Formulações, heurísticas e um limite combinatório para o problema de alocação de salas de aula com demandas flexíveis

Junot Freire
Departamento de Ciência da Computação, Universidade Federal da Bahia
Av. Adhemar de Barros, s/n, Salvador, BA 40170-110, Brazil
junotsantos@dcc.ufba.br

Rafael A. Melo
Departamento de Ciência da Computação, Universidade Federal da Bahia
Av. Adhemar de Barros, s/n, Salvador, BA 40170-110, Brazil
melo@dcc.ufba.br

RESUMO

Este trabalho apresenta um problema de alocação de salas de aula com demandas flexíveis e recursos limitados que ocorre na Universidade Federal da Bahia. Duas variantes do problema são consideradas, e modeladas como problemas de programação inteira, baseadas nos seguintes critérios de otimização: a) maximização de aulas alocadas; b) minimização de aulas conflitantes. São propostas três heurísticas construtivas e um método de busca local. É apresentado um limite inferior combinatório em relação ao número de aulas conflitantes. Os resultados mostram que as heurísticas geram soluções ótimas ou bem próximas das ótimas para as instâncias consideradas. As técnicas propostas foram utilizadas com sucesso para determinar os quadros de horários das aulas práticas do Departamento de Ciência da Computação, possibilitando encontrar inviabilidades no planejamento e uma melhoria na alocação das salas de aula.

PALAVRAS CHAVE. Problema de alocação de salas de aula, programação inteira, heurísticas.

Tópicos: EDU - PO na Educação, OC - Otimização Combinatória

ABSTRACT

This paper presents a classroom allocation problem with flexible demands and limited resources that occurs at the Federal University of Bahia. Two variants of this problem are considered, and modeled as integer programming problems, based on the following optimization criteria: a) allocated classes maximization; b) conflicting classes minimization. Three constructive heuristics and a local search method are proposed. A combinatorial lower bound regarding the number of conflicting classes is also presented. Results show that the proposed heuristics generate optimal or near optimal solutions for the considered instances. The proposed techniques were used to successfully determine the timetable to practical lessons at the Computer Science Department, allowing to identify infeasibilities on the planning and an improvement in classroom allocation.

KEYWORDS. Classroom allocation problem, integer programming, heuristics.

Paper topics: EDU - OR in Education, CO - Combinatorial Optimization

1. Introdução e descrição do problema

Problemas de planejamento acadêmico são recorrentes em instituições de ensino básico e superior e aparecem em diversos casos, como apresentado em Phillips et al. [2015], Rudová et al. [2011] e Wehrer e Yellen [2014]. Esses problemas possuem aplicações tais como geração do quadro de salas de aula utilizadas ou na construção dos horários das aulas de laboratório de um departamento. A cada semestre, o Departamento de Ciência da Computação da Universidade Federal da Bahia encara a situação de distribuir um grande número de aulas práticas em uma quantidade limitada de laboratórios de computação que, quando resolvida de forma manual, demanda um grande período de tempo e nem sempre consegue chegar a uma solução de boa qualidade.

O problema estudado neste trabalho é caracterizado como um problema de alocação de salas de aula no quadro de horários universitário, como visto em Carter e Tovey [1992], sendo que turmas podem ter flexibilidade sobre quais os dias terão aulas alocadas. A flexibilidade ocorre pelo fato de as turmas alternarem suas aulas entre práticas (dadas em laboratórios) e teóricas (dadas em salas de aula comuns). O objetivo principal deste trabalho é desenvolver métodos computacionais baseados em heurísticas que encontrem boas soluções para este problema e visando a criação de um aplicativo resolvedor para ser utilizado pelas diferentes unidades da universidade.

Pode-se definir formalmente o problema como segue. Seja $L = \{l_1, l_2, \ldots, l_{|L|}\}$ um conjunto de salas de aula, onde cada sala l_i possui capacidade c_i; $T = \{t_1, t_2, \ldots, t_{|T|}\}$ um conjunto de turmas onde cada uma possui um número de aulas a serem alocadas m_t (que pode variar entre 1 e o número de aulas da turma), os horários de cada uma de suas aulas e a quantidade de alunos matriculados; $D = \{d_1, d_2, \ldots, d_{|D|}\}$ o conjunto de dias de aula e $H = \{h_1, h_2, \ldots, h_{|H|}\}$ o conjunto de possíveis horários de aula por dia (slots). A tabela de horários é um conjunto de slots $S = \{s_{l,d,h}\}$, onde cada slot é uma posição da tabela em uma determinada sala, dia e horário. Além disso, em cada turma $t \in T$, é definido um conjunto $L_t = \{l_{a_1}, \ldots, l_{a_k}\}$ de salas de aula nas quais t pode ser alocada e o conjunto $D_t = \{d_{a_1}, \ldots, d_{a_k}\}$ de dias nos quais t tem aulas. Também são definidos st_t^d e en_t^d como sendo, respectivamente, o primeiro e o último horário de aula de uma turma t em um dia d. O objetivo do problema consiste em encontrar uma solução que designe as aulas às salas de aula enquanto otimiza uma certa função objetivo. Considera-se neste trabalho a maximização do número de aulas alocadas e a minimização do número de conflitos (sendo um conflito definido como duas aulas designadas ao mesmo slot).

O restante do trabalho é organizado como segue. Na Seção 2 são vistos os dois modelos de programação inteira para o problema. Na Seção 3 são apresentadas as heurísticas construtivas para solução do problema. A Seção 4 descreve os métodos de busca local propostos. A Seção 5 apresenta propriedades adicionais do problema, incluindo o limite combinatório. Na Seção 6 são apresentados resultados preliminares utilizando as técnicas apresentadas e na Seção 7 são feitos comentários finais.

2. Modelos de programação inteira

2.1. Maximizando o número de aulas alocadas

Considere a variável de decisão $y_{t,l,d,h}$ sendo igual a 1 se há uma aula da turma t designada ao slot (l, d, h), e 0 caso contrário. O problema de maximização do número de aulas alocadas pode ser formulado pelo seguinte modelo de programação inteira:

$$\max \sum_{t\in T}\sum_{l\in L_t}\sum_{d\in D_t} y_{t,l,d,st_t^d}, \tag{1}$$

$$y_{t,l,d,st_t^d} = y_{t,l,d,h'}, \quad t \in T, l \in L,\ d \in D_t,\ st_t^d < h' \leq en_t^d \tag{2}$$

$$\sum_{l\in L_t} y_{t,l,d,st_t^d} \leq 1, \quad t = 1,\ldots,|T|,\ d \in D_t \tag{3}$$

$$\sum_{l\in L_t}\sum_{d\in D_t} y_{t,l,d,st_t^d} \leq m_t, \quad t = 1,\ldots,|T| \tag{4}$$

$$\sum_{\substack{t\in T:\\ l\in L_t, d\in D_t, st_t^d\leq h\leq en_t^d}} y_{t,l,d,h} \leq 1, \quad l \in L,\ d \in D,\ h \in H \tag{5}$$

$$y_{t,l,d,h} \in \{0,1\}, \quad t \in T,\ l \in L_t,\ d \in D_t,\ h \in H \tag{6}$$

A função objetivo (1) maximiza o número de aulas alocadas. As restrições (2) determinam que todas as aulas designadas estão alocadas de forma contínua na mesma sala. As restrições (3) obrigam cada aula a ser designada a no máximo uma sala. As restrições (4) determinam que o número de aulas alocadas de cada turma é menor ou igual ao seu número de aulas práticas. As restrições (5) determinam que no máximo uma aula pode ser alocada em cada slot. As restrições (6) definem a integralidade das variáveis.

2.2. Minimizando o número de conflitos

Caracteriza-se um conflito como uma posição da tabela de horários onde duas turmas estão alocadas. Observe que uma solução sem conflito implica que todas as aulas podem ser alocadas. Defina as variáveis $y_{t,l,d,h}$ como na seção anterior e as variáveis $z_{t,t',l,d,h}$, sendo iguais a 1 se existe conflito entre t e t' no slot $s_{l,d,h}$ e 0 caso contrário. O problema da minimização do número de conflitos pode ser modelado como:

$$\min \sum_{t=1}^{|T|-1}\sum_{t'=t_1+1}^{|T|}\sum_{l\in L_t\cap L_{t'}}\sum_{d\in D_t\cap D_{t'}}\sum_{h\in H_{t,d}\cap H_{t',d}} z_{t,t',l,d,h} \tag{7}$$

$$y_{t,l,d,st_t^d} = y_{t,l,d,h'}, \quad t = 1,\ldots,|T|,\ l = 1,\ldots,|L|,\ d \in D_t,\ st_t^d < h' \leq en_t^d \tag{8}$$

$$\sum_{l\in L_t} y_{t,l,d,st_t^d} \leq 1, \quad t = 1,\ldots,|T|,\ d \in D_t \tag{9}$$

$$\sum_{l\in L_t}\sum_{d\in D_t} y_{t,l,d,st_t^d} \leq m_t, \quad t = 1,\ldots,|T| \tag{10}$$

$$z_{t,t',l,d,h} \geq y_{t,l,d,h} + y_{t',l,d,h} - 1, \quad t = 1,\ldots,|T|-1,\ t' = t+1,\ldots,|T|,\ l \in L_t \cap L_{t'},\ d \in D_t \cap D_{t'},\ h \in H_{t,d} \cap H_{t',d} \tag{11}$$

$$z_{t,t',l,d,h} \in \{0,1\}, \quad t = 1,\ldots,|T|-1,\ t' = t+1,\ldots,|T|,\ l \in L_t \cap L_{t'},\ d \in D_t \cap D_{t'},\ h \in H_{t,d} \cap H_{t',d} \tag{12}$$

$$y_{t,l,d,h} \in \{0,1\}, \quad t = 1,\ldots,|T|,\ l \in L_t,\ d \in D_t,\ h \in H_{t,d} \tag{13}$$

A função objetivo (7) minimiza o número de aulas conflitantes. As restrições (8) garantem que todas as aulas designadas sejam alocadas de forma contínua na mesma sala. As restrições (9) obrigam cada aula a ser designada a no máximo uma sala. As restrições (10) determinam que o

número de aulas alocadas em cada turma seja menor ou igual ao seu número de aulas práticas. As restrições (11) forçam a variável $z_{t,t',l,d,h}$ para o valor 1 caso exista conflito entre t e t' no slot $s_{l,d,h}$. As restrições (12) e (13) definem a integralidade das variáveis.

3. Heurísticas construtivas

Três algoritmos construtivos foram implementados com o objetivo de obter boas soluções para o problema. Todos tentam maximizar o número de aulas devidamente alocadas no quadro de horários. O procedimento de caracterização das turmas não alocadas como conflitos em slots será apresentado na seção 3.1. Em seguimento serão apresentadas breves descrições para cada uma das heurísticas.

Const1: a heurística, descrita em Algoritmo 1, atribui aulas para os slots do quadro de horários de forma gulosa. Inicialmente todas as turmas são divididas em suas aulas e cada aula é inserida em uma lista Q_h representando um determinado período h de início de aulas. Cada uma dessas listas são então ordenadas de acordo com as seguintes prioridades: duração (número de slots que ocupa), número de estudantes matriculados para a aula e porcentagem de aulas práticas da respectiva turma. Depois destes passos, as listas são então percorridas seguindo a ordenação e suas aulas são alocadas no quadro de horários até que não existam salas disponíveis ou até que a lista fique vazia.

Const2: a heurística, descrita em Algoritmo 2, é similar ao Algoritmo 1, com a diferença que as turmas não são divididas em suas aulas, e armazena todas as aulas em uma única lista Q que é ordenada segundo as mesmas prioridades usadas em Const1. O algoritmo itera sobre a lista tentando alocar as aulas de cada turma no quadro de horários.

Const3: a heurística, descrita em Algoritmo 3, identifica um subconjunto de turmas regulares (turmas sem sobreposição de horários) que fazem com que o problema de designação de aulas seja computável polinomialmente e resolve esse subproblema como um emparelhamento de cardinalidade máxima em um grafo bipartido. Uma das partes da partição é composta de aulas pertencentes às turmas consideradas e a outra é composta dos slots do quadro de horários. A solução resultante do problema de emparelhamento é então convertida em uma solução parcial do problema original e as turmas restantes, que não foram alocadas pelo algoritmo de emparelhamento, são alocadas usando Const1 ou Const2.

Algoritmo 1 Const 1

```
1: Entrada: Turma T[], Laboratorio L[]
2: tabela ← criaTabela(∅)
3: Q[] ← separaAulas(T)
4: for cada posicao i em Q[] do
5:     while !Q[i].isEmpty() and !todosLaboratoriosOcupados(i) do
6:         aula ← Q[i].desenfileira()
7:         tabela.alocaAula(aula)
8:     end while
9: end for
```

Algoritmo 2 Const 2

```
1: Entrada: Turma T[], Laboratorio L[]
2: tabela ← criaTabela(∅)
3: Q ← T
4: while !Q.isEmpty() do
5:     turma ← Q.desenfileira()
6:     for cada aula a_i em turma do
7:         if alocavel(a_i) then
8:             tabela.alocaAula(a_i)
9:         end if
10:    end for
11: end while
```

Algoritmo 3 Const 3

```
1: Entrada: Turma turmas[], Laboratorio l[]
2: demandasRegulares ← separaDemandasRegulares(turmas)
3: tabela ← criaTabela(∅)
4: grafo ← geraGrafoBipartido(demandasRegulares, tabela)
5: grafoMatch ← geraMatching(grafo)
6: for cada aresta e em grafoMatch do
7:     tabela.alocaAula(e.getAula())
8: end for
```

3.1. Adaptando as heurísticas construtivas para minimizar conflitos

Foi também implementado um algoritmo complementar utilizado em todas as versões construtivas para determinar a posição na tabela de horários em que as turmas não alocadas devem ser dispostas para minimizar o número de conflitos a partir de uma determinada alocação.

O Algoritmo 4 realiza o processo de sugestão da faixa contínua de slots para se colocar uma aula não alocada para minimizar o número de conflitos dado um quadro de horários com aulas já alocadas. Para fazer isso, um quadro de horários especial é criado para alocar os conflitos. Para cada aula não alocada, é calculada a melhor posição que ela pode ocupar nesta tabela para criar o menor valor de conflito. Se a aula precisar ser alocada, ela será armazenada na tabela especial de horários nos slots que lhe foram sugeridos.

Algoritmo 4 Alocação de conflitos

```
1: Entrada: Turma naoAlocadas[], Laboratorio L[]
2: tabelaConflitos ← criaTabelaConflitos(∅)
3: for cada turma t_i em naoAlocadas[] do
4:     for cada aula a_j em t_i do
5:         conflito ← calculaConflito(a_j)
6:         if t_i.faltaAlocar(a_j) then
7:             tabelaConflitos.alocaConflito(a_i, conflito)
8:         end if
9:     end for
10: end for
```

4. Busca local: vizinhanças e algoritmos

Um procedimento de busca local foi implementado para melhorar as soluções obtidas usando uma heurística construtiva que combina Const3 e Const1, conforme explicado na seção 3. A busca local considera que as aulas não alocadas já foram convertidas em conflitos através do Algoritmo 4.

O Algoritmo 5 percorre todas as aulas práticas de todas as turmas não alocadas devidamente verificando se é possível inserí-las no mapa de horários através da chamada ao Algoritmo 6. Caso alguma turma consiga ser inserida no quadro de horários, então esta turma se caracteriza como conflito não-obrigatório e é finalmente alocada na tabela .

O Algoritmo 6 visa realocar as turmas que estão no período de tempo que t necessita de forma que t possa ser alocada naquele período. Para isso, é determinada a melhor posição que t pode ocupar no quadro de horários ao variar os labotarórios ou horários disponíveis. Através desta variação, busca-se encontrar qual posição causa o menor conflito por slots possível. Ao final da execução, o algoritmo retorna se foi possível realocar todas as aulas presentes naquele período para permitir a inserção de t.

Algoritmo 5 Adapta solução

```
1:  Entrada: MapaHorarios tabela, Turma naoAlocadas[]
2:  for cada turma t em naoAlocadas do
3:      for cada aula a em t do
4:          if adaptaRegiao(tabela, a) then
5:              tabela.aloca(a)
6:          end if
7:      end for
8:  end for
```

Algoritmo 6 Adapta região

```
1:  Entrada: MapaHorarios tabela,Aula t
2:  calculaLaboratorioPreferencial(t)
3:  for cada aula x alocada no periodo de t do
4:      tabela.reAloca(x)
5:  end for
6:  if periodo de t não possui aulas alocadas then
7:      return true
8:  else
9:      return false
10: end if
```

4.1. Outras vizinhanças

Além do procedimento anterior de busca local, outras vizinhanças foram criadas para aumentar as chances de otimização das soluções geradas pelos métodos construtivos, como visto em Fonseca e Santos [2014] e Fonseca et al. [2014]. São elas:

1. Move: Consiste em mover uma aula alocada para outra posição disponível. A nova posição deve estar vazia para alocar a aula e a melhoria deste movimento é dada através da redução do número de conflitos em comparação com a solução inicial.

2. Swap: Consiste em trocar uma aula alocada por uma não alocada de características semelhantes (esteja no mesmo dia, comece no mesmo horário e possua duração menor ou igual à aula alocada). A melhoria é dada pelo número de slots que ficarão desocupados após a troca.
3. Suggestion Move: Esta vizinhança consiste em mover uma turma não alocada que está gerando conflitos para outra posição onde ela gere menos conflitos. A melhoria é dado pela quantidade de conflitos que serão removidos pela movimentação da aula não alocada para outra posição.

O algoritmo final de busca local combina as três vizinhanças apresentadas acima com o Algoritmo 5, de forma que as vizinhanças perturbem uma solução ou tentem melhorar alguns valores (como número de conflitos) para que o método consiga alocar mais aulas de forma eficiente.

5. Propriedades do problema considerando modelagem em grafos

5.1. Componentes conexas

Uma forma de resolver o problema tratado neste artigo de forma mais rápida é decompondo-o em subproblemas menores e independentes entre si. No entanto, ao contrário do problema geral de alocação de aulas em salas de aula, ao considerar a existência de demandas flexíveis (aulas que tem mais de uma opção de dia para serem ministradas) não é possível tratar cada dia da semana como um subproblema independente.

Considerando a existência de demandas flexíveis, é possível decompor o problema em subproblemas através de suas componentes conexas (considerando um grafo bipartido tendo suas partes como as demandas e os slots do quadro de horários, respectivamente). Desta forma, é possível resolver separadamente cada componente conexa como um problema de alocação de salas de aula.

5.2. Limite inferior combinatório

Para se mensurar quão bons são os algoritmos construtivos e as buscas locais, sem fazer o uso de um resolvedor comercial, pode-se determinar quantos conflitos existem obrigatoriamente dada uma instância do problema. Aulas obrigatoriamente não alocadas são caracterizadas como aulas nas quais não é possível a sua alocação devido a estrutura da instância fornecida como entrada.

Para instâncias sem sobreposição de horários, isto é, turmas que ao iniciar em horários diferentes não possuem horários como interseção, a solução ótima do problema pode ser determinada polinomialmente como um problema de emparelhamento de cardinalidade máxima em um grafo bipartido, como foi visto no Algoritmo 3. Um limite inferior combinatório para o número de aulas não alocadas pode ser, portanto, definido como o número de turmas não emparelhadas em um emparelhamento de cardinalidade máxima considerando o maior subconjunto da instância utilizada com demandas sem sobreposição de horários.

O limite inferior dado no parágrafo anterior pode ser melhorado ao considerar demandas não regulares como é visto a seguir. Dada uma instância I do problema, um subconjunto de demandas regulares R desta instância e o limite inferior de R sendo igual a k. Se ao fixar previamente uma demanda não regular x na solução final, o novo valor do limite inferior para R for maior que k, então x em conjunto com R indica mais uma aula obrigatoriamente não alocada para I e x pertence a alguma componente conexa de R.

6. Resultados preliminares

Três conjuntos de teste com 5 instâncias cada foram usados para verificar a corretude dos algoritmos e mesurar a qualidade das soluções encontradas. Cada teste possui um percentual de quantas turmas com estrutura regular existem em suas instâncias: 74% (conjunto A), 100% (conjunto B) e 26% (conjunto C). Cada instância possui 45 turmas. Os experimentos computacionais foram realizados utilizando-se uma máquina All-in-One HP, com processador Intel(R) Core(TM) i5-4690T CPU @ 2.50GHz, 8GB de memória, utilizando o sistema operacional Xubuntu. Vale ressaltar que o tempo de execução de todas as heurísticas foi menor que 0,2 segundos para cada uma das instâncias testadas.

Todos os conjuntos foram submetidos a todos os algoritmos e a dois modelos de formulação de programação inteira para comparar as soluções. Os resultados são apresentados na Tabela 1. Os resultados em negrito indicam a melhor solução encontrada usando uma das heurísticas. Os dados indicam que Const3 combinado com o procedimento de busca local (Const3+BL) possui a melhor performance em comparação com as demais, chegando a soluções que são quase ótimas, principalmente em relação ao número de aulas alocadas. É observado também que Const3+BL obtém melhores resultados que Const3 ao considerar o número de conflitos.

Tabela 1: Resultados preliminares usando as heurísticas e a correspondência com as soluções ótimas

Instâncias		Const1		Const2		Const3		Const3+BL		Ótimo	
		#con	#aul	#con	#aul	#con	#aul	#con	#aul	#con	#aul
Grupo A	A10	**6**	**48**	9	47	**6**	**48**	**6**	**48**	6	48
	A20	2	**51**	**1**	**51**	2	50	**1**	**51**	1	51
	A30	8	47	8	**48**	8	46	**7**	47	6	48
	A37	7	**48**	6	**48**	11	46	**5**	**48**	5	48
	A40	10	47	**5**	**49**	9	45	9	47	5	49
Grupo B	B110	4	48	2	49	**0**	**50**	**0**	**50**	0	50
	B120	**6**	**47**	**6**	**47**	**6**	**47**	**6**	**47**	6	47
	B130	**10**	**45**	**10**	**45**	**10**	**45**	**10**	**45**	10	45
	B137	2	49	**0**	**50**	**0**	**50**	**0**	**50**	0	50
	B140	**2**	**49**	**2**	**49**	**2**	**49**	**2**	**49**	2	49
Grupo C	C60	9	46	9	47	10	46	**6**	**48**	6	48
	C70	12	48	11	47	6	48	**5**	**49**	5	49
	C80	**16**	**45**	17	**45**	18	43	**16**	44	15	45
	C90	19	**45**	23	43	**15**	**45**	**15**	**45**	15	45
	C100	14	**44**	14	**44**	14	43	**13**	**44**	13	45

7. Comentários finais

Como é possível observar pelos resultados da Seção 6, as heurísticas apresentadas permitiram encontrar soluções de alta qualidade. Vale ressaltar que os algoritmos apresentados foram cruciais para a determinação do quadro de horários dos laboratórios de computação do Departamento de Ciência da Computação na Universidade Federal da Bahia, ao encontrar muito rapidamente soluções de alta qualidade, reduzindo o trabalho manual e o tempo para se realizar tal tarefa.

Para trabalhos futuros pode-se propor métodos que considerem a indisponibilidade de slots no quadro de horários antes de gerar a alocação, a flexibilização da alocação de turmas para laboratórios que não as comportem completamente, considerando um custo associado a este processo e a elaboração de metaheurísticas.

Agradecimentos: Os autores agradecem ao PIBIC/UFBA-FAPESB e ao CNPQ pelo apoio financeiro.

Referências

Carter, M. e Tovey, C. (1992). When is the classroom assignment problem hard? *Operations Research*, 40(1-supplement-1):S28–S39.

Fonseca, G. e Santos, H. (2014). Variable neighborhood search based algorithms for high school timetabling. *Computers & Operations Research*, 52, Part B:203 – 208. Recent advances in Variable neighborhood search.

Fonseca, G., Santos, H., Toffolo, T., Brito, S., e Souza, M. (2014). Goal solver: a hybrid local search based solver for high school timetabling. *Annals of Operations Research*, p. 1–21.

Phillips, A., Waterer, H., Ehrgott, M., e Ryan, D. (2015). Integer programming methods for large-scale practical classroom assignment problems. *Computers & Operations Research*, 53:42 – 53.

Rudová, H., Müller, T., e Murray, K. (2011). Complex university course timetabling. *Journal of Scheduling*, 14(2):187–207.

Wehrer, A. e Yellen, J. (2014). The design and implementation of an interactive course-timetabling system. *Annals of Operations Research*, 218(1):327–345.