v.4, n.1, p. 88-108, abr. 1997

OTIMIZAÇÃO DE LEIAUTE USANDO BUSCA TABU

João Carlos Furtado
LAC/INPE - Instituto Nacional de Pesquisas Espaciais
Av. dos Astronautas 1758 - São José dos Campos - SP

Luiz Antônio Nogueira Lorena
LAC/INPE - Instituto Nacional de Pesquisas Espaciais
Av. dos Astronautas 1758 - São José dos Campos - SP

Resumo

A busca por leiautes de facilidades de alta qualidade pode ser vista como um problema de otimização combinatória que surge numa grande variedade de contextos de planejamento espacial. Examina-se aqui o problema de otimização de leiaute de facilidades numa área que pode incluir espaços ocupados. O espaço do leiaute e as áreas das facilidades são retangulares e limitados por razões de aspecto. Considerando a complexidade do problema, propõe-se uma heurística de busca tabu para sua solução. Primeiro um leiaute inicial é produzido, usando um procedimento de aglomeração ou aleatoriamente. Considerando n facilidades, a representação é dada por meio de uma árvore binária com um espaço de soluções n!. A busca tabu é então aplicada, usando os leiautes iniciais (aglomeração e aleatoriamente). Resultados com alta qualidade são obtidos para problemas da literatura.

Palavras-chave: busca tabu, leiaute de facilidades, otimização.

1. Introdução

O planejamento de facilidades é um assunto complexo, amplo, e é tema de diferentes disciplinas especializadas. Por exemplo: engenharia civil, elétrica, industrial, e mecânica estão todas envolvidas em planejamento de facilidades.

Num sistema de manufatura, o projetista, que é responsável por projetar o leiaute, deve considerar o número de facilidades, o volume de tráfego entre as facilidades, a

área necessária e restrições geométricas das facilidades individuais.

Para ajudar o projetista no seu trabalho, diversas técnicas, auxiliadas por computador, foram desenvolvidas e são úteis na determinação de leiautes. Leiautes podem ser gerados ou melhorando um leiaute existente ou construindo um leiaute a partir de dados básicos. Quando o algoritmo gera um leiaute mediante o melhoramento de um leiaute existente, o algoritmo é chamado de algoritmo de melhoramento. São exemplos de algoritmos de melhoramento: CRAFT (ARMOUR & BUFFA, 1963) e COFAD (TOMPKINS & REED, 1976) Quando o algoritmo constrói um leiaute em uma área livre, o algoritmo é chamado de algoritmo de construção. São exemplos de algoritmos de construção: PLANET (DEISENROTH & APPLE, 1972), CORELAP (LEE & MOORE, 1967) e ALDEP (SEEHOF & EVANS, 1967). Estas técnicas adotam a programação quadrática para a formulação do problema de leiaute. Foi demonstrado que este procedimento pertence à classe de problemas NP-completo (SAHNI & GONZALEZ, 1976).

Um dos grandes problemas dos algoritmos citados e de outros do gênero (BLAND & DAWSON, 1991; CO & REISMAN, 1989; COHOON *et al.*, 1991; DIAZ, 1992; GLOVER & McMILLAN, 1985; HERAGU & ALFA, 1992; KIM *et al.*, 1991; LANCE & WILLIAMS, 1966; O'BRIEN & BARR, 1980; SKORIN-KAPOV, 1990 e YEAP & SARRAFZADEH, 1993) é a incapacidade de tratar restrições geométricas, tais como uma construção que não seja um retângulo, ou a existência de regiões no interior da construção que não possam ser usadas pelas facilidades, pois já estão ocupadas, como por exemplo, a existência de elevadores, escadas, pilares de sustentação da construção e outras. Um dos poucos trabalhos que consideram restrições geométricas é o de TAM (1992). Nesta tese é proposta uma busca tabu para atacar esse problema, usando como estrutura de dados uma árvore binária. Após obter duas soluções iniciais: uma com procedimento de aglomeração e outra aleatoriamente, uma busca tabu é usada para melhorar essas soluções. Finalmente os dois resultados são comparados.

2. Busca Tabu

A busca tabu é um método heurístico, aplicado com grande êxito a um bom número de problemas de otimização combinatória. A forma básica moderna é de Glover (GLOVER, 1989a e 1989b), bem como seus melhoramentos (GLOVER, 1990).

A idéia básica consiste em evitar mínimos locais na busca, usando algumas estruturas para soluções (ou movimentos) proibidas (tabu).

Seja X o conjunto de possíveis soluções e $S(x)$ o conjunto de *movimentos* que levam de uma solução x para uma outra solução x'. É criado um subconjunto T de S, cujos elementos são chamados *movimentos tabu*. Os elementos de T são determinados por meio de uma função que utiliza informações históricas do processo de busca. Um elemento importante na busca tabu é a incorporação da aspiração $A(f(x))$, critério que permite que um movimento seja realizado mesmo sendo tabu. Uma das características da busca tabu é apresentar-se como uma forma de busca em vizinhança. Desta forma, cada solução $x \in X$, tem um conjunto de vizinhos, $N(x) \subset X$, chamado de *vizinhança* de x. Cada elemento $x' \in N(x)$ pode ser alcançado diretamente a partir de x, mediante um movimento. A qualidade de cada solução x gerada é avaliada mediante o cálculo da função objetivo $f(x)$.

Nós usamos o seguinte o algoritmo de busca tabu (ABT), adaptado de (GLOVER, 1989a):

Inicialização: a árvore binária (usando o algoritmo de aglomeração ou produzida aleatoriamente). Solução inicial para o leiaute $s \in S$, e *nbmax* (*nbmax* representa o número máximo de iterações);
cálculo: f(x);
Inicialização: uma lista tabu *T*; *nbiter :*= 0;
while *nbiter* < *nbmax* ***do***
Gerar rep soluções $x' \in N(x)$ com $[x \rightarrow x'] \notin T$ ou satisfazer a $A(f(x))$;
Verifique a melhor *x'* gerada por meio de *f(x');*
atualize a lista tabu *T*, com o movimento que gerou a melhor *x';*
atualize $A(f(x))$; { valor da melhor *f(x)* gerada até a presente iteração}
x := *x'*;
nbiter := *nbiter* + 1;
endwhile

A árvore binária inicial é produzida de duas formas: pelo processo de aglomeração (seção 3.3) ou aleatoriamente. A solução inicial $x \in X$ é um vetor de dimensão *n* (*n* representa o número de facilidades do problema). Cada componente é uma facilidade diferente e corresponde a uma folha na árvore binária. *X* é o espaço de todas as soluções possíveis, um espaço de dimensão *n!*. A vizinhança *N(x)* é dada por todas as possíveis permutações (movimentos) das diferentes facilidades ou dos diferentes nós internos. O tamanho de *N(x)* é dado pela soma $(n-1 + n-2 + ... + 1) = (n-1)n/2$ para a permutação das folhas da árvore binária. Para a permutação dos nós internos, *N(x)* depende da estrutura da árvore binária, uma vez que não é possível permutar um nó filho com o nó pai ou qualquer outro ancestral. Além disso, a estrutura da árvore binária esta variando ao longo do processo de busca, por meio da permutação de nós internos.

Caracterizamos como sendo um *movimento* duas situações: a) o *movimento* corresponde à permutação de folhas da árvore binária; b) o *movimento* corresponde à permutação de nós internos da árvore binária. Na implementação, as duas situações foram testadas e comparadas.

3. Formulação do Problema

3.1 A estrutura de Dados

A estrutura de dados usada é uma árvore binária, conforme figura 3.1.1. A árvore apresenta números positivos nas folhas que correspondem às facilidades e números negativos nos nós internos como resultado do processo de construção da árvore (ver seção 3.3). Cada facilidade requer uma quantidade de área, conforme coluna 2 da tabela 3.1.1. As colunas 3 e 4 serão discutidas posteriormente.

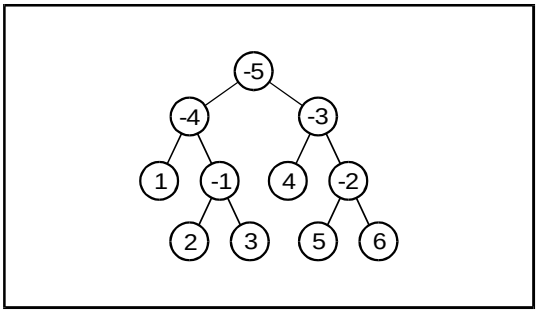

Figura 3.1.1 - Árvore binária

Tabela 3.1.1 - Exemplo de seis facilidades

Facilidade	Área	Razão de aspecto (limite inferior)	Razão de aspecto (limite superior)
1	100	0.5	1.2
2	80	0.4	1.1
3	50	0.3	3
4	60	0.1	5.5
5	120	0.9	1.9
6	40	0.4	1.2

Dispondo da árvore binária e das dimensões da região onde as facilidades serão introduzidas, inicia-se o processo de particionamento dessa região. Para isso, percorre-se recursivamente a árvore binária, a partir da raiz, verificando as áreas correspondentes na região de particionamento. Por exemplo, na figura 3.1.1, inicia-se na raiz (nó -5) e calcula-se o somatório das áreas das facilidades que estão no ramo esquerdo (as áreas dos nós 1, 2 e 3). O valor obtido é usado na região de particionamento, na qual se varre a região até obter a área correspondente, figura 3.1.2. Nesse momento um particionamento é realizado. Em seguida, continua-se a percorrer a árvore binária (seguindo para o nó -4) e novamente a área do ramo esquerdo é calculada (área do nó 1) e traduzida em particionamento na região de particionamento (é colocada a identificação da região particionada, ou seja, facilidade número 1). O processo continua até toda a árvore binária ser percorrida, o que corresponde em um leiaute na região de particionamento, no final do processo. A cada particionamento é necessário decidir se o particionamento será horizontal ou vertical. O critério é o seguinte: verificamos a distância entre os limites inferior e superior para os cortes vertical e horizontal no local considerado naquele instante a ser cortado. Se a distância vertical é menor do que a horizontal realizamos um corte vertical, caso contrário, o corte será horizontal. Usamos este critério por acreditar que desta maneira produziremos leiautes que violem menos as limitações das razões de aspecto máxima e mínima (seção 3.4).

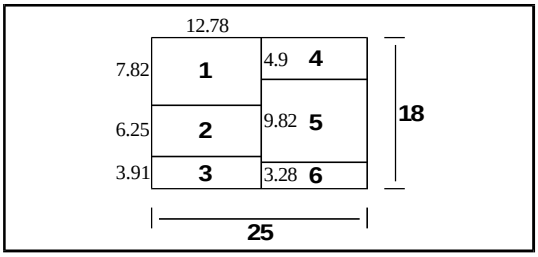

Figura 3.1.2 - Leiaute produzido pelo processo de particionamento

3.2 A Geração de Novos Leiautes

Após a obtenção da solução inicial (árvore binária), obtida pelo método de aglomeração ou aleatoriamente, procuramos melhorar o resultado por meio da busca tabu.

Conforme o conceito de *movimento* (seção 2), implementamos duas situações:

3.2.1 Movimento Mediante Troca de Folhas da Árvore Binária

No primeiro caso, para obter novos leiautes, caracterizamos o *movimento* como sendo a permutação de folhas da árvore binária. Assim, conforme a seqüência abaixo, vemos que para cada permutação de folhas geramos um novo leiaute.

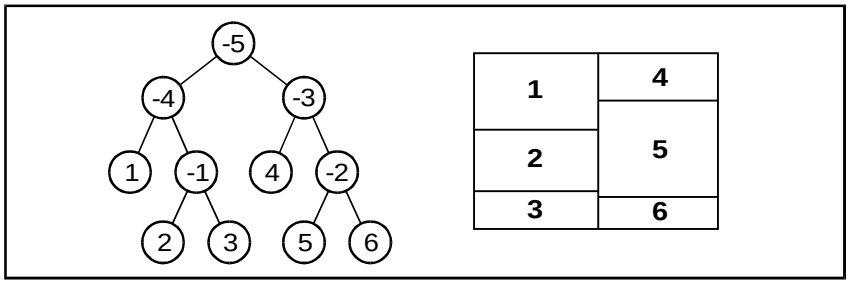

Figura 3.2.1.1 - Árvore binária e leiaute correspondente

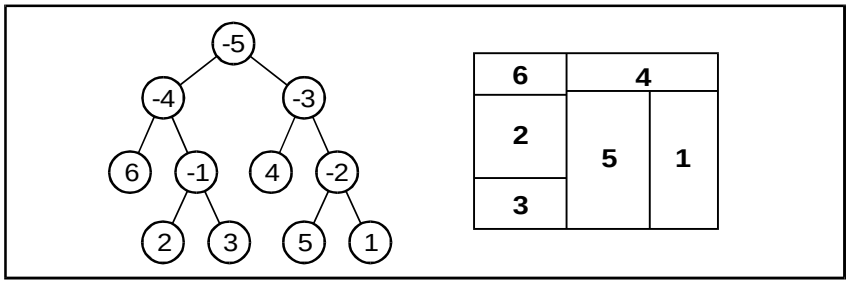

Figura 3.2.1.2 - Árvore binária e leiaute correspondente.

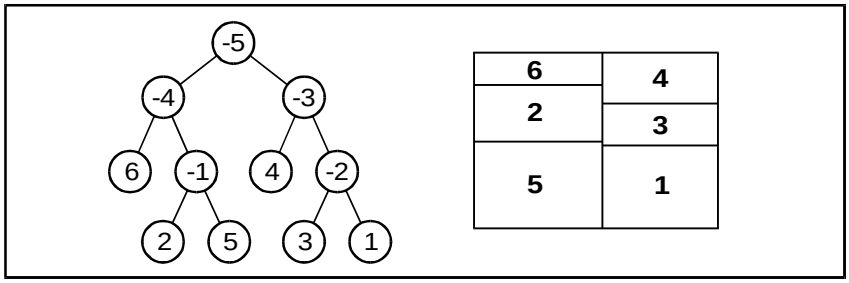

Figura 3.2.1.3 - Árvore binária e leiaute correspondente.

Assim, vemos que na primeira iteração permutamos as folhas *1* e *6*, e geramos conseqüentemente um novo leiaute. Na segunda iteração permutamos as folhas *3* e *5* e um novo leiaute foi produzido. Percebemos que a cada permutação de folhas da árvore binária, geramos um novo leiaute.

3.2.2 Movimento Mediante Troca de Nós Internos

De forma análoga à permutação de folhas na árvore binária, podemos permutar os nós internos na árvore e produzir novos leiautes a cada iteração, conforme a seqüência de figuras abaixo:

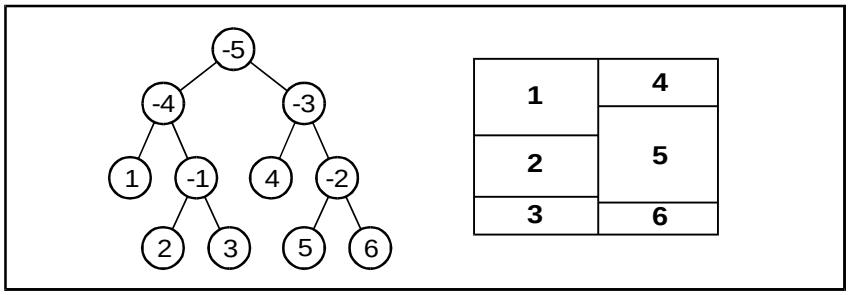

Figura 3.2.2.1 - Árvore binária e leiaute correspondente.

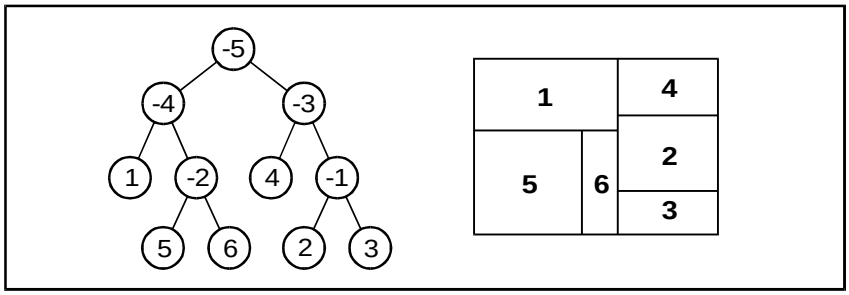

Figura 3.2.2.2 - Árvore binária e leiaute correspondente.

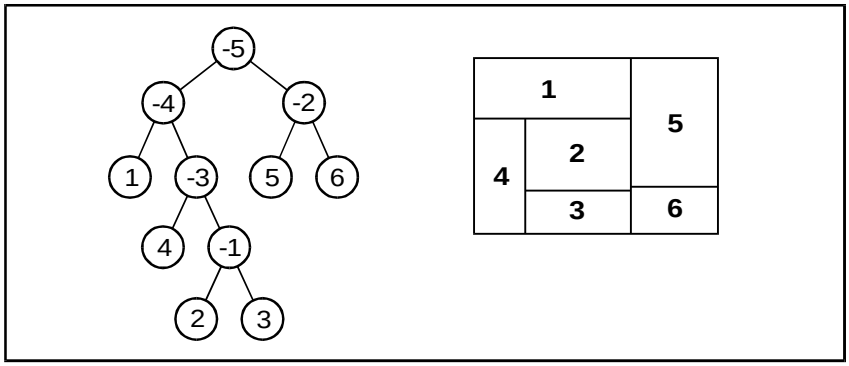

Figura 3.2.2.3 - Árvore binária e leiaute correspondente.

Na primeira iteração permutamos os nós internos *-1* e *-2*, conseqüentemente os filhos destes nós mudaram de posição na árvore binária, produzindo um novo leiaute. Na segunda iteração permutamos os nós internos *-2* e *-3* e produzimos um novo leiaute. É importante observar que quando permutamos nós internos alteramos a estrutura da árvore binária, produzindo modificações mais profundas no leiaute gerado.

3.3 O Processo de Construção da Árvore Binária

Foram construídas duas árvores binárias. Uma aleatoriamente e outra usando um método de aglomeração, no qual facilidades com alto tráfego entre si formam um aglomerado (ANDERBERG, 1973 e ZUPAN, 1982). O processo é descrito pelo seguinte algoritmo de aglomeração (AA):

Passo 1: { Inicialização da matriz de tráfego }
$\Theta_{nXn} = [\theta_{i,j}]$, $i, j = 1,...,n$ (simetricamente) e n aglomerados formados pelas facilidades individuais;

Passo 2: { Geração de um novo aglomerado combinando dois aglomerados }
Geração de um novo aglomerado com os aglomerados i e j para os quais $\theta_{i,j}$ é máxima;

Passo 3: { Atualização na matriz de tráfego }
A nova matriz de tráfego é formada:

- Excluindo as linhas (e colunas) correspondentes aos aglomerados i e j;
- Mantendo os tráfegos antigos θ_{pq} (em que os aglomerados p e q não pertencem aos aglomerados i ou j unidos no passo 2, e
- Incluindo um novo conjunto (linha e coluna) de tráfego $\theta_{(ij),h}$ entre os novos aglomerados (ij) e os aglomerados antigos remanescentes (representados aqui por h);

Passo 4: { Aglomerado final ? }
Parar com um aglomerado final ou voltar ao passo 2.

Existem diferentes possibilidades para o passo 3.3. Nós usamos a seguinte relação para calcular os novos tráfegos:

$$\theta_{(ij),h} = \frac{n_i}{n_{(ij)}}\theta_{ih} + \frac{n_j}{n_{(ij)}}\theta_{jh}$$

onde n_i é o aglomerado i, n_j é o aglomerado j, e $n_{[ij]}$ é o aglomerado formado pela união dos aglomerados i e j.

3.3.1 Exemplo

Usando o procedimento descrito acima, e considerando que a matriz de tráfego seja a da tabela 3.3.1.1, construímos um exemplo.

A cada iteração, as duas facilidades com o maior tráfego entre si são unidas num aglomerado, o que é traduzido num ramo da árvore binária. Vemos na figura 3.3.1.1 a seqüência de aglomerações até a formação da árvore final.

Tabela 3.3.1.1 - Matriz de tráfego

	1	***2***	***3***	***4***	***5***	***6***	***7***
1	---						
2	5	---					
3	2	3	---				
4	4	0	1	---			
5	1	2	0	5	---		
6	0	2	2	2	**10**	---	
7	0	2	5	2	0	5	---

Inicialmente, cada aglomerado corresponde a uma única facilidade, ou seja, temos 7 aglomerados, com uma facilidade para cada aglomerado.

Verificamos o maior tráfego na matriz, e percebemos que os aglomerados 6 e 5 apresentam o maior tráfego entre si. Portanto, os aglomerados 6 e 5 devem ser unidos num único aglomerado, conforme mostrado na iteração 1 da figura 3.3.1.1. Seguindo os passos do algoritmo de aglomeração, temos a seqüência de construção da árvore, conforme a figura 3.3.1.1.

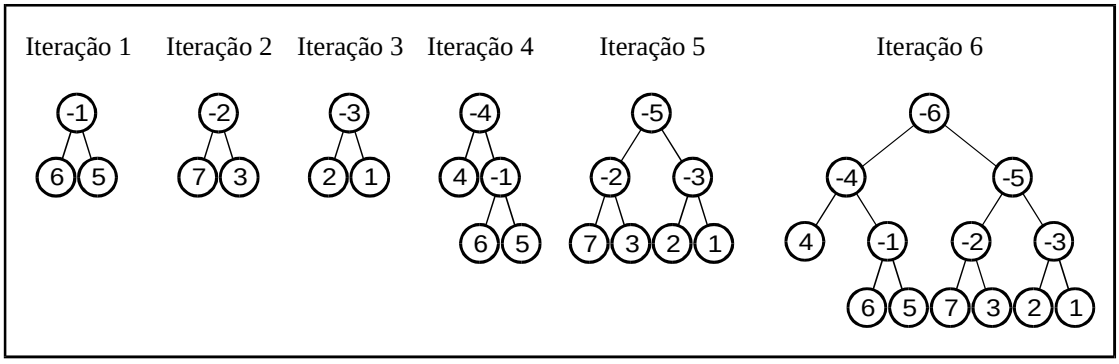

Figura 3.3.1.1 - Seqüência de construção da árvore binária.

Percebemos que ao final do processo encontramos uma única árvore binária, na qual as facilidades com alto tráfego entre si, permanecem mais próximas na estrutura de árvore, produzindo conseqüentemente um leiaute inicial de melhor qualidade.

3.3.2 A Árvore Binária Produzida Aleatoriamente

Vimos na seção anterior como é produzida a árvore binária pelo procedimento de aglomeração. Também produzimos aleatoriamente árvores binárias para a solução inicial. No entanto, na árvore binária aleatória os números de nós internos e de folhas correspondem exatamente aos das produzidas pelo procedimento de aglomeração, ou seja, a estrutura da árvore binária é preservada como base. São geradas aleatoriamente as facilidades e colocadas nas folhas da árvore binária.

3.4 Restrições Geométricas

A forma das facilidades é descrita por dois parâmetros: a razão de aspecto e a razão de área morta. A razão de aspecto a_i da facilidade *i* é definida como:

$$a_i = \frac{hp}{lp}$$

onde: hp = altura da partição alocada à facilidade *i;*

lp = largura da partição alocada à facilidade *i*.

Podemos impor restrições na forma da facilidade, restringindo sua razão de aspecto a $[a_{i(min)}, a_{i(max)}]$, onde $a_{i(min)}$ e $a_{i(max)}$ são os limites inferior e superior da razão de aspecto a_i.

Freqüentemente existem regiões no leiaute que não podem ser alocadas integralmente às facilidades, isto porque já estão parcialmente ocupadas, com construções, tais como elevadores, escadas, pilares, etc. Essas áreas ocupadas precisam ser subtraídas da área a ser alocada à facilidade. Considere os três espaços ocupados mostrados na figura 3.4.1, a área alocada à facilidade 1 é igual à área da partição retangular menos a porção de espaço já ocupada.

A existência de áreas ocupadas dentro da facilidade distorcerá a forma retangular da área usável; portanto, é necessário providenciar uma medida que reflita o grau de distorção da forma. A razão de área morta é introduzida com esse propósito. A razão de área morta da facilidade *i* denotada por σ_i é definida como:

área total do espaço ocupado na partição alocada à facilidade *i*

$$\sigma_i = \frac{\text{área da partição alocada à facilidade } i}{\text{área da partição alocada à facilidade } i}$$

Se o leiaute estiver livre de áreas ocupadas, a razão de área morta de todas as facilidades será zero. Para cada facilidade *i*, podemos restringir o valor de σ_i ao intervalo $[0, \sigma_{i(max)}]$, onde , $\sigma_{i(max)}$ é o limite superior de σ_i.

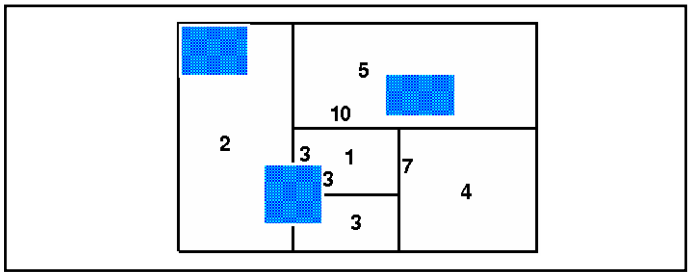

Figura 3.4.1 - Leiaute com espaços ocupados.

3.5 Função Objetivo

O objetivo é alocar espaço às facilidades de tal maneira que facilidades com grande volume de tráfego estejam próximas entre si, enquanto ainda satisfazem individualmente as restrições de área e forma. Dada uma árvore de particionamento $x \in X$, o problema de alocação de espaço pode ser formulado como:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n} \theta_{ij}\, d_{ij}\,, \qquad x \in X$$

sujeito a: $a_{i(min)} \le a_i \le a_{i(max)}$
$0 \le \sigma_i \le \sigma_{i(max)}$
$i = 1, 2, \ldots, n.$

onde

θ_{ij} = Volume de tráfego entre as facilidades *i* e *j*.

d_{ij} = Distância retangular entre os centros das partições alocadas às facilidades *i* e *j*.

a_i = Razão de aspecto da partição alocada a facilidade *i*.

$a_{i(min)}$ = Limite inferior de a_i.

$a_{i(max)}$ = Limite superior de a_i.

$\sigma_{i(max)}$ = Limite superior de σ_i.

Considerando que o centro da partição alocada à facilidade *i* seja (ρ_{i1}, ρ_{i2}) e o centro da partição alocada à facilidade *j* seja (ρ_{j1}, ρ_{j2}), a distância retangular d_{ij} é dada por:
$d_{ij} = |\rho_{i1} - \rho_{j1}| + |\rho_{i2} - \rho_{j2}|$.

Na formulação, as restrições são convertidas em funções de penalidades. Desta forma, temos:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n} \theta_{ij}\, d_{ij} \;+\; \sum_{k=1}^{n} \left(w^{\alpha}_{k}\, \alpha_k + w^{\beta}_{k}\, \beta_k\right) \;,\quad x \in X$$

onde:
$\alpha_k = max\{\, 0,\, max\{(a_k - max\{a_{k(max)},\, 1/a_{k(min)}\}),\, (min\{a_{k(min)},\, 1/a_{k(max)}\} - a_k)\}\},$
$\beta_k = max\{\, 0,\, \sigma_k - \sigma_{k(max)}\},$
$w^{\alpha}_{k},\, w^{\beta}_{k} \ge 0,$
$k = 1, 2, \ldots., n.$

O primeiro termo da função objetivo mede o volume de tráfego. O segundo termo representa uma função de penalidades para restrições geométricas. O primeiro termo da função de penalidades mede a extensão em que a razão de aspecto é violada. Uma vez que cada facilidade tem um único nível de tolerância para a violação destas restrições, α_k é multiplicada por um fator w_k^{α}.

Igualmente, o segundo termo β_k pune a existência de espaços ocupados dentro da facilidade. Pesos individuais $w_k^{\beta} \geq 0$ podem ser atribuídos às facilidades, para refletir seus níveis de tolerância de irregularidade de forma.

4. Testes Realizados

Realizamos sete grupos de testes usando como matriz de tráfego os dados do trabalho de NUGGENT *et al.* (1968) e como áreas das facilidades dimensões da região em que o leiaute será construído, os dados de trabalho de TAM (1992).

O programa de computação foi implementado na linguagem C++, usando compilador Borland, em um computador PC-486, DX2-66.

GRUPO 1: *rep* = 4 (vizinhança, são escolhidos quatro pares de folhas aleatoriamente e permutados),
t = 3 (tamanho da lista tabu),
nbmax = 150, e
$x_{inicial}$ = a solução do algoritmo AA,
movimento: exclusivamente permutação de folhas na árvore binária.

GRUPO 2: *rep* = 4,
t = 3,
nbmax = 400, e
$x_{inicial}$ = a solução do algoritmo AA,
movimento: exclusivamente permutação de folhas na árvore binária.

GRUPO 3: *rep* = 4,
t = 3,
nbmax = 150, e
$x_{inicial}$ = a geração de uma solução aleatória (mas com mesma estrutura de árvore binária gerada pelo algoritmo AA),
movimento: exclusivamente permutação de folhas na árvore binária.

GRUPO 4: *rep* = 4,
t = 3,
nbmax = 400, e
$x_{inicial}$ = a geração de uma solução aleatória (mas com mesma estrutura de árvore binária gerada pelo algoritmo AA),
movimento: exclusivamente permutação de folhas na árvore binária.

GRUPO 5: *rep* = 4,
t = 3,
nbmax = 150, e
$x_{inicial}$ = a solução do algoritmo AA,

movimento: permutação de folhas e de nós internos.

GRUPO 6: *rep* = 4,
t = 3,
nbmax = 400, e
$x_{inicial}$ = a solução do algoritmo AA,
movimento: permutação de folhas e de nós internos.

GRUPO 7: *rep* = toda vizinhança, ou seja, a permutação de todos os nós internos, seguido da permutação de todas as folhas;
t = 3,
nbmax = 150, e
$x_{inicial}$ = a solução do algoritmo AA,
movimento: permutação de folhas e de nós internos.

Os grupos de teste **1** e **2** usam como solução inicial o algoritmo de aglomeração AA. Estas soluções iniciais apresentam bons resultados e conseqüentemente de difícil melhoramento. As figuras 4.1, 4.2, 4.3 e 4.4 mostram os leiautes finais para os problemas teste usando os dados do grupo **1**. As figuras 4.5, 4.6, 4.7 e 4.8 mostram os leiautes finais para os problemas teste usando os dados do grupo **2**. A tabela 4.1 mostra a qualidade dos leiautes iniciais produzidos pelo AA. A tabela 4.2 mostra a qualidade dos leiautes após aplicação do algoritmo ABT, usando o grupo de dados **1**, e tabela 4.3 mostra a qualidade do leiaute para o grupo **2**.

Os grupos de teste **3** e **4** são formados para verificar a eficiência do algoritmo ABT numa inicialização aleatória. Nós usamos a árvore binária gerada pelo algoritmo AA, mas com facilidades introduzidas aleatoriamente nas suas folhas. Cinco instâncias são consideradas para cada problema. A tabela 4.4 mostra os resultados para o grupo **3**, e a tabela 4.5 mostra os resultados para o grupo **4**.

Os grupos de teste **5** e **6** são formados para verificar o comportamento na busca quanto existe troca de nós internos. Como há uma alteração mais profunda na estrutura da árvore binária, acreditamos que a troca de nós internos pode ser vista como um processo de diversificação na busca tabu. Os resultados obtidos para 150 iterações encontram-se na tabela 4.6 e para 400 iterações na tabela 4.7.

No grupo de teste ***7*** usamos como solução inicial o resultado obtido pelo algoritmo AA. A cada iteração é realizada a permutação de todos os nós internos e a permutação de todas as folhas da árvore binária, ou seja, toda a vizinhança é considerada. Nesta implementação, duas listas tabu são usadas. Na primeira lista, os movimentos tabu relativos à permutação de nós internos são armazenados e na segunda os movimentos tabu relativos à permutação de folhas são armazenados. Vemos na tabela 4.8 que os resultados obtidos são melhores que os dos testes anteriores, usando-se menos iterações (50 iterações). No entanto, é necessário destacar que cada iteração apresenta tempo de execução maior que o dos casos anteriormente estudados.

Figura 4.1 - Leiaute após 150 iterações, para 12 facilidades.

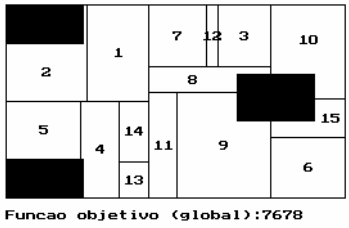

Figura 4.2 - Leiaute após 150 iterações, para 15 facilidades.

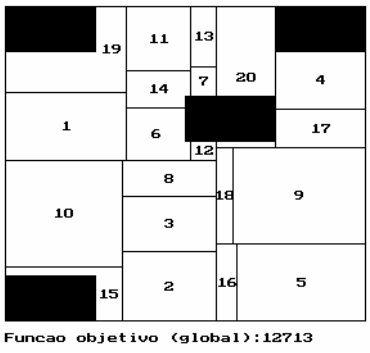

Figura 4.3 - Leiaute após 150 iterações, para 20 facilidades.

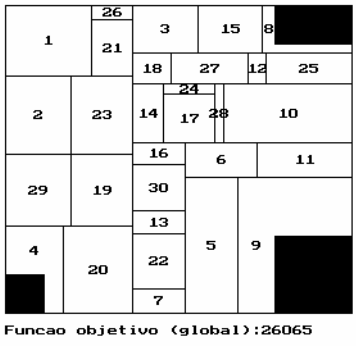

Figura 4.4 - Leiaute após 150 iterações, para 30 facilidades.

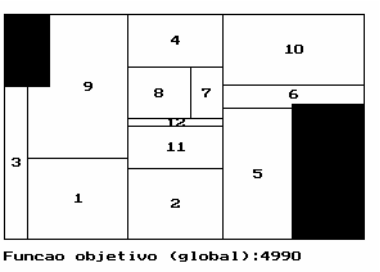

Figura 4.5 - Leiaute após 400 iterações, para 12 facilidades.

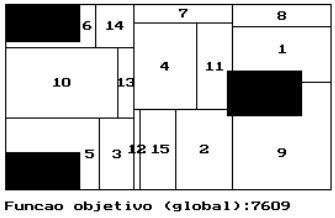

Figura 4.6 - Leiaute após 400 iterações, para 15 facilidades.

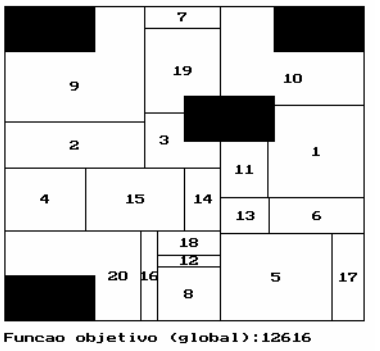

Figura 4.7 - Leiaute após 400 iterações, para 20 facilidades.

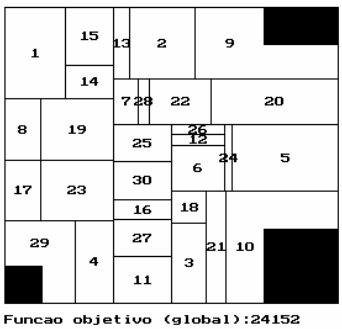

Figura 4.8 - Leiaute após 400 iterações, para 30 facilidades.

Tabela 4.1 - Qualidade dos leiautes iniciais produzidos pelo processo de aglomeração

Número de facilidades	Função objetivo	Tempo(segundos)	Número de violações
12	5083	4	7
15	7961	7	9
20	14667	14	10
30	27392	31	14

Tabela 4.2 - Qualidade dos leiautes produzidos após 150 iterações, com leiaute inicial produzido por meio do procedimento de aglomeração

Número de facilidades	Função objetivo	% Melhoria	Tempo (min)	Violações
12	5041	0,83	13,40	7
15	7678	3,55	14,01	8
20	12713	13,32	19,24	10
30	26065	4,84	23,08	13

Tabela 4.3 - Qualidade dos leiautes produzidos após 400 iterações, com leiaute inicial produzido por meio do procedimento de aglomeração

Número de facilidades	Função objetivo	% Melhoria	Tempo (min)	Violações
12	4990	1,83	34,23	6
15	7609	4,42	36,52	8
20	12616	13,98	49,60	10
30	24152	11,83	58,31	13

Tabela 4.4 - Qualidade dos leiautes iniciais e após 150 iterações, com leiautes iniciais produzidos de forma aleatória

- Para 12 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
6620	5160	22,05	13,41	7
6365	5117	19,61	13,38	8
5439	5353	1,58	13,35	6
7123	5069	28,84	13,44	5
6871	5175	24,68	13,40	8

Média Inicial: 6483,6

Média Final: 5174,8
% Média de Melhoria: 20,19
Média Tempo (min): 13,39
Violações: 6,8

- Para 15 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
8165	7826	4,15	14,08	10
8431	7545	10,51	14,10	4
8405	7598	9,60	13,97	6
8861	8086	8,75	14,01	8
8474	7571	10,66	13,96	9

Média Inicial: 8467,2
Média Final: 7725,2
% Média de Melhoria: 8,76
Média Tempo (min): 14,02
Violações: 7,4

- Para 20 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
15123	13033	13,82	19,20	11
14895	13334	10,48	19,19	10
14937	12966	13,19	19,15	9
15420	12840	16,73	19,21	10
15008	13457	10,33	19,23	10

Média Inicial: 15076,6
Média Final: 13126
% Média de Melhoria: 12,94
Média Tempo (min): 19,19
Violações: 10

- Para 30 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
30393	26559	12,61	23,10	14
32201	27384	14,96	23,20	15
30251	27866	7,88	23,08	13
28857	27172	5,84	22,95	12
31522	26972	14,43	22,90	14

Média Inicial: 30644,8
Média Final: 27190,6
% Média de Melhoria: 11,27
Média Tempo (min): 23,05

Violações: 13,6

Tabela 4.5 - Qualidade dos leiautes iniciais e após 400 iterações, com leiautes iniciais produzidos de forma aleatória

- Para 12 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
6973	5075	27,22	34,27	8
7267	5078	30,12	34,19	6
5966	5081	14,83	34,20	6
6780	4980	26,55	34,20	7
7122	5112	28,22	34,30	6

Média Inicial: 6821,6
Média Final: 5065,2
% Média de Melhoria: 25,75
Média Tempo (min): 34,23
Violações: 6,6

- Para 15 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
7983	7688	3,69	36,40	8
8836	7746	12,34	36,41	7
8430	7693	8,74	36,37	6
8511	7610	10,59	36,40	9
8632	7701	10,78	36,43	8

Média Inicial: 8478,4
Média Final: 7687,6
% Média de Melhoria: 9,33
Média Tempo (min): 36,40
Violações: 7,6

- Para 20 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
14830	12676	14,52	49,73	8
15557	12754	18,02	49,71	9
15030	12710	15,44	49,75	10
14995	12642	15,69	49,61	8
15423	12627	18,13	49,69	11

Média Inicial: 15167
Média Final: 12681,8
% Média de Melhoria: 16,39
Média Tempo (min): 49,70

Violações: 9,2

- Para 30 facilidades:

INICIAL	FINAL	% MELHORIA	TEMPO (MIN)	VIOLAÇÕES
31150	24671	20,79	58,42	15
28055	24530	12,56	58,30	13
30548	25278	17,25	58,45	12
32105	24960	22,26	58,37	16
31482	24734	21,43	58,36	13

Média Inicial: 30668
Média Final: 24834,6
% Média de Melhoria: 19,02
Média Tempo (min): 58,38
Violações: 13,8

Tabela 4.6 - Qualidade dos leiautes após 150 iterações, com leiautes iniciais produzidos pelo processo de aglomeração e movimento caracterizado pela troca de nós internos e de folhas

Número de facilidades	Função objetivo inicial	Função objetivo final	% Melhoria	Tempo (min)
12	5083	5083	0	14,53
15	7961	7421	6,78	15,46
20	14667	13325	9,15	21,54
30	27392	26552	3,07	26,00

Tabela 4.7 - Qualidade dos leiautes após 400 iterações, com leiautes iniciais produzidos pelo processo de aglomeração e movimento caracterizado pela troca de nós internos e de folhas

Número de facilidades	Função objetivo inicial	Função objetivo final	% Melhoria	Tempo (min)
12	5083	4994	1,75	36,58
15	7961	7393	7,13	39,04
20	14667	13078	10,83	54,11
30	27392	26988	1,48	63,50

Tabela 4.8 - Qualidade dos leiautes após 50 iterações, com leiautes iniciais produzidos pelo processo de aglomeração e movimento caracterizado pela troca de nós internos (vizinhança: todos os nós e todas as folhas possíveis)

Número de facilidades	Função objetivo inicial	Função objetivo final	% Melhoria	Tempo (min)
12	5083	4990	1,83	42,23
15	7961	7575	4,85	48,11
20	14667	12301	16,13	61,22
30	27392	23754	13,28	87,02

5. Conclusões

Este trabalho apresenta restrições maiores que as dos normalmente encontrados na literatura. Consideramos áreas ocupadas no interior da região a ser usada para alocação das facilidades, o que torna mais complexo e demorado o processo de busca. O único trabalho que trata destas restrições, citado na literatura, é o de TAM (1992).

No trabalho realizado, podemos observar que a solução inicial encontrada é de qualidade considerável, devido ao processo de aglomeração que aproxima facilidades de alto tráfego e afasta as de baixo tráfego. Vemos também que o processo de aglomeração é bastante rápido.

A busca tabu implementada conseguiu melhorar a qualidade dos leiautes iniciais produzidos com o processo de aglomeração. No entanto, devemos observar que esse aumento na qualidade torna-se difícil de ser muito significativo, uma vez que o leiaute inicial é de uma boa qualidade e que durante o processo de busca, a maioria das permutações (movimentos tabu) realizadas pioram a qualidade da solução inicial.

A busca tabu conseguiu melhorar mais acentuadamente a qualidade das soluções iniciais (em termos percentuais) produzidas aleatoriamente do que quando a busca foi aplicada às soluções iniciais produzidas pelo procedimento de aglomeração. Isso se deve ao fato de que, em regra, as soluções iniciais obtidas pelo procedimento de aglomeração estão mais próximas de uma solução de boa qualidade do que aquelas obtidas aleatoriamente.

A introdução do procedimento de diversificação, representado pela permutação de nós internos foi responsável pelos melhores resultados para 20 e 30 facilidades, quando toda a vizinhança foi considerada. Nos testes realizados com o programa de computação, observamos que o aumento do número de iterações processadas pela busca tabu implicou sistematicamente uma melhoria da solução final. Para os testes em que o número máximo de iterações foi estendido para 400, a busca tabu obteve resultados ligeiramente superiores quando aplicada em soluções iniciais obtidas pelo procedimento de aglomeração se comparada à aplicação da busca tabu em soluções iniciais produzidas aleatoriamente.

Como atividades que podem ser desenvolvidos em trabalhos futuros, destacamos: os critérios de aspiração podem ser aperfeiçoados, o número de iterações pode ser aumentado, a lista tabu pode ser variada. Acreditamos que os valores dos parâmetros w^{α}_{k} e w^{β}_{k} devam ser mais elevados, para que a razão de aspecto das facilidades não seja tão fortemente violada.

Referências Bibliográficas:

ANDERBERG, M.R.: *Cluster Analysis for Applications*, Nova York, Academic Press, 1973.

ARMOUR, G.C. & BUFFA, E.S.: "A heuristic algorithm and simulation approach to the relative location of facilities". *Management Science*, 9:295-309, 1963.

BLAND, J.A. & DAWSON, G.P.: "Tabu search and design optimization". *Computer-aided design*, 23(3):195-201, 1991.

CO, H.; WU, A. & REISMAN A.: "A throughput-maximizing facility planning and layout model". *International Journal Production Research*, 27(1):1-12, 1989.

COHOON, J.P.; HEGDE, S.U.; MARTIN, W.N. & RICHARDS, D.S.: "Distributed Genetic Algorithms for the Floorplan Design Problem". *IEEE transactions on computer-aided design*, 10(4):483-492, 1991.

DEISENROTH, M.P. & APPLE J.M.: "A Computerized Plant Layout Analysis and Evalution Technique (PLANET)", *Technical papers 1962*, American Institute of Industrial Engineers, Norcross, Ga., 1972.

DIAZ, B.A.: "Restricted neighborhood in the tabu search for the flowshop problem". *European Journal of Operational Research*, 62:27-37, 1992.

GLOVER, F.: "Tabu Search - Part I". *ORSA Journal on Computing*, 1(3):190-206, 1989a.

GLOVER, F.: "Tabu Search - Part II", *ORSA Journal on Computing*, 2(1):4-32, 1989b.

GLOVER, F.: "Tabu Search - A tutorial". *Interfaces*, 20(4):74-94, 1990.

GLOVER, F. & McMILLAN, C.: "Interactive decision software and computer graphics for architectural and space planning". *Annals of operations research*, 5:557-573, 1985.

HERAGU, S.S. & ALFA, A.S.: "Experimental analysis of simulated annealing based algorithms for the layout problem". *European Journal of Operational Research*, 57:190-202, 1992.

KIM, Y.; JANG Y. & KIM, M.: "Stepwise-overlapped parallel annealing and its application to floorplan designs". *Computer-aided design*, 23(2):133-144, 1991.

LANCE, G.N. & WILLIAMS, W.T.: "A Generalized sorting strategy for computer classification". *Nature*, 5:212-218, 1966.

LEE, R.C. & MOORE, J.M.: "CORELAP - Computerized Relationship Layout Planning". *Industrial Engineering*, 18:195-200, 1967.

NUGENT, C.E.; VOLLMANN, T.E. & RUML, J.: "An Experimental comparison of techniques for the assignment of facilities to locations". *Operations Research* 16:150-173, 1968.

O'BRIEN, C. & BARR, S.E.Z.A.: "An interactive approach to computer aided facility layout". *International Journal Production Research*, 18(2):201-211, 1980.

SAHNI, S. & GONZALEZ, T.: "P-complete approximation problem". *Journal of Associated Computing Machinery*, 23(3):555-565, 1976.

SEEHOF, J.M. & EVANS, W.O.: "Automated layout design program". *Industrial Engineering*, 18:690-695, 1967.

SKORIN-KAPOV, J.: "Tabu search applied to the quadratic assignment problem". *ORSA Journal on computing*, 2(1):33-45, 1990.

TAM, K.Y.: "Genetic algorithms, function optimization, and facility layout design". *European Journal of Operational Research,* 63:322-346, 1992.

TOMPKINS, J.A. & REED Jr, R.: "An Applied Model for the Facilities Design Problem". *International Journal of Production Research,* vol. 14, no. 5, pp. 583-595, setembro 1976.

YEAP, G.K. & SARRAFZADEH, M.: "A unified approach to floorplan sizing and enumeration". *IEEE transactions on computer-aided design of integrated circuits and systems,* 12(12),1858-1867, 1993.

ZUPAN, J.: *Clustering of large data sets.* Nova York, Research Studies Press, 1982.

FACILITY LAYOUT OPTIMIZATION USING TABU SEARCH

Abstract

The search for high quality facility layouts can be viewed as combinatorial optimization problems arising in a wide variety of spatial planning contexts. We examine here the problem of optimizing a layout of facilities in a supplied space that can include unallowed areas. The layout space and facility areas are rectangular and limited by aspect rations. Considering the problem complexity, we propose a tabu search heuristic for the problem. First, an initial layout is generated using a clustering procedure. Considering n facilities, the representation is given by a binary tree with solution space size n!. The tabu search is then applied twofold, using the initial layout and using a random generated layout that maintains only the tree structure of the first layout. Quality results for both cases are obtained for problems of the literature.

Key words: tabu search; facility layout; optimization.