unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

**PAULA ROCHA ANDRADE**

# UM ALGORITMO DE BUSCA TABU PARA PROBLEMAS DE LOCALIZAÇÃO NÃO CAPACITADOS

Guaratinguetá
2012

PAULA ROCHA ANDRADE

# UM ALGORITMO DE BUSCA TABU PARA PROBLEMAS DE LOCALIZAÇÃO NÃO CAPACITADOS

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia de Produção Mecânica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia de Produção Mecânica.

Orientador: Prof. Dr. Edson L. F. Senne

Guaratinguetá
2012

Andrade, Paula Rocha
A553a Um algoritmo de busca tabu para problemas de localização não capacitados / Paula Rocha Andrade – Guaratinguetá : [s.n], 2012.
50 f : il.
Bibliografia: f. 39

Trabalho de Graduação em Engenharia de Produção Mecânica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2012.
Orientador: Prof. Dr. Edson Luiz França Senne

1. Algoritmos I. Título

CDU 519.712

## AGRADECIMENTOS

Agradeço a todos aqueles que me apoiaram durantes os estudos, principalmente meus pais, *José e Gilda*,

ao meu orientador, *Prof. Dr. Edson L. F. Senne,* pela atenção e todo auxilio, e sem o qual este trabalho não poderia ser realizado.

## EPÍGRAFE

"Enquanto a virtude não compensar já nesta vida,
a ética pregará em vão.

Freud

ANDRADE, P. R. **Um algoritmo de Busca Tabu para problemas de localização não capacitados.** 2012. 50 p. Trabalho de Graduação (Graduação em Engenharia de Produção Mecânica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2012.

## RESUMO

Neste trabalho, apresenta-se um algoritmo de busca tabu para a solução de problemas de localização não-capacitados. O problema de localização não-capacitado é um problema clássico de localização e ocorre em diversas situações práticas. O problema consiste em determinar a melhor localização, em uma rede, para a abertura de um conjunto de instalações (também conhecidas como facilidades) de modo a atender às demandas associadas aos clientes, ao menor custo possível. Admite-se que existe um custo associado à abertura de uma facilidade e um custo de atendimento de cada cliente por uma das facilidades abertas. No caso particular do problema de localização de facilidades não-capacitado não existe limitação quanto à capacidade de uma facilidade em atender às demandas dos seus clientes. O algoritmo proposto possui vários parâmetros que influenciam a qualidade da solução. Tais parâmetros foram testados e foram obtidos valores favoráveis para estes. Os resultados mostram que o algoritmo proposto é capaz de encontrar a solução ótima para todos os problemas pequenos testados, mantendo o compromisso entre a qualidade da solução e o tempo computacional. Já para problemas maiores, o algoritmo deve sofrer melhoras em sua estrutura. O algoritmo implementado está integrado a uma plataforma computacional para solução de problemas logísticos

**PALAVRAS CHAVE.** Metaheurísticas. Localização de facilidades. Busca tabu. Área de classificação principal: Metaheurísticas.

ANDRADE, P. R. **A Tabu Search algorithm for uncapacitated location problems.** 2012. 50 p. Graduate Work (Graduate in Mechanical Production Engineering) - Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2012.

## ABSTRACT

In this work, a tabu search algorithm for solving uncapacitated location problems is presented. The uncapacitated location problem is a classic problem of localization and occurs in many practical situations. The problem consists in determining in a network, at the minimum possible cost, the better localization, in a network, for the installation of facilities in order to attend the customers' associated demands, at the minimum possible cost. One admits that there exists a cost associated with the opening of a facility and a cost of attendance of each customer by any open facilities. In the particular case of the uncapacitated location problem there is no capacity limitation to attend the customers' demands. There are some parameters in the algorithm that influence the solution's quality. These parameters were tested and optimal values for them were obtained. The results show that the proposed algorithm is able to find the optimal solution for all small tested problems keeping the compromise between solution's quality and computational time. However, to solve bigger problems, the structure of the algorithm must be changed in its structure. The implemented algorithm is integrated to a computational platform for solution of logistic problems.

**KEYWORDS**. Metaheuristics. Facility location. Tabu search. Main area: Metaheuristics.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## SUMÁRIO

# 1 INTRODUÇÃO

A Pesquisa Operacional possui várias técnicas para o auxílio à obtenção de soluções de diversos tipos de problemas de otimização. O problema de localização de facilidades é um problema clássico de otimização que consiste em estabelecer os locais onde devem ser abertas facilidades para atender, ao menor custo possível, um conjunto espacialmente distribuído de pontos de demanda identificados como clientes de alguma facilidade (Drezner, 1995). O termo "facilidade" pode se referir a fábricas, depósitos, escolas, postos de saúde, centros de distribuição, pontos de ônibus, dentre muitas outras possibilidades.

Tal problema apresenta uma importância crescente para alguns setores empresariais, à medida que os custos referentes à distribuição de seus produtos influenciam diretamente no preço do produto oferecido ao mercado. Segundo Sun (2006), o sucesso de empresas e instalações públicas depende, em grande parte, da sua localização. Este fato pode ser comprovado analisando-se o valor da logística nas políticas estratégicas das empresas. Redes de suprimentos eficazes acarretam diminuição no custo total de produção e, como consequência, levam à satisfação do cliente, sendo, portanto, uma grande vantagem competitiva.

O problema de localização de facilidades é um problema clássico de Otimização Combinatória com diversas aplicações práticas. Exemplos de aplicações deste problema aparecem em localização de escolas, postos de saúde, corpo de bombeiros, ambulâncias, viaturas de polícia, pontos de ônibus, localização de fábricas, depósitos, torres de transmissão, lojas de franquias, dentre muitos outros.

Como estes problemas demandam grande esforço matemático, é necessário a utilização de ferramentas computacionais para encontrar a melhor solução possível. Neste trabalho, pretende-se implementar, na linguagem C, um algoritmo de Busca Tabu para a solução de problemas de localização de facilidades não-capacitados.

O algoritmo de Busca Tabu tem sido aplicado com sucesso em muitos problemas de localização, em particular, a problemas de localização de facilidades. Este algoritmo possui alguns parâmetros básicos que afetam a qualidade da solução e o tempo de resposta. Além da implementação do algoritmo, este trabalho tem como objetivo discutir a relação destes parâmetros com a eficácia do programa.

Este trabalho está organizado da forma descrita a seguir. No Capítulo 2, Fundamentação Teórica, são apresentados conceitos relevantes para o entendimento do trabalho. No Capítulo 3, o método implementado, apresenta-se as etapas para realização do trabalho e do algoritmo implementado.. No Capítulo 4 são apresentados os resultados computacionais e a discussão sobre estes resultados. Já no capítulo 5, apresenta-se os resultados e a discussão para testes realizados com problemas grandes. E por fim, no Capítulo 6, Conclusões, apresenta-se a conclusão do trabalho.

É importante observar que este trabalho é baseado nos resultados da Iniciação Científica realizada pela aluna, autora deste texto, sendo o orientador o mesmo na Iniciação e neste trabalho.

O principal acréscimo neste trabalho, em relação à iniciação científica é que, além dos resultados obtidos naquela, novos testes foram realizados para problemas maiores. Estes testes podem ser encontrados no capítulo 5, Problemas Grandes.

Além disto, novos testes também foram feitos para os problemas pequenos já testados antes, e novas interpretações foram feitas com base nestes novos testes. Também, alterações foram realizadas no texto da monografia, em todos os capítulos, a fim de aperfeiçoar e garantir a coesão do texto do trabalho anterior com o que foi adicionado neste.

# 2. FUNDAMENTAÇÃO TEÓRICA

## 2.1 Pesquisa operacional

A pesquisa operacional é uma ciência aplicada cujo objetivo é auxiliar a tomada de decisões relacionadas a problemas complexos reais. Esta ciência se originou durante a segunda guerra mundial e devido ao sucesso obtido, após o conflito, passou a ser empregada em empresas. Dentre as disciplinas que constituem a pesquisa operacional tem-se a programação linear, teoria das filas, simulação, programação dinâmica, teoria dos jogos, entre outros.

Como ciência aplicada, a pesquisa operacional possui várias técnicas para o auxílio de solução de problemas de otimização, modelando matematicamente problemas reais.

Uma das classes de problemas de interesse da pesquisa operacional, é a classe dos problemas de localização de facilidades.

## 2.2 O problema de localização de facilidades

Os problemas de localização de facilidades podem ser modelados de diversas formas. Há modelos nos quais as facilidades podem ser alocadas em qualquer local, enquanto em outros, as facilidades só podem ser alocadas em nós ou arcos de uma rede. Este último é conhecido como modelo de localização em redes. Em alguns casos de modelos de localização de facilidades deseja-se obter a distância máxima entre facilidades e clientes. Esta distância, quando definida a priori, é conhecida como distância de cobertura (Toregas *et al*., 1971).

O problema de localização de facilidades (PLF) pode ser classificado como capacitado (PLFC) ou como não-capacitado (PLFNC). No primeiro caso, cada cliente possui uma demanda a ser suprida e cada facilidade possui uma capacidade limitada de atendimento (Ducati, 2003). No PLFNC não existem restrições de atendimento de demandas. Em ambos os casos, o objetivo é minimizar os custos relacionados à abertura de facilidades e à alocação dos clientes às facilidades abertas.

Pode-se imaginar que os locais potenciais para as facilidades e os locais dos clientes a serem atendidos constituem o conjunto V = {1, ..., n} de vértices de uma rede e que os caminhos entre os locais das facilidades e os locais dos clientes constituem o conjunto A = {$a_1$, ..., $a_m$} de arcos da rede, com $a_k$ = (i, j) e i, j ∈ V. Pode-se admitir também que, associado a cada arco $a_k$ = (i, j), k = 1, ..., m, da rede, existe um valor não-negativo $c_{ij}$, o qual pode ser interpretado como o custo de atendimento do cliente j pela facilidade i. Portanto, problemas de localização de facilidades podem ser formulados como modelos de otimização em redes.

O PLFNC pode ser formulado como o seguinte problema de programação inteira binária (Krarup e Pruzan, 1983):

$$v(PLFNC) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i \qquad (1)$$

sujeito a:

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j \in N = \{1, \ldots, n\} \qquad (2)$$

$$x_{ij} \leq y_i \qquad i \in M = \{1, \ldots, m\},\ j \in N \qquad (3)$$

$$x_{ij}, y_i \in \{0,1\} \qquad i \in M,\ j \in N \qquad (4)$$

em que:

- $m$ é o número de locais potenciais para as facilidades;
- $n$ é o número de clientes a serem atendidos;
- $c_{ij}$ é o custo de atendimento do cliente $j$ pela facilidade $i$;
- $f_i$ é o custo fixo para a abertura da facilidade $i$;
- $x_{ij}$ é uma variável de decisão, com $x_{ij}$ = 1 se a demanda do cliente $j$ é satisfeita pela facilidade $i$ e $x_{ij}$ = 0, caso contrário;
- $y_i$ é uma variável de decisão, com $y_i$ = 1 se a facilidade $i$ deve ser aberta e $y_i$ = 0, caso contrário.

Nesta formulação, a função-objetivo (1) corresponde ao custo total, composto pelo custo fixo de abertura das facilidades e pelo custo de atendimento das demandas, que deve ser minimizado. As restrições (2) garantem que cada cliente *j* será atendido por uma única facilidade *i*. As restrições (3) asseguram que um cliente *j* será atendido pela facilidade *i* somente se esta facilidade estiver aberta. As restrições (4) indicam que as variáveis de decisão são binárias.

Portanto, no PLFNC tem-se como objetivo determinar a melhor localização de facilidades, dentre locais potenciais, capaz de suprir todas as demandas dos clientes ao menor custo. O problema consiste em determinar a localização das facilidades, bem como os clientes atendidos pelas facilidades, a fim de minimizar o custo total, composto pelo custo fixo de abertura das facilidades e pelo custo de atendimento dos clientes. Considera-se que as facilidades não apresentam necessariamente o mesmo custo de abertura e os mesmos custos de atendimento.

O termo "não-capacitado" refere-se à ausência de restrições de capacidade de atendimento dos clientes pelas facilidades. Este problema possui ampla aplicação prática. Pode ser utilizado para determinar a melhor localização de escolas, hospitais e outros serviços de atendimento à população. Na indústria, o PLFNC pode ser utilizado para determinar os melhores locais para a instalação de fábricas de um determinado produto para o atendimento dos clientes de vários centros de distribuição deste produto, de modo a minimizar os custos de produção em cada fábrica e os custos de transporte das fábricas para os centros de distribuição.

É interessante observar que, como não existem restrições de capacidade de atendimento das facilidades, na situação ótima cada cliente será inteiramente atendido pela facilidade melhor localizada. Isto pode ser importante, por exemplo, quando se deseja que cada centro de distribuição seja atendido por uma única fábrica, a fim de garantir um bom serviço de atendimento.

O PLFNC é conhecido ser um problema NP-difícil (Garey e Johnson, 1979), implicando que soluções ótimas para o problema podem ser obtidas apenas com algoritmos enumerativos. Assim, para problemas de grande porte uma solução ótima

pode ser obtida somente às custas de um grande esforço computacional (Korkel, 1989; Galvão e Raggi, 1989).

Situações práticas do problema, no entanto, exigem que decisões sejam tomadas rapidamente. Assim, nem sempre uma solução ótima será possível (ou necessária).

## 2.3 Heurísticas e meta-heurísticas

Define-se procedimento heurístico como um método de aproximação das soluções ideais dos problemas, a heurística assume uma solução próxima da ideal baseada em um função de avaliação do resultado. A solução encontrada por uma heurística não é necessariamente a melhor possível.

Uma meta-heurística é um método heurístico para resolver de forma genérica problemas de otimização. Metaheurísticas são geralmente aplicadas a problemas para os quais não se conhece algoritmo eficiente.

As meta-heurísticas utilizam combinações de escolhas aleatórias e conhecimento histórico dos resultados anteriores, adquiridos pelo método, para se guiarem e realizar suas buscas pelo espaço de pesquisa em vizinhanças dentro do espaço de pesquisa, o que evita óticos locais.

A necessidade de se obter boas soluções rapidamente tem levado às técnicas heurísticas e meta-heurísticas para a solução do problema. Vários algoritmos heurísticos têm sido desenvolvidos e aplicados com sucesso ao PLFNC. Alguns destes algoritmos aparecem em (Beasley, 1993). Neste projeto pretende-se implementar um algoritmo de busca tabu para a solução do problema.

## 2.4 Busca tabu

A busca tabu (Glover e Laguna, 1997) é uma meta-heurística que consiste em explorar o espaço de soluções de um problema combinatório evitando-se executar movimentos desnecessários. Para tal, são definidos movimentos proibidos (ou tabus). A imposição de movimentos tabus força a passar por regiões do espaço de soluções ainda inexploradas, orientando a busca em direção à solução ótima do problema.

Assim, pode-se dizer que a busca tabu faz proibições de forma a fugir dos riscos de um caminho não recomendável para a busca que está sendo feita. Uma característica do algoritmo de busca tabu é o uso de memória adaptativa para guiar a busca. Esta memória adaptativa é usada para armazenar os passos já realizados pela busca, de forma a impedir que a busca retorne a posições recém visitadas. Na busca tabu isto é feito definindo-se os movimentos tabus, porém permitindo que um movimento tabu em um dado momento da busca não o seja em um outro momento posterior, dependendo do conteúdo da memória adaptativa.

Normalmente, na busca tabu, os movimentos recém realizados são definidos como tabus e mantidos nesse status por um dado número de iterações. Depois de um determinado número de iterações, um movimento deixa de ser tabu e pode ser realizado novamente, caso a busca seja guiada para isso. A forma de controlar os movimentos tabus é criar uma lista tabu com o número da iteração em que o movimento foi feito. O algoritmo de busca tabu, no entanto, permite que um movimento tabu seja realizado se, com este movimento, encontra-se uma solução melhor do que a melhor solução já visitada.

A forma de realizar a busca, normalmente, é visitar as possíveis soluções e analisar suas vizinhanças, de modo a encontrar a melhor solução possível, realizando uma troca desde que o movimento não seja tabu. Assim, a busca tabu consegue explorar o espaço de soluções evitando os mínimos locais e buscando analisar regiões ainda não visitadas. Com isso, a busca tabu consegue realizar uma busca global e, ao mesmo tempo, alcançar as melhores soluções locais.

A busca tabu vem sendo utilizada para solução de vários problemas de localização. Al-Sultan e Al-Fawzan (1999) apresentam um algoritmo de busca tabu simples para o PLFNC. Os resultados computacionais mostram que este algoritmo foi capaz de encontrar a solução ótima de todos os problemas-teste, sendo muito eficiente em termos de tempo quando comparado com outros algoritmos existentes na literatura.

Michel e Van Hentenryck (2004) desenvolveram um algoritmo simples e rápido para o problema de localização de armazéns não-capacitado. Mladenovic *et al*. (2003) discutem a aplicação da busca tabu ao problema de p-centros. Este problema objetiva

localizar p nós (centros) em uma rede, de tal forma a minimizar a distância máxima de qualquer nó da rede a um centro.

Problemas de p-centros são muito usados no caso de localização de pronto-socorros e bombeiros, onde existe a necessidade da facilidade ser acessível a todos os clientes da forma mais rápida possível. Carello *et al*. (2004) discutem a aplicação de busca tabu na solução de problemas de localização de concentradores (*hubs*) que surgem em projetos de redes de transporte e de telecomunicação.

# 3 O MÉTODO IMPLEMENTADO

Para o desenvolvimento do projeto realizou-se as seguintes atividades:

- Levantamento bibliográfico sobre Busca Tabu e sua aplicação a problemas de localização de facilidades.
- Escolha das estruturas de dados necessárias para a implementação do algoritmo de Busca Tabu.
- Implementação do algoritmo de Busca Tabu para o PLFNC.
- Realização de testes computacionais visando demonstrar a efetividade e eficiência do algoritmo implementado para problemas-teste disponíveis na literatura.

O algoritmo de busca tabu implementado tem como objetivo encontrar a melhor solução para o problema de localização de facilidades não-capacitado. Para tal finalidade, o algoritmo realiza iterações de busca tabu nas quais as soluções encontradas são comparadas a fim de se determinar aquela que apresenta o menor custo total.

O processo de busca parte de uma solução inicial e busca por outras soluções melhores que as obtidas anteriormente. Neste processo, além das informações atuais, são armazenadas também informações sobre as últimas soluções encontradas. Estas informações são armazenadas na lista tabu, que guia a busca pelas novas soluções. No anexo B encontra-se um fluxograma simplificando o algoritmo implementado

Na primeira parte do algoritmo (denominada NBH – *net benefit heuristic*) é encontrada uma solução através de duas fases: inicial e de refinamento. Na fase inicial é encontrada a facilidade que supre cada cliente ao menor custo e calcula-se o custo total para tal configuração. Têm-se, portanto, o estabelecimento de certo número de facilidades que atenderá a demanda de todos os clientes. Na fase de refinamento, cada facilidade é examinada quanto à possibilidade de fechamento, se isto significar uma diminuição do custo total. Isto ocorrerá caso a economia com o fechamento da

facilidade for maior que o custo extra da realocação dos clientes inicialmente atendidas por esta facilidade.

Com a solução encontrada após a fase de refinamento, inicia-se a fase principal do algoritmo BT (busca tabu). Nesta fase, novas soluções vizinhas devem ser encontradas e comparadas com a melhor solução vigente. O algoritmo termina quando a condição determinada por parâmetros pré-estabelecidos é alcançada. Apresenta-se a seguir, o algoritmo de Busca Tabu a ser implementado.

**Algoritmo NBH**:

Sejam:

$v_i$: facilidade que entrega o menor custo à demanda i;

$U_i$: facilidades abertas na fase inicial;

$P_i$: demandas atendidas pelas facilidade i;

$d_i$: diferença de custos entre a melhor solução já encontrada e a solução corrente;

**Fase Inicial:**

*Passo 1*. Para cada nó de demanda *j (j = 1, ..., m),* encontre a facilidade que supra esta demanda ao menor custo de distribuição e denote-a por i*. Para todo *j (j = 1,...,m*), faça $c_{i^*j} = \min_{1 \le i \le n} c_{ij}$, e forme o vetor V dos pares ordenados, *V = {(i*,j)}*.

*Passo 2*. Forme o vetor I a partir de V extraindo o primeiro índice, ou seja: *I = {i* : (i*, j) ∈ V}*. Esta é a lista das facilidades sugeridas para serem abertas.

*Passo 3*. Avalie a solução corrente, que corresponde ao limitante superior *(UB)* da solução ótima, como:

$$UB = \sum_{(i,j) \in V} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i$$

**Fase de Refinamento:**

Passo 1: Forme o vetor U de facilidades abertas na fase inicial, a partir de I.

Para todo *i (i = 0, ..., n),* se $I_i$=1 então $U_{m^*}$ = i;

*Passo 2*. Considere a k-ésima facilidade aberta na fase inicial. Seja $i$ o $k$-ésimo elemento em $I$.

*Passo 3*. Seja $J$ o vetor de índices de todas as demandas satisfeitas pela facilidade $i$, ou seja:

$$J = \{ j \mid (i, j) \in V\}$$

*Passo 4*. Para cada demanda satisfeita pela facilidade $i$, encontrar qual outra facilidade satisfaz esta demanda ao mínimo custo de distribuição e calcule o custo extra devido a esta realocação. Ou seja, para cada $j \in J$ calcule:

$$d_j = c_{ij} - c_{tj}, \text{ no qual } c_{ij} = \min c_{ij},\ t \in I \text{ e } t \neq i.$$

*Passo 5*. Calcule o custo extra, que é igual ao custo de realocação das demandas originalmente satisfeitas pela facilidade $i$, menos a economia devido ao fechamento da facilidade $i$. Ou seja, calcule:

$$\delta = \sum_{j \in J} d_j - f_i$$

*Passo 6*. Se o custo extra calculado no passo 5 for negativo então é melhor fechar a facilidade (passo 7), caso contrário, considere a próxima facilidade (passo 8).

*Passo 7*. $I = I - \{i\}$. $UB = UB - |\delta|$. Se $|I| = 1$, pare; caso contrário, vá para o passo 1.

*Passo 8*. Se $k = m^*$, pare; caso contrário, faça $k = k + 1$ e vá para o passo 2.

**Algoritmo BT:**

**<u>Fase principal</u>:**

O algoritmo BT possui como parâmetros: *nbhsize* (o número de soluções que serão geradas a partir da solução corrente), *TL* (a quantidade de soluções que a lista tabu poderá conter), *MAXSEMMELHORA* (o número máximo de iterações que o algoritmo poderá realizar sem que haja melhora na solução), *ITERMAX* (o número máximo total de iterações) e $\alpha$ (possibilidade de cada facilidade mudar seu status).

Inicialmente, utiliza-se o algoritmo NBH para conseguir um vetor com as facilidades abertas e fechadas *y*. Fazer $y_{corrente} = y$, $y_{melhor} = y$. Seja totcost (y) o custo total de y.

Fazer $TL = \varnothing$, $BV = totcost\ (y)$ (onde *TL* é a lista tabu e *BV* é o melhor valor corrente).

Fazer $k = 1$.

*Passo 1*. Gerar *nbhsize* soluções aleatórias a partir da solução $y_{corrente}$. Cada solução é avaliada e aquela que apresentar o menor custo total dentre as soluções geradas é selecionada como $y_{min}$. Para isso, fazer:

$V_{min}$ = INFINITO.

Para h = 1 até *nbhsize* fazer:

Obter y a partir de $y_{corrente}$ através dos seguintes passos:

Passo 1.1: Seja $\alpha$ uma dada constante.

Passo 1.2: Fazer $k = 1$.

Passo 1.3: Gerar um número aleatório $\beta$ entre 0 e 100.

Passo 1.4: Se $\beta > \alpha$, fazer $y_k = 1 - y_k$ (ou seja, inverter o status de $y_k$ de 0 para 1, ou de 1 para 0, o que corresponde a abrir uma facilidade fechada ou fechar uma facilidade aberta). Se $\beta < \alpha$ ir para o passo 1.5.

Passo 1.5: Se $k = n$ pare. Caso contrário, fazer $k = k+1$ e ir para o passo 1.3.

Passo 1.6: Formar a matriz $J_{hn}$ com as facilidades abertas em cada solução vizinha: se $R_k = 1$, então $J_{hn} = k$;

Avaliar *totcost(y).* Se $totcost(y) < V_{min}$, então $V_{min} = totcost(y)$, $y_{min} = y$.

*Passo 2*. Verificar o status da lista tabu.

Verificar se a solução $y_{min}$, encontrada no passo 1, está ou não na lista tabu, através dos seguintes passos:

Passo 2.1: $l = 1$.

Passo 2.2: Se $y_{min} \notin TL$ *ou* $y_{min} \in TL$ *e* $V_{min} < BV$, então ir para o passo 3. Do contrário, fazer $l = l + 1$. Fazer a l-ésima melhor solução ser $y_{min}$ (esta solução é a melhor solução dentre todas as soluções geradas na iteração excluindo aquelas já consideradas neste passo) e repita este passo.

*Passo 3*. Atualizar a solução corrente.

Trocar a solução corrente pela nova solução $y_{min}$ e o valor da função-objetivo (BV) por $V_{min}$. Incluir $y_{min}$ na lista tabu. fazer $y_{corrente} = y_{min}$. Se $V_{min} \geq BV$, então ir para o passo 4. Do contrário, fazer $BV = V_{min}$, $k = 0$, e ir para o próximo passo.

*Passo 4*. Verificar o critério de parada.

Se o critério de parada for satisfeito pare. Se $k = MAXSEMMELHORA$ *ou* $K = ITERMAX$, ir para o passo 5; caso contrário, fazer $k = k + 1$ e voltar para o passo 1.

*Passo 5*. Parar e reportar os resultados.

Pare. Reportar $y_{melhor}$ como a melhor solução encontrada, com custo total igual a $BV$.

## 4 RESULTADOS E DISCUSSÕES

O algoritmo de busca tabu implementado baseia-se no trabalho de Al-Sultan e Al-Fawzan (1999). No algoritmo implementado, o processo parte de uma solução inicial e busca por soluções melhores nas vizinhanças das melhores soluções obtidas. Neste processo, para diversificar as soluções, as soluções encontradas mais recentemente são armazenadas na lista tabu.

O algoritmo possui duas partes. A primeira parte consiste em determinar uma boa solução inicial para o problema. Isto é feito em duas fases: inicial e de refinamento. Na fase inicial, determina-se a facilidade que supre cada demanda ao menor custo e calcula-se o custo total para esta configuração. Têm-se, portanto, o estabelecimento de certo número de facilidades que atenderá a todas as demandas. Na fase de refinamento, cada facilidade é examinada quanto à possibilidade de fechamento, se isto significar uma diminuição do custo total. Isto irá ocorrer caso a economia com o fechamento da facilidade seja maior que o custo extra da realocação dos clientes inicialmente atendidas por esta facilidade.

Na segunda parte do algoritmo, parte-se da solução obtida na primeira parte e inicia-se efetivamente o processo de busca tabu, no qual novas soluções vizinhas são encontradas e comparadas com a melhor solução vigente. O algoritmo termina quando uma condição de parada é alcançada.

O algoritmo implementado apresenta algumas alterações em relação ao algoritmo apresentado inicialmente, no capítulo sobre a metodologia. Apresenta-se, a seguir, o algoritmo de busca tabu implementado.

*Parâmetros:*

- ITERMAX: Número máximo de iterações;
- MSM: Número máximo de iterações sem melhora;
- NSV: Número de soluções vizinhas;
- TLT: Tamanho da Lista Tabu;

- α: Probabilidade de uma facilidade manter seu *status* atual.

**Parte 1: Construção de solução inicial**

*Fase Inicial:*

*Passo 1.* $V = \varnothing$;

Para $j = 1, ..., n$, fazer:

$c_{ij} = \min\{c_{kj}, k = 1, ..., m\}$;

$V = V \cup \{ (i, j) \}$;

*Passo 2.* $I = \{ i : (i, j) \in V \}$;

*Passo 3.* $LS = \sum_{(i,j)\in V} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i$.

*Fase de Refinamento:*

*Passo 1*: $k = 1$;

*Passo 2*: Seja $i$ o $k$-ésimo elemento em $I$;

*Passo 3.* $J = \{ j \mid (i, j) \in V \}$;

*Passo 4.* Para cada $j \in J$, fazer:

$d_j = c_{mj} - c_{ij}$, onde $c_{mj} = \min \{ c_{kj}, k \in I, k \neq i \}$;

*Passo 5.* $\delta = \sum_{j \in J} d_j - f_i$;

*Passo 6.* Se $\delta < 0$ então:

$I = I - \{ i \}$;

$LS = LS - | \delta |$;

Se $|I| = 1$, parar; Caso contrário, voltar ao Passo 1.

*Passo 7.* Se $k = |I|$, parar; Caso contrário, fazer $k = k + 1$ e voltar ao Passo 2.

**Parte 2: Busca tabu**

Seja $I$ o conjunto de facilidades abertas resultante da Parte 1 do algoritmo. Seja $Y$ um vetor binário tal que $y_i = 1$ se $i \in I$ e $y_i = 0$, caso contrário. Seja custo($Y$) o custo total da solução $Y$.

*Passo 1. Inicialização*

$Y_{atual} = Y$; $Y_{melhor} = Y$; $MV$ = custo($Y$); $LT = \varnothing$; $k = 0$; $iter = 0$.

*Passo 2. Gerar soluções vizinhas*

$k = k + 1$;

$Y = Y_{atual}$;

$S = \varnothing$;

Para $h$ = 1 até NSV fazer:

Para $i$ = 1 até $n$, fazer:

$\beta$ = número aleatório entre 0 e 100;

Se $\beta > \alpha$, fazer $w_i = 1 - y_i$;

S = S $\cup$ { $W$ };

Organizar as soluções $W_h$ ($h$ = 1, ..., NSV) de $S$ em ordem crescente de custo($W_h$).

*Passo 3. Verificar a lista tabu*

Para $h$ = 1 até NSV, fazer:

Seja $Y_{min}$ a $h$-ésima solução de $S$;

Se ($Y_{min} \notin LT$) ou ($Y_{min} \in LT$ e custo($Y_{min}$) < $MV$), então fazer:

Se ($Y_{min} \notin LT$), então fazer:

$LT = LT \cup \{ Y_{min} \}$;

Se | $LT$ | > TLT então remover de $LT$ a solução mais antiga;

$Y_{atual} = Y_{min}$;

Se custo($Y_{min}$) < $MV$, então fazer:

$MV$ = custo($Y_{min}$);

$Y_{melhor} = Y_{min}$;

$k = 0$.

*Passo 4. Verificar o critério de parada*

Se ($k$ = MSM) ou ($iter$ = ITERMAX), parar.

Caso contrário, fazer $iter = iter + 1$ e voltar para o Passo 2.

A partir do algoritmo apresentado acima, foi implementado o programa BUSCA_TABU em Linguagem C. Este programa pode ser encontrado no Anexo A.

## 4.1 Resultados Computacionais

O algoritmo de busca tabu descrito na seção anterior foi escrito na linguagem C. Utilizou-se o software Dev-C++ para a escrita e teste do algoritmo.

Os testes computacionais foram realizados em um microcomputador com processador Intel Celeron 2,13GHz, 2 GB RAM. Para os testes foram utilizados 12 exemplares do problema disponíveis na OR Library (Beasley, 1990) cujas soluções ótimas são conhecidas. Para os testes realizados foram considerados como padrões os seguintes valores dos parâmetros:

- ITERMAX = 1000000;
- MSM = 50;
- NSV = 200;
- TLT = 15;
- $\alpha = 94$.

Observa-se que ao se estabelecer um valor muito alto para ITERMAX, este parâmetro não influenciará no resultado encontrado, sendo necessário apenas em caso de ciclagem do algoritmo.

Para cada exemplar, o algoritmo de busca tabu foi executado dez vezes. Nas tabelas que seguem, os valores da solução obtida e os tempos computacionais são as médias dos dez resultados encontrados.

## 4.2 Problemas Pequenos

A Tabela 1 mostra os resultados obtidos. Esta tabela possui como linha de cabeçalho os seguintes termos:

- ARQUIVO - Nome do Arquivo de Dados;
- M - Número de Clientes;
- N - Número de Facilidades;
- OPT - Valor da Solução Ótima Conhecida;
- SOL - Valor da Solução Obtida;
- GAP - Valor percentual da diferença entre SOL e OPT, calculado como: 100*(SOL-OPT)/OPT
- TC NBH: Tempo Computacional (em segundos) para a execução da primeira parte do algoritmo (NBH);
- TC BT: Tempo Computacional (em segundos) para execução de todo o algoritmo;

Tabela 1 - Resultados computacionais para problemas obtidos na OR Library

| Arquivo | m | n | OPT | SOL | GAP | TC NBH | TC BT |
|---|---|---|---|---|---|---|---|
| Cap71 | 16 | 50 | 932615,750 | 932615,813 | 0,00001 | 0 | 0,07 |
| Cap72 | 16 | 50 | 977799,400 | 977799,438 | 0,00000 | 0 | 0,07 |
| Cap73 | 16 | 50 | 1010641,450 | 1012477,063 | 0,18163 | 0 | 0,07 |
| Cap74 | 16 | 50 | 1034976,975 | 1034977,063 | 0,00001 | 0 | 0,05 |
| Cap101 | 25 | 50 | 796648,437 | 797508,750 | 0,10799 | 0 | 0,1 |
| Cap102 | 25 | 50 | 854704,200 | 854704,188 | 0,00000 | 0 | 0,08 |
| Cap103 | 25 | 50 | 893782,112 | 893782,125 | 0,00000 | 0 | 0,09 |
| Cap104 | 25 | 50 | 928941,750 | 928941,813 | 0,00001 | 0 | 0,05 |
| Cap131 | 50 | 50 | 793439,562 | 803489,375 | 1,26661 | 0 | 0,19 |
| Cap132 | 50 | 50 | 851495,325 | 853941,688 | 0,28730 | 0 | 0,15 |
| Cap133 | 50 | 50 | 893076,712 | 899357,375 | 0,70326 | 0,016 | 0,08 |
| Cap134 | 50 | 50 | 928941,750 | 928941,813 | 0,00001 | 0,016 | 0,08 |

Por meio dos resultados obtidos, indicados na Tabela 1, pode-se afirmar que o algoritmo é eficaz para a resolução dos problemas testados uma vez que, para a maioria dos exemplares, o valor da solução obtida não apresenta diferença significativa para o valor da solução ótima conhecida.

Observa-se, porém, que a diferença entra o resultado ótimo e o obtido aumenta para problemas maiores (com m e n iguais a 50). Isto pode ser um indicativo de que o algoritmo implantado não é recomendado para problemas maiores.

Este fato também pode ser resultado dos valores dos parâmetros utilizados. É possível que ao aumentar o número de interações, a diferença entre o valor obtido e o ótimo diminuirá.

A fim de comparar a qualidade do algoritmo desenvolvido neste trabalho, a Tabela 2 apresenta uma comparação dos resultados obtidos pelo algoritmo BT desenvolvido neste trabalho, com os resultados obtidos pelo algoritmo TS de Sun (2006).

Nesta tabela (e nas próximas), na coluna Desvio, utilizou-se o símbolo "-" para indicar valores nulos. Observa-se que para a maioria dos problemas os desvios e o tempo computacional foram muito pequenos.

Embora os tempos computacionais não possam ser diretamente comparados (o algoritmo TS foi codificado em Fortran e executado em um computador SUN Enterprise 3000), observa-se que, em termos de qualidade da solução, o algoritmo BT é competitivo para a maioria dos exemplares do problema.

Tabela 2 - Comparação com os resultados de Sun (2006)

| | **BT** | | **TS** | |
|---|---|---|---|---|
| **Problema** | **Desvio** | **Tempo** | **Desvio** | **Tempo** |
| Cap71 | - | 0,07 | - | 0,05 |
| Cap72 | - | 0,07 | - | 0,05 |
| Cap73 | 0,07 | 0,07 | - | 0,05 |
| Cap74 | - | 0,05 | - | 0,04 |
| Cap101 | - | 0,1 | - | 0,07 |
| Cap102 | - | 0,08 | - | 0,06 |
| Cap103 | 0,01 | 0,09 | - | 0,05 |
| Cap104 | - | 0,05 | - | 0,06 |
| Cap131 | 1,03 | 0,19 | - | 0,11 |
| Cap132 | 0,86 | 0,15 | - | 0,12 |
| Cap133 | 0,61 | 0,08 | - | 0,13 |
| Cap134 | - | 0,08 | - | 0,13 |

Esta comparação de resultados mostra que a qualidade dos resultados do algoritmo BT diminui quando aplicado a problemas grandes. Os desvios dos problemas Cap131, Cap132 e Cap133 são os mais altos entre os problemas testados e contrastam com os desvios nulos do algoritmo de TS de Sun (2006).

No entanto, o algoritmo pode ser otimizado em relação a sua eficiência, tentando-se minimizar o tempo de execução do algoritmo e diminuindo os desvios para problemas maiores. Essa otimização pode ser obtida encontrando-se um conjunto de valores favoráveis para os parâmetros do algoritmo: NBHSIZE; MAXSEMMELHORA; TL e α.

## 4.3 Estudo Dos Parâmetros

Um estudo foi realizado sobre a influência dos parâmetros na qualidade da solução (desvios) e no tempo computacional. Neste estudo, em primeiro lugar, considerou-se a alteração do tamanho da lista tabu (parâmetro TLT) para valores limites. A Tabela 3 apresenta os resultados obtidos.

Foram alterados também os valores de α e MSM, isoladamente, mantendo-se os demais parâmetros com os valores padrões. A Tabela 4 apresenta os resultados deste estudo.

E por último, considerou-se a alteração do parâmetro NSV (número de soluções vizinhas a cada iteração). Pelos resultados mostrados na Tabela 5, observa-se que este parâmetro possui grande influência na solução obtida e no tempo computacional.

### 4.3.1 Lista Tabu

Pode-se afirmar que a lista tabu é um parâmetro de grande importância já que a heurística empregada neste trabalho tem como base as soluções tabus. No presente algoritmo, esta lista possui um tamanho fixo, e uma vez preenchida, as novas soluções correntes encontradas serão colocadas na lista tabu no lugar da solução tabu mais antiga e, portanto esta última se tornará uma solução válida novamente.

Sendo assim, pode significar uma melhoria no algoritmo a realocação da lista tabu a cada solução corrente encontrada. Dessa forma, nenhuma solução tabu poderá voltar a ser solução corrente.

Para testar a influência da Lista Tabu, um teste foi realizado alterando-se seu valor para o mínimo (um) e também para um valor grande (mil).

Tabela 3 - Resultados do estudo sobre a influência do parâmetro TLT

| | Padrão | | TLT = 1 | | TLT = 1000 | |
|---|---|---|---|---|---|---|
| Problema | Tempo | Desvio | Tempo | Desvio | Tempo | Desvio |
| Cap71 | 0,07 | - | 0,14 | - | 0,07 | - |
| Cap72 | 0,07 | - | 0,12 | - | 0,07 | - |
| Cap73 | 0,07 | 0,07 | 0,08 | 0,18 | 0,05 | 0,18 |
| Cap74 | 0,05 | - | 0,08 | - | 0,05 | - |
| Cap101 | 0,1 | - | 0,14 | 0,11 | 0,13 | - |
| Cap102 | 0,08 | - | 0,11 | - | 0,08 | - |
| Cap103 | 0,09 | 0,01 | 0,12 | 0,02 | 0,08 | - |
| Cap104 | 0,05 | - | 0,08 | - | 0,05 | - |
| Cap131 | 0,19 | 1,03 | 0,27 | 0,58 | 0,18 | 0,8 |
| Cap132 | 0,15 | 0,86 | 0,25 | 0,21 | 0,1 | 0,61 |
| Cap133 | 0,08 | 0,61 | 0,13 | 0,72 | 0,08 | 0,92 |
| Cap134 | 0,08 | - | 0,11 | 0,06 | 0,08 | 0,06 |
| **Média** | **0,09** | **0,21** | **0,14** | **0,16** | **0,08** | **0,22** |

Pelos resultados mostrados na Tabela 3, observa-se que listas tabus muito pequenas podem implicar aumento do tempo computacional sem a contrapartida de aumento na qualidade da solução. Já para lista tabu muito grande, o resultado e tempo computacional se mantêm constantes em relação ao Padrão.

### 4.3.2 Interações Máxima Sem Melhora e alfa

Tabela 4 - Resultados do estudo sobre a influência dos parâmetros α e MSM

| | Padrão | | α = 92 | | α = 96 | | MSM = 35 | |
|---|---|---|---|---|---|---|---|---|
| Problema | Tempo | Desvio | Tempo | Desvio | Tempo | Desvio | Tempo | Desvio |
| Cap71 | 0,07 | - | 0,07 | - | 0,08 | - | 0,07 | - |
| Cap72 | 0,07 | - | 0,06 | - | 0,05 | - | 0,06 | - |
| Cap73 | 0,07 | 0,07 | 0,05 | 0,18 | 0,05 | 0,18 | 0,06 | 0,18 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Cap74 | 0,05 | - | 0,05 | 0,26 | 0,05 | - | 0,05 | - |
| Cap101 | 0,1 | - | 0,1 | 0,03 | 0,08 | 0,11 | 0,1 | 0,09 |
| Cap102 | 0,08 | - | 0,07 | - | 0,08 | - | 0,08 | - |
| Cap103 | 0,09 | 0,01 | 0,09 | - | 0,08 | 0,03 | 0,08 | - |
| Cap104 | 0,05 | - | 0,06 | - | 0,06 | - | 0,05 | - |
| Cap131 | 0,19 | 1,03 | 0,17 | 1,02 | 0,17 | 0,09 | 0,15 | 0,63 |
| Cap132 | 0,15 | 0,86 | 0,1 | 0,97 | 0,13 | 0,61 | 0,1 | 0,88 |
| Cap133 | 0,08 | 0,61 | 0,09 | 0,94 | 0,08 | 0,86 | 0,08 | 0,86 |
| Cap134 | 0,08 | - | 0,08 | 0,06 | 0,06 | 0,06 | 0,08 | 0,06 |
| **Média** | **0,09** | **0,21** | **0,08** | **0,29** | **0,08** | **0,16** | **0,08** | **0,22** |

Analisando a Tabela 4, observa-se que a alteração efetuada no parâmetro MSM não resultou em melhora no tempo ou desvio. Isto mostra que são necessárias muitas iterações até que um resultado muito melhor seja encontrado. Estas iterações demandam tempo e, portanto, deve ser encontrada a melhor relação entre tempo computacional e desvio.

Já para alfa, observa-se que o aumento do seu valor, diminuiu consideravelmente o desvio, sem aumentar o tempo.

### 4.3.3 Número de Soluções Vizinhas

Tabela 5 - Resultados do estudo sobre a influência do parâmetro NSV

| | Padrão | | NSV = 100 | | NSV = 300 | |
|---|---|---|---|---|---|---|
| Problema | Tempo | Desvio | Tempo | Desvio | Tempo | Desvio |
| Cap71 | 0,07 | - | 0,05 | - | 0,22 | - |
| Cap72 | 0,07 | - | 0,05 | - | 0,15 | - |
| Cap73 | 0,07 | 0,07 | 0,04 | 0,18 | 0,18 | 0,11 |
| Cap74 | 0,05 | - | 0,05 | - | 0,12 | - |
| Cap101 | 0,1 | - | 0,08 | 0,11 | 0,23 | 0,05 |
| Cap102 | 0,08 | - | 0,05 | - | 0,19 | - |
| Cap103 | 0,09 | 0,01 | 0,07 | - | 0,18 | 0,06 |
| Cap104 | 0,05 | - | 0,04 | - | 0,14 | - |
| Cap131 | 0,19 | 1,03 | 0,08 | 1,26 | 0,49 | 0,9 |
| Cap132 | 0,15 | 0,86 | 0,07 | 0,86 | 0,27 | 0,28 |
| Cap133 | 0,08 | 0,61 | 0,06 | 0,83 | 0,17 | 0,66 |
| Cap134 | 0,08 | - | 0,05 | 0,06 | 0,17 | 0,06 |
| **Média** | **0,09** | **0,21** | **0,06** | **0,27** | **0,21** | **0,18** |

Analisando a Tabela 5, observa-se que tempo computacional e o desvio possui grande influência em relação à quantidade de soluções vizinhas. Com um número menor de soluções (NSV = 100) o tempo computacional foi baixo, mas o desvio foi mais alto quando comparado com o padrão.

Já para uma quantidade maior de soluções vizinhas (NSV = 300), o tempo computacional apresentou um grande aumento, quando comparado com o padrão, e uma diminuição no desvio.

Tabela 6 - Resumo dos resultados para vários parâmetros

| | Parâmetros Iniciais | | TL = 1000 | | α = 92 | | α = 96 | | MSM = 35 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Problema** | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| Cap71 | 0,07 | 0,00% | 0,072 | 0,00% | 0,067 | 0,00% | 0,078 | 0,00% | 0,074 | 0,00% |
| Cap72 | 0,072 | 0,00% | 0,067 | 0,00% | 0,062 | 0,00% | 0,047 | 0,00% | 0,064 | 0,00% |
| Cap73 | 0,073 | 0,07% | 0,05 | 0,18% | 0,052 | 0,18% | 0,047 | 0,18% | 0,061 | 0,18% |
| Cap74 | 0,051 | 0,00% | 0,047 | 0,00% | 0,047 | 0,26% | 0,046 | 0,00% | 0,05 | 0,00% |
| Cap101 | 0,103 | 0,00% | 0,125 | 0,00% | 0,101 | 0,03% | 0,078 | 0,11% | 0,104 | 0,09% |
| Cap102 | 0,078 | 0,00% | 0,076 | 0,00% | 0,073 | 0,00% | 0,078 | 0,00% | 0,075 | 0,00% |
| Cap103 | 0,094 | 0,01% | 0,08 | 0,00% | 0,094 | 0,00% | 0,078 | 0,03% | 0,084 | 0,00% |
| Cap104 | 0,053 | 0,00% | 0,052 | 0,00% | 0,058 | 0,00% | 0,063 | 0,00% | 0,053 | 0,00% |
| Cap131 | 0,186 | 1,03% | 0,175 | 0,80% | 0,167 | 1,02% | 0,171 | 0,09% | 0,147 | 0,63% |
| Cap132 | 0,148 | 0,86% | 0,097 | 0,61% | 0,095 | 0,97% | 0,125 | 0,61% | 0,102 | 0,88% |
| Cap133 | 0,081 | 0,61% | 0,08 | 0,92% | 0,089 | 0,94% | 0,078 | 0,86% | 0,083 | 0,86% |
| Cap134 | 0,078 | 0,00% | 0,08 | 0,06% | 0,083 | 0,06% | 0,062 | 0,06% | 0,08 | 0,06% |
| **Média** | **0,091** | **0,21%** | **0,083** | **0,22%** | **0,082** | **0,29%** | **0,079** | **0,16%** | **0,081** | **0,22%** |

Pelos resultados mostrados na Tabela 4 pode-se afirmar que uma boa configuração para os parâmetros é encontrada mantendo-se os parâmetros padrões, exceto para α, que deve ser aumentado para 96.

Para o parâmetro NSV, pelos resultados mostrados na Tabela 5, observa-se que grandes vizinhanças resultam em um tempo computacional alto sem melhora considerável na solução encontrada. No entanto, como o tempo de execução para os problemas pequenos não são grandes, pode representar uma melhora na qualidade do resultado, o aumento do número de soluções vizinhas.

O estudo dos parâmetros mostrou que poucas soluções sendo mantidas como tabus já levam a resultados de boa qualidade. Também foi possível concluir que NSV igual a duzentos pode ser considerado como um bom compromisso entre a qualidade da solução e o tempo computacional.

O estudo mostrou também que o valor da probabilidade de uma facilidade manter seu *status* (facilidade aberta ou fechada) deve ser alta ($\alpha = 96$) para se obter bons resultados.

## 4.4 Sistemas de Informação Geográfica

Modelos de localização de facilidades têm sido propostos como ferramentas de auxílio à decisão, principalmente quando é possível usar um Sistema de Informação Geográfica (SIG) na coleta e análise dos dados dos problemas (Lorena *et al*., 2001).

O algoritmo implementado neste trabalho encontra-se integrado ao sistema PCLOG (Trofino e Senne, 2008), uma plataforma computacional que utiliza recursos de SIG disponíveis na biblioteca de classes MapObjects (ESRI, 2003) para solução de problemas logísticos.

Para ilustrar como as soluções são visualizadas na plataforma PCLOG, considerou-se uma rede com duzentos nós, dos quais 10 foram estabelecidos como locais potenciais para a abertura de facilidades e 190 como nós de demanda.

A solução obtida pode ser observada na Figura 1. Nesta figura, os quadrados representam as facilidades e os círculos, os nós de demanda, que são ligadas por segmentos de reta às facilidades que os atendem.

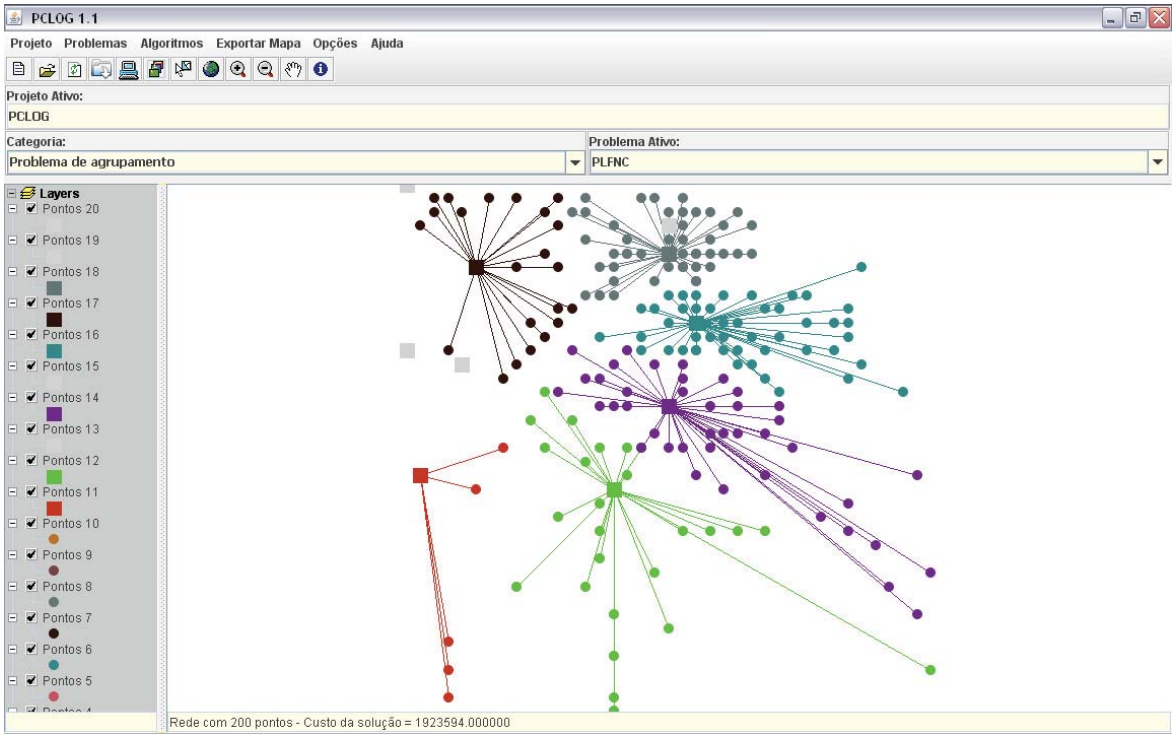



Figura 1. Solução encontrada pelo algoritmo BT mostrado no PCLOG.

# 5 PROBLEMAS GRANDES

Para os problemas pequenos, o algoritmo se mostrou eficaz. Abaixo seguem resultados dos testes realizados para problemas maiores que possuem cem facilidades e mil demandas (100x1000).

Na tabela 7, são comparados os resultados obtidos pelo algoritmo desenvolvido neste trabalho com o algoritmo BT, desenvolvido por Sun (2006). Observa-se que, embora com um tempo de execução muito menor, o algoritmo BT não conseguiu encontrar a solução ótima conhecida, utilizando-se os parâmetros padrões.

Tabela 7. Comparação entre o algoritmo BT e TS

| | **BT** | | **TS** | |
|---|---|---|---|---|
| | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| CAPa | 2,95 | 18,63% | 13,02 | - |
| CAPb | 2,99 | 9,87% | 10,98 | - |
| CAPc | 3,01 | 7,58% | 9,23 | - |

A fim de se obter uma configuração de parâmetros capaz de encontrar a solução ótima para problemas maiores, realizou-se um estudo sobre as influências de cada parâmetro no tempo e no desvio. Na Tabela 8, cada parâmetro tem o seu valor aumentado, e os resultados são comparados com o Padrão.

Tabela 8 - Comparação das influências dos parâmetros

| | **Padrão** | | **a = 96** | | **NSV = 400** | | **TLT = 100** | | **MSM = 150** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| CAPa | 2,95 | 18,63% | 2,13 | 16,95% | 5,58 | 17,73% | 2,70 | 18,78% | 8,52 | 18,70% |
| CAPb | 2,99 | 13,05% | 2,21 | 10,59% | 5,21 | 11,50% | 2,96 | 12,46% | 8,36 | 12,43% |
| CAPc | 3,01 | 11,30% | 2,22 | 8,60% | 5,08 | 9,78% | 2,81 | 10,59% | 8,29 | 10,18% |

O aumento do valor de Alfa diminuiu o tempo de execução e o desvio entre o resultado obtido e a solução ótima conhecida, mesmo comportamento observado para os problemas pequenos.

Com uma geração de mais soluções vizinha a cada iteração (NSV = 400), diminuiu-se o desvio, em contrapartida, aumentou o tempo de execução do algoritmo. Já o aumento da lista tabu acarretou apenas uma pequena diminuição no tempo, enquanto o desvio se manteve constante, como também se manteve constante para o aumento do máximo de soluções sem melhora, só que neste caso, o tempo de execução aumentou em grande quantidade.

Tabela 9 - Análise do parâmetro alfa

| | **a=94 (Padrão)** | | **a = 96** | | **a = 98** | |
|---|---|---|---|---|---|---|
| | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| CAPa | 2,95 | 18,63% | 2,13 | 16,95% | 1,78 | 14,17% |
| CAPb | 2,99 | 13,05% | 2,21 | 10,59% | 2,89 | 6,84% |
| CAPc | 3,01 | 11,30% | 2,22 | 8,60% | 3,02 | 6,70% |

Em relação ao alfa, como pode ser observado na Tabela 9, os testes mostraram que quanto menor a probabilidade de uma facilidade mudar de status, menor é o desvio e o tempo computacional, sendo que a diminuição do desvio é considerável.

Sabendo-se que a qualidade do resultado aumenta com o aumento do alfa, os testes que seguem foram realizados com alfa igual a 98.

Tabela 10 - Análise do parâmetro NSV

| | **Padrão** | | **NSV = 400** | | **NSV= 500** | |
|---|---|---|---|---|---|---|
| | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| CAPa | 5,58 | 17,73% | 3,27 | 14,09% | 6,76 | 13,98% |
| CAPb | 5,21 | 11,50% | 6,22 | 4,89% | 10,23 | 4,98% |

| | | | | | | |
|---|---|---|---|---|---|---|
| CAPc | 5,08 | 9,78% | 6,61 | 6,79% | 11,80 | 5,65% |

Analisando a Tabela 10 acima, nota-se que quanto maior o número de soluções vizinhas geradas a cada iteração, menores os desvios em relação à solução ótima e maior também o tempo computacional, pois a cada iteração mais soluções são geradas e testadas.

Tabela 11 - Análise do tamanho da Lista Tabu

| | **Padrão** | | **TLT = 100** | | **TLT = 500** | | **TLT = 2000** | | **TLT = 1** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| CAPa | 2,95 | 18,63% | 1,65 | 13,47% | 1,85 | 14,10% | 1,68 | 15,56% | 1,64 | 15,66% |
| CAPb | 2,99 | 13,05% | 2,49 | 6,56% | 3,01 | 6,11% | 4,22 | 5,21% | 3,14 | 5,82% |
| CAPc | 3,01 | 11,30% | 3,55 | 5,80% | 3,69 | 7,03% | 3,45 | 6,45% | 2,97 | 6,71% |

Já em relação ao tamanho da Lista Tabu, observa-se que tanto para o tempo, quanto para o desvio, a qualidade do resultado depende do problema, não estando esta proporcionalmente relacionada ao tamanho da lista. No entanto, comparando o resultado com a Tabela 9, onde o alfa também é 98 e TLT igual a 15, observa-se uma melhora nos resultados para maiores valores da lista tabu.

Como a solução ótima não foi encontrada nos testes mostrados acima, novos testes alterando-se simultaneamente mais de um parâmetro foram realizados. Abaixo seguem os resultados:

Tabela 12 - Análise dos parâmetros variados

| | **a = 99; NSV = 1000; MSM = 200, TLT = 1000** | | **a = 99; NSV = 500; MSM = 100; TLT = 100** | | **a = 98; NSV = 500; MSM = 100; TLT = 100** | |
|---|---|---|---|---|---|---|
| | **Tempo** | **Desvio** | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| CAPa | 35,13 | 9,83% | 10,72 | 12,39% | 10,72 | 14,39% |

| | | | | | | |
|---|---|---|---|---|---|---|
| CAPb | 72,14 | 2,83% | 10,75 | 4,25% | 10,75 | 3,66% |
| CAPc | 74,97 | 3,69% | 20,32 | 3,83% | 20,32 | 4,98% |

Na tabela acima, observa-se que aumentando valores de quase todos os parâmetros o desvio diminuem consideravelmente. No entanto, o tempo de execução também aumenta muito e, portanto, a qualidade do algoritmo diminui, pois os desvios continuam grandes e o tempo computacional é muito alto.

Para todos os testes realizados acima, foram considerados valores de alfa acima de 94. Abaixo, segue tabela no qual se avaliou a influência de alfa menor (igual a 85):

Tabela 13 - Resultados comparativos para alfa igual a 85

| | **a = 85; NSV = 400, TLT = 200** | | **a = 99; NSV = 400; TLT = 200** | |
|---|---|---|---|---|
| | **Tempo** | **Desvio** | **Tempo** | **Desvio** |
| CAPa | 7,94 | 25,06% | 3,63 | 11,97% |
| CAPb | 7,68 | 13,94% | 6,10 | 3,70% |
| CAPc | 7,80 | 13,40% | 6,41 | 5,67% |

Observa-se que para alfa igual a 85, mesmo considerando valores mais altos de soluções vizinhas e da lista tabu, o desvio é muito grande, o que indica que alfa tem grande interferência na qualidade do resultado. Conforme comentado anteriormente, com maiores valores de alfa a solução se torna mais aleatória, sendo difícil encontrar a solução ótima.

# 6 CONCLUSÃO

O algoritmo de Busca Tabu implementado neste trabalho tem como objetivo a solução de problemas de localização não-capacitados. Os resultados obtidos nos testes computacionais são satisfatórios, sendo que as soluções obtidas pelo algoritmo são próximas das soluções ótimas conhecidas, principalmente para problemas menores.

Já para problemas maiores o algoritmo apresentou soluções aceitáveis, com desvios de cerca de 5%, mas com um tempo computacional alto. Desta forma, o algoritmo deve sofrer alterações a fim de se tornar mais eficiente para problemas grandes.

Em relação aos parâmetros, observa-se que o alfa e o número de soluções vizinhas (NSV) geradas a cada iteração são aqueles que mais influenciam o resultado, sendo que o alfa maior garante desvios e tempo computacional menor. Já com o aumento do NSV, diminui-se o desvio, mas aumenta o tempo computacional.

Em relação à Lista Tabu, observa-se que o seu tamanho não interfere diretamente nos resultados, sendo que sua principal função é evitar ótimos locais.

Observa-se também, que independente do problema, os tempos de execução do algoritmo são pequenos, sendo o maior tempo obtido, pouco maior que um minuto. Desta forma, mesmo que a diminuição do desvio seja pequena quando comparado ao aumento do tempo computacional, deve-se optar pela melhora da solução, pois o tempo total continuará pequeno.

No entanto, a melhor configuração de parâmetros deve ser obtida de acordo com o que se deseja: menor tempo computacional ou melhor solução.

Apesar dos vários testes realizados neste trabalho, não se pode afirmar com certeza a influência de cada parâmetro, ou interações entre parâmetros, na qualidade dos resultados. Desta forma, um possível desdobramento deste trabalho seria o estudo dos parâmetros utilizando as técnicas de planejamento de experimentos.

## BIBLIOGRAFIA


AL-SULTAN, K.S.; AL-FAWZAN, M.A. A Tabu Search Approach to the Uncapacited Facility Location Problem, **Annals of Operations Research**, v. 86, p. 91-103, 1999.

BEASLEY, J.E. OR-Library: distributing test problems by electronic mail, **Journal of the Operational Research Society**, 41 (11), 1069-1072, 1990.

BEASLEY, J.E**.** Lagrangean heuristics for location problems. **European Journal of Operational Research**, 65, 383-399, 1993.

CARELLO, G.; DELLA CROCE, F.; GHIRARDI, M.; TADEI, R. Solving the Hub Location Problem in Telecommunication Network Design: A Local Search Approach, **Networks**, v. 44, n. 4, p. 94-105, 2004.

DREZNER, Z. (ed.) **Facility Location: A Survey of Applications and Methods**, NewYork: Springer-Verlag, 1995.

DUCATI, E.A., **Busca tabu aplicada ao problema de localização de facilidades com restrições de capacidade**, Dissertação de Mestrado em Engenharia Elétrica, UNICAMP, Campinas, SP, 2003. Disponível wm<http://libdigi.unicamp.br/document/>

ESRI *MapObjects Java Edition Developer's Guide*, Environmental Systems Research Institute, Inc., Redlands, CA, 2003.

GAREY, M.R.; JOHNSON, D.S. **Computers and Intractability: A Guide to the Theory of NP-Completeness.** San Francisco: W.H. Freeman, 1979.

GLOVER, F.; LAGUNA, M. *Tabu Search.* Massachusetts: Kluwer Academic Publishers, 1997.

KORKEL, M. On the Exact Solution of Large-scale Simple Plant Location Problems. **European Journal of Operational Research**, v. 39, p. 157-173, 1989.

KRARUP, J.; PRUZAN, P.M. The Simple Plant Location Problem: Survey and Synthesis. **European Journal of Operational Research**, v. 12, p. 36-81, 1983.

LORENA, L.A.N.; SENNE, E.L.F.; PAIVA, J.A.C.; PEREIRA, M.A. Integração de Modelos de Localização a Sistemas de Informações Geográficas. **Gestão e Produção** *(UFSCar)*, v.8, p.180-195, 2001.

MICHEL, L.; VAN HENTENRYCK, P. A Simple Tabu Search for Warehouse Location. **European Journal of Operational Research**, v. 157, p. 576-591, 2004.

MLADENOVIC, N.; LABBÉ, M.; HANSEN, P. Solving the p-Center Problem with Tabu Search and Variable Neighborhood Search, **Networks**, v. 42, n. 1, p. 48–64, 2003

SUN, M. Solving the Uncapacitated Facility Location Problem Using Tabu Search, **Computers & Operations Research**, 33, 2563-2589, 2006.

TOREGAS, C.; REVELLE, C.; BERGUM, L.; The Location of emergency service facilities, **Operations Research**, 19, 1363-1373, 1971.

TROFINO, S.H.; SENNE, E.L.F**.** *PCLOG:* **Plataforma Computacional para Solução de Problemas Logísticos**, Trabalho de Graduação em Engenharia Elétrica, FEG/UNESP, Guaratinguetá, SP, 2008.

## ANEXO A - Programa escrito na linguagem C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define INFINITO      99999999
#define NBHSIZE       200      // numero de solucoes vizinhas
#define ITERMAX       1000000
#define MAXSEMMELHORA 50       // iteracoes sem melhora
#define TL            1        // tamanho da lista tabu

float BV;            // melhor valor já encontrado
int h;               // h-esima solução vizinha
int m;               // numero de demandas.
int n;               // numero total de facilidades.
int *I;              // facilidades abertas (1) e fechadas (0)
int *BS;             // facilidades abertas na melhor solução já encontrada
int *mstar;          // número de facilidades abertas em cada solução vizinha
float **c;           // custo de distribuição c[n][m]
float *F;            // custo de abertura da facilidade
float best;          // melhor solução conhecida
int **J;             // facilidades abertas nas h-esimas solução vizinha
int **TB;            // lista tabu (TL x n)
float *totcost;      // custo total de cada solução vizinha
int *IC;             // vetor ordenado dos custos totais de cada colução vizinha
int docum;           // usado para depurar o programa (docum = 1)
double totTimeUFL;   // tempo de CPU para executar todo programa
clock_t iTime,fTime;
FILE *arq,*sai,*dat;

double cpuTime(clock_t ITime, clock_t FTime)
{
 double Elapsed;
 Elapsed = (double)(FTime - ITime)/CLK_TCK;
 return(Elapsed);
}

void solin()
{
 int i, j, k, p, q;
 int tot, star, refinando;
 float menor, ub, csalvo, delta;
 int *v;     // facilidade que entrega o menor custo a demanda (índice)
 int *U;     // facildades abertas
 int *P;     // demandas atendidas pelas facilidade (índice)
 float *d;   // diferença de custos

 v = (int *)calloc(m, sizeof(int));
 U = (int *)calloc(n, sizeof(int));
 P = (int *)calloc(m, sizeof(int));
 d = (float *)calloc(m, sizeof(float));

 // FASE INICIAL
 // 1- Encontrar a facilidade que supre cada demanda ao menor custo.
 for(j=0;j<m;j++)
 {
   menor=c[0][j];
   v[j]=0;
   for(i=1;i<n;i++)
   {
     if(c[i][j]<menor)
     {
       v[j]=i;
       menor=c[i][j];
```

```
    }
  }
}

// 2- Vetor de facilidades abertas (1) e fechadas (0).
for(i=0;i<n;i++)
{
  I[i]=0;
}
for(j=0;j<m;j++)
{
  I[v[j]]=1;
}

// 3- Encontrar custo total para a configuração encontrada.
ub=0;
for(j=0;j<m;j++)
{
  i=v[j];
  ub=ub+c[i][j];
}

for(i=0;i<n;i++)
{
  if(I[i]==1)
  {
    ub=ub+F[i];
  }
}

// FASE DE REFINAMENTO
// 1- Criando vetor das facilidades abertas.
refinando = 1;
while (refinando)
{
  refinando = 0;
  star=0;
  for(i=0;i<n;i++)
  {
    if(I[i]==1)
    {
      U[star]=i;
      star++;
    }
  }
  for(j=0;j<m;j++)
  {
    v[j]=0;
    menor=INFINITO;
    for(i=0;i<star;i++)
    {
      if(c[U[i]][j]<menor)
      {
        v[j]=U[i];
        menor=c[U[i]][j];
      }
    }
   }
   for(k=0;k<star;k++)
   {
     tot=0;
     for(j=0;j<m;j++)
     {
        // Encontrando demandas atendidas pela facilidade i (U[k]).
        if(U[k]==v[j])
        {
          P[tot]=j;
          tot++;
```

```
        }
      }
      // Para cada demanda atendida pela facilidade i,
      // encontrar facilidade com segundo menor custo.
      for(p=0;p<tot;p++)
      {
        // Modificar temporariamente o menor custo
        csalvo = c[U[k]][P[p]];
        c[U[k]][P[p]] = INFINITO;
        menor = INFINITO;
        for(q=0;q<star;q++)
        {
          if(c[U[q]][P[p]]<menor)
          {
            menor=c[U[q]][P[p]];
          }
      }
      // Restaurar o menor custo
      c[U[k]][P[p]] = csalvo;
      d[p]=menor - c[v[P[p]]][P[p]];
    }

    delta=0;
    for(p=0;p<tot;p++)
    {
      delta=delta+d[p];
    }
    delta=delta-F[U[k]];
    if(delta<0)
    {
      I[U[k]]=0;
      ub=ub+delta;
      star--;
      if (star > 1)
      {
        refinando = 1;
      }
      break;
    }
  }
 }
 BV=ub;
 free(v);
 free(U);
 free(P);
 free(d);
 return;
}
void gerarvizinhanca()
{
  int t, a, x; // a = parâmtro do programa/ t=relaciona a quantidade de facilidades
  int b, i;    // b = gerado aleatoriamente
  int *R;      // R vetor de facilidades abertas geradas a partir de I

  R = (int *)calloc(n, sizeof(int));

  for(h = 0; h < NBHSIZE; h++)
  {
    a = 96;
    // mstar = numero de facilidades abertas na solucao h. mstar[n]
    mstar[h] = 0;
    for(t = 0; t < n; t++)
    {
      // gerar numero aleatorio entre 0 e 100
      b = (rand()%101);
      if((b > a) && (I[t] == 0))
      {
         R[t] = 1;
```

```
      }
      if((b < a) && (I[t] == 0))
      {
        R[t]=0;
      }
      if((b > a) && (I[t] == 1))
      {
       R[t] = 0;
      }
      if((b < a) && (I[t] == 1))
      {
        R[t] = 1;
      }
    }

    // formar vetor com os indices das facilidades abertas
    for(i = 0; i < n; i++)
    {
    // J[h][n] = matriz das facilidades abertas na solucao h.
      if(R[i] == 1)
      {
        J[h][mstar[h]] = i;
        mstar[h]++;
      }
    }
  } // fim for h
  free(R);

  return;
}

void custotal()
{
  int j, i;
  float menor;

  if (docum)
 {
   printf("\nMatriz J:");
   for(j=0;j<NBHSIZE;j++)
   {
     printf("\n");
     for(i=0;i<n;i++)
     {
       printf("%d ", J[j][i]);
     }
   }
   system("pause");
 }

 for(h=0;h<NBHSIZE;h++)
 {
   totcost[h]=0;
   for(j=0;j<m;j++)
   {
     // encontrar, entre as facilidades abertar na solucao h, aquela que
     // apresenta o menor custo de entrega para a demanda j.
     menor=INFINITO;
     for(i=0;i<mstar[h];i++)
     {
       if(c[J[h][i]][j]<menor)
       {
         menor=c[J[h][i]][j];
       }
     }
     totcost[h]= totcost[h]+menor;
   }
   // somar ao custo total o custo de abertura das facilidades.
```

```
   for(i=0;i<mstar[h];i++)
   {
     totcost[h]=totcost[h]+F[J[h][i]];
   }

 }
 return;
}

void trocaFloat(float *v, int i, int j)
{
  float aux;
  aux = v[i];
  v[i] = v[j];
  v[j] = aux;
}

void trocaInt(int *v, int i, int j)
{
  int aux;
  aux = v[i];
  v[i] = v[j];
  v[j] = aux;
}

void ordenar_por_selecao(float *x, int *l, int b)
{
  int pos;
  float menor;
  int i,a = 0;

  while (a < b)
  {
    menor = INFINITO;
    for (i = a; i < b; i++)
    if (x[i] < menor)
    {
       menor = x[i];
       pos = i;
    }
    trocaFloat(x,a,pos);
    trocaInt(l,a,pos);
    a++;
  }
}

// ordenar os custos do menor para o maior no vetor totcost.
// criar vetor IC[] que representa o indice inicial dos custos agora ordenados
(solucao vizinha h)
void ordenar()
{
  int i,b;

  // criar vetor IC[]
  for(i=0;i<NBHSIZE;i++)
  {
    IC[i]=i;
  }

  // Classificar vetor
  ordenar_por_selecao(totcost,IC,NBHSIZE);
  return;
}

// criar lista tabu.
void listatabu()
{
  int i, j;
```

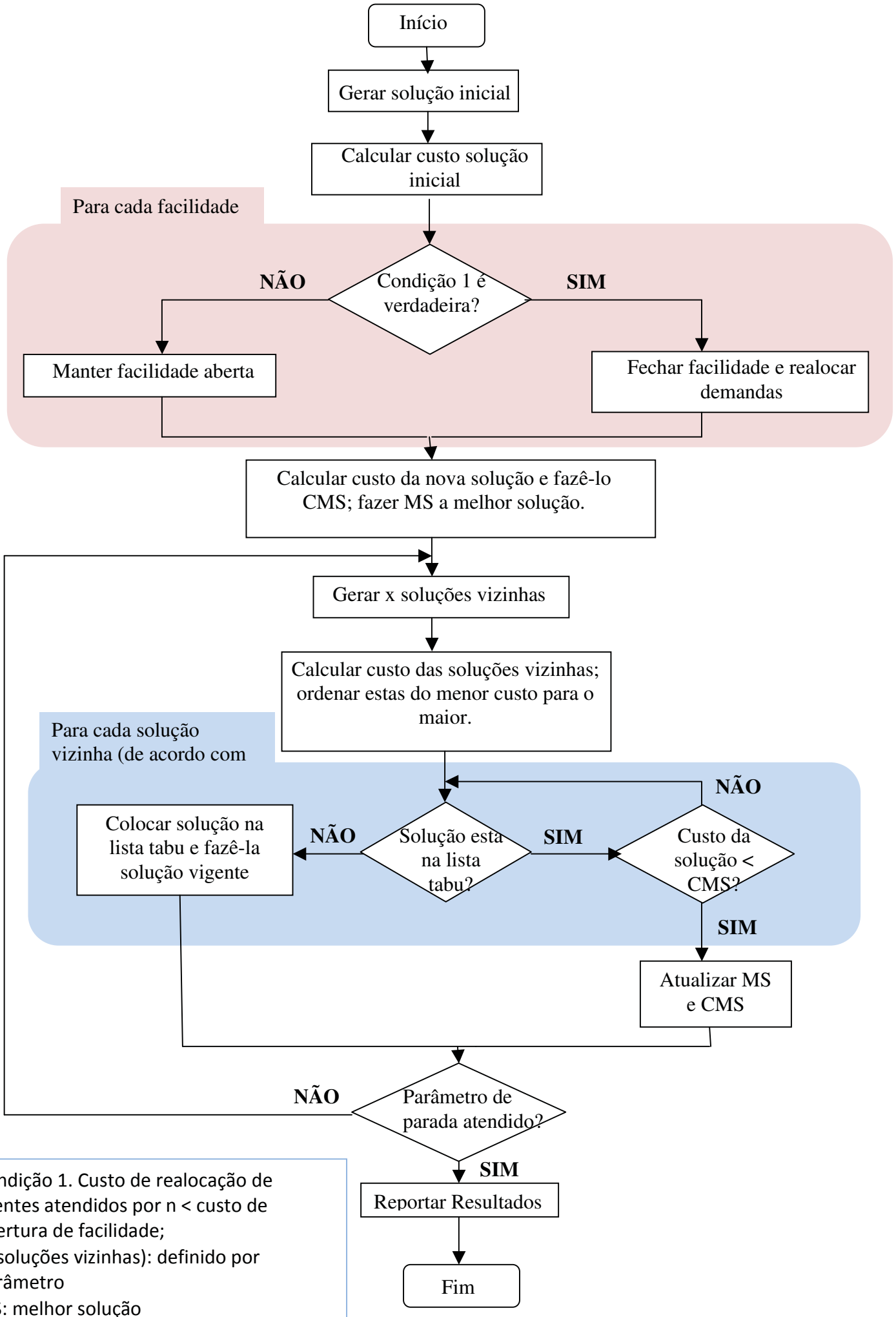
```
  for(i=0;i<TL;i++)
  {
    for(j=0;j<n;j++)
  {
  TB[i][j]=0;
 }
}

return;
}

void alocarMemoria()
{
  int i,j;

  c = (float **)calloc(n, sizeof(float *));
  for (i = 0; i < n; i++)
  {
    c[i] = (float *)calloc(m, sizeof(float));
  }
  F = (float *)calloc(n, sizeof(float));
  I = (int *)calloc(n, sizeof(int));
  BS = (int *)calloc(n, sizeof(int));
  mstar = (int *)calloc(NBHSIZE, sizeof(int));
  J = (int **)calloc(NBHSIZE, sizeof(int *));
  for (i = 0; i < NBHSIZE; i++)
  {
    J[i] = (int *)calloc(n, sizeof(int));
  }

  TB = (int **)calloc(TL, sizeof(int *));
  for (i = 0; i < TL; i++)
  {
    TB[i] = (int *)calloc(n, sizeof(int));
  }

  totcost = (float *)calloc(NBHSIZE, sizeof(float));

  IC = (int *)calloc(NBHSIZE, sizeof(int));
}

void lerDadosProblema()
{
  int i,j;
  int cap,dem;
  char nome[50];

  sai = fopen("saida.txt", "a");
  if (sai == NULL)
  {
    printf("\nErro ao abrir arquivo de saida\n");
    system("pause");
    exit(1);
  }

  fscanf(dat,"%s", nome);
  printf("[%s]\n",nome);

  arq = fopen(nome, "r");
  if (arq == NULL)
  {
    printf("\nErro ao abrir arquivo de dados (%s)\n",nome);
    system("pause");
    exit(1);
  }
  fscanf(arq,"%d %d %f",&n,&m,&best);
  //fprintf(sai,"  %d  %d%              .3f",n,m,best);
```

```
  // Alocar memória para variáveis globais
  alocarMemoria();

  for(i=0;i<n;i++)
  {
    fscanf(arq,"%d %f",&cap,&F[i]);  // cap não é usado
  }

  for(j=0;j<m;j++)
  {
    fscanf(arq,"%d",&dem);           // dem não é usado
    for(i=0;i<n;i++)
    {
      fscanf(arq,"%f",&c[i][j]);
    }
  }
}

int main(int argc, char *argv[])
{

  dat = fopen("problemas.txt", "r");
  if (dat == NULL)
  {
    printf("\nErro ao abrir arquivo de problemas\n");
    system("pause");
    exit(1);
  }
  int l, d, i, j, pi, k, x, iter, cont;
  while (1)
  {
    totTimeUFL = 0;

    // fazer docum = 1, para depurar o programa
    docum = 0;
    srand((unsigned)time(NULL));
    lerDadosProblema();
    iTime = clock();
    listatabu();
    solin();
    iter=0;
    k=1;
    pi=0;
    while ((k < MAXSEMMELHORA) && (iter < ITERMAX))
    {
      iter++;
      gerarvizinhanca();
      custotal();
      ordenar();
      cont=0;
      for(l=0;l<NBHSIZE;l++)
      {
        cont++; //verificar se há solução válida nesta iteração
        i=0;
        // verificar para toda a lista tabu (de tamanho TL)
        for(i=0;i<TL;i++)
        {
          j=0;
          // verificar se a melhor solucao encontrada esta na lista tabu
          while ((j < n) && (TB[i][j]==J[IC[l]][j]))
          {
            j++;
            // se j = n está na lista tabu --> parar de verificar
            if(j==n)
            {
              break;
            }
```

```
          }
         }
         // se a solução encontrada possuir custo menor que BV ou se não estiver
         // na lista tabu: SOLUÇÃO VÁLIDA encontrada
         // x=l;

         if((totcost[l] < BV) || (j != n))
         {
          break;
         }
        }
        // l=x;
        // se não houver solução válida: iteração sem melhora.
        if(cont==NBHSIZE)
        {
           k++;
        }

        if((j != n) || (totcost[l] < BV))
        {
          // se lista tabu estiver cheia, escrever sobre ela
          pi++;
          if(pi==TL)
         {
           pi=0;
         }

         for(j=0;j<n;j++)
         {
           TB[pi][j]=J[IC[l]][j];
         }
         // atualizar solucao, a partir da qual serao geradas novas solucoes
vizinhas.
         for(i=0;i<n;i++)
         {
           I[i]=0;
         }
         for(j=0;j<n;j++)
         {
           I[J[IC[l]][j]]=1;
         }

         if(totcost[l] >= BV)
         {
           k++;
         }
         // atualizar o melhor valor ja encontrado (BV)
         if(totcost[l] < BV)
        {
          BV = totcost[l];
          k=1;
          for(j=0;j<n;j++)
          {
            BS[j]=J[IC[l]][j];
          }
         }
       }// fim do if "solução válida"


    }// fim do while itermax

    fTime = clock();
    totTimeUFL += cpuTime(iTime,fTime);
    fprintf(sai,"                        %.3f", BV);
    fprintf(sai,"                        %.3f\n", totTimeUFL);
  } // fim do while(1)
  system("pause");
  return 0;
```

## ANEXO B – Algoritmo

Abaixo segue fluxograma simplificado das etapas do algoritmo.




- Condição 1. Custo de realocação de clientes atendidos por n < custo de abertura de facilidade;
- X (soluções vizinhas): definido por parâmetro
- MS: melhor solução