Algoritmos *Simulated Annealing* e grasp para o planejamento de aulas de um departamento

Alexandre Xavier Martins (UFOP/DECEA)
xmartins@decea.ufop.br;

Raphael Reis Mauro de Castro (UFOP/DECEA)
raphaelufop@yahoo.com.br

Marcone Jamilson Freitas Souza (PPGEM/UFOP)
marcone@iceb.ufop.br

RESUMO

Este trabalho trata do problema de programação de horários em escolas. Dada sua natureza combinatória, ele é resolvido por meio de dois algoritmos metaeurísticos, um baseado em Simulated Annealing e outro em GRASP. Ambos possuem parâmetros auto-adaptativos, dispensando, assim, a calibragem destes. Para testá-los são utilizados dados reais do departamento de uma universidade. São apresentados resultados computacionais, comparando-se as soluções produzidas pelos algoritmos propostos com aquelas geradas manualmente pela instituição de ensino. Os resultados obtidos mostram a eficiência dos métodos desenvolvidos perante as soluções manuais e a superioridade do Simulated Annealing, em comparação com o GRASP para as instâncias tratadas.

Palavras-chave: *Programação de horários em escolas, simulated annealing e GRASP.*

Simulated annealing algorithms and grasp for the planning of classes of a department

ABSTRACT

This work deals with the school timetabling problem. Due to its combinatorial complexity, two metaheuristic algorithms, one based on Simulated Annealing and the other on GRASP, are proposed. These are adaptive algorithms, and therefore it is not necessary to tune parameters. In order to test the algorithms, real data of a university department are used. It is presented a comparison between the manual solutions and those produced by the proposed algorithms. The computational results show the overall efficiency of the developed methods when confronted with the manuals solutions and the superiority of the Simulated Annealing over the GRASP algorithm when tackling the real instances.

Keywords: *School timetabling, simulated annealing and GRASP.*

1. Introdução

Problemas de programação de horários, conhecidos na literatura inglesa como *timetabling problems*, dizem respeito à alocação, sujeita a restrições, de recursos a objetos colocados no espaço e no tempo, de modo a satisfazer, tanto quanto possível, um conjunto de objetivos desejáveis (WREN, 1996).

Entre as aplicações dessa classe de problemas, incluem-se: (a) programação de horários em escolas (*school timetabling*); (b) programação de cursos universitários (*course timetabling*); (c) programação de horários de exames (*examination timetabling*); (d) alocação de aulas a salas (*classroom assignment*); (e) programação de horários de ônibus (*bus timetabling*); e (f) programação de jogos de competições esportivas (*sports timetabling*).

Neste trabalho, trata-se um problema de programação de horários de uma instituição de ensino superior que se enquadra como um Problema de Programação de Horários em Escolas (PPHE). Problema esse que consiste na confecção de um quadro de horários que represente a rotina semanal dos professores e alunos de uma instituição de ensino, satisfazendo requisitos de vários tipos.

A solução manual desse problema é uma tarefa extremamente árdua e normalmente requer vários dias de trabalho. Além do mais, a solução obtida pode ser insatisfatória com relação a diversos aspectos. Por exemplo, um professor pode ficar descontente se houver muitas janelas ou muitas aulas espalhadas em sua programação semanal de ensino. Muitas vezes, também, a instituição de ensino, por não conseguir montar um quadro de horários devido à existência de conflitos, é obrigada a contratar mais um professor apenas para eliminar os conflitos existentes, o que representa, portanto, aumento de custos para a instituição.

A resolução desse problema por técnicas exatas também não é indicada para a maioria dos casos reais, uma vez que o PPHE é da classe NP-difícil (EVEN et al., 1976). Dessa forma, o problema é comumente abordado por meio de técnicas heurísticas. Entre essas, destacam-se as metaeurísticas, as quais têm condições de produzir soluções finais de melhor qualidade.

Neste trabalho são propostos dois algoritmos heurísticos para resolver o PPHE. O primeiro é baseado na metaeurística *Simulated Annealing* (KIRKPATRICK et al., 1983) e o outro em GRASP (FEO; RESENDE, 1995). Este trabalho se justifica por apresentar algoritmos heurísticos autoadaptativos, que dispensam a calibração tipicamente necessária nessa classe de algoritmos.

Este estudo está assim organizado: na Seção 2 é feita uma revisão bibliográfica sobre as formas de se resolver o problema em questão, bem como sobre metaeurísticas. Na Seção 3, o problema abordado é descrito. Na Seção 4, é apresentada a modelagem heurística do problema. Nas Seções 5 e 6 é feita uma breve descrição dos algoritmos *Simulated Annealing* e *GRASP*, respectivamente, e de como esses métodos são adaptados neste trabalho para resolver o PPHE. Na Seção 7 são apresentados e analisados os resultados. Na Seção 8, encontram-se as conclusões. Por fim, mostra-se o referencial adotado como base de análise.

2. Referencial bibliográfico

O Problema de Programação de Horários em Escolas é NP-difícil (EVEN et al., 1976). Assim, sua resolução na otimalidade por técnicas exatas está, em geral, restrita a problemas de pequenas dimensões (SANTOS; SOUZA, 2007). Souza (2000) apresentou formulação de programação matemática e a usou no otimizador XPRESS. Entretanto, não foi possível obter a solução ótima para nenhum dos sete problemas-teste considerados, os quais envolviam até 33 professores e 20 turmas. Posteriormente, Santos et al. (2007) apresentaram uma formulação estendida com geração de cortes e colunas para resolver esse PPHE, que permitiu resolver na otimalidade três desses problemas-teste, envolvendo até 16 professores e oito turmas.

Dados a sua natureza combinatória e o fato de que a maioria dos problemas reais é de dimensões mais elevadas, o PPHE é, portanto, normalmente resolvido por técnicas heurísticas (SCHAEFER, 1999). Se, por um lado, com esses procedimentos não é possível garantir a distância de otimalidade das soluções finais geradas; por outro, há relatos de utilização bem-sucedida destes procedimentos em vários problemas combinatórios (GLOVER; KOCHENBERGER, 2003; CERNY, 1985). Ademais, ganha-se em flexibilidade, no sentido de que com esses métodos há facilidade para inserir novas restrições, tarefa essa que não é trivial em formulações de programação matemática.

As primeiras heurísticas utilizadas para resolução do problema eram construtivas. Nessa classe de heurísticas, as soluções geradas não conseguem, em geral, contemplar todas as alocações, isto é, algumas

aulas ficam de fora do quadro de horários. Tal fato pode ocorrer porque, nesse procedimento, as aulas são alocadas uma por vez, em uma dada ordem de escolha, e uma vez alocada uma aula, a alocação é definitiva. Em um segundo momento, apareceram as heurísticas de refinamento. Nessa segunda classe de heurísticas são feitas modificações nas alocações das aulas, de forma a se procurar uma solução de melhor qualidade. Quando nenhuma solução melhor é possível com base nessas modificações, então se encerra o procedimento de busca. A grande limitação das heurísticas clássicas de refinamento é que elas ficam presas no primeiro ótimo local encontrado e, assim, impossibilitadas de prosseguir a busca em direção ao ótimo global.

Com o surgimento das metaeurísticas, como Algoritmos Genéticos, *Simulated Annealing*, Busca Tabu etc., apareceu uma nova e promissora alternativa para a resolução de problemas combinatórios, como a programação de horários. O prefixo "meta" é utilizado para descrever uma heurística que está sobreposta a outra heurística, constituindo outro "nível heurístico". Em geral, a metaeurística constitui uma estrutura mais genérica, baseada em princípios ou conceitos, sobreposta a uma heurística específica do problema em estudo. Com esses métodos, é possível escapar das armadilhas dos ótimos locais e, assim, produzir soluções finais de melhor qualidade.

Para tratar o PPHE, várias são as metaeurísticas que têm sido utilizadas, entre elas: Algoritmos Genéticos (COLORNI et al., 1998; CALDEIRA; AGOSTINHO, 1997), *Simulated Annealing* (ABRAMSON, 1991), GRASP (SOUZA et al., 2004) e Busca Tabu (SANTOS et al., 2005; SCHAEFER, 1996).

A metaeurística *Simulated Annealing*, adaptada neste trabalho para resolver o PPHE, foi proposta por KirkPatrick et al. (1983), o qual teve como base o procedimento proposto por Metrópolis (1953).

Existem quatro parâmetros usados pelo *Simulated Annealing*, os quais são combinados em um cronograma de resfriamento: (1) Temperatura inicial (T_0), (2) Quantidade de iterações para cada temperatura (*SAmax*); (3) Regras de decrescimento da temperatura; e (4) Critério de parada do algoritmo.

De acordo com KirkPatrick et al. (1983), T_0 deve ser suficientemente grande para que exista probabilidade inicial de aceitação de movimentos próximo de 100%. Já a quantidade de iterações (*SAmax*) para cada temperatura deve ser proporcional ao tamanho da vizinhança (KOUVELIS; CHIANG, 1992; LIU; ONG, 2002). Com relação às regras de decrescimento da temperatura, há várias prescrições, sendo a mais comum (LIU; ONG, 2002) o decrescimento geométrico, isto é, a temperatura da *k*-ésima iteração é definida seguindo-se a fórmula $T_k = \alpha \times T_{k-1}$, em que α é uma constante com valores menores que 1 e é chamada de taxa de resfriamento, tendo valores típicos entre 0,8 e 0,99 (AARTS; KORST, 1989). Três critérios de parada são normalmente adotados (WANG, 2006): (a) Quando a temperatura chega a zero ou próximo de deste; (b) Quando o valor da função objetivo alcança uma solução predeterminada com qualidade satisfatória para o estudo em questão; e (c) Quando o tempo de processamento excede um limite predeterminado.

A Figura 1 mostra o pseudocódigo de um algoritmo *Simulated Annealing* básico.

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

Figura 1- Pseudocódigo do algoritmo *Simulated Annealing*.
Fonte: SOUZA, 2008.

Uma forma de determinar a temperatura inicial T_0 é por simulação. Em Souza (2008), descreve-se esze procedimento. Parte-se de uma solução inicial qualquer e de uma temperatura inicial com valor igual ao custo de uma solução aleatória. A partir daí, aplicando-se o próprio algoritmo *Simulated Annealing*, são contados quantos

movimentos são aceitos com a temperatura dada em x movimentos na temperatura em questão. Se Y% ou mais movimentos forem aceitos nessas x iterações, então é retornada a temperatura inicial para ser utilizada no método de refinamento *Simulated Annealing*; caso contrário, isto é, se menos que Y% dos movimentos forem aceitos, então é aplicado um aumento de β % na temperatura. O processo é repetido até que se tenha uma temperatura com Y% de movimentos aceitos. A Figura 2 mostra o pseudocódigo desse procedimento.

```
Procedimento DeterminaTemperatura Inicial (Y, β, x, T0, s)
1       T ← T0; {Temperatura corrente}
2       Continua ← true;
3       enquanto (Continua) faça
4               Aceitos ← 0;   {Número de soluções aceitas na temperatura T}
5               para IterT = 1 até x faça
7                       Gere um vizinho qualquer s' ∈ N(s);
8                       Δ = f(s') - f(s);
9                       se (Δ < 0)
10                              então
11                                Aceitos ← Aceitos + 1;
14                              senão
15                                Escolha x ∈ [0, 1];
16                                se (x < e^(-ΔT)) então Aceitos ← Aceitos + 1;
17                      fim-se;
18              fim-para;
9               se (Aceitos ≥ (Y × x)
10                      então Continua ← False;
11                      senão T ← (1 + β) × T;
19              fim-se;
21      fim-enquanto;
23      Retorne T;
fim DeterminaTemperatura Inicial;
```

Figura 2 - Pseudocódigo do procedimento de determinação da temperatura inicial

Fonte: SOUZA, 2008.

Um segundo algoritmo de refinamento utilizado para efeito de comparação e análise de eficiência na resolução do PPHE é baseado na metaeurística *GRASP* (*Greedy Randomized Adaptive Search Procedure*). *GRASP* é um método iterativo, proposto por Feo e Resende (1995), que consiste de duas fases: uma de construção, na qual uma solução é gerada, elemento a elemento; e de uma fase de busca local, em que um ótimo local na vizinhança da solução construída é pesquisado. A melhor solução encontrada ao longo de todas as iterações *GRASP* realizadas é retornada como resultado. A Figura 3 mostra o pseudocódigo do GRASP clássico.

A fase de construção tem como objetivo gerar uma solução viável e se caracteriza como um processo iterativo, adaptativo, randômico e guloso. Ele é iterativo, pois uma solução é construída elemento a elemento. É adaptativo porque a escolha de cada elemento da solução é feita através de uma função g que estima o benefício da inserção de cada candidato na solução parcial e que, portanto, depende da escolha anterior. É randômico e guloso, porque, dos elementos ainda não inseridos na solução, é feita uma lista restrita de candidatos (LRC), composta pelos melhores candidatos, e selecionado um desses elementos aleatoriamente. Para definir a lista restrita de candidatos, é necessário utilizar um parâmetro $\delta \in [0, 1]$. Fazem parte da LRC todos os elementos t, tais que $g(t) \leq min + \delta \times (max - min)$, em que min e max representam, respectivamente, o melhor e o pior valor segundo a função g.

```
Algoritmo GRASP (δ, N, s)
1       f* ← ∞;     {valor da melhor solução obtida até então}
2       IterGRASP ← 0; {Número de iterações GRASP}
3       enquanto (IterGRASP < N) faça
4               IterGRASP ← IterGRASP + 1;
5               s0 ← ConstruaSolucao(δ);
6               s ← BuscaLocal(s0);
7               se (f(s) < f*)
8                       então
9                         s* ← s;
10                        f* ← f(s);
11              fim-se;
12      fim-enquanto;
13      s ← s*;
14      Retorne s;
fim GRASP;
```

Figura 3 - Pseudocódigo do algoritmo GRASP.

Fonte: Adaptado de FEO; RESENDE, 1995.

O parâmetro δ, que determina o tamanho da lista restrita de candidatos, é basicamente o único parâmetro a ser ajustado na implementação de um procedimento *GRASP*. Em Feo e Resende (1995), discute-se o efeito do valor de δ na qualidade da solução e na diversidade das soluções geradas durante a fase de construção. Valores de δ que levam a uma lista restrita de candidatos de tamanho muito limitado (ou seja, valores de δ próximos da escolha gulosa) implicam soluções finais de qualidade muito próxima àquela obtida de forma puramente gulosa, obtidas com um baixo esforço computacional. Em contrapartida, provocam uma baixa diversidade de soluções construídas. Já uma escolha de δ próxima da seleção puramente aleatória leva a uma grande diversidade de soluções construídas, mas, no entanto, muitas das soluções construídas são de qualidade inferior, tornando mais lento o processo de busca local.

A solução encontrada pela fase de construção do *GRASP* não corresponde necessariamente a um ótimo local de um problema de otimização. Dessa forma, é

aconselhável aplicar um método de refinamento dessa solução através de um algoritmo de busca local (FEO; RESENDE, 1995).

3. Descrição do problema abordado

O problema tratado neste trabalho diz respeito ao Departamento de Ciências Exatas e Aplicadas (DECEA) da Universidade Federal de Ouro Preto (UFOP). Trata-se de um departamento em expansão, localizado no campus de João Monlevade, que no primeiro semestre letivo de 2008 trabalhou com 25 professores, 40 disciplinas e aproximadamente 250 alunos no horário noturno. Há cinco horários diários, de segunda a sexta, para a realização das aulas, e considera-se que as turmas estão sempre disponíveis.

Os requisitos considerados na elaboração do quadro de horários semanal são os seguintes: (a) Um mesmo professor não pode ministrar aulas em um mesmo horário para turmas diferentes; (b) Uma mesma turma não pode ter aulas simultaneamente com mais de um professor; (c) As aulas de um mesmo professor para uma mesma turma devem ser geminadas em dois ou três horários consecutivos no dia; e (d) A quantidade de vezes que o professor necessita ir à instituição na semana para ministrar aulas deve ser minimizada.

A alocação de aulas no DECEA ocorre manualmente e é um processo necessariamente demorado, além de desgastante, apesar da aparente simplicidade dos requisitos considerados. Esse fato decorre, como dito anteriormente, da característica combinatória do problema e, pelas dimensões consideradas, apresenta um número proibitivo de soluções a serem analisadas.

4. Modelagem do problema

4.1. Representação de uma solução

Uma solução do problema é representada por uma matriz $S = (s_{ij})_{mxn}$, em que m corresponde aos professores disponíveis para ministrar as aulas e n, os horários da semana. Em cada célula s_{ij} é colocado o número da turma t alocada para o professor i no horário j. Uma célula com valor 0 indica que o horário está vago; caso a célula contenha o valor −1, então o professor i no horário j está indisponível para ministrar aulas. Um exemplo simples de representação é dado na Figura 4.

	Horários											
	Segunda-Feira					Terça-feira					...	
Prof.	1	2	3	4	5	6	7	8	9	10	...	*n*
1	7	7	8	8	8	0	0	0	0	0	...	7
2	0	0	0	-1	-1	2	2	-1	-1	-1	...	0
3	3	3	7	7	2	4	4	2	2	2	...	4
4	1	1	3	3	3	5	5	6	6	6	...	1
5	-1	-1	2	2	7	7	7	7	3	3	...	6
6	0	0	0	0	0	6	6	0	1	1	...	3
7	4	4	4	0	0	-1	-1	3	5	5	...	5
8	2	2	5	5	5	0	0	5	0	0	...	-1
9	7	6	-1	-1	1	1	1	1	4	4	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
p	5	5	1	1	6	3	3	4	7	7	...	2

Figura 4 - Exemplo de uma solução.
Fonte: Elaborado pelo próprio autor.

Essa figura mostra, por exemplo, que o professor 1 dá aulas nos dois primeiros horários da segunda-feira para a turma 7 e nos três últimos horários para a turma 8, enquanto na terça-feira ele está com horários vagos.

Com essa representação, impede-se que um mesmo professor dê aula para mais de uma turma em um mesmo horário. Assim, o requisito (a) da Seção 3 é automaticamente atendido. No entanto, uma mesma turma pode ter aulas com mais de um professor ao mesmo tempo. Na Figura 4, pode-se observar, ainda, que a turma 7 tem aulas com os professores 1 e 9 no primeiro horário da segunda-feira, violando o requisito (b) da Seção 3.

4.2. Estrutura de vizinhança

Para a utilização de metaeurísticas, faz-se necessário definir uma estrutura de vizinhança, a qual é modelada para cada aplicação específica e influencia fortemente a qualidade das soluções geradas.

Para explorar o espaço de busca do PPHE foram utilizados dois tipos de movimentos, realocação e troca, para definir uma vizinhança $N(s)$ de uma solução s. O movimento de realocação consiste em realocar uma aula de determinado professor para algum horário vago. Já o movimento de troca consiste em trocar entre si os horários de duas turmas de um mesmo professor. Assim, uma solução $s' \in N(s)$ é um vizinho de s se ela pode ser acessada a partir desta por meio de um movimento ou de realocação ou de troca.

4.3. Função de avaliação

Para avaliar uma solução, os requisitos do problema foram primeiramente classificados em duas categorias: requisitos fortes e requisitos fracos, que diferem entre si pelo peso dado a cada qual. Os primeiros são aqueles que, se não forem satisfeitos, gerarão uma solução inviável; já os segundos são aqueles cujo atendimento é desejável, mas que, se não satisfeitos, não gerarão soluções inviáveis. No caso em análise, são requisitos fortes os itens (a) e (b) da Seção 3 e fracos, os itens (c) e (d). O resultado da função de avaliação é chamado de custo da solução.

A função de avaliação f que associa cada solução s do espaço de soluções a um número real $f(s)$ e deve ser minimizada, é computada pela equação (1):

$$f(s) = \omega_R \times R + \omega_{Q_1} \times Q_1 + \omega_{Q_2} \times Q_2 \qquad (1)$$

em que:

- R representa o número de vezes, na solução s, em que professores diferentes ministram aulas para uma mesma turma em um mesmo horário (violação ao requisito (b) da Seção 3).
- Q_1 indica o número de vezes em que as aulas de um professor não estão geminadas em dois ou três horários consecutivos no dia (violação ao requisito (c) da Seção 3).
- Q_2 representa o número de dias, na solução s, em que o professor comparece à instituição para dar aulas.
- ω_R, ω_{Q_1} e ω_{Q_2} são pesos que refletem a importância relativa de cada requisito. Os pesos utilizados são apresentados na Seção 7.

O requisito (a) da Seção 3 não é avaliado, pois, como mostrado na Subseção 4.1, ele é automaticamente atendido.

4.4. Geração de uma solução inicial

Uma solução inicial para o PPHE é obtida da seguinte forma: para cada professor, começando do primeiro, e para cada uma de suas aulas, é calculado o custo de inserção dessa aula em todos os horários disponíveis, segundo a equação (1). A seguir é feita uma lista ordenada, em ordem crescente, com os custos das inserções da aula desse professor em todos os horários. Para o algoritmo *Simulated Annealing*, aloca-se a aula no primeiro horário da lista, que é o de menor custo. Já para o GRASP dessa lista de horários, se cria, primeiramente, uma Lista Restrita de Candidatos (LRC) definida pelo parâmetro $\delta \in [0, 1]$. Dessa LRC, um horário é escolhido aleatoriamente, e a aula respectiva é alocada nesse horário. Esse procedimento é repetido até que todas as aulas de todos os professores estejam alocadas.

5. *Simulated annealing aplicado ao* PPHE

Neste trabalho, adaptou-se a metaeurística *Simulated Annealing* da seguinte forma: parte-se de uma solução construída conforme a Subseção 4.4. A seguir, sorteiam-se um professor e dois horários, nos quais esse professor esteja em atividade em turmas diferentes ou esteja em atividade em uma turma qualquer e ocioso no outro horário. Para o professor e os horários escolhidos, analisa-se o custo da modificação, de acordo com a função (1). Caso esse custo seja menor que o anterior, antes da troca, o movimento é aceito; caso seja maior, um cálculo de probabilidade é efetuado e, dependendo da temperatura de arrefecimento, o movimento é aceito ou não. A probabilidade de aceitar um movimento de piora é calculada com base na avaliação da expressão $e^{-(\Delta/T)}$, em que T é a temperatura corrente e Δ, a variação de custo.

No início do processo, a temperatura é alta e, com isso, também é alta a probabilidade de se gerarem soluções de pior custo. No entanto, à medida que a temperatura decresce, essa probabilidade diminui. A analogia que se faz com o processo físico é que, quando a temperatura é alta, as moléculas estão agitadas e podem mover-se para quaisquer lugares, gerando-se outros estados (configurações). Porém, quando a temperatura é baixa, as moléculas possuem baixa energia cinética e

praticamente só movem para estados de menor configuração de energia.

Para que no início do processo a temperatura fosse suficientemente alta, foi implementado o método de determinação da temperatura inicial por simulação, tal como descrito na Seção 2, à exceção da temperatura de partida, a qual é de 10% do valor da função de avaliação da solução inicial. Nesse caso, a solução inicial é obtida conforme a Subseção 4.4.

6. Grasp aplicado ao PPHE

A metaeurística GRASP foi implementada de forma similar ao algoritmo descrito na Seção 2. É gerada uma solução inicial na forma da Subseção 4.4. Sobre a solução inicial é aplicada busca local que, diferentemente da versão clássica, tem um mecanismo de relaxação adaptativa, conforme descrito a seguir. Se a solução proveniente da busca local for melhor que a melhor até o momento, é feita uma atualização da melhor solução. Esse procedimento de construir e refinar uma solução é repetido até que haja N iterações sem melhora na melhor solução.

O procedimento de busca local implementado funciona como segue. A cada iteração são avaliados todos os vizinhos gerados com os movimentos de realocação e troca descritos na Subseção 4.2. Se o melhor vizinho tiver valor para a função de avaliação (1) melhor do que o da solução corrente, move-se para esse vizinho, e a busca continua a partir desta. No entanto, se o melhor vizinho não gerar uma solução melhor, o método para.

A busca local, tal como apresentado, pára no primeiro ótimo local, que pode não ser um ótimo global. Para tentar escapar dessa situação, implementou-se um mecanismo de relaxação adaptativa, com base nas ideias de Schaefer (1996). Inicialmente, o peso de inviabilidade ω_R apresentado na Subseção 4.3 é reduzido para $\omega'_R = \omega_R/15$. Em seguida, é aplicada a busca local e a partir do ótimo local encontrado com esse novo peso, um novo refinamento é realizado com os pesos originais. Se esse ótimo local gerado for melhor que o primeiro, repete-se a mudança de ω_R para ω'_R. Caso contrário, muda-se o peso de inviabilidade ω_R para $\omega''_R = 1$, e prossegue-se como antes, isto é, aplica-se busca local na solução corrente, considerando-se o peso $\omega_R = 1$ na função de avaliação (1), seguida de nova busca local com pesos originais. Sendo essa operação bem-sucedida, isto é, produzindo novo ótimo local melhor, então todo o procedimento é repetido; caso contrário, o refinamento é definitivamente interrompido.

7. Resultados computacionais

Os algoritmos *Simulated Annealing* e GRASP foram implementados na linguagem C++, usando-se o compilador Borland C++ Builder 6.0 e testado em um microcomputador PC Intel Core 2 Duo, 2.1 GHz, com 4 GB de RAM, sob o sistema operacional Windows Vista.

Para testar os algoritmos foram usados dados relativos à distribuição de aulas do primeiro e segundo semestres letivos do ano de 2006 (DECEA 2006/1 e 2006/2), 2007 (DECEA 2007/1 e 2007/2) e 2008 (DECEA 2008/1). A Tabela 1 apresenta algumas características dessas instâncias. A última coluna apresenta o custo das soluções manuais submetidas à função de avaliação apresentada na Subseção 4.3.

Tabela 1- Características das instâncias consideradas

Instância	Número de Professores	Número de Turmas	Número de Horas-Aula	Custo da Solução Manual
DECEA2006/01	21	6	141	3450
DECEA2006/02	23	7	159	3425
DECEA2007/01	21	8	177	-
DECEA2007/02	22	8	175	-
DECEA2008/01	25	9	210	4280

Em relação à Tabela 1, observa-se que as instâncias de 2007 não apresentam soluções manuais, pois nesses semestres foram usadas as soluções obtidas pelo algoritmo *Simulated Annealing* desenvolvido neste trabalho.

Para avaliar uma solução foram aplicadas à função de avaliação, descrita pela equação (1), os seguintes pesos:

1. ω_R (peso pela existência de sobreposição, isto é, professores diferentes ministrando aulas para a mesma turma em um mesmo horário): 300 por horário de sobreposição.
2. ω_{Q_1} (peso por não respeitar à disposição de aulas diárias): 50 por aula.
3. ω_{Q_2} (peso pelo número de dias que o professor comparece à instituição): 40 por dia de comparecimento.

Para a determinação da temperatura (T_0) do *Simulated Annealing*, foram usados os seguintes parâmetros referenciados na Seção 2: Temperatura de

partida: T_0 = 10% do custo da solução inicial; Número de iterações em dada temperatura: x = 100; Incremento na temperatura corrente: β = 20%, e Taxa de movimentos aceitos em dada temperatura: Y = 60%. O valor de *SAmax* é calculado com base na expressão $SAmax = 50 \times p \times h$, em que p é o número de professores e h, o número de horários semanais. Já o valor da taxa de resfriamento α foi fixado em 0,98.

No algoritmo *GRASP*, considera-se como critério de parada um número máximo de iterações dado pelo parâmetro $N = 0{,}5 \times p \times h$. Assim, esse método é interrompido se houver N iterações sem melhora. O outro parâmetro do algoritmo, o valor de δ, que controla o nível de aleatoriedade, foi fixado em 0,1.

Inicialmente, o algoritmo GRASP proposto foi comparado com sua versão clássica, em que a busca local é feita de forma tradicional, sem a relaxação adaptativa descrita na Seção 6 e com um número de iterações $N = 2 \times p \times h$, isto é, quatro vezes maior que o GRASP proposto. Esse número de iterações mais elevado foi considerado para possibilitar ao GRASP tradicional obter soluções melhores. Na Tabela 2 são mostrados os melhores resultados e os resultados médios de 30 execuções de cada uma dessas versões.

Tabela 2 - Comparação GRASP proposto x GRASP tradicional

	GRASP tradicional		GRASP proposto	
	Melhor	Valor	Melhor	Valor
Instâncias	Valor	Médio	Valor	Médio
2006/1	4550	5298.33	2380	2471.00
2006/2	4180	4806.67	2400	2526.67
2007/1	4650	5635.00	2650	2819.00
2007/2	7110	7643.33	4960	5039.33
2008/1	5970	6705.00	3060	3314.00

Como pode ser verificado pela Tabela 2, o GRASP proposto produz soluções finais substancialmente melhores, tanto em relação aos melhores resultados quanto em relação aos resultados médios. Assim, apenas a versão GRASP com relaxação adaptativa foi considerada para comparação com o algoritmo *Simulated Annealing*.

Na Tabela 3 são apresentados os resultados encontrados pelos algoritmos propostos, bem como os da solução manual. Na coluna Melhor Valor, apresenta-se o melhor valor encontrado, segundo a função de avaliação (1). A coluna Valor Médio indica o valor médio de 30 execuções de cada algoritmo. A coluna Tempo indica o tempo médio de processamento, em segundos de cada algoritmo.

Tabela 3 - Resultados da comparação entre os algoritmos

	Simulated Annealing			GRASP			Manual
	Melhor	Valor	Tempo	Melhor	Valor	Tempo	Valor
Instâncias	Valor	Médio	(s)	Valor	Médio	(s)	
2006/1	2010	2236,67	126,87	2380	2471,00	235,55	3450
2006/2	2120	2305,33	180,27	2400	2526,67	277,53	3425
2007/1	2340	2449,67	147,41	2650	2819,00	383,70	-
2007/2	2330	2442,33	152,51	4960	5039,33	316,85	-
2008/1	2780	2973,33	190,66	3060	3314,00	526,95	4280

Como se observa, os algoritmos propostos produzem soluções finais de qualidade melhor que aquelas geradas manualmente pela instituição de ensino e em tempos computacionais reduzidos (menos de 10 min de processamento). Comparando os dois algoritmos propostos entre si, vê-se que o algoritmo *Simulated Annealing* produziu resultados substancialmente melhores que os do GRASP, tanto com relação aos melhores valores quanto com os valores médios.

A Figura 1 mostra os resultados finais dos algoritmos propostos para a instância 2006/1. Estes são apresentados por execução, partindo-se de uma mesma semente de números aleatórios. Observa-se que esta é a instância para a qual o algoritmo GRASP teve o melhor desempenho, em comparação com o *Simulated Annealing*, de acordo com a Tabela 3.

Como se observa pelo Figura 1, o *Simulated Annealing* produziu sempre soluções finais melhores que as do GRASP em cada execução. Nas outras instâncias, o mesmo comportamento foi observado.

A seguir, é feito um teste de significância para mostrar que a superioridade do algoritmo *Simulated Annealing* não foi obtida ao acaso. Para tanto, para cada bateria de testes relativo a cada instância foi aplicado o teste *t*, de Student (FREUND, 2006), com o nível de significância de 0,01, tendo como hipótese nula que a solução encontrada pelo algoritmo GRASP é melhor que a solução obtida pelo algoritmo *Simulated Annealing*, e como hipótese alternativa, o contrário; isto é, que a solução gerada pelo *Simulated Annealing* é melhor que a do GRASP.

Assim, as seguintes hipoteses são testadas:

- H_0: $\bar{x}_{SA} - \bar{x}_{GRASP} \geq 0$
- H_1: $\bar{x}_{SA} - \bar{x}_{GRASP} < 0$

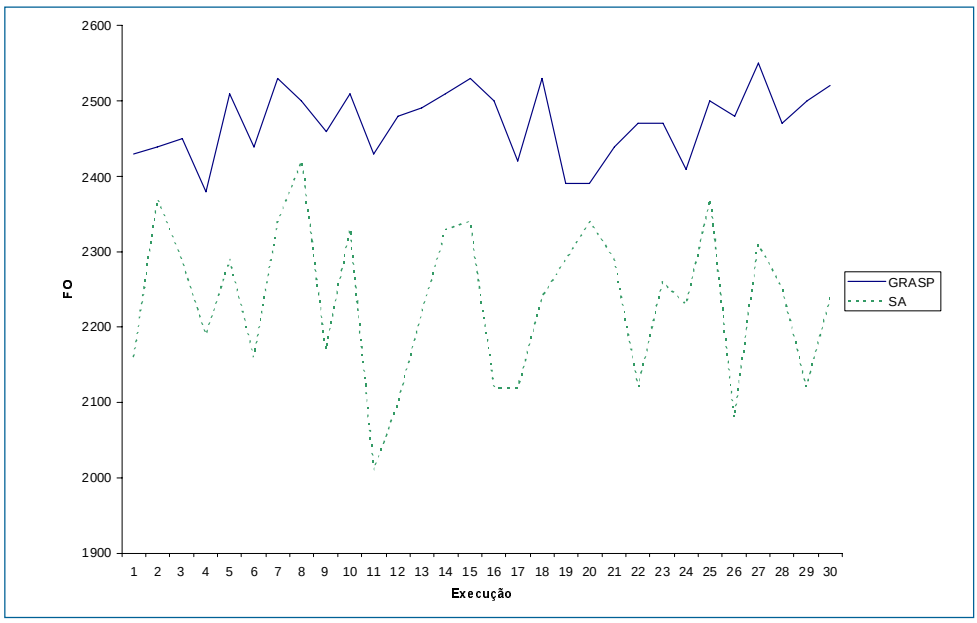

Figura 1 - Resultados finais dos algoritmos para a instância 2006/1.

Na Tabela 4 são apresentados os valores de *t* para cada instância, obtidos pela aplicação da fórmulas (2) e (3) (FREUND, 2006).

$$t = \frac{\bar{x}_{SA} - \bar{x}_{GRASP}}{s_p \sqrt{\frac{1}{n_{SA}} + \frac{1}{n_{GRASP}}}} \qquad (2)$$

$$s_p = \sqrt{\frac{(n_{SA}-1)s_{SA}^2 + (n_{GRASP}-1)s_{GRASP}^2}{n_{SA} + n_{GRASP} - 2}} \qquad (3)$$

Nessas expressões, $\bar{x}$ representa o valor médio da amostra, *n* é o tamanho da amostra e *s* é o desvio-padrão da amostra, todos de acordo com cada algoritmo.

Tabela 4 - Valores de *t* por instância

Instâncias	*t*
2006/1	-11.426
2006/2	-13.051
2007/1	-24.916
2007/2	-148.823
2008/1	-13.887

O valor crítico é t(0,01) = -2,326, ou seja, t < -2,326 rejeita H_0. Como se observa pela Tabela 4, a hipótese nula é rejeitada em todos os testes. Assim, pode-se concluir que o algoritmo *Simulated Annealing* gera soluções melhores que as do GRASP, com 99% de confiança.

8. Conclusões

Neste trabalho, foram apresentados dois algoritmos metaeurísticos, um baseado em *Simulated Annealing* e outro em GRASP, para resolver o problema de programação de aulas de uma instituição de ensino. Para testá-los, foram usados dados de cinco semestres letivos do Departamento de Ciências Exatas e Aplicadas (DECEA) da Universidade Federal de Ouro Preto. Os algoritmos desenvolvidos foram comparados entre si e com as soluções manuais aplicadas nesses semestres. A comparação foi feita em termos de melhor desempenho, desempenho médio e tempo de processamento, bem como uma análise de significância.

Os dois algoritmos mostraram-se capazes de encontrar boas soluções, superando em muito a qualidade das soluções manuais. Além disso, são autoadaptativos e, assim, não requerem calibração de seus parâmetros. No entanto, o algoritmo *Simulated Annealing* mostrou-se mais eficiente, por apresentar melhores soluções em todas as instâncias, tanto com relação aos melhores valores quanto aos valores médios, e através de testes de hipotese foi mostrado que isso não ocorreu ao acaso. Adicionalmente, o algoritmo *Simulated Annealing* requer um tempo de processamento bem inferior ao do algoritmo GRASP. Assim, pode-se concluir que, para o

problema de programação de horários em escolas, o algoritmo *Simulated Annealing* proposto é mais eficiente.

Com a execução deste trabalho, foi disponibilizada uma ferramenta computacional eficiente para o planejamento do horário de aulas do DECEA, que minimiza consideravelmente o tempo de execução necessário para tal atividade e aumenta a qualidade das soluções quando comparadas com as soluções obtidas manualmente.

Agradecimentos

Ao CNPq (Processo 474831/2007-8) e à FAPEMIG (Processo CEX 2991-6.01/07), bem como à Universidade Federal de Ouro Preto, pelo apoio ao desenvolvimento deste trabalho.

Referências

AARTS, E.; KORST, J. **Simulated annealing and boltzmann machines**. New York: John Wiley & Sons, 1989.

ABRAMSON, D. Constructing school timetables using simulated annealing: sequential and parallel algorithms. **Management Science**, v. 37, p. 98-113, 1991.

CALDEIRA, J.; AGOSTINHO, C. School timetabling using genetic search. In: INTERNATIONAL CONFERENCE ON PRACTICE AND THEORY OF AUTOMATED TIMETABLING, 2., 1997, PATAT. **Proceedings**... Toronto, 1997. p. 115-122.

CERNY, V. Thermo dynamical approach to traveling salesman problem: an efficient simulation algorithm. **Journal of Optimization Theory Applications**, v. 45, p. 41-51,1985.

COLLORNI, A.; DORIGO, M.; MANIEZZO, V. Metaheuristics for high school timetabling. **Computational Optimization and Applications**, v. 9, p. 275-298,1998.

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

FREUND, J.E. **Estatística aplicada**: economia, administração e contabilidade. 11. ed. Porto Alegre: Bookman, 2006.

FEO, T.A.; RESENDE, M.G.C. Greedy randomized adaptive search procedures. **Journal of Global Optimization,** v. 6, p. 109-133, 1995.

KIRPATRICK, S.; GELLATI, D.C.; VECCHI, M.P. Optimization by simulated annealing. **Science**, v. 220, p. 671-680, 1983.

KOUVELIS, P.; CHIANG, W.C. A simulated annealing procedure for single row lay-out problems in flexible manufacturing system. **Internacional Journal of Production Research**, v. 30, n. 4, p. 717-732, 1992.

LIU, S. Q.; ONG, H.L. A comparative study of algorithms for the flow-shop scheduling problem, Asia. **Pacific Journal of Operational Research**, v. 19, n. 4, p. 205-222, 2002.

METROPOLIS, M.A.R. Equation of state calculations by fast computing machines. **Journal of Chemical Physics**, v. 21, p. 1987-1992, 1953.

SCHAERF, A. Tabu search techniques for large high-school timetabling problems. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 30., 1996. **Proceedings**... [S.l. : s.n.], 1996. p. 363-368.

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

SANTOS, H.G. **Formulações e algoritmos para o problema de programação de horário de escolas.** 2007. 74 f. Tese (Doutorado) – Instituto de Computação, Universidade Federal Fluminense, Niterói, 2007.

SANTOS, H.G.; OCHI, L.S.; SOUZA, M.J.F. A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem. **ACM Journal of Experimental Algorithmics,** v. 10, art-2.09, p. 1-15, 2005.

SANTOS, H.G.; UCHOA, E.; OCHI, L.S. Extended formulation with cut and column generation for timetabling. In: ORP3 CONFERENCE 2007, 2007, Guimarães. **Anais**... Guimarães, 2007. p. 1-10.

SANTOS, H.G.; UCHOA, E.; OCHI, L.S. Formulação estendida com geração de cortes e colunas para o problema de programação de horarios em escolas. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONA, 39., 2007, Fortaleza. **Anais**... Rio de Janeiro: SOBRAPO, 2007. v. 1. p. 1844-1854.

SANTOS, H.G.; SOUZA, M.J.F. Programação de horários em instituições educacionais: formulações e algoritmos In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 39., 2007, Fortaleza. **Anais**... Rio de Janeiro: SOBRAPO, 2007. v.1, p. 2827-2882.

SOUZA, M.J.F. **Programação de horários em escolas**: uma aproximação por metaheurísticas. 2000. 149 f. Tese (Doutorado em Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2000.

SOUZA, M.J.F. **Inteligência computacional para otimização**. Ouro Preto, MG: UFOP, 2008. (Notas de Aula). Disponível em: <http://www.iceb.ufop.br/prof/marcone>. Acesso em: 22 fev. 2008.

SOUZA, M.J.F.; MACULAN, N.; OCHI, L.S. A GRASP-Tabu search algorithm for solving school timetabling problems. In: RESENDE, M.C.; SOUSA, J.P. (Eds.). **Metaheuristics**: computer decision-making. [S.l.]: Kluwer Academic Publishers, 2004. p. 659-672.

WANG, T.Y. Comparison of scheduling efficiency in two/three-machine no-wait flow-shop problem using simulated annealing and genetic algorithm, Asia. **Pacific Journal of Operational Research**, v. 23, n. 1, p. 41-59, 2006.

WREN, A. Scheduling, timetabling and rostering – A special relationship? In: BURKE, E.K.; CARTER, M.W. (Eds.). **The practice and theory of automated timetabling I.** Springer-Verlag: Lecture Notes in Computer Science, 1996. v. 1153, p. 46-75.

Recebido em 05/02/2009

Publicado em 02/10/2009