UMA ABORDAGEM BASEADA EM BUSCA TABU PARA O PROBLEMA DE ALOCAÇÃO DE CORREDOR

Hannu Ahonen
Departamento de Informática, Universidade Federal do Espírito Santo (UFES), Vitória, ES, 29060-900, Brazil
hannu@inf.ufes.br

Arlindo Gomes de Alvarenga
Departamento de Informática, Universidade Federal do Espírito Santo (UFES), Vitória, ES, 29060-900, Brazil
agomes@inf.ufes.br

André Renato Sales Amaral
Departamento de Informática, Universidade Federal do Espírito Santo (UFES), Vitória, ES, 29060-900, Brazil
amaral@inf.ufes.br

RESUMO

O Problema de Alocação de Corredor estuda como arranjar n facilidades ao longo de um corredor. Os arranjos em ambos os lados do corredor devem começar a partir de um ponto comum na extremidade esquerda do corredor, sem espaço permitido entre duas facilidades adjacentes. Aplicações do problema ocorrem na disposição de salas em edifícios de escritórios, hospitais, centros comerciais e escolas. Um algoritmo de busca tabu é apresentado para minimizar o custo total de fluxo entre facilidades. O algoritmo apresenta um bom desempenho em várias instâncias disponíveis na literatura.

PALAVRAS-CHAVE: Layout de facilidades, busca tabu, otimização combinatória.

ABSTRACT

The Corridor Allocation Problem studies how to arrange n facilities along a corridor. The arrangements on either side of the corridor should start from a common point on the left end of the corridor, with no space allowed between two adjacent facilities. Applications of the problem occur in the arrangement of rooms in office buildings, hospitals, shopping centers or schools. A tabu search algorithm is presented to minimize the total flow cost among facilities. The algorithm presents a good performance on several instances available in the literature.

KEYWORDS: Facilities layout, tabu search, combinatorial optimization.

1 Introdução

O Problema de Alocação de Corredor (Amaral, 2012), doravante abreviado como CAP (*Corridor Allocation Problem*), procura arranjar n facilidades, sem sobreposição, ao longo de um corredor sendo que os arranjos em ambos os lados do corredor devem começar a partir de um ponto comum na extremidade esquerda do corredor, e não é permitido espaço entre duas facilidades adjacentes. São dados o número de facilidades n, o comprimento ℓ_i de cada facilidade i e o valor não negativo do fluxo c_{ij} entre as facilidades i e j. A largura do corredor é considerada negligenciável, pelo que a distância entre duas facilidades i e j em uma solução do CAP é a distância horizontal entre os seus centros. O *custo de fluxo total* é a soma ponderada das distâncias entre cada par de facilidades utilizando os parâmetros c_{ij} como pesos. O objetivo do CAP é encontrar um layout que minimiza o custo do fluxo total sobre todos os layouts possíveis. A Figura 1 exemplifica uma disposição do CAP com sete facilidades.

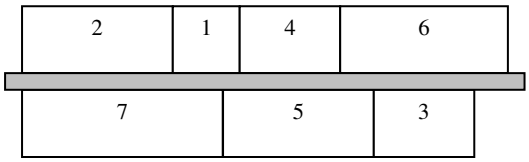

Figura 1. Uma disposição do CAP com sete facilidades.

O CAP tem relações estreitas com outros problemas na literatura. Por exemplo, no problema da disposição das facilidades em fileira única (Single Row Facility Layout Problem, SRFLP) as facilidades são todas colocadas no mesmo lado do corredor (por exemplo, Simmons, 1969; de Alvarenga et al, 2000; Anjos et al, 2005; Anjos e Vannelli, 2008; Anjos e Yen, 2009; Amaral, 2006; Amaral, 2008; Amaral, 2009a; Amaral, 2009b; Amaral e Letchford, 2008; Datta et al., 2011; Hungerländer e Rendl, 2011). O CAP também tem relações com o problema da disposição das facilidades em fileira dupla (Double Row Layout Problem, DRLP) (e.g. Heragu e Kusiak, 1988; Chung e Tanchoco, 2010; Amaral, 2013), que é originalmente motivado pela disposição de máquinas em um sistema de manufatura. No DRLP, as facilidades são colocadas em ambos os lados do corredor, mas os arranjos superior e inferior não têm que começar a partir de um ponto comum e, além disso, algum espaço pode ser permitido entre facilidades adjacentes.

O CAP tem muitas aplicações práticas como, por exemplo, a disposição de salas em edifícios administrativos, hospitais ou supermercados. Amaral (2012) apresentou uma formulação em programação inteira mista para o CAP. No entanto, o CAP é NP- árduo, o que justifica o estudo de abordagens heurísticas que possam lidar com grandes instâncias do problema. Nesse sentido, Amaral (2012) propôs um algoritmo heurístico para o CAP e abordou instâncias com tamanho n = 30.Um recente relatório de pesquisa de Hungerländer and Anjos (2012) estudou o CAP e desenvolveu uma heurística para obter soluções factíveis a partir de relaxações de programação semi-definida.

No presente trabalho, um algoritmo de busca tabu é implementado para o CAP. O algoritmo é avaliado em instâncias da literatura com $n \leq 30$. Posteriormente, o algoritmo é testado para instâncias ainda maiores com tamanho $42 \leq n \leq 70$.

2 Implementação da busca tabu para o CAP

A estrutura de vizinhança utilizada pela busca tabu é apresentada na subseção 2.1. Em seguida, o algoritmo de busca tabu é descrito na subseção 2.2. Os leitores não familiarizados com busca tabu são referidos para Glover e Laguna (1997).

2.1 A estrutura de vizinhança

O algoritmo de busca tabu explora uma estrutura de vizinhança definida por dois conjuntos de movimentos: C de movimentos de coluna e E de movimentos de elemento de linha.

Os movimentos são aplicados a uma matriz 2×n que representa um arranjo de n facilidades, em que um valor diferente de zero de um elemento de linha indica a posição de uma facilidade do lado correspondente do corredor. O valor zero significa que a facilidade não está presente naquele lado do corredor. Um arranjo é uma matriz A = (aij) com elementos

$$a_{ij} = \begin{cases} k, \text{ se a facilidade } j \text{ está na posição } k \text{ no lado } i; \\ 0, \text{ se a facilidade } j \text{ não está no lado } i. \end{cases}$$

Por exemplo, se $n = 5$, a matriz

$$A = \begin{pmatrix} 0 & 2 & 0 & 1 & 0 \\ 3 & 0 & 1 & 0 & 2 \end{pmatrix}$$

representa um arranjo, no qual as facilidades 2 e 4 (veja os elementos da primeira linha nas colunas 2 e 4) estão em um mesmo lado do corredor, na ordem (4, 2), enquanto que a ordem das facilidades no outro lado do corredor é (3, 5, 1).

Um movimento de coluna é definido como uma troca entre duas colunas na matriz. Este tipo de movimento pode resultar em mudanças na ordem das facilidades em apenas um lado do corredor ou em uma facilidade indo de um lado do corredor para o outro. Por exemplo, a troca das colunas 2 e 4 em A, coloca a facilidade 2 como a primeira e a facilidade 4 como a segunda facilidade no lado superior do corredor; e a troca das colunas 1 e 2 modifica os arranjos em ambos os lados do corredor colocando a facilidade 1 como a segunda facilidade no lado superior do corredor e a facilidade 2 como a terceira facilidade no lado inferior do corredor.

Um movimento de elemento de linha troca elementos de uma mesma coluna e associa uma determinada posição para o elemento não nulo. Por exemplo, em A, uma troca dos elementos da terceira coluna, sendo a nova posição dada para o elemento não nulo igual a 3, faz com que a facilidade 3 se torne a terceira facilidade no lado superior do corredor. Se a nova posição dada para o elemento não nulo estiver antes de outras posições no seu novo lado do corredor, pode ser necessária uma atualização das posições subsequentes. Por exemplo, uma troca dos elementos da terceira coluna, sendo a nova posição dada para o elemento não nulo igual a 1: para que a facilidade 3 passe a ser a primeira no lado superior do corredor, temos que as posições das facilidades 4 e 2 devem ser atualizadas para ser 2 e 3, respectivamente. Do mesmo modo as posições devem ser ajustadas no lado a partir do qual foi removida uma facilidade. No nosso exemplo, isso significa atribuir a facilidade 5 para a posição 1 e a facilidade 1 para a posição 2.

2.2 Algoritmo de busca tabu

O algoritmo de busca tabu implementado neste trabalho é constituído pelas seguintes etapas:

• Passo 1. Selecione uma solução inicial s_0 . Definir solução atual $s := s_0$;
Calcule o custo $c(s)$.

• Passo 2. Aplicar uma busca local a s com resultado s';
Se $c(s') < c(s)$ faça $s := s'$.

• Passo 3. Definir a melhor solução encontrada até agora $s^b := s$.

• Passo 4. Inicializar os valores do vetor F (freqüência de trocas) igual a 0;
Inicializar os valores do vetor R (recência de trocas) igual a $-\infty$;
Fazer $stop := false$, $i_{iter} := 1$.

• Passo 5. Repita

- Passo 5-1. Gerar vizinhança $N(s)$ da solução atual s aplicando a s os movimentos de coluna C e os movimentos de elemento de linha E.

- Passo 5-2. Selecione solução s^{NB} com o menor custo modificado c^F em $N(s)$.
Denote por m^{NB} o movimento que gerou essa solução. Determinar se o movimento m^{NB} é tabu ou não.

- Passo 5-3. Se ($c(s^{NB}) < c(s^b)$):
Faça $s^b := s^{NB}$ e $s := s^{NB}$;
Aplicar busca local a s^b com resultado s^{bl};
Se $c(s^{bl}) < c(s^b)$, $s := s^{bl}$ e $s^b := s^{bl}$.

- Passo 5-4. Senão, se ($c(s^{NB}) \geq c(s^b)$):
Se o movimento m^{NB} é tabu, selecione em $N(s)$ uma solução $s^{nb'}$ gerada por um movimento não-tabu $m^{nb'}$ e com o menor custo modificado c^F.
Definir $s := s^{nb'}$.

- Passo 5-5. Incrementar de uma unidade o valor do elemento em F que corresponde ao movimento selecionado m^{NB} ou $m^{nb'}$.
Defina o valor do componente correspondente de R igual a i_{iter} .

- Passo 5-6. Incrementar i_{iter} de uma unidade.

- Passo 5-7. Se o critério de parada se aplica, $stop := true$.

- Passo 5-8. Senão, se o critério de diversificação se aplica, chamar procedimento diversificação $DIV(s, F)$ com resultado s^d. Faça $s := s^d$.

- Passo 5-9. até ($stop$).

• Passo 6. Retornar s^b

Observações:
Passo 1: Uma solução correspondente a uma matriz-arranjo inicializada aleatoriamente é selecionada.

Passo 2: O algoritmo de busca local consiste no algoritmo mais básico, isto é, os movimentos são aplicados à solução atual e o melhor movimento é selecionado. O procedimento é repetido até que nenhuma melhoria no custo de uma solução seja observada.

Passo 4: O vetor de freqüências F armazena as informações sobre quantas vezes um movimento foi selecionado no Passo 5-2 ou Passo 5-4. O valor atual do contador de iteração está registrado no vetor de recência R, guardando assim a iteração da ultima vez que esse movimento foi utilizado.

Passo 5-2: O custo modificado $c^F(s)$ de s é aqui definido como $c^F(s) = (1 + f / (1 + f)) c(s)$, onde f é a freqüência do movimento que gerou a solução s.

Passo 5-3: Este passo significa a aplicação de um critério de aspiração no caso de o movimento m^{NB} ser tabu.

Passo 5-4: Um movimento é tabu, se tiver sido aplicado recentemente, ou seja, se a diferença entre o contador de iteração em curso e o elemento em R correspondente ao movimento for menor que o comprimento da duração tabu. A definição desse comprimento é explicada abaixo.

Passo 5-7: O critério de parada é expresso como um número máximo de iterações (*nMaxIters*) e como um número máximo de iterações sem melhoramentos (*nMaxWithoutImprovement*).

Passo 5-8: O processo de diversificação *DIV (s, F)* é chamado quando não há melhoria durante um determinado período de iterações (*nMaxBeforeDiv*). Para isso, alguns movimentos menos freqüentes – identificados pelo vetor F – são aplicados à solução atual. O número desses movimentos é determinado pelo parâmetro *nDivSteps*.

O critério de recência na determinação do status tabu foi implementado como uma duração tabu de comprimento variável, isto é, o comprimento começa com um valor máximo *MaxTabuDuration* e é reduzido por um coeficiente *TabuFactor* até um valor mínimo de *MinTabuDuration* ser alcançado. Depois disso, a sequência repete-se começando de novo com a duração máxima.

3 Experimentos computacionais

O algoritmo de busca tabu foi implementado em Java e testado em JVM 1.6. O algoritmo foi executado em um processador Intel (R) Core (TM) 2 Quad CPU (P9550) com 2,83 GHz e 4 GB de RAM no sistema operacional Linux.

O desempenho do algoritmo foi avaliado em várias instâncias da literatura (note-se que as instâncias usadas aqui para testar o CAP foram, em sua maioria, originalmente usadas na literatura para testar o SRFLP e estão disponíveis no conjunto de instâncias para o SRFLP no endereço: http://www.gerad.ca/files/Sites/Anjos/flplib.html). Para cada instância o algoritmo de busca tabu é executado 30 vezes. Ao longo das experiências, os seguintes valores de parâmetros foram utilizados:

nMaxIters = 100.000,
nMaxWithoutImprovement = 50.000;
nMaxBeforeDiv = 12500,
nDivSteps = 50,
MinTabuDuration = *n* / 8,
MaxTabuDuration = 2*MinTabuDuration,
TabuFactor = 0,995.

3.1 Instâncias com $n \leq 15$

Inicialmente, o algoritmo de busca tabu foi testado em um número de instâncias da literatura tendo $n \leq 15$:
• As maiores instâncias de Simmons (1969) com $9 \leq n \leq 11$;
• duas instâncias com $n = 12$ e duas com $n = 13$ de Amaral (2012);
• uma instância de Amaral (2006), com $n = 15$;
Soluções ótimas são conhecidas apenas para as instâncias com $n \leq 13$ (Amaral, 2012).

A Tabela 1 mostra para cada instância: o melhor valor (Min.), a média (Média), o pior valor (Max.) e o desvio padrão (SD) em 30 execuções do algoritmo para:
i) o custo da solução
ii) o número de avaliações (isto é, quantas vezes o valor da função de custo foi calculado) e
iii) tempos de CPU (em segundos).

Observou-se que a busca tabu facilmente atingiu a solução exata conhecida para $n \leq 13$ e a melhor-conhecida (Amaral, 2012) para $n = 15$ em cada uma das 30 execuções.

	Custos				*# avaliações (x1000)*				*Tempo*			
Name	Min.	Max.	Avg.	S. D	Min.	Max.	Avg.	S. D	Min.	Max.	Avg.	S. D
S9	**1181.5**	**1181.5**	**1181.5**	0.0	2250	2251	2250	0	1.09	1.49	1.14	0.08
S9H	**2294.5**	**2294.5**	**2294.5**	0.0	2250	2256	2251	1	1.09	1.51	1.15	0.09
S10	**1374.5**	**1374.5**	**1374.5**	0.0	2750	2888	2811	52	1.46	2.03	1.58	0.14
S11	**3439.5**	**3439.5**	**3439.5**	0.0	3300	3378	3337	28	1.88	2.29	2.02	0.09
Am12a	**1529.0**	**1529.0**	**1529.0**	0.0	3901	4149	3987	88	2.32	2.71	2.43	0.09
Am12b	**1609.5**	**1609.5**	**1609.5**	0.0	3900	4016	3931	34	2.34	2.77	2.43	0.08

Am13a	**2467.5**	**2467.5**	**2467.5**	0.0	4550	5091	4715	148	2.84	3.44	2.99	0.13
Am13b	**2870.0**	**2870.0**	**2870.0**	0.0	4551	5145	4770	162	2.83	3.42	3.01	0.13
Am15	3195.0*	3195.0*	3195.0*	0.0	6020	6537	6234	173	4.48	5.17	4.70	0.16

valores em negrito indicam que a busca tabu atingiu custos sabidos serem ótimos.
(*) Busca tabu atingiu a melhor solução conhecida.
Tabela 1: instâncias com $n \leq 15$.

3.2 Instâncias com n = 30

Em seguida, o desempenho do algoritmo de busca tabu foi avaliado utilizando-se cinco instâncias de Anjos e Vannelli (2008) com $n = 30$. As melhores soluções conhecidas para estas instâncias são dadas por Amaral (2012).

Os resultados destes experimentos são mostrados na Tabela 2. Foi verificado que para cada instância do problema, o melhor custo de solução alcançado pelo algoritmo de busca tabu corresponde ao melhor custo de solução da heurística de Amaral (2012).

	Custos				*# avaliações (x1000)*				*Tempo*			
Name	Min.	Max.	Avg.	SD	Min.	Max.	Avg.	SD	Min.	Max.	Avg.	SD
N30_01	4115.0*	4115.0*	4115.0*	0.0	12381	13634	12789	367	26	28	27	1
N30_02	10779.5*	10783.5	10780.4	1.2	12536	24750	18066	4275	26	52	38	9
N30_03	22702.0*	22709.0	22703.7	2.2	12434	24750	16804	4126	26	52	35	9
N30_04	28401.5*	28411.5	28403.4	3.3	12376	24750	17735	3981	25	51	36	8
N30_05	57400.0*	57434.0	57415.2	11.0	12422	24750	18962	4584	25	51	39	9

(*) Busca tabu atingiu a melhor solução conhecida.
Tabela 2: instâncias com $n = 30$.

Note que para as instâncias com n=30, os tempos médios requeridos pelo algoritmo de busca tabu são bem pequenos, variando de 27 a 39 segundos.

3.3 Instâncias com $42 \leq n \leq 70$

Nesta subseção, vamos experimentar com instâncias de tamanhos bem maiores, com $42 \leq n \leq 70$. Para esses testes, vamos utilizar um conjunto de instâncias "sko" (Anjos e Yen, 2009) com tamanhos n= 42, 49, 56 e 64, e com duas instâncias "AKV" (Anjos et al., 2005) de tamanhos n= 60 e 70.

3.3.1 Avaliação de tempos computacionais

Aqui queremos observar o comportamento dos tempos computacionais requeridos pelo algoritmo de busca tabu para as instâncias com $42 \leq n \leq 70$. Dos resultados mostrados na Tabela 3, vemos que para as instâncias com n=42, os tempos médios

variam de 111 a 134 segundos. Para as instâncias com n=49, os tempos médios variam de 136 a 151 segundos. Para n=56, os tempos médios variam de 209 a 240 segundos. Finalmente, para as instâncias com tamanho $60 \leq n \leq 70$ os tempos médios variam de 281 a 459 segundos. Esses resultados indicam que, mesmo para grandes instâncias, o algoritmo executa em tempo razoável como requerido em aplicações práticas.

3.3.2 Comparação com a heurística de Hungerländer e Anjos

Algumas das instâncias que utlizamos, particularmente as de maior tamanho, foram também utilizadas por Hungerländer e Anjos (2012) para testar a sua heurística, o que permite fazer uma comparação com os custos das soluções obtidos pela busca tabu. A Tabela 4 mostra tal comparação. Vemos que a busca tabu obtém menores custos de soluções em todos os casos, especialmente para as instâncias de maior tamanho.

4 Conclusões

Neste trabalho, consideramos o Problema de Alocação de Corredor (CAP), o qual possui muitas aplicações no mundo real. Uma estrutura adequada de vizinhança para o CAP foi apresentada, a qual foi explorada por um algoritmo de busca tabu descrito neste trabalho. O algoritmo foi avaliado em várias instâncias da literatura com $n \leq 30$, atingindo a solução exata (quando disponível) ou a melhor-conhecida para essas instâncias.

Além disso, o algoritmo foi testado para instâncias ainda maiores, com tamanho $42 \leq n \leq 70$ onde foi observado que, mesmo para essas instâncias de tamanhos maiores, o algoritmo executa em tempo razoável, como é desejável em aplicações práticas. A comparação dos custos de solução alcançados pela busca tabu com aqueles obtidos pela heurística de Hungerländer e Anjos (2012) mostrou que a busca tabu obtém menores custos de soluções para todas as instâncias testadas em comum, especialmente para aquelas de maior tamanho.

Desta forma, acreditamos que o algoritmo apresentado se traduz em boa opção para se abordar o problema de alocação de corredor.

	Custos				*# avaliações (x1000)*				*Tempo*			
Name	Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D
sko42_01	12731.0	12739.0	12733.6	3.5	23769	47250	29527	7756	89.59	178.27	111.49	29.33
sko42_02	108020.5	108110.5	108065.9	23.3	23629	46511	29732	6072	88.14	172.90	111.26	22.65
sko42_03	86657.5	86715.5	86681.2	15.9	24033	47250	32368	8185	90.95	179.11	122.54	30.86
sko42_04	68711.0	68801.0	68747.4	19.9	24278	47250	35935	8684	91.45	177.64	134.90	32.34
sko42_05	124018.5	124142.5	124081.6	30.9	23722	47250	32912	8088	88.86	177.21	123.43	30.15
sko49_01	20470.0	20488.0	20479.9	4.6	22192	44100	30211	7116	110.51	222.05	151.20	35.45
sko49_02	208157.0	208329.0	208260.9	53.0	22893	44100	30356	7155	113.94	219.37	151.20	35.21
sko49_03	162249.0	162423.0	162337.1	40.1	22187	44100	28928	6433	106.96	213.48	139.72	30.99
sko49_04	118302.5	118391.5	118340.9	24.9	22355	44100	28012	8351	108.60	218.86	136.40	40.60
sko49_05	332972.0	333170.0	333087.3	55.1	22435	44100	29251	6792	109.09	214.37	142.48	32.80
sko56_01	31972.0	31984.0	31978.5	3.3	29086	56778	38175	8250	179.12	361.90	235.88	52.29
sko56_02	248247.0	248543.0	248391.4	73.2	28474	56933	37065	10245	175.60	353.15	230.74	61.53
sko56_03	85209.0	85498.0	85293.0	79.2	28651	56933	39220	9685	174.40	346.19	240.03	58.84
sko56_04	156701.0	156918.0	156835.6	48.2	28541	56933	33960	7294	175.26	372.43	209.38	47.02
sko56_05	296315.5	296565.5	296424.8	55.4	28476	56830	36568	8476	175.23	348.55	225.92	52.15
AKV_6	159715	159973.	159838.	67.0	3271	6500	4155	8188	220.5	437.8	281.2	54.80

0_5	.0	0	8		1	0	7		8	2	2	
sko64_05	250955.5	251160.5	251048.9	58.4	36835	73600	46752	9168	275.82	552.30	351.54	68.36
AKV_70_5	2110357.5	2113448.5	2111623.3	752.4	43755	79843	52106	11421	384.72	701.62	459.60	99.74

Tabela 3: Performance da busca tabu para as instâncias sko/AKV

Name	Size	Busca Tabu	Hungerländer and Anjos
sko42_01	42	12731.0	-
sko42_02		108020.5	-
sko42_03		86657.5	-
sko42_04		68711.0	-
sko42_05		124018.5	127639.5
sko49_01	49	20470.0	-
sko49_02		208157.0	-
sko49_03		162249.0	-
sko49_04		118302.5	-
sko49_05		332972.0	349137.0
sko56_01	56	31972.0	-
sko56_02		248247.0	-
sko56_03		85209.0	-
sko56_04		156701.0	-
sko56_05		296315.5	306133.5
AKV_60_05	60	159715.0	171280.0
sko64_05	64	250955.5	261257.5
AKV_70_05	70	2110357.5	2196942.5

Tabela 4: Comparação dos custos de solução alcançados pela busca tabu com aqueles obtidos pela heurística de Hungerländer e Anjos.

Referências

Amaral, A. R. S., 2006. On the exact solution of a facility layout problem. European Journal of Operational Research 173 (2), 508 – 518.
Amaral, A. R. S., 2008. An exact approach to the one-dimensional facility layout problem. Operations Research 56 (4), 1026–1033.
Amaral, A. R. S., 2009a. A mixed 0-1 linear programming formulation for the exact solution of the minimum linear arrangement problem. Optimization Letters 3, 513–520.
Amaral, A. R. S., 2009b. A new lower bound for the single row facility layout problem. Discrete Applied Mathematics 157 (1), 183 – 190.
Amaral, A. R. S., 2012. The corridor allocation problem. Computers & Operations Research 39 (12), 3325 – 3330.
Amaral, A. R. S., 2013. Optimal solutions for the double row layout problem. Optimization Letters 7 (2), 407-413.
Amaral, A. R. S., Letchford, A. N., 2008. A polyhedral approach to the singlerow facility layout problem. Technical report, Department of Management Science, Lancaster University, UK.
Anjos, M. F., Kennings, A., Vannelli, A., 2005. A semidefinite optimization approach for the single-row layout problem with unequal dimensions. Discrete Optimization 2 (2), 113 – 122.
Anjos, M. F., Vannelli, A., 2008. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. INFORMS Journal on Computing 20 (4), 611–617.
Anjos, M. F., Yen, G., 2009. Provably near-optimal solutions for very large single-row facility layout problems. Optimization Methods and Software 24 (4), 805 – 817.
Chung, J., Tanchoco, J. M. A., 2010. The double row layout problem. International Journal of Production Research 48 (3), 709 – 727.
Datta, D., Amaral, A. R. S., Figueira, J. R., 2011. Single row facility layout problem using a permutation-based genetic algorithm. European Journal of Operational Research 213 (2), 388 – 394.
de Alvarenga, A. G., Negreiros-Gomes, F. J., Mestria, M., 2000. Metaheuristic methods for a class of the facility layout problem. Journal of Intelligent Manufacturing 11, 421–430.
Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA.
Heragu, S. S., Kusiak, A., 1988. Machine layout problem in flexible manufacturing systems. Operations Research 36 (2), 258–268.
Hungerländer, P., Anjos, M. F., 2012. A semidefinite optimization approach to space-free multi-row facility layout. Technical report, Les Cahiers du GERAD,Montréal (Québec), Canada.
Hungerländer, P., Rendl, F., 2011. A computational study and survey of methods for the single-row facility layout problem. Technical report, Operations Research Group, Mathematics, University of Klagenfurt, Austria.
Simmons, D. M., 1969. One-dimensional space allocation: An ordering algorithm. Operations Research 17 (5), 812–826.