Departamento de Ciência da Computação - UFJF

DCC163 - Pesquisa Operacional

Problema de alocação de corredor

Tales Lopes Silva
Filipe Barreto
Igor Tibiriçá

Novembro / 2018

1 Introdução

O problema de alocação de corredor [Santana and dos Santos,], ou CAP (Corridor Allocation Problem) , é um problema NP-difícil e possui como objetivo a alocação de instalações ao longo de um corredor, sem sobreposição, de forma a otimizar sua comunicação, ou seja, minimizar o custo de comunicação entre suas instalações. Essa organização física recebe o nome de *leiaute*.

Para desenvolvimento do problema são consideradas duas filas horizontais ao longo do corredor, além de duas regras para o desenvolvimento da solução: (i) o início do corredor representa um ponto de origem em comum para ambas as filas, (ii) não é permitido a existência de espaços vazios entre as instalações. A **figura 1** exemplifica o uso e não uso dessas duas regras.

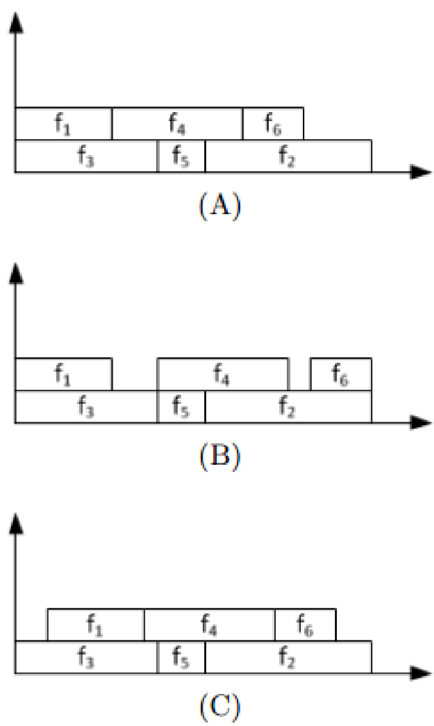

Figura 1. Exemplo de solução viável (a) e soluções inviáveis (b) e (c).

A ideia de minimizar o custo de comunicação entre as instalações de um corredor surge a partir da necessidade de organização que alguns lugares, como escolas, mercados e hospitais, possuem devido ao intenso tráfego entre salas. Esse fato serve, também, como justificativa para a escolha do tema desse trabalho, visto que é um problema atual e rotineiro com aplicações práticas, sendo de extrema importância seu estudo para um melhor planejamento de locais onde o tempo atua de forma determinante.

O trabalho possui como objetivo a aplicação do *método simplex*, com as respectivas definições de regras e restrições, para a resolução e desenvolvimento do problema que envolve testes e análises de resultados.

2 Trabalhos Relacionados

O problema de alocação de instalações foi primeiramente proposto por [Simmons, 1969] e recentemente introduzido por [Amaral, 2012]. [Amaral, 2012] utiliza programação inteira e obtém solução ótima para instâncias de até 13 instalações, utilizando cplex 12.1.0 como resolvedor. Já para instâncias contendo 15 instalações a utilização do cplex 12.1.0 não obteve solução ótima, mesmo após 8,6 horas de execução, em decorrência disso heurísticas de busca em vizinhança também foram utilizadas para abordar o problema em instâncias de até 30 instalações.

O tema também foi estudado em diversos outros trabalhos que abordam o problema de diferentes perspectivas. [Ghosh and Kothari, 2012a] ressaltam que CAP é parte de um grupo de problemas de disposição de instalações que envolve a organização de instalações em fileira única (single row facility layout problem, SRFLP) e a organização de instalações em fileira dupla (double row facility layout problem, DRFLP), sendo esse último o foco de seu trabalho. Em decorrência dos resultados obtidos por [Amaral, 2012], [Ghosh and Kothari, 2012a] propõem 2 metaheurísticas a ser aplicadas em instâncias de médio e grande porte do problema CAP, mais em específico o SRFLP.

Outras estratégias que abordam SRFLP foram realizadas em [Kothari and Ghosh, 2013], onde é apresentada uma heurística de busca em vizinhança, LK-INSERT, para tratar o problema, e [Kothari and Ghosh, 2014], onde são apresentados quatro algoritmos de dispersão para resolver instâncias de grande porte. Além disso, [Kothari and Ghosh, 2012] faz uma revisão literária para uma maior compreensão desse tipo de abordagem para o problema.

[Chung and Tanchoco, 2010] introduzem a abordagem que utiliza fileira dupla, DRFLP, porém, as duas regras descritas na seção 1 desse trabalho, não precisam, necessáriamente, serem cumpridas, como mostra a **figura 2**.

A abordagem de [Amaral, 2012] para resolver o problema CAP com programação inteira se mostrou ineficiente para instâncias maiores devido ao problema pertencer à classe NP-

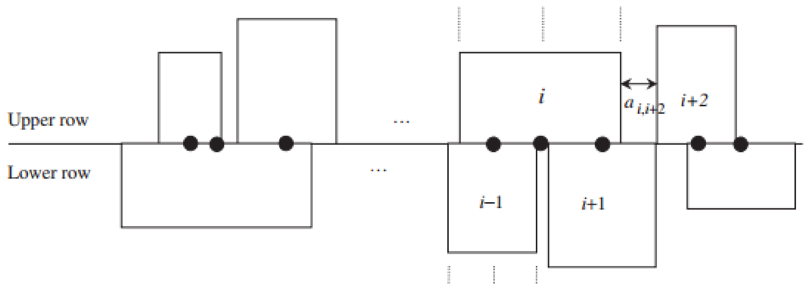

Figura 2. DRFLP

Difícil, com isso outros trabalhos que abordam a solução do problema CAP surgiram afim de fazer uso de diferentes heurísticas para lidar com instâncias maiores. [Santana and dos Santos,] utilizam uma busca local iterada, [Ahonen et al., 2014] abordam o problema utilizando busca Simulated Annealing e busca Tabu, enquanto que, [Ghosh and Kothari, 2012b] utilizam um algoritmo genético híbrido afim de solucionar o problema.

Todos os trabalhos descritos acima foram estudados e servem como embasamento teórico para compreensão do problema e resolução desse trabalho. Além disso, foram observadas quais instâncias foram utlizadas, pelos autores, para trabalhar com o problema, afim de que as mesmas instâncias pudessem ser obtidas e utilizadas no presente trabalho com o propósito de comparação com os resultados da literatura, senão, que ideias de construção das mesmas pudessem ser aplicadas.

3 Definição do problema

O problema de alocação de corredor, CAP, tem como objetivo a organização de salas ou instalações ao longo de um corredor, minimizando o custo de comunicação entre as mesmas. Os corredores podem ser representados através de linhas horizontais paralelas ao eixo x, no plano cartesiano, onde cada instalação possui um comprimento específico ao longo desse eixo. Um lado do corredor, com as respectivas salas alocadas, recebe o nome de *alocação*. A distância, largura, entre as duas alocações é desprezada nesse problema.

Como mencionado na seção 1, as duas regras impostas para o problema são: (i) o início do corredor representa um ponto de origem em comum para ambas as filas, (ii) não é permitido a existência de espaços vazios entre as instalações.

Um *leiaute* representa uma solução para a organização das instalações, ou a melhor combinação de alocações, cujo objetivo é encontrar o melhor *leiaute* possível. O custo de um *leiaute* se dá pela soma do centro de cada par de instalações do modelo. A utilização do centro de cada instalação é justificada pelo fato de que a porta de cada sala está situada no centro de sua respectiva instalação.

De acordo com [Amaral, 2012] o *leiaute* de uma solução pode ser representado como uma permutação P em dois componentes $[p_1, p_2]$, onde p_1 e p_2 representam alocações constituídas pelo arranjo das instalações em cada um dos lados do corredor. Um exemplo retirado de [Santana and dos Santos,] pode ser visto na **figura 3**, onde P = (5,1,7,2,4,6,3), p_1 = (5,1,7,2) e p_2 = (4,6,3). É importante notar que, nesse exemplo, as 2 regras para o problema CAP são obedecidas.

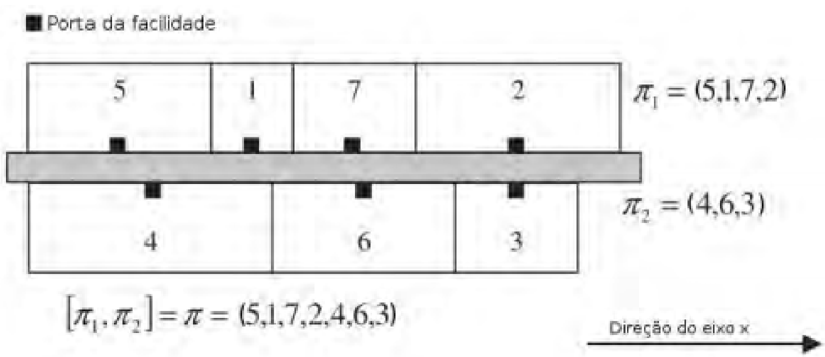

Figura 3. Exemplo de solução de CAP, retirado de [Santana and dos Santos,], adaptaado de [Ahonen et al., 2014]

Alguns valores a ser considerados para o CAP são:

n	número de instalações
$C_{\pi_r(i),\pi_s(j)}$	média de tráfego entre i-ésima e j-ésima instalação da linha s $(1 \leq r \leq s \leq 2)$
$x_{\pi_r(i)}$	abscissa do centro da i-ésima instalação na linha r $(r \in 1, 2)$

De acordo com [Amaral, 2012], a função objetivo a ser minimizada está explícita em (1), presente na **figura 4**, e se traduz pela soma do custo de comunicação entre cada par de sala. A função objetivo considera custos de comunicação entre elementos de uma mesma fila e filas opostas. A restrição que garante a não existência de espaços entre instalações adjacentes, e a não sobreposição de instalações, é demonstrada em (2), também presente na **figura 4**.

Uma instância do problema CAP pode ser representada por (F,L,C), onde F é o conjunto de instalações, L é um vetor contendo o comprimento das instalações e C é a matriz de

$$\min_{[\pi_1,\pi_2]:[\pi_1,\pi_2]\in\Pi_n} \left\{ \sum_{i=1}^{|\pi_1|-1} \sum_{j=i+1}^{|\pi_1|} c_{\pi_1(i),\pi_1(j)} |x_{\pi_1(i)} - x_{\pi_1(j)}| \right.$$

$$+ \sum_{i=1}^{|\pi_2|-1} \sum_{j=i+1}^{|\pi_2|} c_{\pi_2(i),\pi_2(j)} |x_{\pi_2(i)} - x_{\pi_2(j)}|$$

$$\left. + \sum_{i=1}^{|\pi_1|} \sum_{j=1}^{|\pi_2|} c_{\pi_1(i),\pi_2(j)} |x_{\pi_1(i)} - x_{\pi_2(j)}| \right\} \qquad (1)$$

onde,

$$x_{\pi_r(j)} = \frac{l_{\pi_r(j)}}{2} + \sum_{i=1}^{j-1} l_{\pi_r(i)}, r \in 1,2; 1 \leq j \leq n. \qquad (2)$$

e $l_{\pi_r(j)}$ é o tamanho da *j-ésima* facilidade na linha r.

Figura 4.

pesos.

4 Formulações matemáticas

O modelo de programação linear a seguir define como devem ser organizadas as salas ao longo dos dois lados de um corredor. Considerando o conjunto de salas (instalações) I = {1...n} e, seja A = {a_1,a_2,...,a_n} o conjunto de composições de lados do corredor (alocações), temos as seguintes informações:

DADOS

1. s_{ia}: 1 caso a sala i esteja em a, 0 caso contrário
2. p_{ia}: posição da porta (abscissa) da sala i em a
3. t_{ij}: tráfego entre sala i e j

VARIÁVEIS

1. x_a: variável do tipo inteira que possui valor 1 caso alguma alocação de a tenha sido selecionada.

2. y_i: variável fracionária que indica a posição da porta da sala i.

3. d_{ij}: variável fracionária que indica a distância da sala i para a sala j.

$$\text{Minimizar} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij}d_{ij} \tag{1a}$$

sujeito a

$$\sum_{a} x_a = 2 \qquad \forall a \in A \tag{1b}$$

$$\sum_{a} s_{ia}x_a = 1 \qquad \forall i \in I \tag{1c}$$

$$\sum_{a} p_{ia}x_a = p_i \qquad \forall i \in I \tag{1d}$$

$$p_i - p_j \leq d_{ij} \qquad \forall i, j \in I \tag{1e}$$

$$p_j - p_i \leq d_{ij} \qquad \forall i, j \in I \tag{1f}$$

A função objetivo, definida em (1a), consiste em minimizar o custo total de tráfego entre salas de um corredor, com $d_{ij} \geq 0$ e $t_{ij} \geq 0$. Analisando as restrições temos que, (1b) garante a existência de, apenas, duas alocações, na solução, com $x_a \in \{0, 1\}$. (1c) define que toda sala está em apenas uma das alocações. (1d) garante, apenas, a utilização da porta p_{ia} referente à alocação a, utilizada. Ambas restrições (1e) e (1f) estão relacionadas com o cálculo da distância entre as salas e, garantem que a distância entre a abiscissa de duas salas será, no máximo, igual a distância entre essas salas, respeitando $y_i \geq 0$.

5 Algoritmo proposto

A abordagem proposta, para resolver o problema, consiste na utilização de uma heurística construtiva para criação de uma solução inicial, seguida da geração de múltiplas soluções, utilizando a solução obtida pelo construtivo, como ponto de partida. Por fim, é empregado a aplicação do resolvedor Gurobi, para geração da solução final. A explicação de cada algoritmo é descrita a seguir.

Todo o problema foi desenvolvido na linguagem Python, utilizando o sistema operacional Ubuntu 14.4, uma máquina com processandor i5 e 4mb de memória ram.

5.1 Leitura das Instâncias

Primeiramente, para o desenvolvimento do trabalho, é necessário a realização da leitura das instâncias. Cada instância consiste em dois arquivos .txt, um contendo o tamanho das partições (*tamanho.txt*), outro contendo os tréfegos entre cada sala presente na instância (*trafego.txt*). O tamanho das salas é lido e salvo em um vetor, enquanto que, o tráfego entre as salas é representado através de um matriz triangular superior. Os valores abaixo da diagonal principal, incluindo a diagonal principal da matriz, tiveram valor atribuído igual a -1, visto que existem valores de tráfego iguais a 0, nas instâncias.

5.2 Abordagem Construtiva

A abordagem construtiva, baseada nas estretégia descrita em [Santana and dos Santos,], consiste em minimizar a distância entre salas que possuam alto tráfego entre si. A ideia se baseia no balanceamento de alocações, no que diz respeito à tráfego e distância. A função objetivo, de maximização, é atendida, tanto aproximando salas pequenas, quanto salas que possuam um alto valor de tráfego entre si.

Sendo assim, dado duas alocações i e j, o valor que queremos maximizar é calculado pela razão definida em (2).

$$z = trafego(i,j)/distancia(i,j) \tag{2}$$

Inicialmente, duas salas, da instância, são colocadas na solução. Uma é inserida na primeira alocação, outra na segunda alocação. Em seguida, para cada umas das sala que ainda não está na solução, é realizado o cálculo de z com as últimas salas inseridas em cada uma das alocações. Logo, a execução do procedimento construtivo pode ser considerada como uma estratégia $gulosa$, maximizando z para salas adjacentes. Um exemplo, de como é realizado o procedimento, pode ser visto na **figura 5**.

5.3 Geração de Alocações

A partir da primeira solução, conténdo duas alocações, gerada pelo algoritmo construtivo, outras soluções, também conténdo duas alocações, são geradas.

Duas são as formas para a geração das outras soluções: (i) realizar a troca entre duas posições, aleatórias, da mesma alocação, **figura 6**, (ii) realizar a troca entre duas posições, aleatórias, de alocações distindas, **figura 7**, da mesma solução. Apenas a estratégia (i), **figura 6**, foi adotada nesse trabalho.

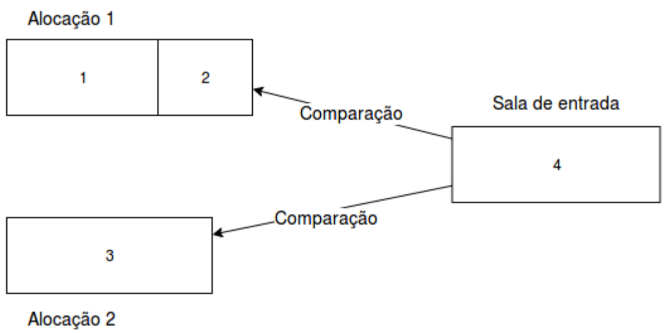

Figura 5. Comparação entre salas fora da solução e a última sala inserida em cada alocação.

Toda a abordagem construtiva é realizada N vezes, cada uma das vezes é gerada uma nova solução inicial, e, com base na solução gerada, x alocações são geradas. Todas as alocações geradas, após as N execuções do construtivo, são utilizadas como variáveis para o resolvedor.

Por fim o modelo de programação linear (PL) é implementado no *Gurobi* seguindo o modelo definido na **seção 4** desse trabalho.

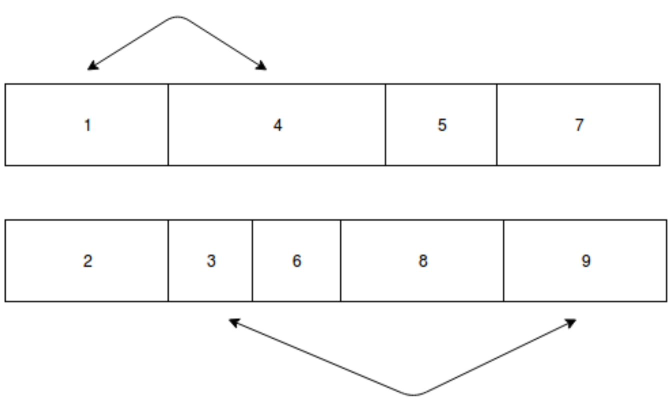

Figura 6. Troca entre duas salas da mesma alocação.

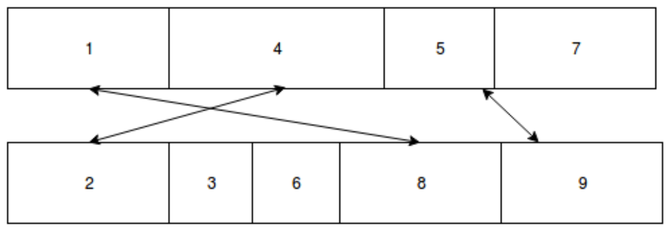

Figura 7. Troca entre duas posições de diferentes alocações.

6 Experimentos computacionais

Para a experimentação, primeiramente, foi gerada uma instância pequena, através de uma função geradora de instância, implementada, para validação. Em seguida, testes com instâncias da literatura, disponíveis em *http://www.miguelanjos.com/flplib*, foram realizados afim de comparar os resultados obtidos, pelo método implementado, com os resultados ótimos, conhecidos, de cada instância.

6.1 Validação

A instância de validação, gerada, possui três salas e um valor ótimo conhecido, referente à função objetivo, igual a 4,5.

A validação foi utilizada com a ideia de avaliar e solucionar problemas possíveis de ocorrer, durante a execução do programa, assim como, verificar a corretude do algoritmo em todas etapas da implementação. A estrutura da instância pode ser visualizada na figura 8, assim como a melhor solução obtida, que equivale ao valor ótimo de 4,5.

6.2 Experimentação

Instâncias, retiradas da literatura, foram utilizadas para análise de desempenho do algoritmo, principalmente, pelo fato de se conhecer a solução ótima para essas instâncias. O número de salas varia entre 5 e 15, para essas instâncias.

Encontrar a melhor solução depende diretamente das alocações fornecidas pelo construtivo, que, no caso, são geradas aleatóriamente trocando-se posições de salas. Sendo assim, devido ao fato de que, em apenas uma execução, as alocações criadas podem não favorecer à construção da melhor solução, foi adotado a execução do algoritmo em 50 iterações, ou seja,

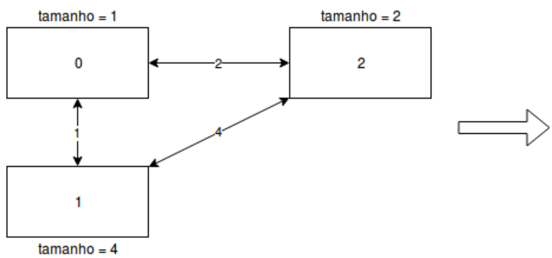

Figura 8. Instância de validação.

o algoritmo construtivo é executado 50 vezes, produzindo, a cada iteração, uma nova solução e realizando a geração das alocações. Os resultados podem ser vistos nas **tabela 1** e **tabela 2**.

Instâncias Heku				
Instância	Valor Ótimo	Valor Obtido	percentual	Tempo (s)
HeKu_5	450.0	450.0	0	2.120002
HeKu_6	720.0	720.0	0	3.417602
HeKu_7	1700.0	1700.0	0	5.511390
HeKu_8	2385.0	2385.0	0	12.239387
HeKu_12	8995.0	9865.0	9.67	5889.755081
HeKu_15	16640.0	18100.0	8.77	8895.12313

Tabela 1. Resultados para instâncias Heku.

6.3 Conclusão

Os experimentos realizados demonstram a capacidade do algoritmo para encontrar um valor, de solução, ótimo para instâncias com até nove salas. Em comparação com a literatura, os resultados obtidos indicam a falta de competitividade para instâncias de médio e grande porte, porém, o método é promissor para instâncias de pequeno porte, principalmente em questões de viabilidade de tempo.

Instâncias Si				
Instância	Valor Ótimo	Valor Obtido	percentual	Tempo (s)
Si_8_set1	408.0	416.0	1.96	12.268386
Si_8_set2	1135.5	1137.5	0.17	17.163927
Si_9_set1	1181.5	1186.5	0.42	81.745487
Si_9_set2	2295.5	2295.5	0	423.226576
Si_10_set1	1374.5	1434.5	4.36	1430.560340
Si_11_set1	3439.5	3632.5	5.61	4779.172566

Tabela 2. Resultados para instâncias Si.

Os experimentos realizados demonstram a capacidade do algoritmo para encontrar um valor de solução ótimo, em **tempo viável**, para instâncias de pequeno porte.

Considerando o método aleatório, implementado, para geração de alocações, a incorporação de métodos mais inteligentes, como busca local ou outros tipos de metaheurísticas, combinados com a utilização de um resolvedor matemático, criam a hipótese da otimização na busca pela solução ótima e melhoria no resultado obtido, principalmente para instâncias de médio e grande porte.

7 Referências

[Ahonen et al., 2014] Ahonen, H., de Alvarenga, A. G., and Amaral, A. (2014). Simulated annealing and tabu search approaches for the corridor allocation problem. *European Journal of Operational Research*, 232(1):221–233.

[Amaral, 2012] Amaral, A. R. (2012). The corridor allocation problem. *Computers & Operations Research*, 39(12):3325–3330.

[Chung and Tanchoco, 2010] Chung, J. and Tanchoco, J. (2010). The double row layout problem. *International Journal of Production Research*, 48(3):709–727.

[Ghosh and Kothari, 2012a] Ghosh, D. and Kothari, R. (2012a). Population heuristics for the corridor allocation problem.

[Ghosh and Kothari, 2012b] Ghosh, D. and Kothari, R. (2012b). Population heuristics for the corridor allocation problem.

[Kothari and Ghosh, 2012] Kothari, R. and Ghosh, D. (2012). The single row facility layout problem: state of the art. *Opsearch*, 49(4):442–462.

[Kothari and Ghosh, 2013] Kothari, R. and Ghosh, D. (2013). Insertion based lin–kernighan heuristic for single row facility layout. *Computers & Operations Research*, 40(1):129–136.

[Kothari and Ghosh, 2014] Kothari, R. and Ghosh, D. (2014). A scatter search algorithm for the single row facility layout problem. *Journal of Heuristics*, 20(2):125–142.

[Santana and dos Santos,] Santana, C. A. and dos Santos, A. G. Uma heurística baseada na busca local iterada para o problema de alocação de corredor.

[Simmons, 1969] Simmons, D. M. (1969). One-dimensional space allocation: an ordering algorithm. *Operations Research*, 17(5):812–826.