de inicialização do algoritmo uma população inicial é gerada aleatoriamente.

A cada geração, o algoritmo cria uma nova população a partir da construída na geração anterior, através das fases descritas a seguir:

- Avaliação: Nesta fase, as aptidões dos indivíduos da população são alocadas pela função objetivo.

- Seleção: Um determinado número de indivíduos da população corrente é escolhido, com probabilidade de escolha proporcional à aptidão de cada indivíduo, calculada pela função objetivo. Como um mesmo indivíduo pode ser selecionado mais de uma vez, as melhores soluções geram mais cópias enquanto as piores soluções tendem a desaparecer da população.

- Reprodução: Nesta fase são aplicados nos indivíduos selecionados operadores específicos, tais como o cruzamento (*crossover*) e a mutação, gerando os novos indivíduos que irão compor a nova população. No cruzamento, indivíduos selecionados são recombinados aos pares, através de troca mútua (*swap*) de alelos de genes entre os dois cromossomos. Na mutação, ocorre a alteração do alelo de um gene de um cromossomo, escolhidos aleatoriamente.

Para assegurar que os melhores indivíduos sejam preservados de uma geração para a outra, pode ser utilizado o elitismo. Este processo consiste em passar um percentual dos melhores indivíduos automaticamente para a próxima geração, sem alteração das suas características genéticas.

Após as operações descritas, que ocorrem segundo probabilidades específicas, a população corrente é atualizada com os novos indivíduos gerados, gerando assim uma nova população.

O método é finalizado quando um determinado número de populações é atingido ou quando a melhor solução atinge um determinado valor calculado pela função objetivo ou ainda quando o algoritmo não faz nenhuma melhora do valor da função em um número limite de gerações.

Na abordagem do problema de Programação de Grade Horária Educacional através de algoritmos genéticos (BURKE et al., 1997), a representação de uma grade horária (*timetable*) é geralmente realizada na forma de uma longa seqüência (*string*) de *bits*, codificada com a informação de quando (e onde) cada evento (aula ou prova) deve ocorrer. A função objetivo avalia a aptidão das soluções em função da satisfação das restrições rígidas e flexíveis do problema. Operadores genéticos (cruzamento, mutação) são aplicados às seqüências de bits selecionadas em cada iteração.

Burke, Elliman e Weare (1994) apresentam uma abordagem do problema de Programação de Grade Horária Educacional baseada em Algoritmo Genético, através da implementação das fases descritas anteriormente (Figura 13).

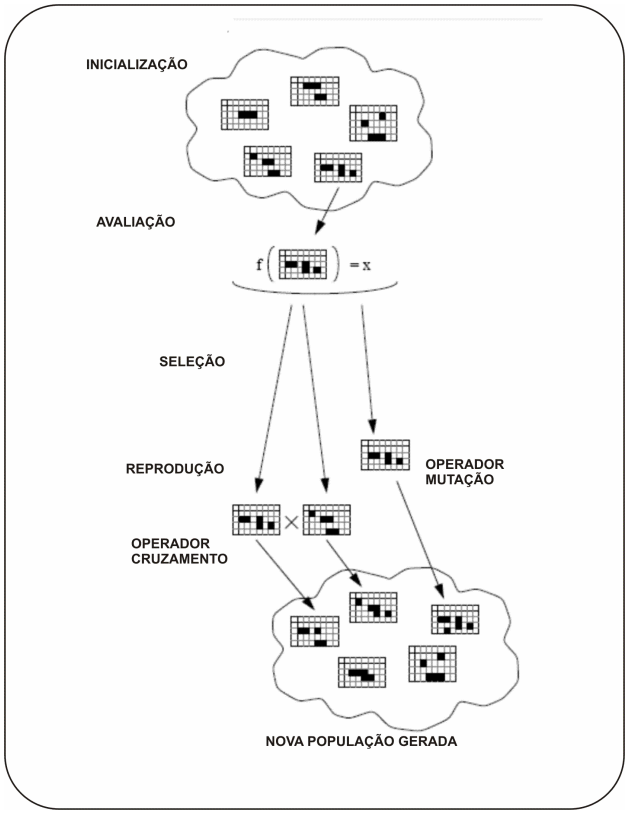


Figura 13 - Exemplo da aplicação do algoritmo genético ao problema de Programação de Grade Horária Educacional. Adaptado de (BURKE; ELLIMAN; WEARE, 1994)

3.3.11 Algoritmos Meméticos

Os Algoritmos Meméticos foram introduzidos em Moscato (1989) e Moscato e Norman (1992, apud RADCLIFFE; SURRY, 1994) para descrever a hibridação de algoritmos genéticos com métodos de busca local, utilizados de forma intensiva.

São inspirados no *Meme*, termo cunhado no livro "*The Selfish Gene*" (DAWKINS, 1976, apud MOSCATO,1989) que pode ser definido como uma unidade de informação na transmissão cultural que reproduz a si mesmo passando de uma mente para outra, à medida que as pessoas trocam idéias, sendo em alguns aspectos análogo ao gene, uma unidade de informação na transmissão genética que também se reproduz e passa de um indivíduo para outro (RADCLIFFE; SURRY, 1994). A principal diferença entre genes e *memes* é que antes de um *meme* ser propagado, ele é adaptado pela pessoa que o transmite, na medida que essa pessoa pensa, compreende e processa o *meme*, enquanto que genes são propagados sem modificações.

A adição de um método de busca local ao algoritmo genético, aplicado a cada novo indivíduo gerado na fase de reprodução, antes dele ser inserido na população, reduz a pesquisa no espaço de soluções a uma pesquisa no subespaço de ótimos locais. Os operadores genéticos utilizados na fase de reprodução, tais como o cruzamento (*crossover*) e a mutação, aplicados sobre soluções existentes, freqüentemente produzirão soluções que estão situadas fora deste subespaço de ótimos locais. Através do método de busca local estas soluções são restauradas, de forma a produzir soluções situadas neste subespaço de ótimos locais.

Este processo é ilustrado na Figura 14 tomando como exemplo o operador de cruzamento aplicado a duas soluções existentes X e Y, produzindo a solução Z' situada fora do subespaço de ótimos locais. A solução Z' é então otimizada pelo método de busca local para produzir a solução Z, situada dentro do subespaço de ótimos locais.

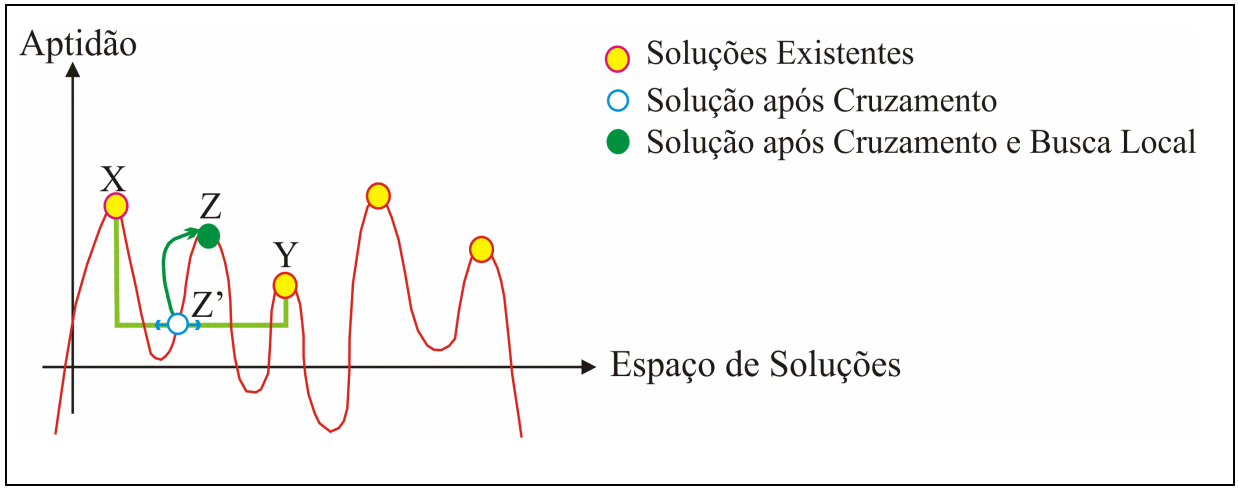


Figura 14 – Pesquisa efetuada pelos algoritmos meméticos sobre o subespaço de ótimos locais, através da utilização de métodos de busca local. Adaptado de (RADCLIFFE; SURRY, 1994)

Em Burke, Newall, Weare (1996) e Rossi-Doria, Paechter (2004a) algoritmos meméticos são utilizados para tratar problemas de Programação de Grade Horária Educacional.

3.3.12 Otimização por Colônia de Formigas (*Ant Colony Optimization-ACO*)

Meta-heurística empregada na solução de problemas de otimização combinatória introduzida por Colorni, Dorigo e Maniezzo (1991, apud DORIGO; STUTZLE, 2002), utilizando o conceito de aprendizado por reforço. Inspirada na analogia biológica de formigas de algumas espécies que depositam uma substância química denominada feromônio sobre o caminho que percorrem, formando uma trilha que é identificada pelas outras formigas, influenciando o comportamento da colônia na escolha do caminho a ser percorrido.

O processo ocorre da seguinte forma: no percurso do formigueiro até a fonte de alimento, ao chegar a um ponto inexplorado do percurso (ainda sem a presença da trilha de feromônio) onde precisam decidir entre dois caminhos a seguir, cada formiga inicialmente escolhe aleatoriamente um dos caminhos, sobre o qual libera o

feromônio. Aproximadamente metade das formigas percorrerá cada um dos caminhos, uma vez que a probabilidade de escolha é igual para ambos. Entretanto, considerando que as distâncias percorridas em cada caminho sejam diferentes, as formigas que escolheram o caminho mais curto são as primeiras a chegar à fonte de alimento e a retornar ao formigueiro. Neste ponto, haverá maior acúmulo de feromônio sobre o caminho mais curto, influenciando as formigas a escolherem este caminho. Após algum tempo, esta diferença na quantidade de feromônio fará com que a maior parte da colônia siga pelo caminho mais curto.

A Figura 15 ilustra este processo.

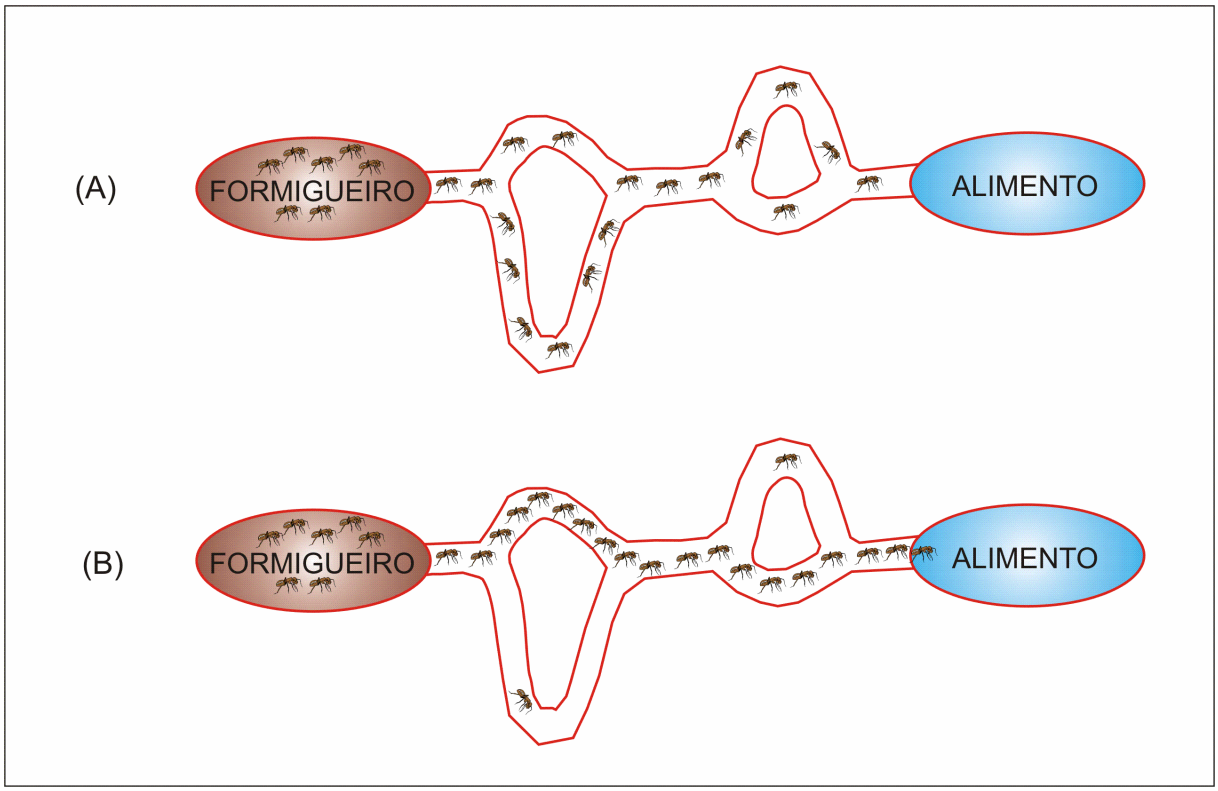


Figura 15 – (a) Formigas exploram os caminhos até o alimento - (b) A maior parte das formigas escolhe o caminho mais curto – Adaptado de Goss et al. (1989, apud DORIGO; DI CARO; GAMBARELLA, 1999)

Os algoritmos ACO foram construídos a partir desta analogia, onde agentes denominados formigas artificiais, utilizando trilhas de feromônio artificiais, constroem soluções efetuando percursos em um grafo completo (totalmente conexo), denominado grafo de construção (*construction graph*), no qual os vértices são os

componentes da solução e as arestas do grafo, denominadas conexões, interligam os componentes.

O primeiro algoritmo ACO proposto foi o Sistema de Formigas (*Ant System-AS*), utilizado inicialmente na resolução do problema do caixeiro viajante (*Traveling Salesman Problem-TSP*) (DORIGO; MANIEZZO; COLORNI, 1991, apud DORIGO; STUTZLE, 2002) e depois em diversos outros problemas de otimização combinatória.

Diversos aperfeiçoamentos foram introduzidos no Sistema de Formigas, buscando melhorar o seu desempenho, dando origem a diversas variantes deste algoritmo. Uma delas (STÜTZLE; HOOS, 1997a, apud DORIGO; DI CARO; GAMBARELLA, 1999) é o Sistema de Formigas Max-Min (*MAX-MIN Ant System - MMAS*), que introduz limites inferior e superior para os valores da trilha de feromônio artificial e utiliza uma inicialização diferente para estes valores. Além disso, no algoritmo MMAS somente à melhor formiga artificial é permitido adicionar feromônio após cada iteração e o algoritmo normalmente utiliza Busca Local para aperfeiçoar seu desempenho.

Os algoritmos ACO representam uma das aplicações na área de *swarm intelligence* definida na Subseção 3.3.1.

Socha, Knowles e Sampels (2002) abordam a variante ACO MMAS aplicada ao problema de Programação de Grade Horária de Cursos, mapeando o problema em um grafo de construção, de forma que um caminho através do grafo representa uma solução para o problema.

Eley (2006) aplica o ACO MMAS ao problema de Programação de Grade Horária de Provas.

3.3.13 Otimização por Nuvem de Partículas (*Particle Swarm Optimization-PSO*)

Método evolutivo desenvolvido por Kennedy e Eberhart (1995, apud HU, EBERHART e SHI, 2004) inspirado no comportamento coletivo de animais na natureza, como os movimentos coordenados em cardumes de peixes (*fish schooling*) e as formações em bandos de pássaros *(bird flocking*) (Figura 16).

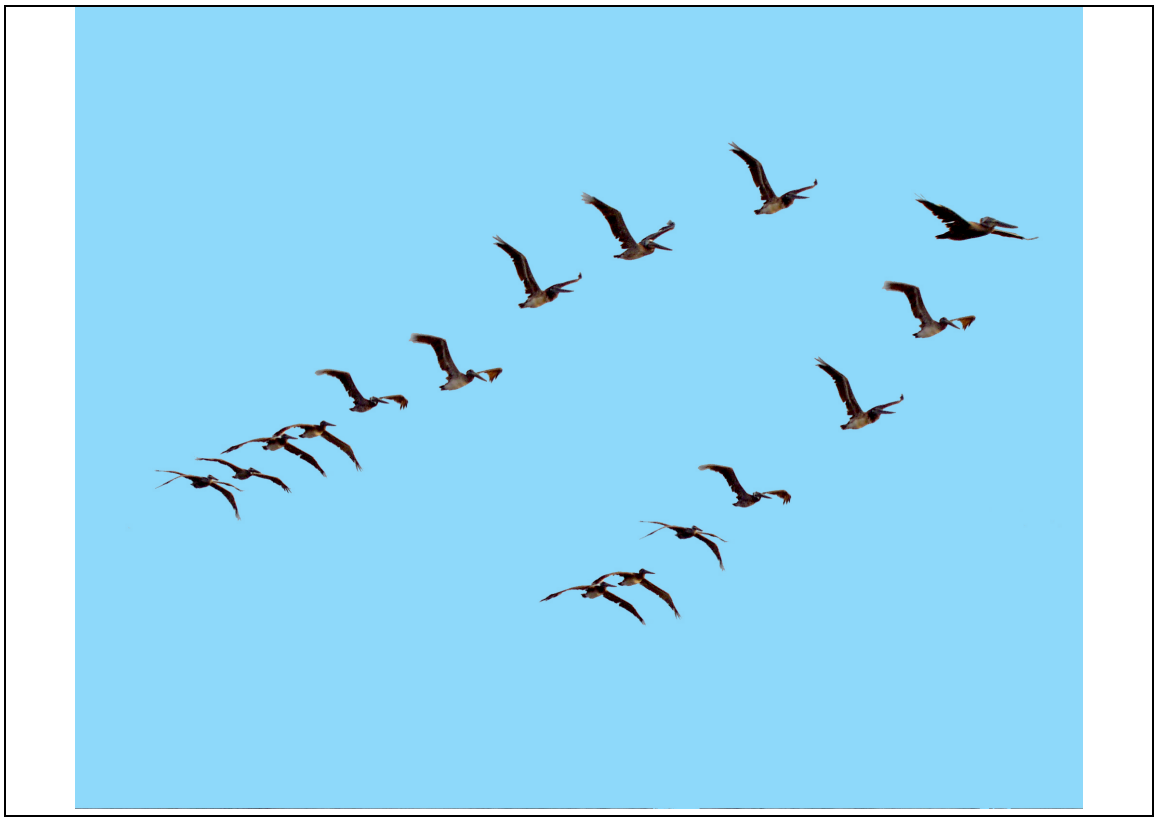


Figura 16 – Exemplo de Formação em V de um bando de pássaros.

O algoritmo *PSO* é apresentado na Figura 17:

Início

1. Gere uma população inicial de n partículas aleatoriamente distribuídas no espaço de soluções S
2. $gbest \leftarrow x_0$
3. **para** i **de** 1 **a** n **faça**
4. $pbest_i \leftarrow x_0$
5. **fim para**
6. **enquanto** (critérios de parada não são atingidos) **faça**
7. **para** i **de** 1 **a** n **faça**
8. **se** $f(x_i) < f(pbest_i)$ **então**
9. $pbest_i \leftarrow x_i$
10. **fim se**
11. **se** $f(pbest_i) < f(gbest)$ **então**
12. $gbest \leftarrow pbest_i$
13. **fim se**
14. **fim para**
15. **para** i **de** 1 **a** n **faça**
16. $v_i = w \times v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times Rand() \times (gbest - x_i)$
17. $x_i = x_i + v_i$
18. **fim para**
19. **fim enquanto**

Fim

Figura 17 – Algoritmo de Otimização por Nuvem de Partículas

Neste algoritmo, cada solução isolada é denominada partícula. Inicialmente um determinado número de partículas é distribuído aleatoriamente no espaço de soluções n-dimensional. Uma função objetivo a ser otimizada avalia a qualidade das soluções (*fitness*). A cada partícula estão associados valores de posição x_i e velocidade v_i, calculados de acordo com as respectivas equações descritas a seguir:

$$v_i^{t+1} = w \times v_i^t + c_1 \times rand() \times (pbest_i - x_i^t) + c_2 \times Rand() \times (gbest - x_i^t) \quad (1)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (2)$$

onde:

- v_i^t é a velocidade atual da partícula *i* ;
- v_i^{t+1} é a nova velocidade da partícula *i* ;

- x_i^t é a posição atual da partícula *i* ;
- x_i^{t+1} é a nova posição da partícula *i* ;
- $Rand()$ e $rand()$ são números aleatórios gerados de forma independente ;
- c_1 e c_2 são denominados coeficientes de aprendizagem (*learning factors*) e controlam a influência de $pbest_i$ e $gbest$ no processo de busca ;
- w é uma função denominada peso de inércia (*inertia weight*) utilizada para melhorar o desempenho em determinadas aplicações.

Em cada iteração, a velocidade de cada partícula no espaço de soluções é atualizada de acordo com a equação (1), a partir da velocidade atual, em função da melhor solução já obtida por aquela partícula até o momento e da melhor solução obtida por todo o grupo de partículas, utilizando pesos diferentes. A posição de cada partícula é atualizada através da adição da velocidade atualizada à posição atual, de acordo com a equação (2).

Portanto, o movimento da partícula no espaço de soluções é direcionado por dois valores: a posição da melhor solução já obtida pela partícula e a posição da melhor solução global, ou seja, as soluções potenciais se movem no espaço de soluções acompanhando as melhores soluções correntes (Figura 18).

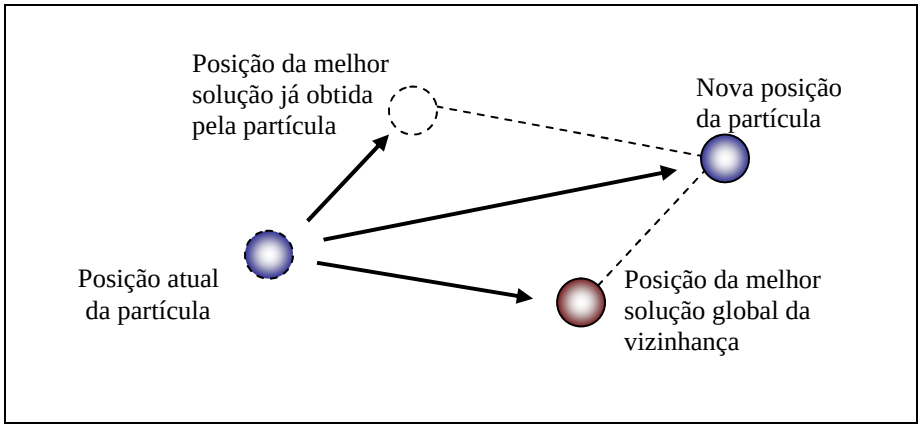


Figura18 – Movimento de uma partícula no espaço de soluções

Uma metáfora que ilustra este comportamento do algoritmo baseado no comportamento de um bando de pássaros é apresentada, segundo Hu, Eberhart e Shi (2004), através do seguinte cenário: um grupo de pássaros, onde cada pássaro representa uma solução isolada (partícula) no espaço de soluções, procura alimento aleatoriamente em uma determinada área. O alimento está localizado em um único local da área que está sendo explorada. Os pássaros não sabem onde o alimento está, mas sabem a sua distância e a posição dos outros pássaros. Neste cenário, a melhor estratégia é seguir o pássaro que estiver localizado à menor distância do alimento.

O algoritmo *PSO* compartilha várias características com os métodos evolutivos. Na fase de inicialização, uma população inicial é gerada aleatoriamente e a busca da solução ótima é efetuada por intermédio da atualização das populações a cada geração, embora o algoritmo *PSO* não utilize os operadores evolutivos. Uma comparação entre os algoritmos genéticos e a otimização por nuvem de partículas é realizada em Eberhart e Shi (1998).

Embora o algoritmo original tenha sido desenvolvido para otimização de problemas contínuos, diversos trabalhos posteriores estenderam a Otimização por Nuvem de Partículas para problemas de otimização combinatória com restrições. A primeira versão discreta do algoritmo *PSO* foi definida em Kennedy e Eberhart (1997, apud HU, EBERHART e SHI, 2004). Um método para resolução de problemas de otimização não-linear com restrições é apresentado em Hu e Eberhart (2002).

Assim como os algoritmos *ACO*, os algoritmos *PSO* também constituem uma aplicação bem-sucedida baseada nos conceitos de *swarm intelligence*.

Em Chu, Chen e Ho (2006, apud POLI, 2007) a Otimização por Nuvem de Partículas é aplicada ao problema de Programação de Grade Horária.

3.3.14 *Opinion Algorithm*

Abordagem proposta por Rossi-Doria e Paechter (2004b) para o problema de Programação de Grade Horária Educacional, baseada na premissa, derivada da observação deste problema, de que a posição relativa dos horários aos quais os eventos (aulas ou provas) estão alocados é mais importante do que os horários em si, na determinação da qualidade da solução.

No *opinion algorithm*, quando um novo evento precisa ser programado na grade horária, cada um dos demais eventos calcula valores representando suas "opiniões", por exemplo, o quanto é bom ou ruim que a alocação seja feita no mesmo horário ou em um horário diferente do dele, no mesmo dia ou em um dia diferente do dele, em um horário adjacente ou não-adjacente ao dele e assim por diante. Estes valores são armazenados em listas de opinião associadas a cada evento, uma para cada tipo de opinião. Quanto maior cada valor, maior é o peso daquela opinião do evento com relação à posição da grade horária onde o evento em análise deve ser alocado.

As listas de opinião exercem no *opinion algorithm* uma função semelhante ao exercido pelo feromônio no algoritmo de Otimização por Colônia de Formigas, porque auxiliam na realização de escolhas probabilísticas durante o processo construtivo e porque são atualizados em cada iteração com informações locais e globais relativas à solução que está sendo construída. A principal diferença com relação ao algoritmo ACO é que pode existir mais de uma informação de feromônio, uma para cada tipo de opinião. Por esta razão, conforme a informação se torna mais extensa, é conveniente a utilização de grafos parciais, ao invés de grafos completos com arestas entre cada par de eventos. Outro motivo é que um evento normalmente não precisa opinar sobre todos os outros eventos, mas somente em poucos relacionamentos mais críticos, sendo a tarefa do algoritmo fazê-los emergir, através das listas de opinião de eventos para cada evento que está sendo atualizado. Rossi-Doria e Paechter (2004b) consideraram listas de comprimento igual a seis, ou seja, cada evento considera a opinião de outros seis eventos em cada tipo de opinião.

Os valores dos votos correspondentes aos diversos eventos e opiniões são somados e a escolha dos horários é efetuada de forma probabilística, de acordo com a proporção de votos atribuídos a cada horário.

Como o algoritmo memético, o *opinion algorithm* é utilizado associado à métodos de busca local.

As listas de opinião são inicializadas a partir de uma solução inicial construída aleatoriamente e refinada através de um método de busca local.

O algoritmo considera para cada evento somente horários viáveis, ou seja, que não provocam violação de restrições rígidas. Se nenhum horário factível está disponível, o número limite total de horários é temporariamente ampliado e um método de busca local é empregado para reduzir este número de volta para o limite de horários disponíveis na grade horária.

O método de busca local também é utilizado em cada iteração para melhorar a qualidade das soluções construídas pelo algoritmo por intermédio da redução do número de restrições flexíveis violadas.

3.3.15 Problema de Satisfação de Restrições (*Constraint Satisfaction Problem-CSP*)

Conforme Barták (1999) uma restrição define uma relação entre um conjunto de variáveis, a qual limita os valores que as variáveis podem assumir simultaneamente.

A Programação por Restrições (*Constraint Programming - CP*), cujas origens estão na Inteligência Artificial, aborda a resolução, através de algoritmos computacionais, de problemas descritos a partir de um conjunto de restrições definidas inicialmente, encontrando uma solução que satisfaça a todas estas restrições.

É dividida em duas áreas (Figura 19):

- Satisfação de Restrições (*Constraint Satisfaction*), que trata de problemas definidos sobre domínios finitos, através de algoritmos baseados em métodos combinatórios;
- Resolução de Restrições (*Constraint Solving*) que lida com restrições definidas sobre domínios infinitos ou mais complexos, utilizando algoritmos baseados em outros métodos matemáticos, como séries de Taylor, por exemplo.

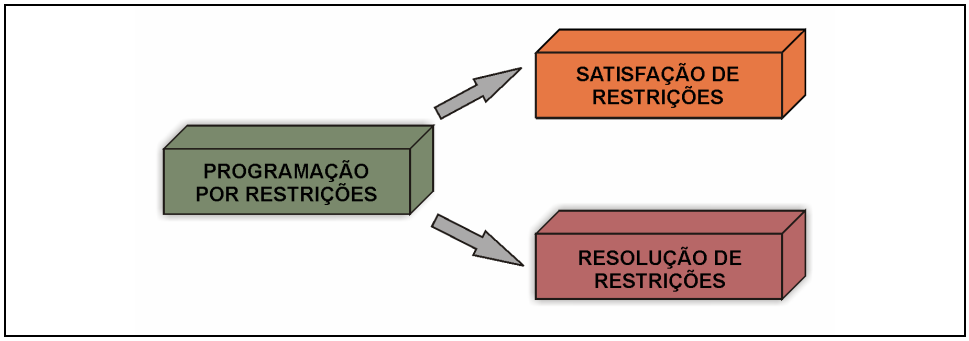


Figura 19 - Programação por Restrições e Problema de Satisfação de Restrições

O Problema de Satisfação de Restrições (CSP) é definido (KUMAR, 1992) como um conjunto de variáveis, um domínio discreto e finito para cada variável e um conjunto de restrições. Cada restrição é definida sobre um subconjunto de variáveis, limitando as combinações de valores possíveis para aquelas variáveis. O objetivo é encontrar um conjunto de valores para as variáveis que satisfaça todas as restrições. Foi tratado inicialmente como um problema algorítmico (MACKWORTH 1977 apud ROY; LIRET; PACHET, 2000), (LAURIÈRE 1978 apud ROY; LIRET; PACHET, 2000).

A programação por Satisfação de Restrições (*Constraint Satisfaction Programming*) (ROY; LIRET; PACHET, 2000) é um paradigma para resolução de problemas combinatoriais, utilizado na resolução do Problema de Satisfação de Restrições, percorrendo de forma sistemática e efetuando cortes no espaço de soluções, através de métodos que verificam as relações de determinadas restrições e atribuições de variáveis.

O exemplo algébrico mais intuitivo de um CSP (KROKHIN, BULATOV e JEAVONS, 2005) é o problema de resolução de um sistema de equações lineares em um domínio finito, onde cada equação individualmente representa uma restrição.

Para o problema de coloração de grafos, os vértices do grafo geralmente correspondem ao conjunto de variáveis da definição do CSP, o conjunto de cores corresponde ao domínio finito e as arestas correspondem às restrições. Uma solução do CSP corresponde a uma coloração do grafo e para minimizar o número de cores são adicionadas restrições artificiais adicionais.

O problema de Programação de Grade Horária Educacional possui domínios finitos, podendo assim ser abordado através do CSP. Em uma formulação possível para este problema, pode-se considerar as aulas como as variáveis do CSP, o conjunto de horários como o domínio finito e as restrições do problema como restrições do CSP. Em uma solução do CSP, o valor atribuído a cada variável corresponderá ao horário para o qual a aula associada à variável foi programada. Yoshikawa e outros (1996 apud SCHAERF,1999a) apresentam uma abordagem baseada em satisfação de restrições para problemas de Programação de Grade Horária Educacional.

3.3.16 Programação em Lógica (*Logic Programming-LP*)

Conforme Kowalski (1988) a programação em lógica utiliza sentenças da lógica matemática para representar conhecimento e é capaz de fazer uso da dedução para a resolução de problemas através da derivação de conseqüências lógicas.

Schaerf (1999a) incluiu a utilização da programação em lógica nas abordagens para problemas de Programação de Grade Horária Educacional, embora caracterize-se mais como uma ferramenta para o desenvolvimento da solução de problemas do que propriamente uma técnica de solução.

Kang e White (1992, apud SCHAERF ,1999a) aplicaram a programação em lógica ao problema de Programação de Grade Horária Escolar, adotando o *Prolog* como linguagem de programação para a implementação da grade horária, de forma a tirar proveito da habilidade desta abordagem em expressar as restrições do problema de

forma declarativa, embora restringida por heurísticas limitadas com relação à capacidade de reprogramação de aulas conflitantes.

3.3.17 Programação em Lógica com Restrições (*Constraint Logic Programming - CLP*)

Gallaire (1985, apud BARTÁK, 1999) e Jaffar e Lassez (1987, apud BARTÁK, 1999) observaram que a Programação em Lógica (*Logic Programming-LP*) é um tipo particular de Programação por Restrições (*Constraint Programming-CP*) (Figura 20).

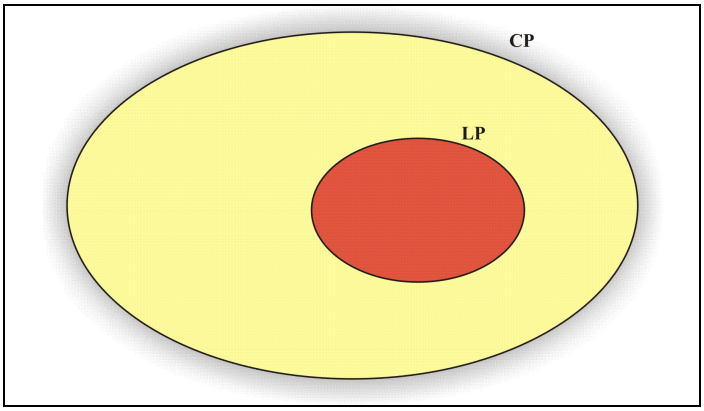


Figura 20 - Programação em Lógica

Foi visto anteriormente que o Problema de Satisfação de Restrições (*CSP*) é um ramo da Programação por Restrições. A Programação em Lógica com Restrições (BURKE et al., 1997) é baseada no desenvolvimento e aplicação de linguagens de Programação em Lógica declarativas, tais como o *Prolog*, aos Problemas de Satisfação de Restrições (Figura 21). Foram implementadas linguagens específicas para CLP, que proporcionam um ambiente declarativo e técnicas voltadas para a resolução do CSP.

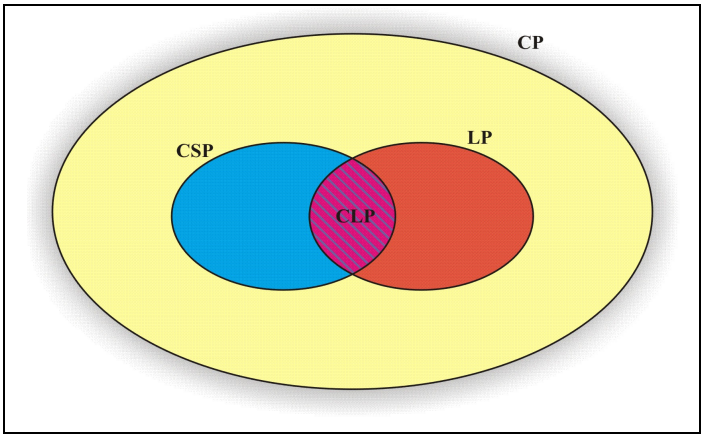


Figura 21 - Programação em Lógica com Restrições

Como o problema de Programação de Grade Horária Educacional pode ser modelado como um CSP, também foram elaboradas abordagens para este problema utilizando CLP. A CLP utiliza uma estratégia de rotulação que determina a ordem em que o espaço de busca é percorrido (ordem na qual as aulas são programadas, ordem em que os horários e as salas são alocados), assegurando a eficácia da exploração. Guéret e outros (1995), Kambi e Gilbert (1996), e Reis e Oliveira (1999) apresentam exemplos de abordagens de problemas de Programação de Grade Horária Educacional através da CLP.

3.3.18 Raciocínio Baseado em Casos (*Case-Based Reasoning – CBR*)

Conforme Burke e outros (2000), na abordagem do Raciocínio Baseado em Casos, um problema é resolvido a partir da adaptação e re-utilização de problemas similares resolvidos anteriormente (*source cases*) armazenados em uma base de casos (*case base*). Neste processo, é utilizada uma medida de similaridade entre os casos contidos na base de casos e o novo problema. Após a resolução, a base de casos é atualizada com o novo caso tratado. Burke e outros (2000, 2006), e Burke, Petrovic e Qu, (2006) apresentam aplicações do Raciocínio Baseado em Casos aos problemas de Programação de Grade Horária Educacional.

3.3.19 Lógica Difusa (*Fuzzy Logic*)

Conforme Asmuni e outros (2005), os sistemas difusos (*fuzzy systems*) são utilizados para representar e aplicar conhecimento que é impreciso, incerto ou não confiável. Petrovic, Patel e Yong (2005) aplicaram métodos difusos a um problema de Programação de Grade Horária Educacional, a fim de modelar e avaliar a satisfação de restrições flexíveis, conforme a seguinte descrição:

Nas abordagens tradicionais do problema de Programação de Grade Horária, a função objetivo geralmente utiliza lógica binária para avaliar a satisfação das restrições flexíveis na solução, ou seja, a restrição é satisfeita ou não. Entretanto, este modelo pode acarretar dificuldades em determinadas situações.

Um exemplo típico ocorre no problema de Programação de Grade Horária de Provas, na restrição de que provas mais extensas devem ser programadas primeiro na grade horária. É difícil avaliar tal restrição utilizando lógica binária. Por isso, para que a satisfação da restrição seja realizada de forma gradual, podem ser empregados os conjuntos difusos (*fuzzy sets*): **baixo, médio-baixo, médio, médio-alto e alto**. Regras Se-Então difusas (*fuzzy IF-THEN rules*) podem ser definidas para derivar o grau de satisfação da restrição, utilizando as variáveis lingüísticas **Tamanho da prova (pequeno, médio e grande)** e **Período de tempo** para o qual a prova é alocada **(início, meio e fim)** com relação à sua posição na grade horária.

Outro exemplo é a restrição de proximidade de horários de provas, que requer que alunos não tenham provas marcadas em períodos de tempo adjacentes. Analogamente, podem ser definidas as variáveis lingüísticas **Distância** entre duas provas e **Número de alunos** que as duas provas têm em comum. Termos lingüísticos **pequeno, médio e grande** podem ser utilizados na avaliação destas duas variáveis lingüísticas.

O resultado da utilização das regras Se-Então difusas definidas para cada restrição é um conjunto difuso que expressa o grau de satisfação e a medida da violação destas restrições flexíveis.

Asmuni e outros (2005) aplicam métodos difusos na construção das grades horárias em um problema de Programação de Grade Horária de Provas, de forma a considerar múltiplas heurísticas de ordenação simultaneamente, a fim de ordenar as provas pelo grau de dificuldade em serem programadas.

3.3.20 Redes Neurais

Redes Neurais Artificiais são sistemas inspirados na analogia com o processamento paralelo e distribuído de informações pela rede de neurônios do cérebro humano. Os estímulos processados no cérebro por um neurônio são transmitidos aos neurônios vizinhos por meio de conexões denominadas sinapses. Este processo pode se repetir em várias camadas de neurônios, que formam a rede neural (FERNEDA, 2006).

A Figura 22 mostra a representação simplificada de um neurônio e uma rede neural do cérebro humano (CORMACK, 2003).

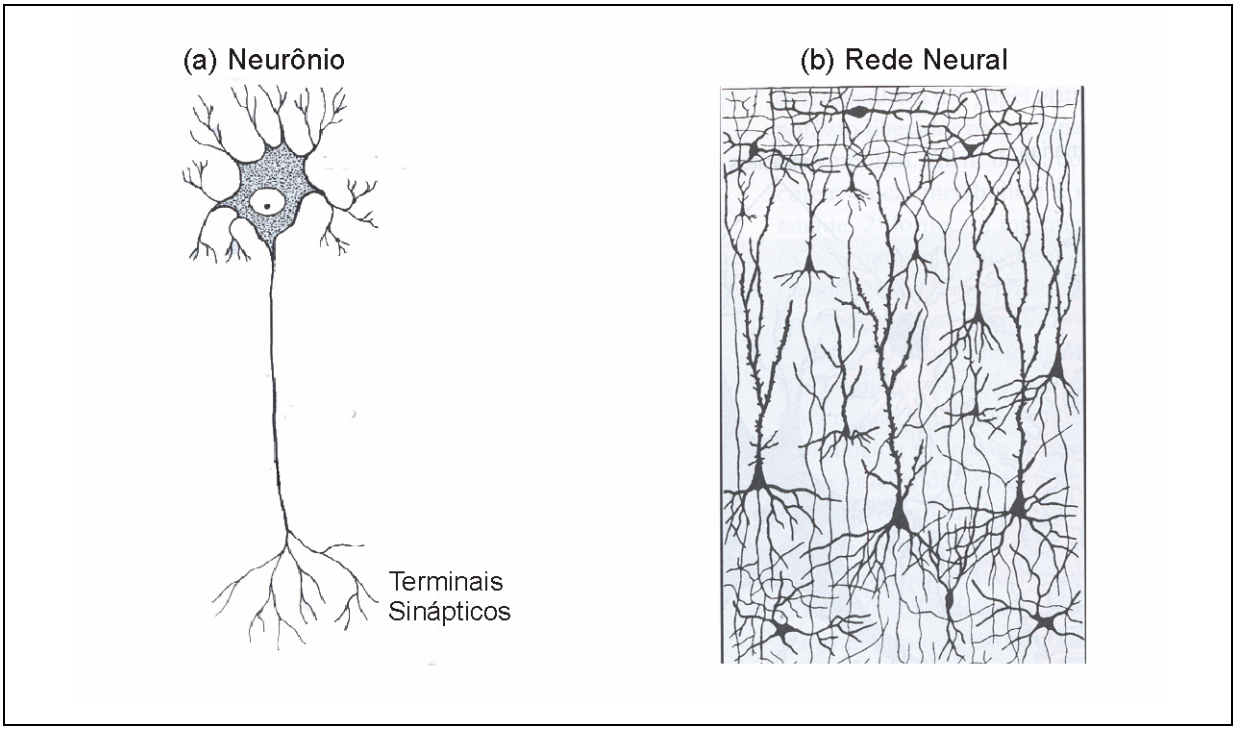


Figura 22 – (a) Neurônio (b) Rede neural (CORMACK, 2003)

Hopfield e Tank (1985 apud Carrasco; Pato, 2001) apresentaram um modelo de rede neural, utilizado na resolução de problemas de otimização.

A rede neural é uma rede (digrafo com valores numéricos associados aos arcos) onde os nós são agrupados em múltiplas camadas (*multilayer*), sendo uma camada de nós de entrada, uma camada de nós de saída e camadas intermediárias de nós internos ou ocultos (*hidden*). Os nós internos de cada camada são totalmente interconectados com os nós da camada seguinte.

Em função da analogia biológica citada acima, na rede neural os nós são denominados neurônios binários e os arcos são denominados conexões sinápticas.

O comportamento das conexões sinápticas entre os neurônios é determinado pelos seus pesos, denominados pesos sinápticos. O peso sináptico de um neurônio *i* para um neurônio *j* é representado por p_{ij}.

A Figura 23 apresenta um exemplo de representação simplificada de uma rede neural.

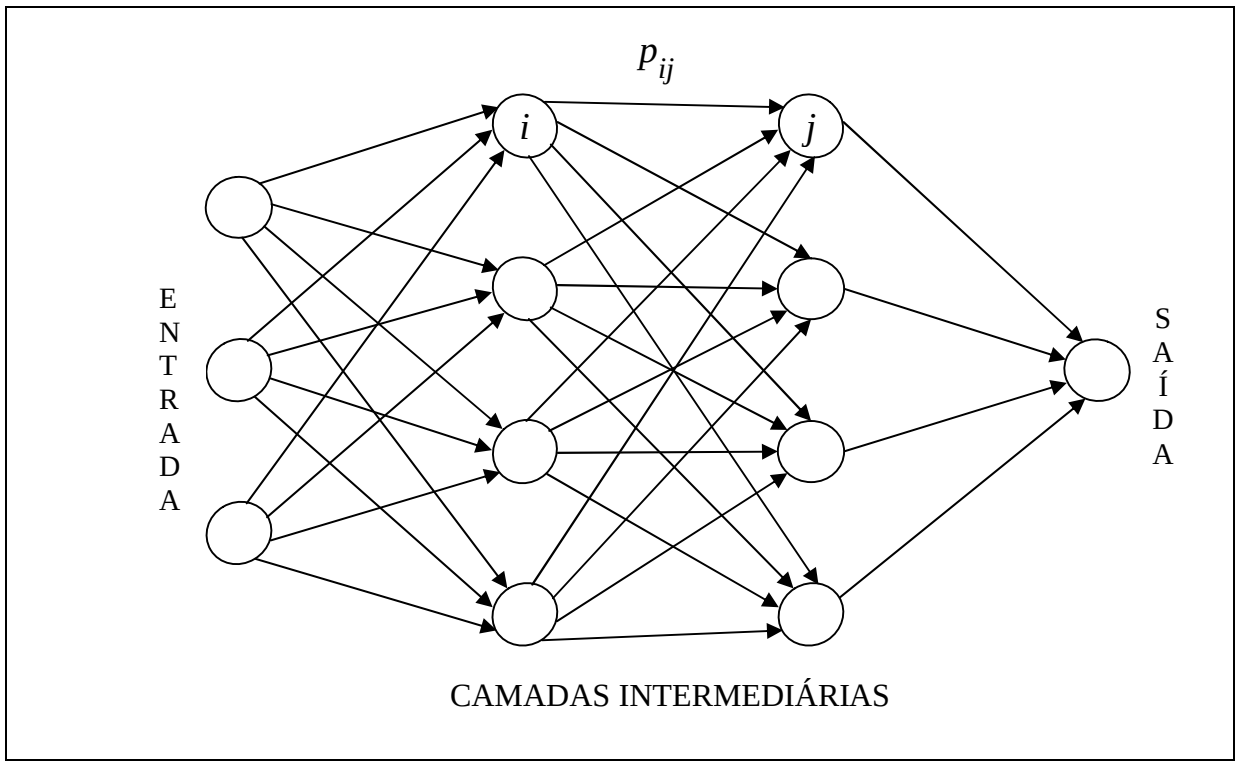


Figura 23 – Representação simplificada de uma rede neural

Cada neurônio *i* possui um valor de entrada X_i e um valor de saída Y_i.

Para os neurônios que representam nós de entrada, o valor de entrada é atribuído.

Para os neurônios das camadas subseqüentes, o valor de entrada X_i de um neurônio *i* é obtido através da soma ponderada dos valores de saída que chegam até ele, provenientes de outros neurônios da camada anterior, adicionada a uma corrente de polarização negativa (*bias*) I_i, cujo efeito é o de um peso sináptico conectado a uma entrada constante, conforme expressão apresentada a seguir:

$$X_i^{(t)} = X_i^{(t-1)}(1-\Delta t) + \Delta t\left(\sum_j p_{ij} Y_j^{(t-1)} + I_i\right) \qquad (1)$$

O valor de saída Y_i, limitado ao intervalo [0,1], é calculado por uma função de ativação $f_i(X_i)$, como a seguinte função sigmóide (o gráfico da função é em forma de S):

$$Y_i = f_i(X_i) = \frac{1}{2}\left(1 + \tanh\left(\frac{X_i}{T}\right)\right) \qquad (2)$$

onde *T* é um parâmetro de temperatura que determina a declividade.

A Figura 24 apresenta o comportamento desta função sigmóide para três valores diferentes do parâmetro T e a Figura 25 mostra o modelo esquemático de um neurônio artificial contendo os elementos descritos acima.

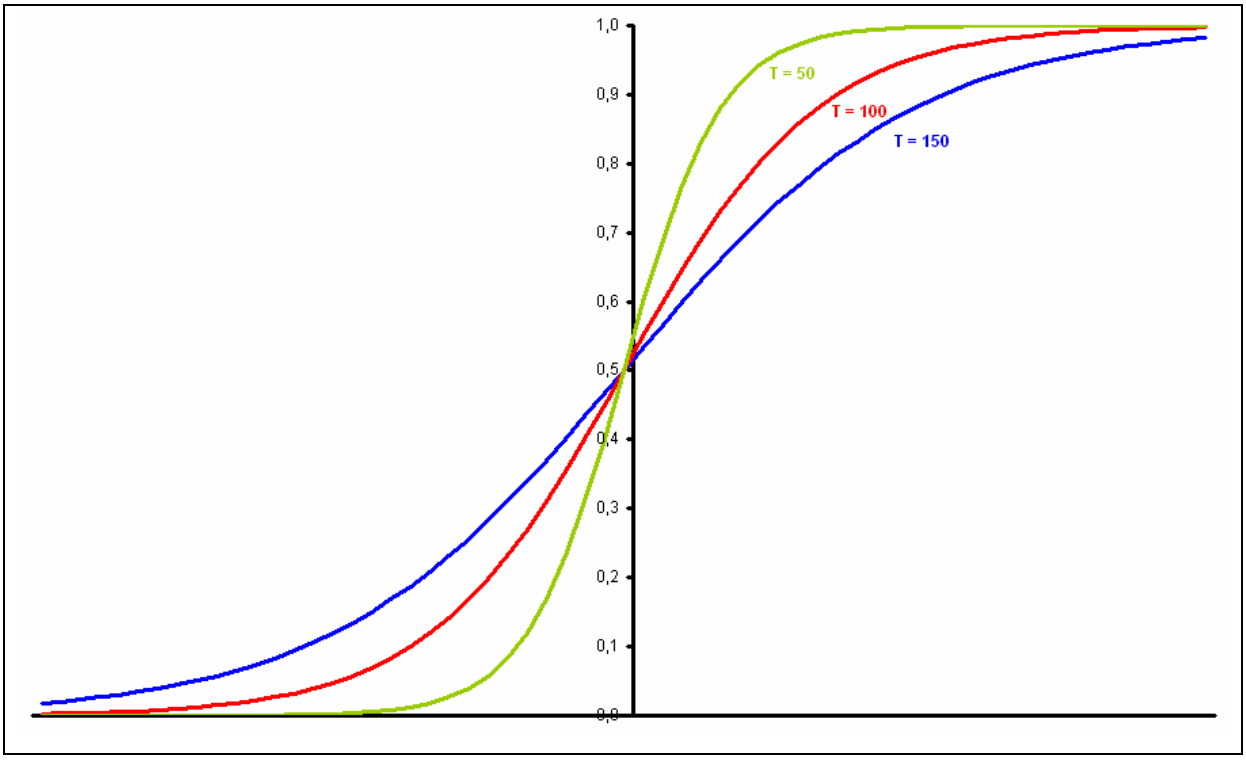


Figura 24 – Função sigmóide $Y_i = f_i(X_i) = \frac{1}{2}\left(1 + \tanh\left(\frac{X_i}{T}\right)\right)$, para diferentes valores do parâmetro T.

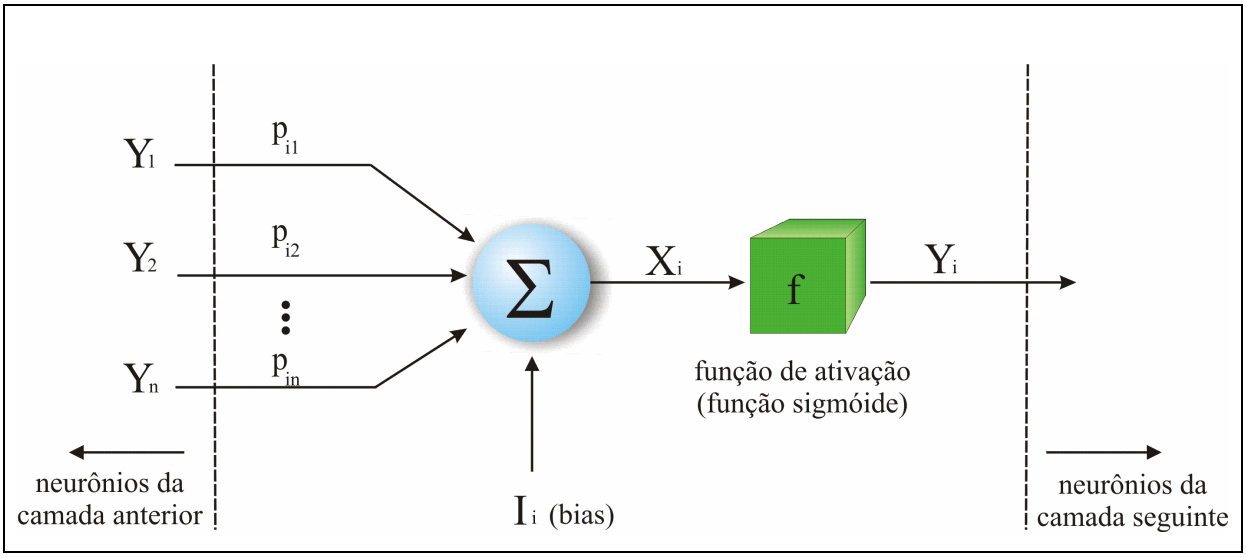


Figura 25 - Modelo esquemático de um neurônio artificial – adaptado de Ferneda (2006)

O processo se repete e os resultados são propagados através das camadas de neurônios, até alcançar os nós de saída.

Hopfield e Tank provaram que, através da utilização das equações acima, para uma matriz simétrica de pesos ($p_{ij} = p_{ji}$) e um parâmetro T apropriado, a rede neural converge para o valor mínimo da função apresentada a seguir, denominada função de energia:

$$E(\overline{Y}) = -\frac{1}{2}\sum_{i}\sum_{j} p_{ij} Y_i Y_j - \sum_{i} I_i V_i \qquad (3)$$

Diversos problemas de otimização combinatória podem ser minimizados utilizando a abordagem de redes neurais apresentada. Para mapear cada problema, deve ser definida uma função de energia apropriada, de forma que o valor mínimo corresponda ao valor ótimo do problema a ser otimizado. Normalmente a função é definida como uma soma ponderada, cujos pesos são as restrições do problema, acrescida dos termos correspondentes à função objetivo, efetuando o balanceamento necessário entre cada restrição e objetivo (Carrasco; Pato, 2001).

Gislen, Peterson e Soderberg (1989), Gianoglio (1990, apud MCGREEVY et al., 2004), Lim e Loe (1991, apud MCGREEVY et al., 2004), Kovacic (1993, apud MCGREEVY et al., 2004), Pellerin e Herault (1994, apud MCGREEVY et al., 2004), Tsuchiya e Takefuji (1997, apud MCGREEVY et al., 2004), Smith, Abramson e Duke (1999, 2003) e Carrasco e Pato (2001) abordaram a resolução de problemas de Programação de Grade Horária Educacional utilizando Redes Neurais.

3.3.21 Hiper-Heurísticas

A seguir são apresentados exemplos da aplicação de hiper-heurísticas, definidas na Subseção 3.3.1, a problemas de Programação de Grade Horária Educacional.

Em Burke e outros (2007) uma hiper-heurística utiliza Busca Tabu como heurística de alto-nível para pesquisar listas de permutações em um conjunto de heurísticas construtivas de coloração de grafo, funcionando como heurísticas de baixo nível, as quais são utilizadas para a construção das grades horárias (soluções) em problemas de Programação de Grade Horária Educacional.

Em Burke, Kendall, Soubeiga, (2003) também é utilizada a Busca Tabu como heurística de alto-nível na hiper-heurística para problemas de Programação de Grade Horária Educacional. Porém, neste trabalho as heurísticas de baixo-nível são heurísticas simples, tais como: deslocamento de uma aula, escolhida de forma aleatória, de seu horário atual para outro horário também escolhido aleatoriamente; troca mútua (*swap*) dos horários de duas aulas escolhidas aleatoriamente; variações destas duas heurísticas, porém primeiro reduzindo o número de violações de restrições rígidas ou ainda primeiro reduzindo o número de violações de restrições flexíveis sem aumentar o número de violações de restrições rígidas.

Rossi-Doria e Paechter (2003) abordam o problema de Programação de Grade Horária Educacional através de uma hiper-heurística que utiliza Algoritmo Evolutivo como heurística de alto-nível e heurísticas construtivas de coloração de grafo como heurísticas de baixo nível.

Segundo Bai e outros (2006) o *Simulated Annealing* é utilizado como heurística de alto-nível na hiper-heurística para abordagem do problema de Programação de Grade Horária Educacional, juntamente com as heurísticas de baixo nível utilizadas em Burke, Kendall e Soubeiga (2003), descritas anteriormente.

4 COLORAÇÃO DE GRAFOS

4.1 HISTÓRICO, FORMULAÇÕES E APLICAÇÕES

Conforme Diestel (2005), a coloração de grafos é uma área da teoria dos grafos que se originou no estudo do problema das quatro cores, apresentado pelos irmãos Francis e Frederick Guthrie em 1852, que consiste em provar a hipótese que qualquer mapa planar pode ser colorido com no máximo quatro cores, de forma que as regiões adjacentes recebam cores diferentes (Figura 26).

Figura 26 – Mapa colorido com 4 cores - Mapa de Divisões Territoriais do Brasil
Fonte: Instituto Brasileiro de Geografia e Estatística-IBGE, acesso em 02 out. 2007.

Associando as regiões do mapa a serem coloridas aos vértices de um grafo e ligando com arestas pares de vértices do grafo que representam no mapa regiões fronteiriças

(Figura 27), o problema de coloração de grafos correspondente consiste em provar o teorema das quatro cores, apresentado a seguir:

Teorema 8.1 (Teorema das quatro cores): Todo grafo planar é 4 - colorível.

A primeira demonstração para este teorema foi obtida por K. Appel e W. Haken em 1977, com o auxílio de computadores.

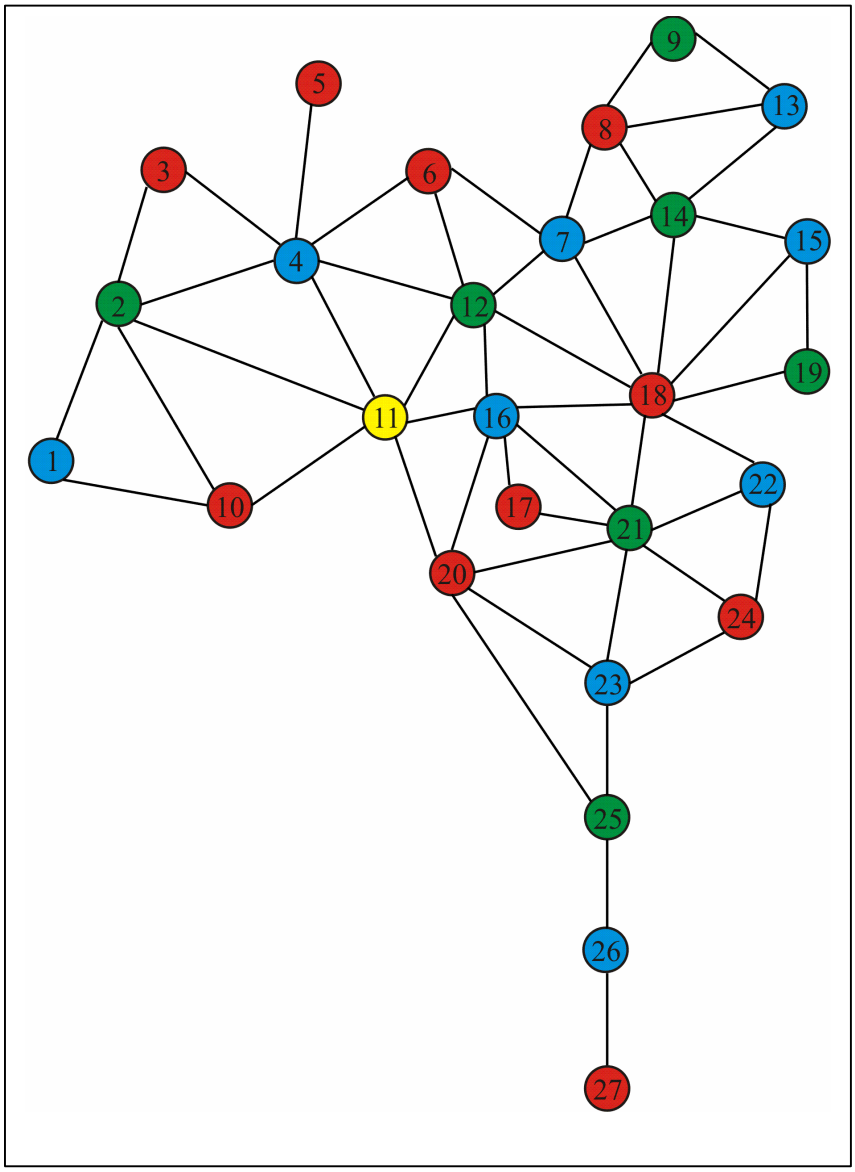


Figura 27 – Uma 4-coloração do grafo planar correspondente ao mapa da Figura 26

Conforme conceitos apresentados na Subseção 2.1.2, o problema de coloração de grafos (*Graph Coloring Problem-GCP*) pode ser formulado como problema de decisão, que consiste em determinar se existe uma coloração de um determinado grafo G com um determinado número de cores *k*, de modo que vértices adjacentes tenham cores diferentes e como problema de otimização, onde o objetivo é encontrar uma coloração com um número mínimo de cores (número cromático *c* (G)), de modo que vértices adjacentes tenham cores diferentes. O primeiro é NP-Completo e o segundo é NP-Difícil. No restante deste trabalho, sempre que o problema de coloração de grafos for mencionado, estará se referindo ao problema de otimização.

Conforme conceitos apresentados na Subseção 2.2.4, do ponto de vista da Otimização Combinatória, o *GCP* pode ser estabelecido como um problema de atribuição, onde um conjunto de *k* cores deve ser atribuído a um conjunto de *n* vértices do grafo; ou como um problema de partição de conjuntos, no qual uma k-coloração viável corresponde a uma partição do conjunto de vértices do grafo em *k* subconjuntos independentes, de maneira que não exista nenhuma aresta ligando dois vértices de uma mesma classe de cores.

O *GCP* pode ser aplicado na resolução de diversos problemas, entre os quais o problema de Programação de Grade Horária Educacional (apresentado na Seção 3.2), alocação de registros de computadores (CHOW; HENNESSY, 1984, 1990, apud MEHROTRA; TRICK, 1996), problemas de alocação de freqüência (GAMST, 1986, apud MEHROTRA; TRICK, 1996), entre outras áreas.

4.2 O FORMATO *DIMACS* PARA ARQUIVOS CONTENDO GRAFOS

Dimacs (*Center for Discrete Mathematics and Theoretical Computer Science*) é um centro de ciência e tecnologia, fundado em 1989 em Nova Jérsei, Estados Unidos. Atua como um projeto colaborativo entre universidades e centros de pesquisas de empresas norte-americanas, dedicado ao desenvolvimento dos campos inter-

relacionados da matemática discreta e da ciência da computação teórica, abrangendo tanto o desenvolvimento teórico destas áreas quanto suas aplicações práticas.

Desde 1991 são realizados seminários e conferências, conhecidos como Desafios *Dimacs* (*Dimacs Implementation Challenges*), com o propósito de estimular pesquisas nas áreas de interesse citadas, sendo abordado uma ou mais classes de problemas diferentes em cada evento. Um dos problemas pesquisados no Segundo Desafio *Dimacs*, ocorrido em 1993, foi o problema de coloração de grafos.

Um resultado obtido através dos Desafios *Dimacs* foi a definição de um formato padrão de arquivos de entrada e de saída, conhecido como formato *Dimacs*, destinado a armazenar grafos para utilização por algoritmos implementados em programas de computador. Este padrão foi definido com o objetivo de facilitar testes e comparações entre os algoritmos, permitindo a utilização das mesmas instâncias.

A seguir serão descritos alguns parâmetros do formado *Dimacs* de arquivo de entrada:

- Linhas de Comentário: São utilizadas para registrar informações no arquivo com o propósito de documentação. Podem ser escritas em qualquer ponto do arquivo e são ignoradas pelos programas. Cada linha de comentário começa com o caractere minúsculo ***c***.

  Sintaxe: ***c*** Esta é uma linha de comentário.

- Linha de descrição do problema: Informa o número de vértices e o número de arestas do grafo. Cada arquivo de entrada contém apenas uma linha de descrição do problema, escrita antes das linhas de descrição de aresta, com o seguinte formato:

  Sintaxe: ***p FORMAT NODES EDGES***

  O caractere minúsculo ***p*** indica que a linha é de descrição do problema. O campo ***FORMAT*** se deve a compatibilidade com versões anteriores do Desafio

Dimacs e deve conter a palavra "***edge***". O campo ***NODES*** contém um valor inteiro especificando o número de vértices do grafo. O campo ***EDGE*** contém um valor inteiro especificando o número de arestas do grafo.

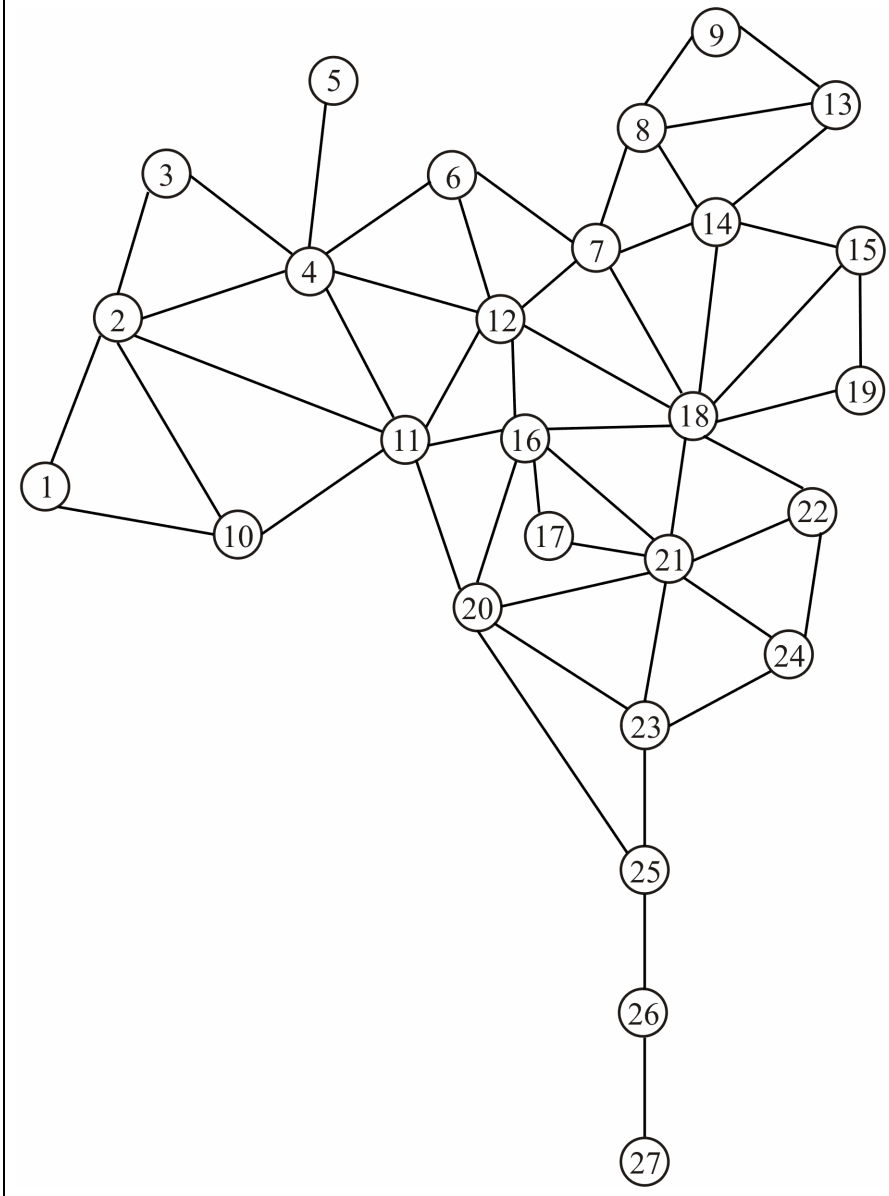
- Linha de descrição de aresta: Cada aresta ***(v, w)*** do grafo é representada por uma linha de descrição de aresta, com o seguinte formato:

  Sintaxe: ***e V W***

  O caractere minúsculo ***e*** indica que a linha é de descrição de aresta. Os campos ***V*** e ***W*** especificam os vértices terminais da aresta.

A Figura 28 descreve o arquivo de entrada no formato *Dimacs* para o grafo apresentado na Figura 27 da Seção 4.1. As primeiras três linhas são comentários, a quarta linha contém o número de vértices e o número de arestas do grafo e as demais linhas informam os números dos vértices terminais de cada aresta. (CLIQUE..., 1993), (MISSION, acesso em 02 out. 2007), (DIMACS, acesso em 02 out. 2007)

Este formato para arquivos de entrada é utilizado nos testes computacionais desta dissertação.




```
c Grafo do Mapa do Brasil
c formato Dimacs
c Geraldo Simonetti Bello
p edge  27  51
e  1  2
e  1  10
e  2  3
e  2  4
e  2  10
e  2  11
e  3  4
e  4  5
e  4  6
e  4  11
e  4  12
e  6  7
e  6  12
e  7  8
e  7  12
e  7  14
e  7  18
e  8  9
e  8  13
e  8  14
e  9  13
e  10  11
e  11  12
e  11  16
e  11  20
e  12  16
e  12  18
e  13  14
e  14  15
e  14  18
e  15  18
e  15  19
e  16  17
e  16  18
e  16  20
e  16  21
e  17  21
e  18  19
e  18  21
e  18  22
e  20  21
e  20  23
e  20  25
e  21  22
e  21  23
e  21  24
e  22  24
e  23  24
e  23  25
e  25  26
e  26  27
```

Figura 28 – (a) Grafo de 27 vértices e 51 arestas – (b) Listagem de arquivo contendo a representação deste grafo no formato *Dimacs.*

4.3 ABORDAGENS PARA RESOLUÇÃO DO PROBLEMA DE COLORAÇÃO DE GRAFOS

Devido ao GCP ser NP-difícil, não é esperado que um algoritmo exato possa encontrar, em tempo polinomial, uma solução para uma instância qualquer do problema. Apenas alguns grafos com pequeno número de vértices e arestas podem ser resolvidos por algoritmos exatos, sendo necessária a utilização de algoritmos aproximados para a coloração de grafos maiores. Diversas heurísticas e meta-heurísticas vêm sendo empregadas para este fim. A seguir, serão apresentadas algumas das principais abordagens propostas ao longo do tempo.

4.3.1 Coloração Seqüencial – Algoritmo Guloso (*Greedy Algorithm*)

Segundo Klotz (2002) e Culberson (1992, 1997), um método simples de coloração é a coloração seqüencial.

Um algoritmo de coloração de grafos seqüencial é denominado algoritmo guloso. Esta heurística construtiva efetua uma coloração de vértices $c:V \rightarrow \{1, 2, \ldots, k\}$ a partir de uma determinada ordenação $O = \{v_{p_1}, \ldots, v_{p_n}\}$ dos elementos do conjunto de vértices $V = \{v_1, \ldots, v_n\}$ do grafo $G = (V, E)$, onde p é uma permutação de $\{1 \ldots n\}$.

O algoritmo guloso simples é apresentado na Figura 29.

Início

1. c(v_{p_1}) = 1
2. **para** i **de** 2 **a** n **faça**
3. k = 1
4. f = 0
5. **enquanto** f = 0 **faça**
6. c(v_{p_i}) = k
7. f = 1
8. **para** j **de** 1 **a** i – 1 **faça**
9. **se** $\{v_{p_i}, v_{p_j}\} \in E$ **e** c(v_{p_i}) = c(v_{p_j}) **então**
10. f = 0
11. **fim se**
12. **fim para**
13. **se** f = 0 **então**
14. k = k + 1
15. **fim se**
16. **fim enquanto**
17. **fim para**

Fim

Figura 29 – Algoritmo Guloso Simples – adaptado de Culberson (1992)

O algoritmo guloso simples seleciona um vértice de cada vez em alguma ordem pré-definida e tenta colorir o vértice selecionado com uma das cores já utilizadas, ou seja, tenta adicioná-lo a uma das classes de cores existentes. Se isto não é possível, então uma nova classe de cores é criada e o vértice recebe a cor desta classe.

Variantes do algoritmo guloso apresentado acima podem ser obtidas, estabelecendo diferentes ordenações das classes de cores disponíveis para a pesquisa da cor a ser atribuída ao vértice que está sendo colorido, selecionando a primeira classe de cores encontrada na ordem de pesquisa indicada. A seguir são apresentadas cinco algoritmos de coloração seqüencial obtidos da forma descrita:

- Algoritmo Guloso Simples (*Simple Greedy*): percorre as classes de cores na ordem *1, . . ., k.* Esta é a técnica utilizada no algoritmo apresentado acima. Quando um algoritmo de coloração seqüencial é mencionado sem nenhuma outra especificação adicional, normalmente está se referindo ao algoritmo guloso simples.

- Algoritmo Guloso Maior Classe Primeiro (*Largest First Greedy*): ordena as classes de cores pela quantidade de vértices associados a cada uma delas no

momento da ordenação e então efetua a pesquisa por ordem decrescente de tamanho. Esta técnica tende a formar classes de cores maiores que os outros métodos.

- Algoritmo Guloso Menor Classe Primeiro (*Smallest First Greedy*): ordena as classes de cores pela quantidade de vértices associados a cada uma delas no momento da ordenação e então efetua a pesquisa por ordem crescente de tamanho. Esta técnica tende a equalizar o tamanho das classes de cores.

- Algoritmo Guloso de Seqüência Aleatória (*Random Sequence Greedy*): pesquisa as classes de cores em uma ordem aleatória.

- Algoritmo Guloso de Ordem Reversa (*Reverse Order Greedy*): percorre as classes de cores na ordem *k, . . ., 1*.

Além disso, de acordo com a definição dada anteriormente para o algoritmo guloso, cada um dos cinco algoritmos gulosos apresentados acima pode utilizar diferentes ordenações iniciais para os vértices do grafo. A seguir são apresentadas sete ordenações que podem ser utilizadas para os vértices:

- Ordem Direta (*In Order*): apresenta os vértices na ordem *1, . . ., n*.

- Ordem Decrescente de Grau (*Decreasing Degree*), também denominada Maior Grau Primeiro (*Largest First-LF*): ordena os vértices por ordem decrescente de grau.

- Ordem Crescente de Grau (*Increasing Degree*), também denominada Menor Grau Primeiro (*Smallest First-SF*): ordena os vértices por ordem crescente de grau.

- Ordem Aleatória (*Random*): apresenta os vértices em uma ordem aleatória.

As três últimas ordenações a serem apresentadas são baseadas em uma variante da Busca em Largura (*Breadth-First Search - BFS*) denominada Busca em Largura Lexicográfica (*Lexicographic Breadth-First Search - LBFS*), proposta por Rose, Tarjan e

Lueker (1976, apud HSU; MA, 1999). Na Ordenação Lexicográfica, cada vértice do grafo recebe um rótulo de *n* dígitos, todos inicialmente iguais a zero, onde *n* é o número de vértices do grafo. Após o vértice *i* da ordenação ser escolhido, o *i-ésimo* dígito dos rótulos de todos os vértices adjacentes a *i* é alterado para "1". Então o próximo vértice *(i+1)* é selecionado, escolhendo o vértice de maior rótulo entre os vértices ainda não ordenados, considerando o primeiro dígito do rótulo como o mais significativo. Os vértices podem ser classificados previamente por ordem decrescente de grau, ordem crescente de grau ou ordem aleatória, com o propósito de assegurar um critério de desempate quando mais de um vértice for elegível de ser incluído na ordenação lexicográfica (HSU; MA, 1999). Portanto, as três ordenações adicionais são:

- Ordem Lexicográfica Aleatória (*LBFS Random*): utiliza Busca Lexicográfica, com os vértices classificados previamente em ordem aleatória, para escolha em caso de empate.
- Ordem Lexicográfica Decrescente de Grau (*LBFS Decreasing Degree*): utiliza Busca Lexicográfica, com os vértices classificados previamente em ordem decrescente de grau, de forma a ser escolhido o vértice de maior grau em caso de empate.
- Ordem Lexicográfica Crescente de Grau (*LBFS Increasing Degree*): utiliza Busca Lexicográfica, com os vértices classificados previamente em ordem crescente de grau, de forma a ser escolhido o vértice de menor grau em caso de empate.

Combinando as sete ordenações de vértices aos cinco algoritmos gulosos apresentados acima, podem ser obtidas trinta e cinco variações diferentes de algoritmos gulosos, identificadas pelo número N_g e relacionadas no Quadro 1.

N_g	TIPO DE ALGORITMO GULOSO - ORDEM INICIAL DOS VÉRTICES DO GRAFO
1	Algoritmo Guloso Simples - Ordem Direta
2	Algoritmo Guloso Simples - Ordem Decrescente de Grau
3	Algoritmo Guloso Simples - Ordem Crescente de Grau
4	Algoritmo Guloso Simples - Ordem Aleatória
5	Algoritmo Guloso Simples – Ordem Lexicográfica Aleatória
6	Algoritmo Guloso Simples – Ordem Lexicográfica Decrescente de Grau
7	Algoritmo Guloso Simples - Ordem Lexicográfica Crescente de Grau
8	Algoritmo Guloso Maior Classe Primeiro - Ordem Direta
9	Algoritmo Guloso Maior Classe Primeiro - Ordem Decrescente de Grau
10	Algoritmo Guloso Maior Classe Primeiro - Ordem Crescente de Grau
11	Algoritmo Guloso Maior Classe Primeiro - Ordem Aleatória
12	Algoritmo Guloso Maior Classe Primeiro – Ordem Lexicográfica Aleatória
13	Algoritmo Guloso Maior Classe Primeiro – Ordem Lexicográfica Decrescente de Grau
14	Algoritmo Guloso Maior Classe Primeiro - Ordem Lexicográfica Crescente de Grau
15	Algoritmo Guloso Menor Classe Primeiro - Ordem Direta
16	Algoritmo Guloso Menor Classe Primeiro - Ordem Decrescente de Grau
17	Algoritmo Guloso Menor Classe Primeiro - Ordem Crescente de Grau
18	Algoritmo Guloso Menor Classe Primeiro - Ordem Aleatória
19	Algoritmo Guloso Menor Classe Primeiro – Ordem Lexicográfica Aleatória
20	Algoritmo Guloso Menor Classe Primeiro – Ordem Lexicográfica Decrescente de Grau
21	Algoritmo Guloso Menor Classe Primeiro - Ordem Lexicográfica Crescente de Grau
22	Algoritmo Guloso de Seqüência Aleatória - Ordem Direta
23	Algoritmo Guloso de Seqüência Aleatória - Ordem Decrescente de Grau
24	Algoritmo Guloso de Seqüência Aleatória - Ordem Crescente de Grau
25	Algoritmo Guloso de Seqüência Aleatória - Ordem Aleatória
26	Algoritmo Guloso de Seqüência Aleatória – Ordem Lexicográfica Aleatória
27	Algoritmo Guloso de Seqüência Aleatória – Ordem Lexicográfica Decrescente de Grau
28	Algoritmo Guloso de Seqüência Aleatória - Ordem Lexicográfica Crescente de Grau
29	Algoritmo Guloso de Ordem Reversa - Ordem Direta
30	Algoritmo Guloso de Ordem Reversa - Ordem Decrescente de Grau
31	Algoritmo Guloso de Ordem Reversa - Ordem Crescente de Grau
32	Algoritmo Guloso de Ordem Reversa - Ordem Aleatória
33	Algoritmo Guloso de Ordem Reversa – Ordem Lexicográfica Aleatória
34	Algoritmo Guloso de Ordem Reversa – Ordem Lexicográfica Decrescente de Grau
35	Algoritmo Guloso de Ordem Reversa - Ordem Lexicográfica Crescente de Grau

Quadro 1 – Variações de Algoritmos Gulosos

4.3.2 Algoritmo Guloso Iterativo (*Iterated Greedy Algorithm*)

Conforme Culberson (1992, 1997) esta heurística parte de uma solução inicial e tenta melhorá-la através de repetidas utilizações dos algoritmos gulosos descritos na Subseção 4.3.1, a fim de obter uma nova coloração dos vértices do grafo com número de cores menor que a anterior.

A escolha do tipo de algoritmo guloso, em função da forma de ordenação das classes de cores e da forma de ordenação inicial dos vértices do grafo é efetuada de modo probabilístico pelo algoritmo, a partir de pesos definidos como dados de entrada.

4.3.3 Algoritmo Maior Primeiro Recursivo (*Recursive Largest First-RLF)*

O algoritmo *RLF*, proposto por Leighton (1979, apud VESEL; ZEROVNIK, 2000), constrói as classes de cores seqüencialmente, determinando os vértices de uma classe de cores de cada vez.

Seja:

- C_i a classe de cores i a ser construída.
- U o conjunto de vértices não coloridos que podem pertencer a C_i.
- B o conjunto de vértices não coloridos que não podem pertencer a C_i por serem adjacentes a pelo menos um vértice em C_i.

A construção de cada classe de cores C_i pelo algoritmo *RLF* é efetuada conforme apresentado na Figura 30.

Início
1. $B \leftarrow f$;
2. Escolha $v \in U$ | v é o vértice de maior grau em U ;
3. Mova v de U para C_i ;
4. Mova de U para B todos os vértices $u \in U$ | u é adjacente a v ;
5. **Enquanto** $U \neq f$ faça
6. Escolha $v \in U$ | v é vértice terminal da maior quantidade de arestas incidentes a vértices de B ;
7. Mova v de U para C_i ;
8. Mova de U para B todos os vértices $u \in U$ | u é adjacente a v ;

Fim

Figura 30 – Trecho do algoritmo *RLF* (VESEL; ZEROVNIK, 2000)

Uma variante do *RLF* com um fator de aleatoriedade, denominada XRLF, foi apresentada por Johnson e outros (1991, apud LAGUNA; MARTÍ, 2001).

4.3.4 Algoritmo *Dsatur* (*Saturation Degree*)

Brelaz (1979) definiu o grau de saturação de um vértice v de um grafo G como o número de cores diferentes atribuídas aos vértices adjacentes a v em uma coloração parcial de G.

Na Figura 31 é apresentado o algoritmo *Dsatur* proposto por Brelaz.

Início

1. Ordene os vértices por ordem decrescente de grau.
2. Atribua a cor 1 ao vértice de maior grau.
3. Escolha o vértice não colorido com o maior grau de saturação. Em caso de empate, escolha o vértice não colorido de maior grau.
4. Atribua ao vértice escolhido a menor cor possível.
5. Se todos os vértices foram coloridos, pare. Caso contrário, retorne ao passo 3.

Fim

Figura 31 – O algoritmo *Dsatur*

4.3.5 Algoritmo *Dsatur* com Retrocesso (*Backtrack Dsatur)*

Trata-se de uma versão do algoritmo *Dsatur* com mecanismo de *backtrack* (retrocesso) incorporado, o qual retorna ao vértice colorido mais recentemente para explorar novas alternativas de coloração, quando não é possível atribuir uma cor válida para o vértice de maior grau de saturação, selecionado para ser colorido durante a execução do algoritmo (BRELAZ, 1979), (CULBERSON, 1997).

Klotz (2002) utiliza a denominação Coloração Seqüencial com Retrocesso (*Backtracking Sequential Coloring-BSC*) ao apresentar uma versão deste algoritmo.

4.3.6 Busca Local Iterativa (*Iterated Local Search-ILS)*

Chiarandini e Stützle (2002) abordam o problema de Coloração de Grafos utilizando a Busca Local Iterativa, que consiste em repetidas aplicações de busca local sobre soluções iniciais obtidas através da modificação progressiva de uma coloração inicial gerada previamente pelo algoritmo *Dsatur*. A coloração inicial foi implementada por Mehrotra e Trick (1996). A modificação consiste em seqüencialmente diminuir de uma unidade o número de cores da solução e reaplicar o algoritmo *Dsatur*, até que a coloração obtida deixe de ser factível. Então a Busca Local é utilizada para encontrar uma solução factível. A nova solução passa a ser tratada pelo algoritmo *Dsatur*, e o número de cores volta a ser diminuído sucessivamente. O processo se repete até que a Busca Local não consiga mais obter uma solução viável, para um determinado número de cores.

4.3.7 Outras abordagens para coloração de grafos

Abordagens apresentadas na Seção 3.3 para modelagem e resolução do problema de Programação de Grade Horária Educacional podem ser aplicados também para a resolução do problema de coloração de grafos.

A seguir são relacionadas algumas destas abordagens:

- Programação Inteira (MEHROTRA; TRICK, 1996);
- *Simulated Annealing* (CHAMS; HERTZ; DE WERRA, 1987, apud GALINIER; HERTZ, 2006), (JOHNSON et al., 1991, apud GALINIER; HERTZ, 2006);
- Busca Tabu (HERTZ; DE WERRA,1987, apud HERTZ; TAILLARD; DE WERRA, 1997); (GONZÁLEZ-VELARDE; LAGUNA, 2002); (AGUIAR et al., 2005). A meta-heurística Busca Tabu será detalhada na Seção 4.4;
- GRASP (LAGUNA; MARTÍ, 2001);
- Algoritmos Genéticos e Meméticos (DAVIS, 1991, apud GALINIER; HERTZ, 2006), (EIBEN; VAN DER HAUW E; VAN HEMERT, 1997), (DORNE; HAO, 1998), (GALINIER; HAO, 1999), (MARINO; PRÜGEL-BENNETT; GLASS, 1999), (CROITORU et al., 2002);
- Problema de Satisfação de Restrições (CSP), onde as variáveis são os vértices $v_1, v_2, ..., v_n$, o domínio são as cores {1, 2, . . . , k} e as restrições são $v_i \neq v_j$ se v_i e v_j são adjacentes (VAN HEMERT, 2003). A Programação em Lógica com Restrições (CLP) também pode ser utilizada, conforme descrito na Subseção 3.3.17;
- Redes Neurais (DAHL, 1987, apud LAGUNA; MARTÍ, 2001), (JAGOTA, 1996, apud LAGUNA; MARTÍ, 2001);
- Colônia de Formigas: (COMELLAS; OZON, 1998), (VESEL; ZEROVNIK, 2000), (SHAWE-TAYLOR; ZEROVNIK, 2001), (HERTZ; ZUFFEREY, 2006).

4.4 RESOLUÇÃO DO PROBLEMA DE COLORAÇÃO DE GRAFOS UTILIZANDO BUSCA TABU

Conforme Hertz, Taillard e De Werra (1997), embora as origens da Busca Tabu remontem a década de 1970, as primeiras apresentações desta técnica, na forma utilizada atualmente, foram propostas em trabalhos separados por Glover (1986) e Hansen (1986).

Uma característica importante da Busca Tabu é a utilização sistemática de memória. Enquanto a maioria dos outros métodos de busca mantém memorizado apenas o melhor valor encontrado da função objetivo, a Busca Tabu mantém também a informação dos caminhos para as soluções visitadas mais recentemente.

Dado um conjunto S de soluções factíveis e uma função objetivo $f : S \rightarrow \Re$, o processo de busca em S é descrito em termos de movimentos de uma solução atual para a próxima. Para cada solução $s \in S$ é definido o conjunto $M(s)$ de movimentos m_i que podem ser aplicados a s para obter uma nova solução s_i (esta operação é representada por $s_i = s \oplus m_i$). A vizinhança $N(s)$ de uma solução s é definida por $N(s) = \{s_i \mid \exists m_i \in M(s), s_i = s \oplus m_i\}$. Os movimentos são reversíveis: para cada movimento m_i existe um movimento m_i^{-1}, tal que $(s \oplus m_i) \oplus m_i^{-1} = s$. Ao invés de armazenar as últimas $|\overline{T}|$ soluções visitadas, uma lista tabu $\overline{T}$ mantém memória dos últimos $|\overline{T}|$ movimentos ou movimentos reversos correspondentes.

A Busca Tabu utiliza esta informação para evitar a ocorrência de ciclos durante a escolha do movimento para a próxima solução na vizinhança da solução atual, restringindo movimentos que levam às soluções visitadas recentemente. Por isso, a vizinhança *N(s)* de uma solução *s* varia a cada iteração.

Desde que um critério de aspiração seja atendido, é possível aceitar a aplicação de um movimento m em uma solução corrente s, ainda que m esteja na lista tabu. Para isso, são calculados os valores $a(s,m)$ e $A(s,m)$, denominados nível de aspiração e função de aspiração, respectivamente. Se $a(s,m) < A(s,m)$ então a condição tabu de

m com relação a s é cancelada. Por exemplo, se forem definidos $a(s,m) = f(s \oplus m)$ e $A(s,m) = f(s^*)$, onde s^* é a melhor solução obtida até aquele momento, então o *status* tabu de m será removido se a solução $s' = s \oplus m$ for melhor que s^*.

A Busca Tabu possui critérios de parada: quando o valor de f para a solução corrente s é suficientemente próxima de um limitante inferior f^* do valor mínimo de f ou quando não ocorre melhoria de s^* durante um número estipulado *nmax* de iterações.

A Figura 32 apresenta o algoritmo Busca Tabu descrito acima (HERTZ, 1992).

Início
1. $s \leftarrow s_0$; (* solução inicial em S *)
2. *nbiter* $\leftarrow 0$; (* iteração atual *)
3. $s^* \leftarrow s$; (* melhor solução *)
4. *bestiter* $\leftarrow$ 0; (* iteração na qual a melhor solução foi obtida *)
5. $\overline{T} \leftarrow f$; (* Lista tabu *)
6. inicialize a função de aspiração A;
7. **enquanto** ($f(s) > f^*$) **e** (*nbiter* - *bestiter* < *nmax*) **faça**
8. *nbiter* $\leftarrow$ *nbiter* + 1;
9. gere conjunto N^* de soluções $s_i = s \oplus m_i$ em $N(s)$ tal que $m_i \notin \overline{T}$ ou $a(s,m_i) < A(s,m_i)$;
10. escolha a melhor solução s' em N^*;
11. atualize a lista tabu $\overline{T}$ e a função de aspiração A;
12. **se** $f(s') < f(s^*)$ **então**
13. $s^* \leftarrow s'$, *bestiter* $\leftarrow$ *nbiter*;
14. **fim se**
14. $s \leftarrow s'$;
15. **fim enquanto**

Fim

Figura 32 – O algoritmo Busca Tabu – adaptado de (HERTZ, 1992).

Dois recursos adicionais, intensificação e diversificação, podem ser incorporados na Busca Tabu com o propósito de aperfeiçoar a pesquisa das soluções.

- Intensificação: Concentra a busca em regiões específicas de S, durante algumas iterações, através da priorização das soluções que possuem características comuns com a solução atual. Para isso, é acrescentado um termo adicional na função objetivo que penaliza soluções distantes da solução corrente.

- Diversificação: Distribui a busca sobre diferentes regiões de S, introduzindo um termo adicional na função objetivo que penaliza soluções próximas da solução atual.

É possível alternar as fases de intensificação e diversificação durante a busca, modificando os pesos dos termos correspondentes durante o processo.

O processo de armazenamento dos caminhos para as soluções visitadas mais recentemente através da lista tabu é denominado memória de curto prazo. Estratégias de intensificação e diversificação caracterizam a utilização de memória de longo prazo (HERTZ; TAILLARD; DE WERRA, 1997), (HERTZ, 1992).

Hertz e De Werra (1987, apud GALINIER; HERTZ, 2006) desenvolveram o algoritmo *Tabucol*, uma versão da meta-heurística Busca Tabu voltada para a coloração de vértices de grafos. Desde então, este algoritmo vem sendo utilizado em diversos trabalhos, algumas vezes incorporado como uma sub-rotina (AVANTHAY; HERTZ; ZUFFEREY, 2003), (GALINIER; HERTZ; ZUFFEREY, 2003), (GALINIER; HAO, 1999), (DORNE; HAO, 1998). Embora alguns autores tenham introduzido posteriormente modificações no algoritmo, as principais características do *Tabucol* original permanecem as mesmas (GALINIER; HERTZ, 2006).

Galinier e Hertz (2006) apresentaram uma versão do *Tabucol*, descrita a seguir, que representa um balanço deste algoritmo e uma sinopse de adaptações efetuadas na versão inicial nestes vinte anos de existência.

O *Tabucol* gera aleatoriamente uma s-coloração inicial, normalmente contendo arestas conflitantes. A cada iteração, o algoritmo modifica a cor de um único vértice terminal de uma aresta conflitante, com o objetivo de diminuir progressivamente o número de arestas conflitantes, até que uma s-coloração viável seja obtida. Uma lista tabu é utilizada para evitar a retenção em ótimos locais e a ocorrência de ciclos. Os critérios de parada são a obtenção de uma s-coloração viável ou a não ocorrência de melhoria da solução durante um número estipulado de iterações. Outros critérios podem ser utilizados como segundo critério de parada, tais como um valor limite

para o número total de iterações ou ainda um valor limite para o tempo de processamento.

O *Tabucol* não utiliza estratégias de memória de longo prazo, tais como intensificação ou diversificação.

O espaço de busca S explorado pelo algoritmo é o conjunto de s -colorações de um grafo $G=(V,E)$. Conforme apresentado na Subseção 2.2.4, uma solução $c \in S$, correspondendo a uma coloração dos vértices de G em s cores, é uma partição do conjunto de vértices V em s subconjuntos independentes $C_1,...C_s$, denominados classe de cores.

Seja $E(C_k)$ o conjunto das arestas com ambos vértices terminais na classe de cores C_k. Para uma solução $c=(C_1,...,C_s)$, a função objetivo a ser minimizada é $\overline{f}(c)=\sum_{k=1}^{s}|E(C_k)|$, a qual totaliza o número de arestas conflitantes. Portanto, uma solução $c \in S$ tal que $\overline{f}(c)=0$ corresponde a uma s -coloração viável de vértices de G.

Seja $m_{v,k}$ o movimento elementar de um único vértice v para uma classe de cores C_k (alteração da cor de um único vértice v). O desempenho de um movimento $m_{v,k}$ em uma solução $c \in S$ é avaliado pelo parâmetro $d(v,k)=\overline{f}(c \oplus m_{v,k})-\overline{f}(c)$.

A vizinhança $N(c)$ de uma solução $c \in S$ consiste de todas as s -colorações que podem ser obtidas a partir de c, pela aplicação de um único movimento $m_{v,k}$ em um vértice v, passando de uma classe de cores $C_{c(v)}$ para outra classe C_k, com $k \neq c(v)$. Portanto, $N(c)$ contém $|V|.(s-1)$ soluções.

Os vértices terminais de arestas conflitantes são denominados vértices críticos. Em uma solução $c \in S$, o conjunto de todos os vértices críticos será denotado por V_c e o número de vértices críticos será denotado $|V_c|$. Um movimento $m_{v,k}$ que envolve um

vértice crítico v é denominado movimento crítico. Por questões de eficiência, todo movimento realizado no algoritmo *Tabucol* é aplicado em um vértice crítico, ou seja, o algoritmo somente realiza movimentos críticos. O conjunto de todos os $|V_c|.(s-1)$ movimentos críticos será denotado por M_c.

Quando um vértice v é movido de $C_{c(v)}$ para C_k em uma solução c, o movimento $m_{v,c(v)}$ (ou o movimento reverso $m^{-1}_{v,c(v)}$) é inserido na lista tabu $\overline{T}$, proibindo que o vértice v volte para C_k durante um determinado número de iterações, denominado período tabu (*tabu tenure*). Nesta versão do algoritmo, o período tabu varia em função do número de vértices críticos na solução c e depende ainda de dois parâmetros p e q. Assim, quando o movimento $m_{v,k}$ é aplicado na solução c, o movimento $m_{v,c(v)}$ se torna tabu durante $(p.|V_c|+q)$ iterações. Os parâmetros p e q são determinados experimentalmente, em função das instâncias utilizadas. Galinier e Hao (1999), por exemplo, após testar várias combinações, sugeriram a utilização de $p=10$ e $q=0{,}6$ para os grafos tratados em seus experimentos.

O algoritmo utiliza como critério de aspiração $\overline{f}(c \oplus m_{v,k}) = 0$, ou seja, a condição de tabu de um movimento $m_{v,k}$ é cancelada se o movimento leva a uma coloração viável c. Neste caso, o processo de busca é terminado pelo critério de parada correspondente e o algoritmo retorna a coloração viável c.

A Figura 33 apresenta o algoritmo *Tabucol* descrito acima (GALINIER; HERTZ, 2006), (HERTZ; DE WERRA, 1987, apud GALINIER; HERTZ, 2006), (HERTZ; TAILLARD; DE WERRA, 1997).

Início

1. $c \leftarrow c_0$; (* solução inicial em S gerada aleatoriamente *)
2. nbiter $\leftarrow$ 0; (* iteração atual *)
3. $c^* \leftarrow c$; (* melhor solução *)
4. bestiter $\leftarrow$ 0; (* iteração na qual a melhor solução foi obtida *)
5. $\overline{T} \leftarrow f$; (* inicialmente nenhum movimento é tabu *)
6. **enquanto** ($\overline{f}(c^*) > 0$) **e** (nbiter - bestiter < nmax) **faça**
7. nbiter $\leftarrow$ nbiter + 1;
8. para $v \in V_c$ escolha o movimento crítico $m_{v,k} \in M_c$ com mínimo valor para o parâmetro $d(v,k) = \overline{f}(c \oplus m_{v,k}) - \overline{f}(c)$, tal que $m_{v,k} \notin \overline{T}$ ou $\overline{f}(c \oplus m_{v,k}) = 0$;
9. introduza o movimento crítico $m_{v,c(v)}$ na lista tabu durante $p.|V_c| + q$ iterações; (* atualiza a lista tabu $\overline{T}$ e o *tabu tenure* *)
10. $c' \leftarrow c \oplus m_{v,k}$;
11. **se** $\overline{f}(c') < \overline{f}(c^*)$ **então**
12. $c^* \leftarrow c'$, bestiter $\leftarrow$ nbiter;
13. **fim se**
14. $c \leftarrow c'$;
15. **fim enquanto**

Fim

Figura 33 – O algoritmo *Tabucol* (GALINIER; HERTZ, 2006), (HERTZ; DE WERRA, 1987, apud GALINIER; HERTZ, 2006), (HERTZ; TAILLARD; DE WERRA, 1997).

Adicionalmente, duas abordagens foram propostas para utilização em conjunto com o algoritmo *Tabucol*, para melhorar o desempenho na coloração de grafos grandes (a partir de 300 vértices). A primeira abordagem (HERTZ; DE WERRA, 1987 apud GALINIER; HERTZ, 2006) consiste em utilizar um algoritmo de pré-processamento que extrai *n* conjuntos independentes do grafo, obtendo um grafo residual que é tratado pelo algoritmo *Tabucol*, e então são adicionadas *n* cores à coloração obtida pelo *Tabucol*. A segunda abordagem consiste na utilização do *Tabucol* integrado a variantes de algoritmos meméticos, de duas formas: combinado com a extração de conjuntos independentes através de algoritmo de pré-processamento (FLEURENT; FERLAND, 1996 apud GALINIER; HERTZ, 2006) e sem efetuar a extração de conjuntos independentes (DORNE; HAO, 1998), (GALINIER; HAO, 1999).

Joseph Culberson, da Universidade de Alberta, disponibilizou em sua página na Internet (GRAPH, 2004) o código-fonte de um programa de computador escrito em linguagem C, contendo a implementação de uma versão do algoritmo *Tabucol*. Esta implementação apresenta diferenças em relação ao algoritmo *Tabucol* original de

Hertz e De Werra e em relação à versão apresentada acima (GALINIER; HERTZ, 2006). A seguir são relacionadas algumas destas diferenças:

Ao invés de gerar a solução inicial aleatoriamente, como no algoritmo mostrado na Figura 33, a implementação de Culberson do algoritmo *Tabucol* utiliza uma solução gerada por outros algoritmos de coloração de grafos. Alguns destes algoritmos também foram implementados em linguagem C e disponibilizados no site (GRAPH, 2004). Entre estes algoritmos estão o algoritmo guloso, o algoritmo guloso iterativo, o algoritmo *Dsatur* e o algoritmo *Dsatur* com retrocesso, apresentados na Seção 4.3. Caso nenhuma solução inicial seja disponibilizada externamente da forma descrita, o programa gera uma solução trivial, a qual consiste em atribuir uma cor diferente para cada vértice, produzindo assim uma $|V|$-coloração, onde $|V|$ é o número de vértices do grafo.

Outra diferença diz respeito à estrutura de vizinhança utilizada. No algoritmo da Figura 33, a cada iteração é escolhido o melhor movimento entre todos os movimentos críticos, ou seja, considerando todos os vértices críticos $v \in V_c$. Na versão do *Tabucol* implementada por Culberson, assim como no algoritmo original de Hertz e De Werra, o melhor movimento é selecionado, a cada iteração, dentro de um subconjunto M_c^* ($M_c^* \subset M_c$) de movimentos críticos $m_{v,k}$ escolhidos aleatoriamente sobre uma amostra V_c^* de vértices críticos v, escolhidos também aleatoriamente em V_c, ou seja, $v \in V_c^*$ e $V_c^* \subset V_c$. O programa armazena os vértices críticos em uma lista (*conflictList*) e seleciona aleatoriamente os elementos do conjunto V_c^* a partir desta lista. O número de elementos do conjunto V_c^* varia entre os limites $n_{\min_V_c^*}$ e $n_{\max_V_c^*}$, da seguinte forma: Se durante a geração de V_c^* e M_c^* for selecionado um vértice v e um movimento $m_{v,k}$ que produzam um resultado melhor que a melhor solução encontrada até o momento ($\overline{f}(c \oplus m_{v,k}) < \overline{f}(c^*)$), o algoritmo conclui a seleção, desde que V_c^* contenha no mínimo $n_{\min_V_c^*}$ vértices. Caso isto não ocorra, o processo continua até que V_c^* complete $n_{\max_V_c^*}$ vértices.

O tamanho da lista tabu varia proporcionalmente ao número de vértices críticos ($p.|V_c|+q$) na versão do algoritmo apresentada por Galinier e Hertz (2006). Na implementação de Culberson, assim como no algoritmo original de Hertz e De Werra, o tamanho da lista tabu é constante durante todo o processo e definido como um parâmetro de entrada (*tabusize*). Esta opção é adequada, ao ser combinada com o uso de uma amostra aleatória de vértices críticos V_c^* (ao invés do conjunto completo V_c).

O desempenho de um movimento $m_{v,k}$ em uma solução $c \in S$ é avaliado através do valor de $\overline{f}(c \oplus m_{v,k})$ na implementação de Culberson.

Na Figura 34 é apresentado o algoritmo *Tabucol* correspondente à implementação de Culberson descrita acima.

Início

1. $c \leftarrow c_0$; (* solução inicial em S gerada por outro algoritmo ou solução trivial *)
2. nbiter $\leftarrow$ 0; (* iteração atual *)
3. $c^* \leftarrow c$; (* melhor solução *)
4. bestiter $\leftarrow$ 0; (* iteração na qual a melhor solução foi obtida *)
5. $\overline{T} \leftarrow f$; (* inicialmente nenhum movimento é tabu *)
6. $|\overline{T}| \leftarrow tabusize$; (* tamanho da lista tabu *)
7. **enquanto** ($\overline{f}(c^*) > 0$) **e** (nbiter - bestiter < nmax) **faça**
8. nbiter $\leftarrow$ nbiter + 1;
9. **enquanto** ($|V_c^*| < n_{\max_V_c^*}$) **e** (($\overline{f}(c \oplus m_{v,k}) \geq \overline{f}(c^*)$) **ou** ($|V_c^*| < n_{\min_V_c^*}$)) **faça**
10. gere aleatoriamente um subconjunto $V_c^* \subset V_c$ de vértices críticos e um subconjunto $M_c^* \subset M_c$ de movimentos críticos tal que $m_{v,k} \notin \overline{T}$;
11. **fim enquanto**
12. para $v \in V_c^*$ escolha o movimento crítico $m_{v,k} \in M_c^*$ com mínimo valor de $\overline{f}(c \oplus m_{v,k})$;
13. $\overline{T} \leftarrow \overline{T} \bigcup \{m_{v,c(v)}\}$; (* atualiza a lista tabu $\overline{T}$ *)
14. $c' \leftarrow c \oplus m_{v,k}$;
15. **se** $\overline{f}(c') < \overline{f}(c^*)$ **então**
16. $c^* \leftarrow c'$, bestiter $\leftarrow$ nbiter; (* atualiza melhor solução e função de aspiração *)
17. **fim se**
18. $c \leftarrow c'$;
19. **fim enquanto**

Fim

Figura 34 – Versão do algoritmo *Tabucol* implementada por Joseph Culberson (GRAPH, 2004).

No Quadro 2 abaixo, é mostrado um comparativo das duas versões apresentadas do algoritmo *Tabucol*, contendo um resumo das características descritas acima.

	VERSÃO DO *TABUCOL*	
	GALINIER E HERTZ	**CULBERSON**
SOLUÇÃO INICIAL	Aleatória	Externa ou Trivial
VÉRTICES CONSIDERADOS	V_c (todos os vértices críticos)	V_c^* (amostra aleatória de vértices críticos)
MOVIMENTOS CONSIDERADOS	M_c (todos os movimentos críticos)	M_c^* (amostra aleatória de movimentos críticos)
PARÂMETRO DE AVALIAÇÃO DO MOVIMENTO $m_{v,k}$	$d(v,k) = \overline{f}(c \oplus m_{v,k}) - \overline{f}(c)$	$\overline{f}(c \oplus m_{v,k})$
TAMANHO DA LISTA TABU/PERÍODO TABU	Variável ($p.\|V_c\| + q$)	Fixo

Quadro 2 – Quadro comparativo de duas versões do algoritmo *Tabucol*.

5 REFORMULAÇÃO DE UM PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA ESCOLAR COM RESTRIÇÕES RÍGIDAS E FLEXÍVEIS

Neste capítulo, é proposto um processo para resolução de um Problema de Programação de Grade Horária Escolar com restrições rígidas e flexíveis escolhido na literatura, a partir de sua reformulação como um problema de coloração de grafos.

5.1 O PROBLEMA DE PROGRAMAÇÃO DE GRADE HORÁRIA ESCOLAR COMO PROBLEMA DE COLORAÇÃO DE GRAFOS

O Problema de Programação de Grade Horária Escolar, que será denotado no restante deste trabalho como PPGHE, foi definido por Gotlieb (1963, apud NEUFELD; TARTAR, 1974) da forma apresentada a seguir:

Dado:

- Um conjunto de professores $T = \{t_i\}, i = 1,...,a$;
- Um conjunto de turmas $C = \{c_j\}, j = 1,...,b$;
- Um conjunto de horários $H = \{h_k\}, k = 1,...s$;
- Uma matriz de requerimentos $a \times b$ $R = [r_{ij}]$, onde $r_{ij} \geq 0$, e r_{ij} é igual à quantidade de horários que o professor t_i se encontra com a turma c_j para lecionar;

A grade horária deve satisfazer restrições de indisponibilidade e eventos pré-fixados.

As restrições de indisponibilidade são descritas pelas matrizes *D* e *E*:

- $D = [d_{ik}]$ é uma matriz $a \times s$, com $d_{ik} = 1$ se o professor t_i está indisponível no horário h_k e $d_{ik} = 0$ caso contrário;
- $E = [e_{jk}]$ é uma matriz $b \times s$, com $e_{jk} = 1$ se a turma c_j está indisponível no horário h_k e $e_{jk} = 0$ caso contrário;

Os eventos pré-fixados são descritos por conjuntos $P_{ij}, i = 1,...,a; j = 1,...,b$:

- $P_{ij} = \{h_{ij}^k \in H \mid$ o professor t_i está com a turma c_j no horário $h_k\}$. $P_{ij} = f$ se r_{ij} = 0 ou se não existem eventos pré-fixados envolvendo o professor t_i e a turma c_j.

A seguir apresentamos resultados propostos por Neufeld e Tartar (1974, 1975) e baseados nestas proposições, que são utilizados neste trabalho:

- O professor t_i administra r_{ij} aulas para a turma c_j.
- A r-ésima aula do professor t_i para a turma c_j é denotada por m_{ij}^r $(1 \leq r \leq r_{ij})$.
- O conjunto $M_{ij} = \{m_{ij}^1, m_{ij}^2, ..., m_{ij}^{r_{ij}}\}$ representa as r_{ij} aulas do professor t_i com a turma c_j.
- O conjunto $M_j = \bigcup_{i=1}^{a} M_{ij}$ contém as $\sum_{i=1}^{a} r_{ij}$ aulas da turma c_j $(\sum_{i=1}^{a} r_{ij} \leq s)$.
- O conjunto $M_i = \bigcup_{j=1}^{b} M_{ij}$ contém as $\sum_{j=1}^{b} r_{ij}$ aulas do professor t_i $(\sum_{j=1}^{b} r_{ij} \leq s)$.
- $M = \bigcup_{i=1}^{a} M_i = \bigcup_{j=1}^{b} M_j = \bigcup_{i=1}^{a} \bigcup_{j=1}^{b} M_{ij}$ é o conjunto de todas as $\sum_{i=1}^{a} \sum_{j=1}^{b} r_{ij}$ aulas.

Conforme apresentado na Subseção 3.3.4, o grafo associado ao PPGHE é denominado grafo adjunto.

O grafo adjunto *G=(V,E)* é definido da seguinte forma:

- Defina o conjunto de vértices V tal que cada vértice $v \in V$ corresponda a uma aula $m_{ij}^r \in M$ e seja denotado por v_{ij}^r $(r = 1,...,r_{ij}; i = 1,...,a; j = 1,...,b)$.
- Defina $V_{ij}, V_i, V_j \subseteq V$ correspondendo a $M_{ij}, M_i, M_j \subseteq M$, respectivamente.
- Defina o conjunto de arestas $E = E^1 \cup E^2$, onde:

- $E^1 = \bigcup_{j=1}^{b} E_j$ e $E_j = \{\{v, v'\} \mid v, v' \in V_j, v \neq v'\}$

- $E^2 = \bigcup_{i=1}^{a} E_{ijj'}$ e $E_{ijj'} = \begin{cases} \{\{v, v'\} \mid v \in V_{ij}, v' \in V_{ij'}\}, se\ j \neq j', r_{ij} > 0, r_{ij'} > 0 \\ f, caso\ contrário \end{cases}$

As arestas no conjunto E^1 correspondem no PPGHE às restrições que as turmas não podem estar com mais de um professor em um mesmo horário e precisam estar com cada professor o número requerido de horários.

As arestas no conjunto E^2 correspondem no PPGHE às restrições que os professores não podem estar com mais de uma turma em um mesmo horário e precisam estar com cada turma o número requerido de horários.

Uma solução do PPGHE descreve os horários $h_k \in H$, $k = 1, \ldots, s$ programados para todas as $\sum_{i=1}^{a} \sum_{j=1}^{b} r_{ij}$ aulas $m_{ij}^r \in M$ dos professores $t_i \in T$, $i = 1, \ldots, a$ com as turmas $c_j \in C$, $j = 1, \ldots, b$.

A seguir são apresentados lemas, corolários e teoremas destacados do trabalho de Neufeld e Tartar (1974, 1975), que são utilizados neste trabalho.

O Lema 12.1 a seguir relaciona as soluções de um PPGHE sem restrições de indisponibilidade e eventos pré-fixados às s -colorações do grafo adjunto *G*:

Lema 12.1 (Neufeld e Tartar): Para o PPGHE sem restrições de indisponibilidade e eventos pré-fixados, existe uma relação um-para-um entre o conjunto de todas as soluções *S* do problema e o conjunto *U* de todas as s -colorações do grafo adjunto *G*.

O Corolário 12.1 a seguir estabelece a condição necessária e suficiente para a existência de uma solução para o PPGHE sem restrições de indisponibilidade e eventos pré-fixados:

Corolário 12.1 (Neufeld e Tartar): Existe uma solução para o PPGHE sem restrições de indisponibilidade e eventos pré-fixados se, e somente se, existe uma s -coloração para o grafo adjunto *G*.

Os Lemas 12.2 e 12.3 mostram como cada restrição de indisponibilidade do PPGHE corresponde a uma condição que impede que determinados vértices do grafo adjunto G sejam coloridos com uma cor específica e o Corolário 12.2 generaliza o resultado do Lema 12.2:

Lema 12.2 (Neufeld e Tartar): Para o PPGHE com uma restrição de indisponibilidade descrita pelo elemento $d_{ik} = 1$ da matriz *D* e sem eventos pré-fixados, existe uma relação um-para-um entre o conjunto de todas as soluções *S* do problema e o conjunto *U* de todas as s -colorações do grafo adjunto *G*, sujeito à condição que nenhum vértice nos conjuntos V_{ij}, $j = 1, \ldots, b$ seja colorido com a cor l_k.

Corolário 12.2 (Neufeld e Tartar): Para o PPGHE com restrições de indisponibilidade descritas pelos elementos $d_{ik} = 1$ da matriz *D* e sem eventos pré-fixados, existe uma relação um-para-um entre o conjunto de todas as soluções *S* do problema e o conjunto *U* de todas as s -colorações do grafo adjunto *G*, sujeitas à condição que, para cada $d_{ik} = 1$, nenhum vértice nos conjuntos V_{ij}, $j = 1, \ldots, b$ seja colorido com a cor l_k.

Lema 12.3 (Neufeld e Tartar): Para o PPGHE com restrições de indisponibilidade descritas pelos elementos $e_{jk} = 1$ da matriz *E* e sem eventos pré-fixados, existe uma relação um-para-um entre o conjunto de todas as soluções *S* do problema e o conjunto *U* de todas as s -colorações do grafo adjunto *G*, sujeitas à condição que, para cada $e_{jk} = 1$, nenhum vértice nos conjuntos V_{ij}, $i = 1, \ldots, a$ seja colorido com a cor l_k.

O Lema 12.4 mostra como cada evento pré-fixado do PPGHE corresponde a uma condição que obriga que determinados vértices do grafo adjunto G sejam coloridos com cores específicas e o Corolário 12.3 generaliza o resultado do Lema 12.4:

Lema 12.4 (Neufeld e Tartar): Para o PPGHE sem restrições de indisponibilidade e com um evento pré-fixado $P_{i_0 j_0} \neq f$ e $P_{i_0 j_0} = \{h^1_{i_0 j_0},...,h^K_{i_0 j_0}\}$, existe uma relação um-para-um entre o conjunto de todas as soluções *S* do problema e o conjunto *U* de todas as s -colorações do grafo adjunto *G*, sujeitas à condição que os vértices $v^1_{i_0 j_0},...,v^K_{i_0 j_0}$ sejam coloridos com as cores $l_{k_1},...,l_{k_K}$ ($l^1_{i_0 j_0},...,l^K_{i_0 j_0}$), respectivamente.

Corolário 12.3 (Neufeld e Tartar): Para o PPGHE sem restrições de indisponibilidade e com eventos pré-fixados descritos por conjuntos P_{ij}, existe uma relação um-para-um entre o conjunto de todas as soluções *S* do problema e o conjunto *U* de todas as s -colorações do grafo adjunto *G*, sujeitas à condição que, correspondendo a cada conjunto não-vazio $P_{ij} = \{h^1_{ij},...,h^K_{ij}\}$, os vértices $v^1_{ij},...,v^K_{ij}$ sejam coloridos com as cores $l^1_{ij},...,l^K_{ij}$, respectivamente.

Os Teoremas 12.1 e 12.2 estabelecem que grafos sujeitos a condições que obrigam ou impedem que determinados vértices sejam coloridos com cores específicas, podem ser transformados em grafos sem estas condições.

Teorema 12.1 (Neufeld and Tartar): Ao grafo *G*, sujeito à condição de que determinados vértices sejam coloridos com determinadas cores, corresponde o grafo G' sem esta condição, tal que cada vértice em *G* é um vértice em *G'*. O grafo *G* é *n*-colorível se, e somente se, *G'* é *n*-colorível para algum número inteiro positivo *n*.

Teorema 12.2 (Neufeld and Tartar): Ao grafo *G*, sujeito à condição de que determinados vértices não sejam coloridos com determinadas cores, corresponde o grafo *G''* sem esta condição, tal que cada vértice em *G* é um vértice em *G''*. O grafo *G* é *n*-colorível se, e somente se, *G''* é *n*-colorível para algum número inteiro positivo *n*.

O Teorema 12.3 a seguir utiliza os resultados dos teoremas, lemas e corolários anteriores e estabelece a condição necessária e suficiente para a existência de uma solução para o PPGHE com restrições de indisponibilidade e/ou eventos pré-fixados.

Teorema 12.3 (Neufeld and Tartar): Ao PPGHE com restrições de indisponibilidade e eventos pré-fixados corresponde um grafo *H*. Existe uma solução para o PPGHE se, e somente se, o grafo *H* é s -colorível.

O Teorema 12.3 também mostra, baseado nos resultados dos Teoremas 12.1 e 12.2, que um PPGHE com restrições de indisponibilidade e eventos pré-fixados pode ser associado a um grafo que não está sujeito a condições pré-estabelecidas. A forma de obter este grafo será detalhada na Seção 5.2, após a apresentação do PPGHE escolhido para ser utilizado neste trabalho.

5.2 REFORMULAÇÃO DE UM PPGHE COM RESTRIÇÕES RÍGIDAS E FLEXÍVEIS A PARTIR DE UM PROBLEMA DE COLORAÇÃO DE GRAFOS

Conforme apresentado no Capítulo 1, a proposta desta dissertação é:

- Escolher um PPGHE com restrições adicionais, associado às suas respectivas instâncias, que já tenha sido tratado em algum trabalho anterior sem a utilização de Coloração de Grafos;
- Reformular o PPGHE escolhido como um problema de Coloração de Grafos;
- Modificar a versão do algoritmo *Tabucol*, disponibilizada por Joseph Culberson, (CULBERSON, 1997), (GRAPH, 2004) e descrita na Seção 4.4, para que possa resolver o PPGHE escolhido;
- Resolver o problema de Coloração de Grafos correspondente ao PPGHE escolhido, através da meta-heurística Busca Tabu, utilizando o algoritmo modificado;
- Verificar a influência desta reformulação em resultados existentes na literatura.

Dos cinco tópicos relacionados acima, os três primeiros serão abordados nesta seção e os dois últimos serão abordados no Capítulo 6.

5.2.1 Escolha e Descrição do PPGHE

A seguir será apresentado o processo de escolha do PPGHE a partir de suas instâncias associadas.

No cenário mundial, a maioria das instâncias associadas a formulações de problemas de Programação de Grade Horária Educacional publicamente disponíveis referem-se a problemas do tipo Cursos, conforme exemplos apresentados no Quadro 3. Uma das poucas exceções é o conjunto de instâncias disponibilizados por Smith, Abramson e Duke (2003), que refere-se a um PPGHE.

REFERÊNCIA DE TRABALHOS ONDE A INSTÂNCIA É UTILIZADA	TIPO	REFERÊNCIA DE PÁGINAS NA INTERNET ONDE AS INSTÂNCIAS ESTÃO DISPONÍVEIS
(LEWANDOWSKI, 1994, apud LEWANDOWSKI; CONDON, 1996)	Cursos	(GRAPH, 1998) – Instâncias da família *Sch* - formato *Dimacs*
(ROSSI-DORIA et al., 2003) (SOCHA; SAMPELS; MANFRIN, 2003)	Cursos	(METAHEURISTICS,2007)
(ROSSI-DORIA; PAECHTER, 2004a, 2004b) (SOCHA; SAMPELS; MANFRIN, 2003)	Cursos	(INTERNATIONAL, 2003) - Primeiro *International Timetabling Competition*
(SOCHA; KNOWLES; SAMPELS, 2002)	Cursos	(A MAX-MIN, 2007)
(DI GASPERO; SCHAERF, 2003, 2006)	Cursos	(COURSE, 2007) (INTERNATIONAL, 2007) - Segundo *International Timetabling Competition*
(CARTER; LAPORTE; LEE, 1996)	Provas	(MECHANICAL, 2005)
(SMITH; ABRAMSON; DUKE, 2003)	Escolar (PPGHE)	(TIMETABLING, 2004)

Quadro 3 – Exemplos de instâncias de problemas de Programação de Grade Horária Educacional publicamente disponíveis – Acessos em 20 out. 2007.

Entretanto, para o propósito desta dissertação, procurou-se um PPGHE cuja formulação e instâncias associadas fossem baseadas em escolas brasileiras. Foi escolhido o PPGHE (e respectivas instâncias) utilizado por Souza (2000), Souza, Maculan e Ochi (2003) e Santos, Ochi e Souza (2007).

Este PPGHE é descrito a seguir, adaptado ao formato e nomenclatura apresentados na Seção 5.1 (GOTLIEB, 1963, apud NEUFELD; TARTAR, 1974). As instâncias associadas são descritas no Capítulo 6.

Dado:

- Um conjunto de professores $T = \{t_i\}, i = 1,...,a$;
- Um conjunto de turmas $C = \{c_j\}, j = 1,...,b$;
- Um conjunto de horários semanais $H = \{h_k\}, k = 1,...s$; constituído a partir de l dias da semana com m períodos diários, definindo $s = l.m$ horários distintos;
- Uma matriz de requerimentos $a \times b$ $R = [r_{ij}]$, onde $r_{ij} \geq 0$, e r_{ij} é igual à carga horária semanal de aulas do professor t_i para a turma c_j;
- Uma matriz de indisponibilidades $a \times s$ $D = [d_{ik}]$, com $d_{ik} = 1$ se o professor t_i está indisponível no horário h_k e $d_{ik} = 0$ caso contrário;
- Uma matriz de limites de aulas diárias $a \times b$ $U = [u_{ij}]$, onde $0 \leq u_{ij} \leq 2$, e u_{ij} é igual à quantidade máxima diária de aulas do professor t_i para a turma c_j;
- Uma matriz de aulas duplas $a \times b$ $S = [s_{ij}]$, onde $s_{ij} \geq 0$, e s_{ij} é igual à quantidade mínima de aulas duplas (duas aulas alocadas em períodos consecutivos de um mesmo dia) requeridas pelo professor t_i para a turma c_j;

O PPGHE possui as características descritas a seguir:

- Todas as turmas $c_j, \forall j = 1,...,b,$ estão sempre disponíveis. Portanto, a matriz de indisponibilidades $b \times s$ $E = [e_{jk}]$, definida na Seção 5.1, será igual à matriz nula $0_{b,s}$ e não será considerada;

- Não existem requerimentos de aulas pré-fixadas. Portanto, conforme especificado na Seção 5.1, $P_{ij} = f, \forall i = 1,...,a; j = 1,...,b$, e estes conjuntos não serão considerados;
- Uma turma é um conjunto de estudantes que cursam as mesmas disciplinas;
- Os estudantes de cada turma não pertencem a nenhuma outra turma, ou seja, os conjuntos formados pelas turmas são disjuntos;
- Cada disciplina cursada por uma turma está associada a um único professor, estabelecido previamente;
- Todas as turmas tem carga horária igual a s , ou seja, $\sum_{i=1}^{a} r_{ij} = s$, $\forall c_j, j = 1,...,b$.

Uma solução do PPGHE precisa satisfazer às seguintes restrições rígidas e flexíveis (SANTOS; OCHI; SOUZA, 2007),(SOUZA; MACULAN; OCHI, 2003),(SOUZA, 2000).

Restrições Rígidas:

1. Nenhum professor pode ser alocado para mais de uma aula em um mesmo horário;
2. Nenhuma turma pode ser alocada para mais de uma aula em um mesmo horário;
3. Professores não podem ser alocados a horários que estejam indisponíveis;
4. Cada professor precisa cumprir sua carga horária semanal de aulas com cada turma;
5. Nenhuma turma pode ter mais de dois horários de aula por dia com o mesmo professor.

Restrições Flexíveis:

6. As aulas programadas para cada professor devem estar concentradas no menor número possível de dias;
7. As aulas duplas requeridas pelos professores devem ser satisfeitas sempre que possível;

8. Na programação das aulas dos professores deve ser evitada a ocorrência de lacunas (gaps), isto é, horários sem atividade entre dois horários de aula de um mesmo dia.

Souza, Maculan e Ochi (2003) e Santos, Ochi e Souza (2007) apresentaram soluções para o PPGHE acima baseadas na meta-heurística Busca Tabu. A grade horária correspondente a uma solução do PPGHE é representada de forma tabular, a partir de uma matriz $a \times s$ $Q = [q_{ik}]$, com $q_{ik} \in \{-1,0,1,2,...,b\}$, onde:

- $q_{ik} = j$ se o professor t_i está lecionando no horário h_k para a turma c_j;
- $q_{ik} = 0$ se o professor t_i está disponível para alocação;
- $q_{ik} = -1$ se $d_{ik} = 1$, ou seja, se o professor t_i está indisponível no horário h_k.

A Figura 35 mostra um exemplo de grade horária e respectiva matriz Q com 8 professores ($a = 8$), 3 turmas ($b = 3$) e 25 horários ($s = 25$). As indisponibilidades dos professores, contidas na matriz D, estão representadas na grade horária pelo caractere "X", enquanto os horários disponíveis estão em branco.

$$Q = \begin{bmatrix}
3 & 3 & 1 & 2 & 0 & 1 & 1 & 2 & 2 & 3 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 2 & 2 & 1 & 1 & 0 & 2 & 2 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 2 & 2 & 3 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 0 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 2 & 1 & 3 & 3 & 2 & 1 & 0 & 0 & 3 & 2 & 1 & 0 & 0 \\
-1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 3 & 3 & 0 & 0 & 0 & 2 & 3 & 3 & 0 & 0 & 2 & 2 & 3 \\
1 & 2 & 2 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 2 & 3 & 0 & 0 & 0 & 2 & 3 & 3 & 1 & 1 \\
0 & 0 & 3 & 3 & 1 & -1 & -1 & -1 & -1 & -1 & 3 & 3 & 1 & 0 & 1 & 0 & 0 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 3 & 3 & 1 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

t_i \ h_k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	3	3	1	2		1	1	2	2	3	X	X	X	X	X										
2	2	1									1	1		2	2	1	1		2	2	X	X	X	X	X
3						2	2	3	3	1						X	X	X	X	X	1	1		3	2
4						X	X	X	X	X			2	1	3	3	2	1			3	2	1		
5	X	X	X	X	X						2	2	3	3				2	3	3			2	2	3
6	1	2	2	1	3						X	X	X	X	X	2	3				2	3	3	1	1
7			3	3	1	X	X	X	X	X	3	3	1		1			3	1	1					
8					2	3	3	1	1	2						X	X	X	X	X					

Figura 35 – Exemplo de grade horária e matriz Q correspondente

Para maior simplicidade de notação, no restante deste trabalho, uma grade horária correspondente a uma matriz Q será referida como grade horária Q.

Uma solução representada por uma grade horária Q é avaliada por uma função objetivo $f(Q)$, a ser minimizada. Esta função objetivo é descrita a seguir (SANTOS; OCHI; SOUZA, 2007), (SOUZA; MACULAN; OCHI, 2003):

$$f(Q) = w.f_1(Q) + d.f_2(Q) + p.f_3(Q) \qquad (1)$$

e $$f_3(Q) = \sum_{i=1}^{a} (q_i.g_i + j_i.v_i + y_i.l_i) \qquad (2)$$

Onde:

- $f_1(Q)$ contabiliza o número de ocorrências de uma turma c_j ser alocada a mais de uma aula em um horário h_k. Esta ocorrência está atrelada à ocorrência de uma turma c_j não ser alocada a nenhuma aula em um horário $h_{k'}$ ($k, k' \in \{1,...,s\}$), uma vez que a solução do PPGHE abrange a programação de todas as $\sum_{i=1}^{a}\sum_{j=1}^{b} r_{ij}$ aulas dos professores t_i para as turmas c_j e $\sum_{i=1}^{a} r_{ij} = s$.
- $f_2(Q)$ contabiliza o número de alocações que ultrapassam a quantidade máxima diária de aulas u_{ij} do professor t_i para a turma c_j;
- $f_3(Q)$ contabiliza a satisfação dos requerimentos dos professores, da seguinte forma:
    - g_i contabiliza o número de lacunas (*gaps*) na programação das aulas de cada professor t_i;
    - v_i contabiliza o número de dias da semana que cada professor t_i precisa comparecer à escola para lecionar;
    - l_i contabiliza a diferença não-negativa entre o número mínimo de aulas duplas $\sum_{j=1}^{b} s_{ij}$ requerido por cada professor t_i e o número efetivo de aulas duplas na pauta do professor t_i.

A seguir serão apresentadas as relações entre a satisfação das restrições do PPGHE e as definições da grade horária Q e das componentes da função objetivo $f(Q)$:

- A restrição rígida 1 é sempre satisfeita, em decorrência da própria definição da matriz $Q=[q_{ik}]$, uma vez que cada elemento q_{ik} não pode assumir dois valores j diferentes simultaneamente;

- A satisfação da restrição rígida 2 é avaliada pela componente $f_1(Q)$ da função objetivo;

- A restrição rígida 3 é sempre satisfeita, em decorrência da consideração da matriz de indisponibilidades $D=[d_{ik}]$ na solução $Q=[q_{ik}]$ do PPGHE;

- A restrição rígida 4 é sempre satisfeita, devido ao método de solução considerado, Busca Tabu, partir sempre de uma solução inicial completa, ou seja, com todas as $\sum_{i=1}^{a}\sum_{j=1}^{b} r_{ij}$ aulas programadas, ainda que a solução possa não ser viável em função da violação de outras restrições.

- A satisfação da restrição rígida 5 é avaliada pela componente $f_2(Q)$ da função objetivo;

- A satisfação das restrições flexíveis 6, 7 e 8 é avaliada pelas componentes v_i, l_i e g_i da função objetivo, respectivamente;

Assim, as componentes $f_1(Q)$ e $f_2(Q)$ avaliam a violação de restrições rígidas e a componente $f_3(Q)$, através das componentes g_i, v_i e l_i, avalia a não-satisfação de restrições flexíveis. Portanto, a solução representada pela grade horária Q é uma solução viável se $f_1(Q)=0$ e $f_2(Q)=0$. Por este motivo os pesos w, d e p são escolhidos de forma que $w>d>>p$. Analogamente, os pesos q_i, j_i e y_i são escolhidos de forma a refletir a importância relativa das componentes g_i, v_i e l_i, respectivamente.

Considerando a nomenclatura apresentada para o algoritmo Busca Tabu na Figura 32 da Seção 4.4 e a definição da matriz Q apresentada acima, serão descritas a seguir as principais características dos algoritmos Busca Tabu utilizados por Santos, Ochi e Souza (2007) e por Souza, Maculan e Ochi (2003) para resolver o PPGHE escolhido.

Os algoritmos utilizam soluções iniciais $s_0 = Q_0$ geradas por algoritmos gulosos e aleatórios baseados na meta-heurística GRASP apresentada no Subseção 3.3.9. A busca é efetuada em toda a vizinhança $N(Q)$. O movimento m_i consiste na troca mútua (*swap*) de dois valores distintos e não-negativos na linha da matriz Q correspondente ao professor $t_i \in T$, representado pela tripla $(i,k,k') \mid q_{ik} \neq q_{ik'}$, $q_{ik} \geq 0$, $q_{ik'} \geq 0, k < k'$, $k,k' \in \{1,...,s\}$. O critério de aspiração é $f(Q \oplus m_i) < f(Q^*)$, ou seja, $a(s,m_i) = f(Q \oplus m_i)$ e $A(s,m_i) = f(Q^*)$, onde a matriz Q corresponde à solução atual e a matriz Q^* corresponde à melhor solução encontrada até aquele momento.

No algoritmo utilizado por Souza, Maculan e Ochi (2003), o período tabu (*tabu tenure*) é fixo. Para acelerar a obtenção de soluções de melhor qualidade é utilizada a técnica Intraturmas-Interturmas, baseado em um algoritmo de caminho mais curto de um grafo. Para ajudar a evitar a retenção em ótimos locais, além da lista tabu é utilizado um mecanismo para implementar diversificação denominado reinício aleatório (*random restart*), no qual a execução completa dos algoritmos GRASP e Busca Tabu é repetida (reiniciada) em seqüência por um determinado número de vezes, de maneira que, a cada iteração, o algoritmo Busca Tabu utiliza uma nova solução inicial gerada pelo algoritmo GRASP, portanto sem relação com as soluções produzidas nas iterações anteriores. A melhor solução obtida no processo global é mantida como resultado.

No algoritmo utilizado por Santos, Ochi e Souza (2007) o período tabu é variável e determinado aleatoriamente para cada movimento entre os limites $\bar{c} \pm \bar{j}.\bar{c}$, onde $\bar{c}$ é um parâmetro de entrada e $\bar{j} \in [0,1]$. Na estratégia de diversificação utilizada, a escolha dos movimentos correspondentes é efetuada considerando o histórico do

processo de pesquisa, através de dois tipos de memória de longo prazo: baseada em transição (*transition based long term memory*) e baseada em ocupação (*residence based long term memory*). No primeiro tipo, o número de movimentos efetuados envolvendo cada professor t_i com cada turma c_j é contabilizado e armazenado em uma matriz $a \times b$. No segundo tipo, são contabilizadas e armazenadas o número de iterações que cada aula m_{ij}^r $(1 \leq r \leq r_{ij})$, envolvendo cada professor t_i com cada turma c_j, ocupou cada horário h_k. A partir da manipulação destes valores, armazenados para cada tipo de memória de longo prazo, são calculadas as penalidades utilizadas na estratégia de diversificação, separadamente e em conjunto.

Souza, Maculan e Ochi (2003) e Santos, Ochi e Souza (2007) incluíram ainda em seus trabalhos os resultados obtidos a partir de versões dos algoritmos citados sem a implementação de qualquer estratégia de melhoramento e diversificação.

5.2.2 Reformulação do PPGHE como um problema de Coloração de Grafos

A seguir será descrita a reformulação do PPGHE escolhido, apresentado na Subseção 5.2.1, como um problema de coloração de grafos.

Desconsiderando a matriz de indisponibilidades $a \times s$ $D = [d_{ik}]$ neste primeiro momento e utilizando os resultados desenvolvidos por Neufeld e Tartar (1974, 1975) apresentados na Seção 5.1, o grafo adjunto associado ao PPGHE com $\sum_{i=1}^{a}\sum_{j=1}^{b} r_{ij}$ aulas representadas pelo conjunto $M = \bigcup_{i=1}^{a}\bigcup_{j=1}^{b} M_{ij}$, $M_{ij} = \{m_{ij}^1, m_{ij}^2, ..., m_{ij}^{r_{ij}}\}$, é definido por:

$G = (V, E)$, com $V = \{v = v_{ij}^r \mid (m_{ij}^r, v_{ij}^r) \in F\}$ e $E = \left(\bigcup_{i=1}^{a} E_{ijj'}\right) \cup \left(\bigcup_{j=1}^{b} E_j\right)$, onde:

- $F : M \rightarrow V$ é um mapeamento do conjunto M para o conjunto V;

- $E_{ijj'} = \begin{cases} \{\{v,v'\} \mid v \in V_{ij}, v' \in V_{ij'}\}, se\ j \neq j', r_{ij} > 0, r_{ij'} > 0 \\ f, caso\ contrário \end{cases}$;
- $E_j = \{\{v,v'\} \mid v,v' \in V_j, v \neq v'\}$;
- V_{ij} corresponde a $M_{ij} = \{m_{ij}^1, m_{ij}^2, ..., m_{ij}^{r_{ij}}\}$, $V_{ij} \subseteq V$, $M_{ij} \subseteq M$;
- V_j corresponde a $M_j = \bigcup_{i=1}^{a} M_{ij}$, $V_j \subseteq V$, $M_j \subseteq M$;
- V_i corresponde a $M_i = \bigcup_{j=1}^{b} M_{ij}$, $V_i \subseteq V$, $M_i \subseteq M$.

Para incorporar as restrições de indisponibilidade da matriz $D = [d_{ik}]$ ao grafo $G = (V,E)$, serão utilizados os resultados apresentados por Neufeld e Tartar (1975) descritos a seguir, adaptados à nomenclatura utilizada até aqui:

Pelo Corolário 12.2 (Neufeld e Tartar) apresentado na Seção 5.1, o PPGHE com restrições de indisponibilidade descritas pelos elementos $d_{ik} = 1$ da matriz D está associado ao grafo adjunto *G*, cuja coloração está sujeita à condição que, para cada $d_{ik} = 1$, nenhum vértice nos conjuntos V_{ij}, *j = 1, . . . ,* b seja colorido com a cor l_k.

Estas restrições são expressas pela matriz de restrições $|V| \times s$ $P' = [p'_{nk}]$, onde $p'_{nk} = 1$ se o vértice $v_n \in V_{ij}$, *j = 1, . . . ,* b não pode ser colorido com a cor l_k e $p'_{nk} = 0$ caso contrário.

O $G = (V,E)$ sem eventos pré-fixados e com a matriz de restrições P' é denotado por $G = (V,E,f,P')$.

De acordo com o Teorema 12.2 (Neufeld and Tartar) apresentado na Seção 5.1, ao grafo $G = (V,E,f,P')$ corresponde o grafo G'' sem pré-condições, denotado por $G'' = (V'',E'',f,f)$.

Neufeld e Tartar (1975) demonstraram que $V'' = V \cup V'$ e $E'' = E \cup E'$, onde:

- $V^{'} = \{v_{n+1},...,v_{n+s}\}$ com $c(v_{n+1}) = l_1$, . . ., $c(v_{n+s}) = l_s$. Se $P^{'} = [p^{'}_{nk}]$ é tal que existam somente s' cores ($s' < s$) com as quais vértices não possam ser coloridos, então somente s' vértices adicionais $v_{n'}$ irão compor o conjunto V'.

- $E' = E^1 \bigcup E^2$, onde:

  - E^1 é composto pelas arestas necessárias a interligar cada vértice $v_n \in V_{ij}$, $j = 1, \ldots, b$, tal que $p^{'}_{nk} = 1$, ao vértice $v_{n'} \in V'$, tal que $c(v_{n'}) = l_k$.

  - E^2 é composto pelas arestas necessárias a interligar todos os s' vértices de V' entre si, de maneira a formarem uma clique, ou seja, um subgrafo completo;

Estes conceitos são melhor compreendidos com o exemplo apresentado a seguir: Suponha que, na grade horária de 7 aulas semanais associada ao grafo adjunto $G = (V, E)$ da Figura 36(a), o professor t_2 ministre um total de duas aulas semanais, sendo uma para a turma c_1 e outra para a turma c_3, representadas por m^1_{21} e m^1_{23}, respectivamente, e o professor t_4 ministre apenas uma aula semanal para a turma c_2, representada por m^1_{42}. Suponha que o professor t_2 não esteja disponível no horário h_7 e o professor t_4 não esteja disponível no horário h_9, ou seja, $d_{27} = 1$ e $d_{49} = 1$ na matriz D, e que estas sejam as únicas indisponibilidades de horário na grade. Suponha ainda que $v_1 = v^1_{21}$, $v_6 = v^1_{23}$ e $v_5 = v^1_{42}$. Então $p'_{17} = 1$, $p'_{67} = 1$ e $p'_{59} = 1$ na matriz P', $V^{'} = \{v_8, v_9\}$, $E^1 = \{\{v_1 v_8\}, \{v_6, v_8\}, \{v_5 v_9\}\}$, $E^2 = \{\{v_8, v_9\}\}$ e o grafo correspondente é $G^{"} = (V^{"}, E^{"}, f, f)$, onde:

- $V^{"} = V \bigcup \{v_8, v_9\}$
- $E^{"} = E \bigcup \{\{v_1, v_8\}, \{v_6, v_8\}, \{v_5, v_9\}, \{v_8, v_9\}\}$

O grafo $G^{"} = (V^{"}, E^{"}, f, f)$ é mostrado na Figura 36(b).

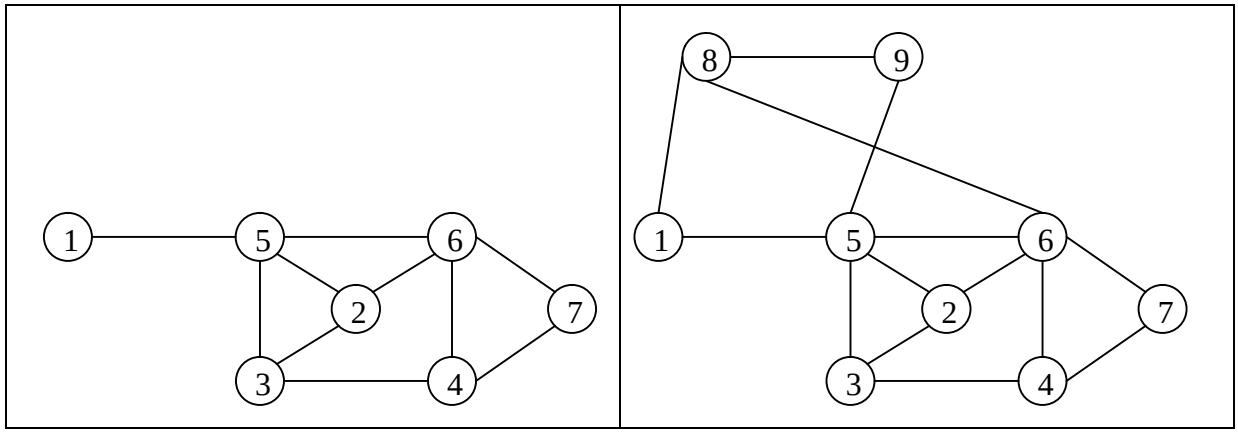


Figura 36 – (a) Grafo adjunto G - (b) Grafo $G^{"}$ correspondente

5.2.3 Adaptação do algoritmo *Tabucol* às restrições adicionais do PPGHE

Na Subseção 5.2.2 foi apresentado o processo de obtenção do grafo adjunto $G^{"} = (V^{"}, E^{"}, f, f)$ associado ao PPGHE escolhido. Este grafo pode ser colorido utilizando a versão do algoritmo *Tabucol*, disponibilizada por Joseph Culberson, (CULBERSON, 1997), (GRAPH, 2004). Entretanto, conforme apresentado na Seção 4.4, a função objetivo (a ser minimizada) utilizada neste algoritmo é $\overline{f}(c) = \sum_{k=1}^{s} |E(C_k)|$, onde $E(C_k)$ é o conjunto das arestas com ambos vértices terminais na classe de cores C_k, conforme proposto por Hertz e De Werra (1987, apud GALINIER; HERTZ, 2006) no *Tabucol* original.

Entretanto, conforme descrito na Subseção 5.2.1, a função objetivo do PPGHE escolhido é $f(Q) = w.f_1(Q) + d.f_2(Q) + p.f_3(Q)$, onde a componente $f_1(Q)$ contabiliza os conflitos de horários. Assim, uma s -coloração de vértices de $G^{"}$ correspondente à solução $c = (C_1, ..., C_s)$ tal que $\overline{f}(c) = 0$, corresponde à minimização da componente $f_1(Q)$ na função $f(Q)$. Portanto, existe uma componente $\overline{f}_1(c)$ na função $\overline{f}(c)$, correspondendo à componente $f_1(Q)$ na função $f(Q)$. Esta componente é definida por: $\overline{f}_1(c) = \sum_{k=1}^{s} |E(C_k)|$.

Utilizando as nomenclaturas da Seção 4.4 e da Subseção 5.2.1, as s classes de cores C_k, $k = 1,...,s$, são agrupadas em um conjunto de l m-uplas ordenadas $L_r = (C_{k_{r(1)}},...,C_{k_{r(m)}})$, $r = 1,...,l$, $s = l.m$ denominadas grupos de cores.

É dito que um vértice v_{ij}^r está no grupo de cores L_r, se este vértice está associado a uma classe de cores $C_k \in L_r$.

Defina a matriz de limites de vértices por grupo $a \times b$ $\overline{U} = [\overline{u}_{ij}]$, correspondendo à matriz de limites de aulas diárias $a \times b$ $U = [u_{ij}]$, onde $\overline{u}_{ij} = u_{ij}$, $0 \le \overline{u}_{ij} \le 2$, e $\overline{u}_{ij}$ é igual à quantidade máxima de vértices v_{ij}^r em um mesmo grupo de cores.

Defina uma matriz de vértices duplos $a \times b$ $\overline{S} = [\overline{s}_{ij}]$, correspondendo à matriz de aulas duplas $a \times b$ $S = [s_{ij}]$, onde $\overline{s}_{ij} = s_{ij}$, $s_{ij} \ge 0$, e $\overline{s}_{ij}$ é igual à quantidade mínima requerida de vértices v_{ij}^r duplos (dois vértices associados a classes de cores consecutivas $C_{k_r}, C_{k_{r+1}}$ de um mesmo grupo de cores);

Defina $\overline{f}(c) = w.\overline{f}_1(c) + d.\overline{f}_2(c) + p.\overline{f}_3(c)$, com $\overline{f}_1(c)$, $\overline{f}_2(c)$ e $\overline{f}_3(c)$ correspondendo às funções $f_1(Q)$, $f_2(Q)$ e $f_3(Q)$ componentes de $f(Q)$, respectivamente, definidas da seguinte forma:

- $\overline{f}_1(c) = \sum_{k=1}^{s} |E(C_k)|$ onde $E(C_k)$ é o conjunto das arestas com ambos vértices terminais na classe de cores $C_k, k = 1,...,s$;
- $\overline{f}_2(c) = \sum_{r=1}^{l} |V(L_r)|$ onde $|V(L_r)|$ é o número de vértices v_{ij}^r no grupo de cores L_r que ultrapassa o limite de vértices por grupo $\overline{u}_{ij}$;
- $\overline{f_3}(c) = \sum_{i=1}^{a} (q_i.\overline{g_i} + j_i.\overline{v_i} + y_i.\overline{l_i})$, onde:

- $\overline{g_i} = \sum_{r=1}^{l} |C^{(i)}(L_r)|$, onde $|C^{(i)}(L_r)|$ é igual ao número de classes de cores de um grupo de cores L_r que não estão associadas a nenhum vértice $v \in V_i$ e estão situadas na m-upla ordenada entre classes de cores que estão associadas cada uma a pelo menos um vértice $v \in V_i$;
- $\overline{v}_i$ é igual ao número de grupos de cores com pelo menos um vértice $v \in V_i$;
- $\overline{l_i}$ é igual à diferença não-negativa entre o valor requerido $\sum_{j=1}^{b} \overline{s}_{ij}$ e o número de pares de vértices $v, w \in V_i$, tal que v e w sejam vértices duplos, ou seja, estão associados a classes de cores consecutivas C_{k_r}, C_{k_r+1} de um mesmo grupo de cores.

A partir dos novos conceitos acima, são descritas a seguir as principais alterações efetuadas no algoritmo *Tabucol* e na implementação de Joseph Culberson deste algoritmo em linguagem C, a fim de que uma s-coloração de vértices de G'' considere a minimização de todas as restrições do PPGHE e não apenas a minimização dos conflitos de horários:

- A função objetivo $\overline{f}(c) = \overline{f}_1(c)$ foi substituída pela função objetivo $\overline{f}(c) = w.\overline{f}_1(c) + d.\overline{f}_2(c) + p.\overline{f}_3(c)$.
- A lista que armazena os vértices críticos (*conflictList*), descrita na Seção 4.4, foi transformada em uma lista penta-dimensional, passando a armazenar os vértices críticos correspondentes a cinco componentes: $\overline{f}_1(c)$, $\overline{f}_2(c)$ e as componentes $\overline{g}_i$, $\overline{v}_i$ e $\overline{l}_i$ de $\overline{f}_3(c)$. O programa continua selecionando aleatoriamente os elementos do conjunto V_c^* a partir desta lista. Porém, agora, primeiro é selecionado aleatoriamente uma das listas e só então é selecionado aleatoriamente um elemento do conjunto V_c^* na lista selecionada.

O programa de Joseph Culberson com a implementação do algoritmo *Tabucol* utiliza o formato *Dimacs* descrito na Seção 4.2 para o arquivo de entrada contendo o Grafo a ser colorido. Este formato foi mantido no programa modificado.

A partir do algoritmo *Tabucol* modificado, pode-se estabelecer o seguinte processo para resolução de uma instância do PPGHE descrito na Subseção 5.2.1:

1. Obter a instância do PPGHE a ser utilizada, representada pelas matrizes $R = [r_{ij}]$, $D = [d_{ik}]$, $U = [u_{ij}]$ e $S = [s_{ij}]$ e pelos valores a, b, l e m, conforme descrito no Capítulo 6 e na Subseção 5.2.1;
2. Obter o grafo adjunto $G'' = (V'', E'', f, f)$ associado à instância do PPGHE, conforme descrito na Subseção 5.2.2;
3. Obter as matrizes $\overline{U} = [\overline{u}_{ij}]$ e $\overline{S} = [\overline{s}_{ij}]$, conforme descrito nesta subseção;
4. Representar o grafo G'' no formato *Dimacs*, conforme descrito na Seção 4.2;
5. Gerar uma solução inicial $c^{(0)} = (C_1^{(0)}, ..., C_s^{(0)})$ através de um algoritmo de coloração de grafos, conforme descrito nas Seções 4.3 e 4.4.
6. Resolver o problema de Coloração de Grafos correspondente ao PPGHE, através da meta-heurística Busca Tabu, utilizando o algoritmo *Tabucol* modificado, a partir da solução inicial $c^{(0)}$, do grafo G'', das matrizes $\overline{U}$ e $\overline{S}$ e dos valores a, b, l e m, obtendo uma s -coloração de vértices $c = (C_1, ..., C_s)$ do grafo G'', tal que minimize $\overline{f}(c) = w.\overline{f}_1(c) + d.\overline{f}_2(c) + p.\overline{f}_3(c)$, conforme descrito nesta subseção;
7. Obter a grade horária $Q = [q_{ik}]$, descrita na Subseção 5.2.1, correspondente à s -coloração c obtida.

A Figura 37 ilustra o processo acima.

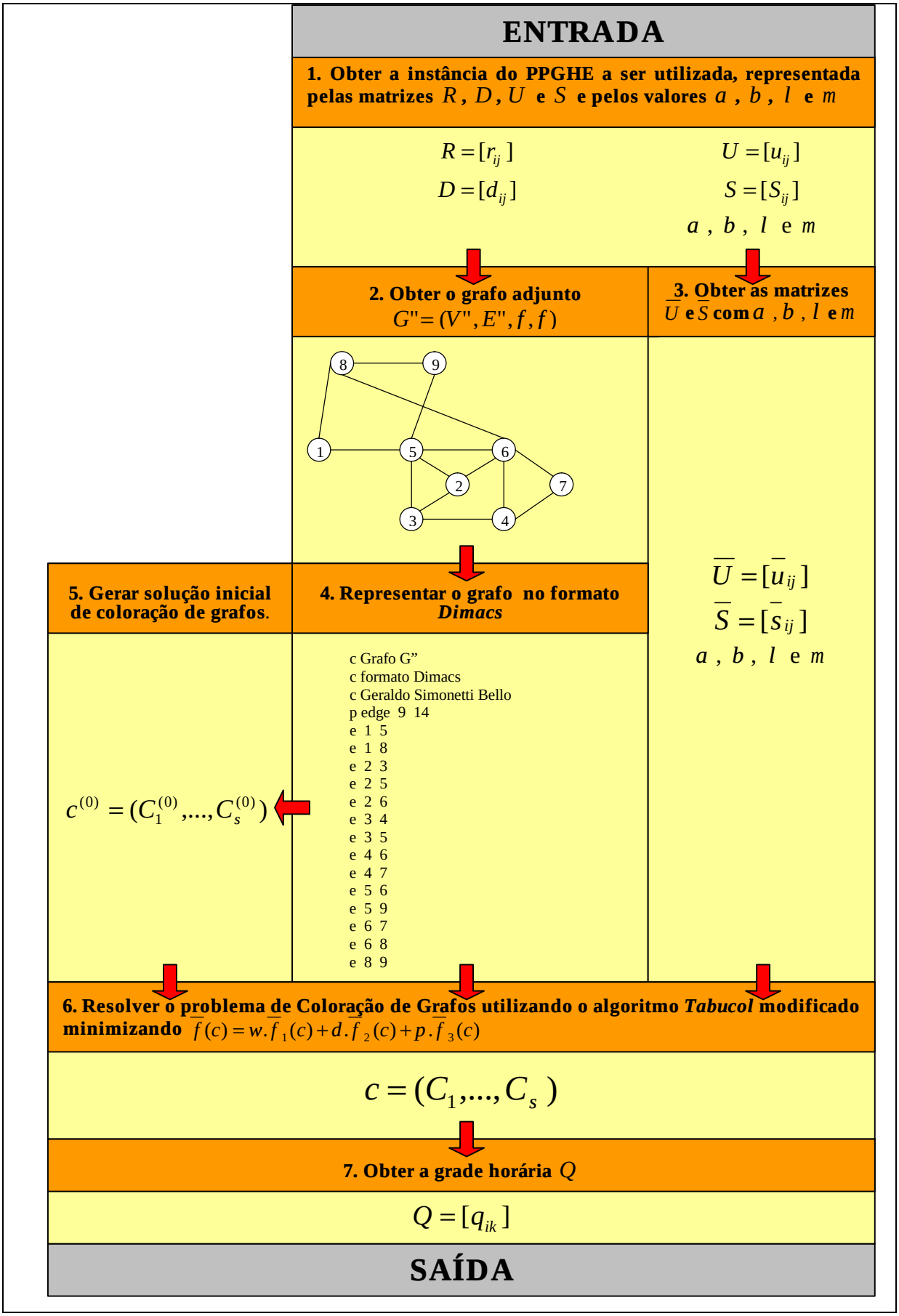


Figura 37 – Processo proposto para resolução do PPGHE

Este processo representa uma proposta diferente da grande maioria das abordagens encontradas na literatura. Nas pesquisas bibliográficas efetuadas, foi encontrada uma única referência a um processo cuja descrição parece assemelhar-se a este (BOUFFLET; NEGRE, 1994, apud GUÉRET et al., 1995). Entretanto, os artigos descritos por Guéret e outros (1995) não foram localizados, após pesquisa exaustiva realizada por intermédio do Programa de Comutação Bibliográfica (Comut). Por este motivo, estes trabalhos não foram considerados nesta dissertação.

6 RESULTADOS COMPUTACIONAIS

Para avaliar o desempenho do processo descrito no Capítulo 5, foi utilizado o mesmo conjunto de sete instâncias usado por Souza (2000), Souza, Maculan e Ochi (2003) e Santos, Ochi e Souza (2007), correspondente ao PPGHE escolhido. A instância 1 é composta de dados fictícios, usados para testar parâmetros do algoritmo. As instâncias 2, 4, 5 e 6 são compostas de dados de uma escola de ensino médio de Mariana (MG) e os dados da instância 3 são de uma escola de ensino médio de Brasília (DF), correspondentes aos anos letivos e turnos relacionados no Quadro 4. A instância 7 refere-se a uma simulação considerando maior quantidade de turmas para a escola mineira. As instâncias 1 e 3 são originadas de Mata (1989, apud SOUZA, 2000) e as instâncias 2, 4, 5, 6 e 7 são originadas de Souza (2000).

INSTÂNCIA	TIPO DE DADOS	LOCALIZAÇÃO DA ESCOLA DE ENSINO MÉDIO	TURNO	ANO LETIVO
1	Fictício	-	-	-
2	Real	Mariana (MG)	Tarde	2000
3	Real	Brasília (DF)	Manhã	1989
4	Real	Mariana (MG)	Manhã	2000
5	Real	Mariana (MG)	Manhã	1998
6	Real	Mariana (MG)	Noite	2000
7	Simulação	Mariana (MG)	-	-

Quadro 4 – Descrição das instâncias utilizadas (SOUZA, 2000)

A Tabela 1 apresenta as características dos conjuntos de dados das instâncias e dos grafos adjuntos correspondentes $G=(V,E)$ e $G^{"}=(V^{"},E^{"},f,f)$. Em todas as instâncias consideradas a grade horária contém 25 horários semanais (s =25), distribuídos em 5 dias da semana (l = 5) com 5 períodos diários (m = 5). A razão de dispersão (rd) incluída na Tabela 1 é calculada pela seguinte expressão (SOUZA; MACULAN; OCHI, 2003):

$$rd = \frac{a \times s - (|M| + |I|)}{a \times s} \qquad (1)$$

onde:

- a é o número total de professores;
- s é o número total de horários da grade horária (s =25);
- $|M| = \sum_{i=1}^{a} \sum_{j=1}^{b} r_{ij}$ é o número total de aulas;
- $|I| = \sum_{i=1}^{a} \sum_{k=1}^{s} d_{ik}$ é o número total de horários indisponíveis.

Quanto menor o valor de rd, mais difícil é a resolução dos problemas PPGHE e de Coloração de Grafos para a instância correspondente.

Tabela 1 - Características das instâncias e dos grafos adjuntos G e G'' correspondentes

Nº da Instância	Nº de Professores (a)	Nº de turmas (b)	Nº de aulas ($\|M\|$)	Nº de aulas duplas	Nº de horários Indisponíveis ($\|I\|$)	Razão de Dispersão (rd)	Grafo adjunto G		Grafo adjunto G''	
							Nº de vértices	Nº de arestas	Nº de vértices	Nº de arestas
1	8	3	75	21	40	0,425	75	1116	100	1791
2	14	6	150	29	25	0,500	150	2560	170	2940
3	16	8	200	4	80	0,300	200	3360	225	4660
4	23	12	300	66	170	0,183	300	5788	325	7401
5	31	13	325	71	0	0,581	325	5845	325	5845
6	30	14	350	63	10	0,520	350	6321	355	6396
7	33	20	500	84	0	0,394	500	9223	500	9223

Conforme descrito nas Seções 4.4 e 5.2, o algoritmo utilizado consiste em uma modificação da implementação em linguagem C efetuada por Joseph Culberson, (CULBERSON, 1997), (GRAPH, 2004) do algoritmo *Tabucol* (HERTZ; DE WERRA, 1987, apud GALINIER; HERTZ, 2006) e foi testado utilizando microcomputador com processador *Pentium* II 450 *MHz*, 128 *MB* de memória *RAM* e Sistema Operacional *Linux CentOS* versão 4.3. O compilador utilizado foi o *GCC* versão 3.4.5.

Foram utilizados os seguintes pesos na função objetivo: w =100, d =30, p =1, q_i =3, j_i =9 e y_i =1, $i = 1,...,a$ (SOUZA; MACULAN; OCHI, 2003), (SANTOS, OCHI; SOUZA, 2007).

Para a geração da solução inicial do algoritmo *Tabucol* modificado foram utilizados algoritmos gulosos descritos no Quadro 1 da Subseção 4.3.1. Testes iniciais foram efetuados utilizando também o algoritmo *Dsatur* e o algoritmo *Dsatur* com retrocesso, apresentados nas Subseções 4.3.4 e 4.3.5, respectivamente, mas foram abandonados em função do melhor desempenho geral apresentado pelos algoritmos gulosos para esta aplicação, durante estes testes iniciais.

Na Tabela 2 são apresentados os resultados dos experimentos computacionais para as sete instâncias descritas acima com o algoritmo *Tabucol* modificado, correspondendo a combinações de valores dos parâmetros descritos a seguir:

- N_g: Identificação da variante do algoritmo guloso utilizado para a geração da solução inicial, conforme relacionado no Quadro 1 da Subseção 4.3.1.

- $n_{\max_V_c^*}$: Número máximo de vizinhos representativos. É o tamanho máximo da amostra de vértices críticos V_c^*, gerada a cada iteração, da qual é escolhido o vértice onde o movimento será aplicado, conforme descrito na Seção 4.4.

- $n_{\min_V_c^*}$: Número mínimo de vizinhos representativos. É o tamanho mínimo da amostra de vértices críticos, a ser verificado pelo algoritmo antes de concluir a geração de V_c^* e M_c^*, se for selecionado um vértice v e um movimento $m_{v,k}$ que produzam um resultado melhor que a melhor solução encontrada até o momento ($\overline{f}(c \oplus m_{v,k}) < \overline{f}(c^*)$), conforme descrito na Seção 4.4.

- $|\overline{T}|$: Tamanho da lista tabu e o período tabu. Um vértice movido de uma determinada partição é proibido de ser movido de volta para a partição de origem durante $|\overline{T}|$ iterações.

No Quadro 5 são relacionados os subconjuntos de valores para os parâmetros descritos acima considerados nas execuções do algoritmo, por terem produzido os melhores resultados nos testes iniciais.

N_g	{1, 2, 13, 34}
$n_{\max_V_c^*}$	{500, 800, 900, 1000}
$n_{\min_V_c^*}$	{2, 3, 13, 19, 20}
$\lvert \overline{T} \rvert$	{7, 10}

Quadro 5 – Subconjuntos de valores dos parâmetros do algoritmo *Tabucol* modificado

As execuções foram realizadas com um número fixo de iterações (30.000) para cada instância. A inicialização foi efetuada com um único valor de semente para os números pseudo-aleatórios (semente = 1) para todas as instâncias, uma vez que este valor proporcionou os melhores resultados gerais nos testes iniciais. São informados os melhores e os piores resultados obtidos com os valores dos parâmetros correspondentes aos melhores resultados.

Na Tabela 2, os resultados obtidos são comparados com os resultados de Souza, Maculan e Ochi (2003) e Santos, Ochi e Souza (2007), referentes às versões dos respectivos algoritmos sem a implementação das estratégias de melhoramento e diversificação correspondentes, denominadas *GTS* e *TS*, respectivamente. Esta escolha foi baseada no seguinte motivo: o propósito da comparação é verificar qual o efeito que a formulação do PPGHE como um problema de Coloração de Grafos provoca na solução do problema. A visualização deste efeito seria dificultada, caso não fosse isolada dos outros efeitos provocados pelas estratégias de memória de longo prazo, uma vez que não foi implementado originalmente na Busca Tabu do algoritmo *Tabucol* nenhuma estratégia de memória de longo prazo, tal como intensificação ou diversificação, conforme apresentado na Seção 4.4.

Os resultados relativos ao *TS* foram recalculados com os mesmos números de iterações e o mesmo valor de semente para números pseudo-aleatórios utilizados no *Tabucol* modificado. Também foi utilizado o mesmo microcomputador. A busca é efetuada em toda a vizinhança, ou seja, o número de vizinhos representativos do algoritmo é $|N(Q)|$, conforme apresentado na Subseção 5.2.1. O período tabu é variável para cada movimento e neste experimento foi configurado para ser

escolhido aleatoriamente entre os limites 27 e 33 ($\bar{c} = 30, \bar{j} = 0{,}1$), conforme proposto por Santos, Ochi e Souza (2007).

Para o *GTS* foram transcritos os resultados apresentados por Souza, Maculan e Ochi (2003), nos quais a instância 1 não está incluída. O propósito foi fornecer uma fonte adicional para comparação dos resultados, ainda que sem a mesma padronização utilizada na comparação do *Tabucol* modificado com o *TS*. Para atenuar este aspecto, foi utilizado um microcomputador com configuração próxima da utilizada por Souza, Maculan e Ochi (2003).

Tabela 2 – Resultados computacionais com número fixo de iterações

Nº da Instância	***GTS***	***TS***	***Tabucol* Modificado**					
			Melhor Valor	N_g	$n_{\max_V_c^*}$	$n_{\min_V_c^*}$	$\lvert\bar{T}\rvert$	**Pior Valor**
1	-	204	202	2	900	13	7	214
2	368	344	347	1	1000	20	7	368
3	491	448	436	2	800	19	10	900
4	749	671	683	13	500	3	10	801
5	831	795	808	13	800	20	7	868
6	847	783	872	34	900	19	10	1236
7	1164	1066	1117	34	1000	2	10	1208

Pela análise da Tabela 2 verifica-se que o método apresenta um bom desempenho para grafos pequenos e médios, quando comparado com o TS, mas este desempenho cai para grafos grandes (a partir de 300 vértices). Quando comparado ao *GTS*, o algoritmo apresentou bom desempenho em todas as instâncias, exceto na instância 6.

Em contrapartida, o tempo médio de execução de cada iteração pelo algoritmo *Tabucol* modificado é muito menor, quando comparado com o *TS*. Este comportamento é verificado em todas as instâncias, mas a diferença se acentua para as instâncias maiores. No Gráfico 1 são apresentados os tempos médios de execução de cada iteração, para as sete instâncias no algoritmo *Tabucol* modificado, com os parâmetros apresentados na Tabela 2 e o mesmo número fixo de iterações (30.000), comparado com o tempo médio de execução de cada iteração no algoritmo *TS*, para as mesmas instâncias e número de iterações.

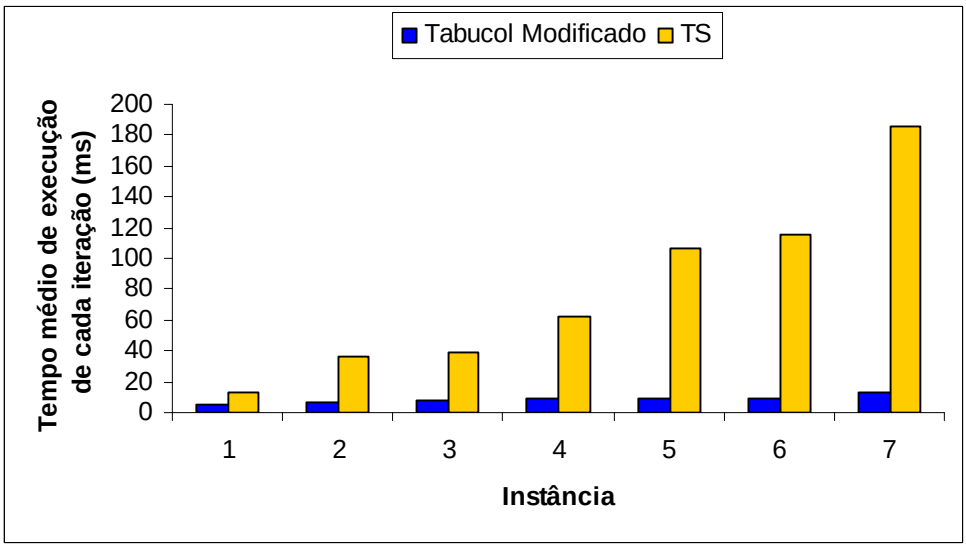


Gráfico 1 – Tempos de execução para um número fixo de iterações

7 CONCLUSÕES E TRABALHOS FUTUROS

Foram descritos os problemas de Programação de Grade Horária Educacional e de Coloração de Grafos, apresentando definições, subdivisões, características e uma visão geral das principais abordagens propostas para solução destes problemas.

A dissertação focou um tipo de problema de Programação de Grade Horária Educacional em particular: o Escolar (PPGHE), com restrições rígidas e flexíveis adicionais. Para este problema, foi proposto um processo para resolução, a partir da sua reformulação como um problema de Coloração de Grafos, utilizando algoritmos voltados para esta modelagem. Embora a correspondência entre estes dois problemas seja bem conhecida na literatura, normalmente o PPGHE considerado contém apenas restrições básicas. As principais contribuições desta dissertação consistiram em desenvolver um modelo estendido desta correspondência, contemplando também restrições adicionais e implantar modificações em um algoritmo para coloração de grafos existente, a partir de uma implementação em linguagem C disponível, de forma a contemplar o PPGHE com restrições adicionais. O processo para resolução do problema consiste na coloração do grafo associado ao PPGHE com esta formulação, utilizando Busca Tabu a partir de uma solução inicial gerada por outro algoritmo de coloração de grafos, por exemplo, algoritmos gulosos.

Além disso, foram escolhidos uma formulação de PPGHE e um conjunto de instâncias, propostos na literatura, que apresentam as características de conter restrições adicionais e ser baseado em escolas brasileiras, respectivamente, conforme desejado para esta dissertação. O PPGHE escolhido foi reformulado como um problema de Coloração de Grafos, suas instâncias foram resolvidas através do processo proposto na dissertação e foi observado o efeito desta reformulação nos resultados, em comparação com os existentes na literatura, que não empregam esta reformulação. Verificou-se que o algoritmo apresenta bom desempenho ao tratar grafos correspondentes a grades horárias de pequeno e médio porte, mas deve ser estudado a implantação de recursos adicionais no algoritmo, com o propósito de melhorar seu desempenho com instâncias maiores. De forma geral, o algoritmo

encontrou boas soluções para o PPGHE escolhido em um tempo computacional bastante curto, quando comparado com outros métodos.

São apontadas a seguir algumas perspectivas para trabalhos futuros:

- Realizar os experimentos computacionais utilizando o algoritmo *Tabucol* modificado, porém com a implementação, isoladamente e em conjunto, de outros recursos descritos ao longo desta dissertação, os quais são relacionados a seguir:
    - Incorporar ao *Tabucol* modificado adaptações com o propósito de melhorar o desempenho na coloração de grafos correspondentes a instâncias de PPGHE de grande porte, porém também levando em conta todas as cinco componentes da função objetivo ($\overline{f}_1(c)$, $\overline{f}_2(c)$ e as componentes $\overline{g}_i$, $\overline{v}_i$ e $\overline{l}_i$ de $\overline{f}_3(c)$);
    - Para a escolha do vértice que será movimentado de uma classe de cores para outra a cada iteração do *Tabucol* modificado, considerar como candidatos todos os vértices críticos, ao invés de uma amostra aleatória destes vértices;
    - Utilizar período tabu variável;
    - Utilizar outros parâmetros para avaliação dos movimentos aplicados aos vértices críticos a cada iteração;
    - Incorporar ao *Tabucol* modificado estratégias de memória de longo prazo, tais como diversificação e intensificação.
    - Gerar a solução inicial para o *Tabucol*, utilizando outros algoritmos que não foram testados neste trabalho, como por exemplo, o algoritmo guloso iterativo (*Iterated Greedy*).
- Adaptar e aplicar o processo descrito nesta dissertação a outras formulações de PPGHE, considerando restrições adicionais diferentes das utilizadas;

- Adaptar e aplicar o processo descrito aos outros tipos de problemas de Programação de Grade Horária Educacional (Cursos e Provas), utilizando as instâncias correspondentes publicamente disponíveis relacionadas nesta dissertação.
- Modificar outros algoritmos de coloração de grafos que utilizem função objetivo, de forma que passem a considerar as restrições adicionais do PPGHE associado, assim como foi feito com o *Tabucol*, e utilizá-los no processo descrito nesta dissertação, em substituição ao *Tabucol*.

Portanto, existe espaço para o desenvolvimento de diversos outros trabalhos, a partir das idéias apresentadas nesta dissertação.

REFERÊNCIAS

A MAX-MIN Ant System For the University Timetabling Problem. In: **IRIDIA, Université Libre de Bruxelles.** 01 Out. 2007. Disponível em: <http://iridia.ulb.ac.be/~msampels/tt.data/> Acesso em 20 out. 2007.

ACADEMIA Brasileira de Letras, [mensagem pessoal]. Mensagem recebida em 22 de outubro de 2007.

AGUIAR, F. et al. Metaheuristica Busca Tabu para o Problema de Coloração de Grafos. In: **XXXVII Simpósio Brasileiro de Pesquisa Operacional - XXXVII SBPO.** Gramado:Anais do XXXVII SBPO. RJ: SOBRAPO, 2005. v. 1. p. 2497-2504.

ALEXANDERSON, G. L. **About the Cover: Euler and Königsberg's Bridges: A Historical View**. In: Bulletin (New Series) of the American Mathematical Society. [S.I.: s.n.], Oct. 2006 v. 43, n. 4, p. 567–573

ASMUNI et al. Fuzzy Multiple Heuristic Ordering for Examination timetabling, Practice and Theory of Automated Timetabling V. In: BURKE, E. K.; TRICK, M. **Lecture Notes in Computer Science.** [S.I.]: Springer, 2005. v. 361.

AVANTHAY, C.; HERTZ, A.; ZUFFEREY, N. A variable neighborhood search for graph coloring. In: **European Journal of Operational Research 151**. [S.I.: s.n.], 2003. p. 379–388.

BAI, R. et al. A Simulated Annealing Hyper-heuristic for University Course Timetabling Problem. In: **6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)**. Czech Republic: [s.n.], Aug. 2006. p. 345–350.

BARTÁK, R. Constraint Programming: In Pursuit of the Holy Grail. In: **Proceedings of the Week of Doctoral Students, Part IV**. Prague: MatFyzPress, 1999. p. 555-564.

BIAJOLI, F. L. et al. Escala de jogos de torneios esportivos: uma abordagem via Simulated Annealing. In: **Anais do XXXV Simpósio Brasileiro de Pesquisa Operacional – SBPO**. Natal: [s.n.], Nov. 2003, p. 1295-1306.

BOAVENTURA NETTO, P. O. **Grafos: teoria, modelos, algoritmos**. 3.ed. São Paulo: Edgard Blucher, 2003.

BRELAZ, D. New Methods to Color the Vertices of a Graph. In: **Communications of the ACM.** [S.I.: s.n.], 1979. v. 22, p. 251-256.

BURKE, E. K.; BYKOV, Y. Solving Exam Timetabling Problems with the Flex-Deluge Algorithm. In: **6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)**. Czech Republic: [s.n.], August 2006. p. 370-372.

BURKE, E. K. et al. Structured Cases in Case-Based Reasoning - Re-using and Adapting Cases for Timetabling Problems. In: **Knowledge-Based Systems**. [S.I.: s.n.], 2000. v. 13, n. 2-3, p. 159-165.

BURKE, E. K. et al. A Time-Predefined Approach to Course Timetabling, Yugoslav. In: **Journal of Operational Research (YUJOR).** [S.I.: s.n.], 2003. v. 13, n. 2, p. 139-151.

BURKE, E. K. et al. Automated university timetabling: The state of the art. In: **The Computer Journal**. [S.I.: s.n.], 1997. v. 40, n. 9, p. 565-571.

BURKE, E. K. et al. Multiple-Retrieval Case-Based Reasoning for Course Timetabling Problems. In: **Journal of Operations Research Society (JORS).** [S.I.: s.n.], 2006. v. 57, n. 2, p. 148-162.

BURKE, E. K. et al., A graph-based hyper-heuristic for educational timetabling problems. In: **European Journal of Operational Research** [S.I.: s.n.], 2007. v. 176, n. 1, p. 177-192.

BURKE, E. K.; ELLIMAN, D. G.; WEARE, R. F. A University Timetabling System based on Graph Colouring and Constraint Manipulation. In: **Journal of Research on Computing in Education.** [S.I.: s.n.], 1994. v. 27, n. 1, p. 1-18

BURKE, E. K.; KENDALL, G.; SOUBEIGA, E. A Tabu-Search Hyperheuristic for Timetabling and Rostering. In: **Journal of Heuristics.** [S.I.: s.n.], 2003. v. 9, p. 451-470.

BURKE,E. K.; NEWALL, J. P.; WEARE, R. F. A Memetic Algorithm for University Exam Timetabling. In: **1st International Conference on Practiceand Theory of Automated Timetabling (PATAT 1995), LNCS 1153**. [S.I.]: Springer-Verlag, 1996. p. 241-250.

BURKE,E. K.; PETROVIC, S., QU, R., Case Based Heuristic Selection for Timetabling Problems. In: **Journal of Scheduling.** [S.I.: s.n.], 2006. v. 9, p. 115-132.

CARRASCO, M.; PATO, M. A Potts Neural Network Heuristic for the Class / Teacher Timetabling problem Proceedings. In: **Fourth Metaheuristics International Conference.** Porto: [s.n.], July, 2001.

CARTER, M.; LAPORTE, G.; LEE, S. Examination Timetabling: Algorithmic Strategies and Applications. In: **Journal of Operational Research Society**. [S.I.: s.n.], 1996. P. 373-383.

CHIARANDINI, M.; STÜTZLE, T. An application of Iterated Local Search to Graph Coloring. In: JOHNSON, D. S.; MEHROTRA, A.; TRICK, M. **Computational Symposium on Graph Coloring and its Generalizations.** [S.I.: s.n.], 2002. v. 47, p. 112-125.

CHIARANDINI, M.; SCHAERF, A., TIOZZO, F. Solving Employee Timetabling Problems with Flexible Workload using Tabu Search. In: BURKE E.; ERBEN W. **3th International Conference on the Practice and Theory of Automated Timetabling**, **PATAT 2000.** Germany: Konstanz, 2000. p. 298-302.

CLIQUE and Coloring Problems Graph Format. [S.I.: s.n.], May 1993.

COMELLAS, F.; OZÓN, J. An ant algorithm for the graph colouring problem. In: **Ants 98 From Ant Colonies to Artificial Ants: First International Workshop on Ant Colony Optimization.** Belgium: [s.n.], 1998.

CORMACK, D. H. **Fundamentos de Histologia.** 2. ed. Rio de Janeiro: Guanabara Koogan, 2003.

COSTA, D. A tabu search algorithm for computing an operational timetable. In: **European Journal of Operational Research**. [S.I.: s.n.], 1994. v. 76, p. 98-110.

COURSE Timetabling page @ SaTT - DIEGM - University of Udine. In: **Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica.** 13 July 2007. Disponível em: <http://www.diegm.uniud.it/satt/projects/EduTT/> Acesso em 20 out. 2007.

CROITORU, C. et al. A new genetic graph coloring heuristic. In: JOHNSON, D. S.; MEHROTRA, A.; TRICK, M. **Proceedings of the Computational Symposium on Graph Coloring and its Generalizations**. New York: [s.n.], 2002. p. 63-74.

CULBERSON, J. Iterated greedy graph coloring and the difficulty landscape, Technical Report 92-07. **Department of Computing Science, The University of Alberta.** Edmonton, Alberta, 1992.

CULBERSON, J. Graph Coloring Programs Manual. **Department of Computing Science, University of Alberta.** Edmonton, 1997.

DE WERRA, D. An introduction to timetabling. In: **European Journal of Operational Research.** [S.I.: s.n.], 1985. v. 19, p.151-162.

DI GASPERO, L.; SCHAERF, A. Multi-neighbourhood local search with application to course timetabling. In BURKE, E. ; DE CAUSMAECKER, P. **Practice and Theory of Automated Timetabling IV, n. 2740 in Lecture Notes in Computer Science**. Berlim: Springer-Verlag, 2003. p. 263-278.

DI GASPERO, L.; SCHAERF, A. Neighborhood portfolio approach for local search applied to timetabling problems. In: **Journal of Mathematical Modeling and Algorithms**. [S.I.: s.n.], 2006. v. 5, n. 1, p. 65-89.

DI GASPERO, L.; SCHAERF, A. Tabu search techniques for examination timetabling. Lecture Notes in: **Computer Science**. Berlin: Springer, 2001. v. 2079, p. 104-117

DIESTEL, R. **Graph Theory.** 3rd ed. New York: Springer-Verlag, 2005.

DIMACS Implementation Challenges. **Center for Discrete Mathematics and Theoretical Computer Science (DIMACS)**. 22 mar. 2006. Disponível em: <http://dimacs.rutgers.edu/Challenges/>. Acesso em 2 out. 2007.

DORIGO, M. et al. Evolving Self-Organizing Behaviors for a Swarm-bot. In: **Autonomous Robots**. [S. I.]: Springer, 2004. v. 17, n. 2-3, p. 223-245.

DORIGO, M.; STUTZLE, T. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In: KOCHENBERGER, G.; GLOVER, F. Chapter 9, **Handbook of Metaheuristics**. Norwell: Kluwer Academic Publishers, 2002. v.57 p. 251-285.

DORIGO, M; DI CARO, G.; GAMBARELLA, L. M. Ant algorithms for discrete optimization**. Artificial Life**. [S.I.: s.n.], 1999. v. 5, p. 137–172.

DORNE, R.; HAO, J-K. A New Genetic Local Search Algorithm for Graph Coloring. In: **5th International Conference on Parallel Problem Solving from Nature.** [S.I.: s.n.], Sept. 27-30, 1998. p. 745-754.

DREZET,L-E.; CHESNES D.; BELLENGIEZ-MORINEAU O. An employee timetabling problem in a maintenance service of a software company. In: **6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)**. Czech Republic: [s.n.], Aug. 2006. p. 391-393.

DUECK, G. New optimisation heuristics. The great deluge algorithm and record-to-record travel. In: **Journal of Computational Physics**. [S.I.: s.n.], 1993. v. 104 , p. 86-92.

EBERHART, R. C.; SHI Y. Comparison between genetic algorithms and particle swarm optimization, Evolutionary Programming VII. In: **Lecture Notes in Computer Science 7th Int. Conf. Evolutionary Programming.** New York: Springer, 1998. v. 1447, p. 611–616.

EIBEN, A. E.; VAN DER HAUW, J. K.; VAN HEMERT, J. I. Graph Coloring with Adaptive Evolutionary Algorithms. In: **Dept. of Computer Science, Leiden University, The Netherlands.** [S.I.: s.n.], 1997.

ELEY, M. Ant algorithms for the exam timetabling problem. In: **Aschaffenburg University of Applied Science Logistics Laboratory (LIAb).** Germany, 2006. p. 167-180.

ELMOHAMED, M. A. S.; CODDINGTON, P.; FOX, G. A comparison of annealing techniques for academic course scheduling. In: **Computer Science Lecture Notes.** [S.I.: s.n.], 1998. v. 1408, p. 92-112.

EVEN, S.; ITAI, A.; SHAMIR, A. On the complexity of timetabling and multicommodity flow problems. In: **SIAM Journal of Computation**. [S.I.: s.n.], 1976. v. 5, p. 691-703.

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. In: **Ciência da Informação.** [S.I.: s. n.], 2006. v. 35, p. 25-30.

GALINIER, P.; HAO, J-K., Hybrid evolutionary algorithms for graph coloring. In: **Journal of Combinatorial Optimization**. Netherlands: Kluwer, 1999. v. 3, p. 379-397.

GALINIER, P.; HERTZ, A. A Survey of Local Search Methods for Graph Coloring. In: **Computers & Operations Research**. [S.I.: s.n.], 2006. v. 33, p. 2547-2562.

GALINIER, P.; HERTZ, A.; ZUFFEREY, N. An adaptive memory algorithm for the k-colouring problem. In: **Les cahiers du GERAD G–2003–35, GERAD**. Montreal: [s.n.], 2003.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness.** New York: W. H. Freeman and Co., 1979

GISLEN, L.; PETERSON, C.; SODERBERG, B. Teachers and Classes with Neural Networks. In: **International Journal of Neural Systems.** [S. I.: s.n.], 1989. v. 1, p. 167-176.

GLOVER, F. Future Paths for Integer Programming and Links to Artificial Intelligence. In: **Computers and Operations Research 13.** [S.I.: s.n.], 1986. p. 533-549.

GOLDBERG, D. E. **Genetic Algorithms In Search, Optimization and Machine Learning**. Massachusetts: Addison-Wesley Publishing Company, 1989.

GONZÁLEZ-VELARDE, J. L.; LAGUNA, M. Tabu Search with Simple Ejection Chains for Coloring Graphs. In: **Annals of Operations Research, Volume 117, Numbers 1- 4.** [S.I.: s.n.], Nov. 2002 , p. 165-174.

GRAPH Coloring Programs. In**: Joseph Culberson's Graph Coloring Page.** 31 mar. 2004. Disponível em: <http://web.cs.ualberta.ca/~joe/Coloring/index.html>. Acesso em 2 out. 2007.

GRAPH Coloring Instances. In: **Michael Trick's Operations Research Page**. 08 Jan. 1998. Disponível em: <http://mat.gsia.cmu.edu/COLOR/instances.html> Acesso em: 20 out. 2007.

GUÉRET, C. et al. Building University Timetables Using Constraint Logic Programming. In: BURKE, E.; ROSS, P. **Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95**. LNCS 1153. [S.I.]: Springer-Verlag, 1995. p. 130-145.

HANSEN, P. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. In: **Congress on Numerical Methods in Combinatorial Optimization.** Capri, 1986.

HASTAD, J. **On Approximating NP-Hard Optimization Problems.** [S.I.]: Documenta Mathematica, 1998. v. Extra ICM III p. 441-450.

HERTZ, A. Finding a feasible course schedule using tabu search. In: **Discrete Applied Mathematics.** [S.I.: s.n.], 1992. v. 35, n. 3, p. 255-270.

HERTZ, A. Tabu search for large scale timetabling problems. In: **European Journal of Operational Research**, [S.I.: s.n.], 1991. v. 54, p. 39-47.

HERTZ, A.; TAILLARD, E.; DE WERRA, D. A tutorial on tabu search. In: AARTS, E. H. L.; LENSTRA, J. K. **Local Search in Combinatorial Optimization.** Chichester: John Wiley & Sons , 1997. p. 121–136.

HERTZ, A.; ZUFFEREY, N. New Ant Algorithm for Graph Coloring. In: **Workshop on Nature Inspired Cooperative Strategies for Optimization, NICSO.** Granada:[s.n.], 2006. p. 51-60.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. Ann Arbor: University of Michigan Press, 1975.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. 2.ed. Cambridge: The MIT Press / A Bradford Book, 1992.

HSU, W.-L. ; MA, T.-H. Fast and Simple Algorithms for Recognizing Chordal Comparability Graphs and Interval Graphs. In: **SIAM Journal of Computation**. [S.I.: s.n.], 1999. v. 28, n. 3, p. 1004-1020.

HU, X.; EBERHART, R. C. Solving constrained nonlinear optimization problems with particle swarm optimization. In: **6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)**. Orlando: [s.n.], 2002.

HU, X.; EBERHART, R. C.; SHI Y. Recent advances in particle swarm, In: **IEEE Congress on Evolutionary Computation 2004.** Portland: [s.n.], 2004.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA - IBGE. **Mapas Interativos.** Brasília, 2005. Disponível em: <http://mapas.ibge.gov.br/divisao/viewer.htm>. Acesso em 2 out. 2007.

INTERNATIONAL Timetabling Competition. In: **IDSIA – Dalle Molle Institute for Artificial Intelligence.** 31 March 2003. Disponível em: <http://www.idsia.ch/Files/ttcomp2002/> Acesso em 20 out. 2007.

INTERNATIONAL Timetabling Competition. In: **Queen's University Belfast - School of Electronics, Electrical Engineering and Computer Science.** 31 July 2007. Disponível em: <http://www.cs.qub.ac.uk/itc2007/> Acesso em 20 out. 2007.

JUNGINGER, W. Timetabling in Germany - a survey. In: **Interfaces**. [S.I.: s.n.], 1986. v. 16, p. 66-74.

KAMBI M.; GILBERT, D. R. Timetabling in constraint logic programming. In: **INAP-96**: **Symposium and Exhibition on Industrial Applications of PROLOG.** Tokyo: [s.n.], 1996. p. 79-88.

KLOTZ W. Graph coloring algorithms. Tech. Rep. Mathematik-Bericht 5. **Clausthal University of Technology.** Clausthal, 2002.

KOWALSKI, R. A. The Early Years of Logic Programming. In: **Communications of the ACM.** [S.I.: s.n.], 1988. v. 31, n. 1, p. 38-43.

KROKHIN, A.; BULATOV, A.; JEAVONS, P. The complexity of constraint satisfaction: an algebraic approach. In: **Structural Theory of Automata, Semigroups, and Universal Algebra, volume 207 of NATO Science Series II: Math., Phys., Chem.** [S.I]: Springer Verlag, 2005. p. 181-213.

KUMAR, V. Algorithms for Constraint Satisfaction Problems: A Survey. In: **AI Magazine. Department of Computer Sciences. University of Minnesota.** Minneapolis, 1992. v. 13, n.1, p. 32-44.

LAGUNA, M.; MARTÍ, R. A GRASP for Coloring Sparse Graphs. In: **Computational Optimization and Applications.** [S.I.: s.n.], 2001. v. 19, n. 2, p. 165-178.

LAPLAGNE, I.; KWAN, R. S. K.; KWAN, A. S. K. Time Windows and Constraint Boundaries for Public Transport Scheduling. In: **6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)**. Czech Republic: [s.n.], Aug. 2006. p. 458-459.

LEWANDOWSKI, G.; CONDON, A. Experiments with parallel graph coloring heuristics and applications of graph coloring. In: **DIMACS Series in Discrete Mathematics and Theoretical Computer Science**. Providence:American Mathematical Society, 1996. v. 26, p. 309-334.

MARINO; PRÜGEL-BENNETT, A.; GLASS, C. A. Improving graph colouring with linear programming and genetic algorithms. In: **EUROGEN 99**. Jyväskyiä: [s.n.], 1999. p. 113–118.

MCGREEVY, M. A. G. et al. A Neural Network based Construction Heuristic, PATAT '04. In: **5th International Conference on the Practice and Theory of Automated Timetabling**. Pittsburgh: [s.n.], Aug. 2004 .

MECHANICAL and Industrial Engineering. In: **University of Toronto.** 09 June 2005. Disponível em: <ftp://ftp.mie.utoronto.ca/pub/carter/testprob/> Acesso em 11 out. 2007.

MEHROTRA, A.; TRICK, M. A. A column generation approach for graph coloring. In: **INFORMS Journal on Computing.** [S. I.: s.n.], 1996. v. 8, n. 4, p. 344 -354.

METAHEURISTICS on the Timetabling Problem – Results of a Comparison within the Metaheuristics. In: **IRIDIA - Supplementary Information Page.** 01 Out. 2007. Disponível em: <http://iridia.ulb.ac.be/~msampels/ttmn.data> Acesso em: 20 out. 2007.

MICHALEWICZ, Z.; FOGEL, D. B., **How to Solve It : Modern Heuristics**. 3rd ed. Berlim: Springer-Verlag, 2002.

MISSION Statement. **Center for Discrete Mathematics and Theoretical Computer Science (DIMACS)**. 13 out. 2003. Disponível em: <http://dimacs.rutgers.edu/About/mission.html>. Acesso em 2 out. 2007.

MOSCATO, P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. In: **Technical Report 826, Caltech Concurrent Computation Program.** Pasadena: California Institute of Technology, 1989.

NEUFELD, G. A.; TARTAR, J. Graph coloring conditions for the existence of solutions to the timetable problem. In: **Communications of the ACM.** [S.I.: s.n.], 1974. v. 17, n. 8, p. 450-453.

NEUFELD, G. A.; TARTAR, J. Generalized graph colorations. In: **SIAM Journal of Applied Mathematics.** [S.I.: s.n.], 1975. v. 29, n. 1, p. 91-98.

OSTERMANN, R.; DE WERRA, D. Some experiments with a timetabling system. In: **OR Spektrum**, [S.I.: s.n.], 1983. v. 3, p. 199-204.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial Optimization: Algorithms and Complexity**. 2nd ed. New York: Dover, 1998.

PETROVIC, S.; PATEL, V.; YONG, Y. University Timetabling with Fuzzy Constraints. In: BURKE, E. K.; TRICK, M. **Practice and Theory of Automated Timetabling V, Lecture Notes in Computer Science 3616.** [S.I.]: Springer, 2005. p. 313-333.

POLI, R. An analysis of publications on particle swarm optimization applications. In: **Technical Report. CSM-469, Department of Computer Science, University of Essex,** ISSN: 1744-8050, 2007.

RADCLIFFE, N. J., SURRY, P. D. Formal memetic algorithms. In: FOGARTY, T. **Evolutionary Computing: AISB Workshop.** [S.I.]: Springer-Verlag, 1994.

REIS, L. P.; OLIVEIRA, E. A Constraint Logic Programming Approach to Examination. Scheduling. In: **AICS'99, Artificial Intelligence and Cognitive Science - 10th Irish Conference on Artificial Intelligence & Cognitive Science, Cork**. Ireland: [s.n.], Sept. 1999.

RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures. In: GLOVER, F.; KOCHENBERGER, G. (Org.) **Handbook of Metaheuristics.** Boston: Kluwer, 2003. p. 219-249.

ROSSI-DORIA et al. A comparison of the performance of different metaheuristics on the timetabling problem. In: BURKE, E.; DE CAUSMAECKER, P. **Practice and Theory of Automated Timetabling IV: 4th International Conference, PATAT 2002, Lecture Notes in Computer Science.** Berlin: Springer- Verlag, Heidelberg, 2003. v. 2740, p. 329-51.

ROSSI-DORIA, O.; PAECHTER B. A memetic algorithm for university course timetabling**.** In: **Technical report, Napier University**. Edinburgh, UK, 2004a.

ROSSI-DORIA, O.; PAECHTER, B. An hyperheuristic approach to course timetabling problem using evolutionary algorithm. In: **Technical report, Napier University.** Edinburgh: [s.n.], 2003.

ROSSI-DORIA, O.; PAECHTER, B. An opinion algorithm for university course timetabling. In: **Technical report, Napier University.** Edinburgh: [s.n.], 2004b.

ROY, P.; LIRET, A.; PACHET, F. The framework approach for constraint satisfaction. In: **ACM Computing Surveys.** [S.I.: s.n.], 2000. v. 32, p. 13-16.

SANTOS, H. G.; OCHI, L. S.; SOUZA, M. J. F. A Tabu Search Heuristic with Efficient Diversification Strategies for the Class/Teacher Timetabling Problem. In: **ACM Journal of Experimental Algorithms.** [S.I.], 2007. To Appear.

SCHAERF, A. A Survey of Automated Timetabling. In: **Artificial Intelligence Review.** [S.I.]: Springer. v. 13 n. 2, p. 87-127, 1999a

SCHAERF, A. **Local search techniques for large high-school timetabling problems, IEEE Transactions on Systems, Man, and Cybernetics** [S.I.: s.n.], 1999b. v. 29, n. 4, p. 368-377.

SHAWE-TAYLOR, J.; ZEROVNIK, J. Ants and graph coloring. **In: Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA'01.** Praga: Springer-Verlag, 2001. p. 276-279.

SMITH, K. A.; ABRAMSON, D.; DUKE, D. Efficient Timetabling Formulations for Hopfield Neural Networks. In: DAGLI, C. et al. **Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining, and Complex Systems**. [S. I.]: ASME, 1999. v. 9, p. 1027-1032.

SMITH, K. A.; ABRAMSON, D.; DUKE, D. Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparative Results. In: **International Journal of Computers and Industrial Engineering.** [S. I.: s.n.], 2003. v. 44, n. 2, p. 283 - 305.

SOCHA, K.; KNOWLES, J.; SAMPELS, M. A MAX- MIN Ant System for the University Course Timetabling Problem. In: DORIGO, M. ; DI CARO, G.; SAMPELS, M. **Third International Workshop on Ant Algorithms (ANTS'2002). Lecture Notes in Computer Science 2463.** Berlin: Springer Verlag, Sept. 2002. p. 1-13.

SOCHA, K.; SAMPELS, M.; MANFRIN, M. Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In: CAGNONI S. et al. **Applications of Evolutionary Computing, Proceedings of the EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, volume 2611 of Lecture Notes in Computer Science.** Germany: Springer-Verlag, Heidelberg, 2003. p. 334-345.

SOUZA, M. J. F. **Programação de Horários em Escolas: Uma Aproximação por Metaheurísticas.** 2000. Tese D. Sc., Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2000.

SOUZA, M. J. F.; GUIMARÃES, I. F.; COSTA, F. P. Um Algoritmo Evolutivo Híbrido para o Problema de Programação de Horários em Escolas. In: XXII Encontro Nacional de Engenharia de Produção - ENEGEP, Outubro 2002, Curitiba, Brasil. **Anais** do XXII ENEGEP, CD-ROM, 8 p.

SOUZA, M. J. F.; MACULAN, N.; OCHI, L. S. A GRASP-Tabu search algorithm for solving school timetabling problems. In: RESENDE, M. G. C.; SOUZA, J. P. **Metaheuristics: Computer Decision-Making.** Boston: Kluwer Academic Publishers, 2003. p. 659-672.

TIMETABLING. In: **OR-Library**. 07 Sept. 2004. Disponível em: <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/tableinfo.html> Acesso em 20 out. 2007.

TRICK, M. A.. A schedule-then-break approach to sports timetabling. In: **Practice and Theory of Automated Timetabling III, of Lecture Notes in Computer Science.** [S.I.]: Springer-Verlag, 2000. v. 2079, p. 242–253.

VAN HEMERT, J. Evolving binary constraint satisfaction problem instances that are difficult to solve. In: **IEEE 2003 Congress on Evolutionary Computation.** [S. I.]: IEEE, 2003. p. 1267-1273.

VESEL, A.; ZEROVNIK, J. How good can ants color graphs? In: **Journal of computing and Information Technology – CIT.** [S.I.: s.n.], 2000. v. 8, p. 131–136.