Ministério da
Ciência e Tecnologia

INPE-15744-TDI/1489

MÉTODOS HEURÍSTICOS PARA O PROBLEMA DE LOCALIZAÇÃO DE CONCENTRADORES

Wesley Gomes de Almeida

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Edson Luiz França Senne, aprovada em 27 de fevereiro de 2009.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2009/02.16.17.46>

INPE
São José dos Campos
2009

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6911/6923
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência e Tecnologia

INPE-15744-TDI/1489

MÉTODOS HEURÍSTICOS PARA O PROBLEMA DE LOCALIZAÇÃO DE CONCENTRADORES

Wesley Gomes de Almeida

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Edson Luiz França Senne, aprovada em 27 de fevereiro de 2009.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2009/02.16.17.46>

INPE
São José dos Campos
2009

Dados Internacionais de Catalogação na Publicação (CIP)

Almeida, Wesley Gomes de.
A64m Métodos heurísticos para o problema de localização de concentradores / Wesley Gomes de Almeida. – São José dos Campos : INPE, 2009.
112p. ; (INPE-15744-TDI/1489)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2009.
Orientador : Dr. Edson Luiz França Senne.

1. Probelma de localização de concentradores. 2. Metaheurística. 3. Algoritmo genético. 4. Lista tabu. 5. Recozimento simulado. 6. Busca por agrupamentos I.Título.

CDU 004.023

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de Mestre em Computação Aplicada

Dr. Horacio Hideki Yanasse

Presidente / INPE / SJCampos - SP

Dr. Edson Luiz Franca Senne

Orientador(a) / UNESP/GUARA / Guaratinguetá - SP

Dr. Marcos Antonio Pereira

Convidado(a) / UNESP / Guaratinguetá - SP

Dr. Marco Cesar Goldbarg

Convidado(a) / UFRN / Natal - RN

Aluno (a): Wesley Gomes de Almeida

São José dos Campos, 27 de fevereiro de 2009

A meus pais.

AGRADECIMENTOS

Primeiramente agradeço a Deus por ter me abençoado e dado forças nesta caminhada. A meus pais Pedro Gomes e Maria Helena que sempre me apoiaram, durante meus estudos. À minha noiva Angélica por todo apoio e compreensão durante este trabalho.

Aos professores mestres e doutores das Faculdades Integradas de Caratinga (FIC): André G. dos Santos, Paulo E. dos Santos, Valdenir de Souza Junior, Antônio P. B. Scarpelli por terem me incentivado a seguir a carreira acadêmica e à todos os outros professores das FIC.

Aos meus amigos de república pelo apoio nas horas difíceis. Aos amigos do CCS, Cristiano, Sóstenes, Rodolfo, Douglas, Rogério, Rudinei, Aline, Adriana, e a todos que de alguma forma contribuíram neste trabalho.

Ao Marcos Roberto Silva orientando do Dr. Cláudio Barbieri Cunha pelas explicações dadas sobre o conjunto de dados. Ao Prof. Dr. Haluk Topcuoglu pelo envio dos custos fixos utilizados com o conjunto de dados. Ao Antonio Chaves pelos conhecimentos passados durante as implementações.

Agradeço também a todos os professores do INPE, pelos ensinamentos nas disciplinas de computação aplicada, e em especial aos professores Luis Antonio N. Lorena e Horácio H. Yanasse pelas correções e críticas propostas para o melhoramento da qualidade deste trabalho. À secretária da CAP por todo o suporte necessário durante esses anos.

À CAPES (Cordenção de Aperfeiçoamento de Pessoal de Nível Superior) pelo auxílio financeiro.

E ao meu orientador Dr. Edson L. F. Senne pela oportunidade, e pelos ensinamentos durante o desenvolvimento deste trabalho.

RESUMO

O problema de localização de concentradores é comumente encontrado em redes de transporte e de telecomunicação. Trata-se de problema de Otimização Combinatória NP-difícil e que ocorre em diversas situações práticas, tais como, no transporte aéreo, nos serviços de entregas postais, nos serviços de atendimento de emergência, no abastecimento de supermercados e de redes de lojas, na localização de antenas de comunicação, dentre muitas outras. A decisão sobre a localização de concentradores e os nós da rede que serão alocados a cada um destes depende do número de concentradores necessários, da demanda de cada nó, da capacidade de atendimento dos concentradores, dentre outros fatores. Metaheurísticas têm sido usadas com sucesso para obter soluções em diversos problemas de Otimização Combinatória. Em geral, a construção de algoritmos heurísticos eficientes requer bons mecanismos de intensificação de busca. Neste trabalho estuda-se o problema de localização de concentradores e propõem-se métodos heurísticos. Os métodos heurísticos propostos utilizam a busca por agrupamentos, uma técnica de intensificação de busca capaz de identificar as regiões do espaço de busca mais promissoras para a obtenção de boas soluções. Dois algoritmos são propostos: um algoritmo genético (AG) e um algoritmo *simulated annealing* com lista tabu (SATL). Testes computacionais foram realizados com os algoritmos propostos e com estes algoritmos combinados com a busca por agrupamentos (*clustering search*). Os testes demonstram que a aplicação da busca por agrupamentos a estes algoritmos permite obter soluções de melhor qualidade e em menor tempo computacional, em relação às soluções obtidas pelos algoritmos AG e SATL isoladamente.

HEURISTIC METHODS FOR THE HUB LOCATION PROBLEM

ABSTRACT

The hub location problem is commonly found in transportation networks and in telecommunications networks. It is a NP-hard Combinatorial Optimization problem that occurs in many practical situations, such as air transportation, postal delivery services, emergency care services, supplying networks of supermarkets and shops, location of antennas for communication, among many others. The decision about the hubs location and for which hub the other network nodes should be allocated depends of the number of hubs needed, the demand of each node, the service capacity of the hubs, among other factors. Metaheuristcs have been used successfully to obtain solutions to various Combinatorial Optimization problems. In general, the construction of efficient heuristics requires good search intensification mechanisms. In this work the hub location problem is studied and heuristic methods are proposed. The proposed methods use the clustering search, a technique of search intensification which is capable of identifying the more promising regions of the search space for obtaining good solutions. Two algorithms are proposed: a genetic algorithm (AG) and a simulated annealing with tabu list algorithm (SATL). Computational tests were accomplished with the proposed algorithms and with these algorithms combined with the clustering search. The tests demonstrate that the application of the clustering search to these algorithms allows to obtain solutions of better quality and in smaller computational times, in relation to the solutions obtained by the algorithms AG and SATL separately.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

AG	Algoritmo Genético
AP	*Australian Post*
CAB	*Civil Aeronautics Board*
CS	*Clustering Search*
ECS	*Evolutionary Clustering Search*
PLC	Problema de Localização de Concentradores
SATL	*Simulated Annealing Tabu List*
USAHLP	*Uncapacited Single Allocation Hub Location Problem*
USApHMP	*Uncapacitated Single Allocation p-Hub Median Problem*

LISTA DE SÍMBOLOS

$\|C\|$	Número de *clusters* existentes
α	Custo de transferência entre concentradores
β	Estratégia de busca associada aos *clusters*
γ_i	Quantidade de soluções pertencentes ao *cluster i.*
δ	Custo de distribuição entre um concentrador e seu destino
λ	Custo de coleta entre um nó de origem e seu concentrador
μ	Limite para que um *cluster* se torne promissor
C_i	Centro do *cluster i*
D_i	Custo total do nó *i*
d_{ij}	Distância entre os nós *i* e *j*
E_i	Quantidade de fluxo que chega no ponto *i*
f_j	Custo fixo para que um *hub* seja localizado no ponto *j*
I_i	Índice usado para escolha dos *hubs* iniciais no SATL.
NS	Número de soluções geradas a cada intervalo de análise dos *clusters*
N_c	Número máximo de *clusters*
N_g	Número de gerações do AG
N_p	Número de indivíduos em uma população
n	Número de nós da rede
PD	Pressão de densidade
P_c	Porcentagem de *crossover*
P_e	Porcentagem de elite

P_m	Probabilidade de mutação
p	Número de concentradores a serem localizados
pr	Probabilidade de aceitar um movimento de piora
Q_i	Capacidade do concentrador i
r_i	Índice de ineficácia do *cluster* i
r_{max}	Número máximo de execuções consecutivas sem melhora para o índice de ineficácia r_i
S_i	Quantidade de fluxo que deixa o ponto i
V	Conjunto de nós da rede
w_i	Fluxo total que entra e sai do nó i
w_{ij}	Quantidade de fluxo transferido entre os nós i e j
x_{ik}	Variável de decisão tal que $x_{ik} = 1$ se o nó i está alocado ao concentrador k ($x_{ik} = 0$, caso contrário). Se $x_{kk} = 1$, significa que o nó k é um concentrador, caso contrário, $x_{kk} = 0$
Y_{ikl}	Quantidade de fluxo transferido entre os concentradores k e l originados a partir do nó i

1. INTRODUÇÃO

Em alguns problemas definidos em redes, a comunicação entre os nós da rede não acontece de forma direta, mas por meio de nós especiais denominados concentradores. Isto ocorre com freqüência, por exemplo, em redes de transporte e em redes de telecomunicação. Nestes casos, diz-se que a rede é do tipo *hub-and-spoke* (AYKIN, 1994).

Um modelo desse tipo de rede pode ser exemplificado imaginando-se um serviço de transporte rodoviário de cargas em que a demanda individual dos clientes não é suficiente para lotar um veículo em uma única viagem. Por esse motivo, as cargas são agregadas e transportadas em conjunto. Para isto, empresas que operam este tipo de serviço possuem instalações físicas localizadas em diversas regiões para consolidar as cargas oriundas de diversas origens. Portanto, este tipo de serviço compreende as operações de coleta (de um cliente até um terminal de consolidação de origem), transferência (de um terminal de consolidação de origem para um terminal de consolidação de destino) e distribuição (do terminal de consolidação de destino até o cliente final). Para uma empresa deste tipo, um bom planejamento da rede de transporte, com os terminais de consolidação (concentradores) bem localizados, pode implicar em ganhos financeiros significativos.

O problema de localização de concentradores em uma rede consiste em determinar o número de instalações de consolidação (concentradores), a localização de cada um dos concentradores e a alocação dos demais nós da rede (denominados como *spokes*, ou nós de demanda) aos concentradores, de forma a minimizar o custo total de operação, que pode incluir os custos variáveis de transporte e os custos fixos de localização das instalações. Essas instalações podem ser fábricas, portos, pontos de venda de produtos, armazéns, postos de serviço de rotina ou de emergência, postos de correio, pontos de incineração de lixo, centros de atendimento médico, aeroportos, antenas de comunicação, escolas, bibliotecas, dentre muitas outras.

Este trabalho tem como principais objetivos propor algoritmos para encontrar soluções com custo total de operação reduzido em redes do tipo *hub-and-spoke*, utilizando a busca por agrupamentos (*Clustering Search*, CS), e a verificação da eficiência destes algoritmos em relação aos mesmos algoritmos sem a utilização da CS. Para isto, dadas as localizações dos clientes e os fluxos de transporte entre eles, o que se deseja é determinar quantos e onde localizar os concentradores necessários. Como visto acima, os concentradores são tipos especiais de facilidades que têm por tarefa permitir o transporte de alguma entidade entre dois nós de demanda de uma rede. Desta forma, os concentradores recebem fluxos de diferentes origens e para diferentes destinos de modo a promover uma economia de escala que decorre desta concentração, em relação ao serviço em que cada par origem-destino possa ser atendido diretamente.

A Figura 1.1 mostra um exemplo de rede do tipo *hub-and-spoke*, onde os concentradores (representados por quadrados) atendem aos nós de demanda (representados por círculos). Um fluxo w_{ij}, com origem no nó *i* e destino no nó *j*, é transportado da seguinte forma: inicialmente, a carga de *i* é enviada para o nó *k*, onde é consolidada com outros fluxos de origens diferentes, e enviada para o concentrador *l*. A partir daí, os fluxos são distribuídos para os nós atendidos por *l*, inclusive *j*.

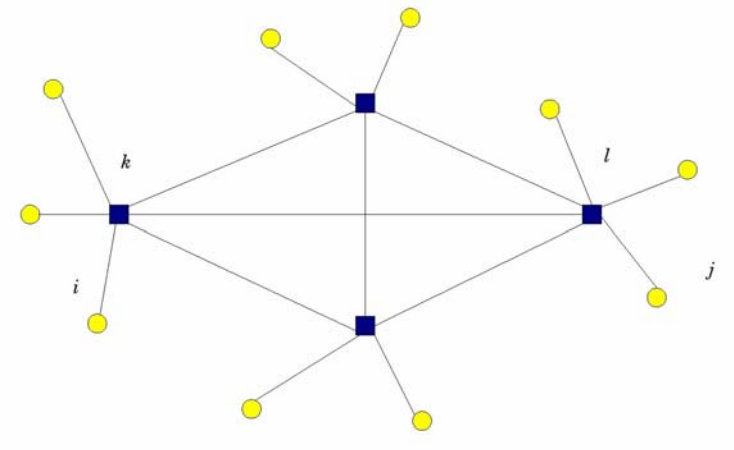

Figura 1.1– Rede do tipo *hub-and-spoke*

Algumas vezes, a localização dos terminais de consolidação em redes deste tipo é determinada por meio da experiência e para resolver problemas específicos. Isso nem sempre leva a uma boa solução. Neste trabalho pretende-se desenvolver algoritmos que levem a boas soluções do problema. Devido ao alto grau de complexidade do problema, o desenvolvimento de métodos eficientes capazes de obter boas soluções constitui um grande desafio.

O problema de localização de concentradores pertence à classe de problemas NP-difíceis (GAREY E JOHNSON, 1979; ZIVIANI, 2004). Portanto, a determinação de solução ótima para o problema por meio de algoritmos exatos, deve utilizar métodos enumerativos, que exigem grande esforço computacional e podem ser impraticáveis para exemplares do problema de grandes dimensões. Por isto, diversas propostas usando heurísticas têm sido consideradas, tais como: Busca Tabu (KLINCEWICZ, 1991), *Simulated Annealing* (ABDINNOUR-HELM E VENKATARAMANAN, 1993), Redes Neurais Artificiais (SMITH ET AL, 1996) e Algoritmos Genéticos (ABDINNOUR, 1998; TOPCUOGLU ET AL, 2005; CUNHA E SILVA, 2007).

Neste trabalho, propõe-se duas metaheurísticas para a solução do problema de localização de concentradores: um algoritmo genético e um algoritmo *simulated annealing* com lista tabu. Propõe-se também a aplicação da busca por agrupamentos (OLIVEIRA E LORENA, 2004, 2007) tendo estes algoritmos como geradores de soluções factíveis para o problema. Testes computacionais foram conduzidos com o objetivo de verificar a eficácia e a eficiência dos algoritmos propostos, por meio da análise do desempenho computacional e da qualidade das soluções obtidas para exemplares do problema disponíveis na literatura.

O principal motivo pelo qual este trabalho utiliza métodos heurísticos é devido ao fato que problemas de localização de concentradores serem problemas quadráticos, ou seja, não lineares o custo computacional para a solução desses tipos de problemas é muito alto.

Este trabalho está organizado da forma descrita a seguir.

O Capítulo 2 corresponde à uma revisão bibliográfica sobre o problema, onde são apresentados conceitos e definições básicas para o desenvolvimento dos próximos capítulos, além dos tipos de problemas de localização de concentradores e seus respectivos modelos matemáticos. Descreve-se também neste capítulo os exemplares do problema disponíveis na literatura que foram usados nos testes computacionais.

No Capítulo 3 é apresentado o método de Busca por Agrupamentos com uma breve explicação sobre cada um dos seus métodos, mostrando suas características e funcionamento.

O Capítulo 4 descreve os algoritmos propostos para a solução do problema de localização de concentradores. Explica-se, de maneira detalhada, como foram implementados o algoritmo genético e o algoritmo *simulated annealing* com lista tabu propostos e como estes algoritmos foram integrados à busca por agrupamentos.

No Capítulo 5 são apresentados os resultados obtidos pelos algoritmos propostos neste trabalho. Os testes foram realizados com os conjuntos de dados AP e CAB disponíveis na literatura, para exemplares do problema definidos em redes de 10 a 200 nós.

No Capítulo 6 são apresentadas as conclusões do trabalho e as sugestões para continuação deste trabalho em futuras pesquisas.

2. REVISÃO DA LITERATURA

2.1 – Problema de Localização de Concentradores

O problema de localização de concentradores (PLC) tem com principal objetivo minimizar o custo total de transporte de alguma entidade em uma rede com *n* nós de demanda, dado o fluxo entre cada par de nós origem-destino. A solução do problema busca encontrar os nós que devem se tornar concentradores e a alocação dos demais nós a estes concentradores de forma que o custo total da rede seja minimizado.

Existem diferentes versões do problema de localização de concentradores (*hubs*). Alguns casos podem apresentar restrições de capacidade (AYKIN, 1994), ou seja, um limitante no volume de informações que um concentrador consegue transportar, ou ainda um custo fixo associado a cada concentrador, além dos custos de alocação dos nós de demanda da rede aos concentradores.

Quando não existe restrição quanto ao fluxo (de pessoas ou de dados, por exemplo) que passa por um concentrador e cada nó de demanda não pode ser alocado a mais que um concentrador, o problema denomina-se Problema Não-Capacitado de Localização de Concentradores (USAHLP - *Uncapacitated Single Allocation Hub Location Problem*). Neste problema o número de concentradores é uma variável de decisão. No caso do número de concentradores ser fixo (por exemplo, igual a *p*), o problema é denominado de USApHMP (*Uncapacitated Single Allocation p-Hub Median Problem*) (CHEN, 2008; EBERY, 2001). No entanto, quando um nó de demanda pode ser alocado a mais do que um concentrador, o problema denomina-se problema de localização de concentradores com alocação múltipla (do inglês, *Uncapacitaded Multiple Allocation Hub Location Problem*, UMAHLP) e quando existe restrição de capacidade quanto ao fluxo máximo de um concentrador, o problema é conhecido como problema de localização de concentadores capacitado (*Capacitated Single Allocation Hub Location Problem*, *CSAHLP*).

Alumur e Kara (2008) apresentam um estudo detalhado sobre problemas de localização de concentradores em que são citados mais de 100 trabalhos.

Goldman (1969) foi o primeiro a apresentar o problema de localização de concentradores. No entanto, o primeiro modelo matemático com função objetivo quadrática para o problema de localização de p-concentradores não-capacitado (USApHMP) é devido a O'Kelly (1987), que formulou o problema como:

$$f(x) = Min \sum_i \sum_j w_{ij} \sum_k \lambda\, d_{ik} x_{ik} + \sum_i \sum_j w_{ji} \sum_l \delta\, d_{jl} x_{jl} + \sum_i \sum_k x_{ik} \sum_j \sum_l \alpha\, x_{jl} d_{kl} w_{ij} \quad (2.1)$$

Sujeito a:

$$\sum_k x_{kk} = p \quad (2.2)$$

$$\sum_k x_{ik} = 1, \quad \forall i \in V \quad (2.3)$$

$$x_{kk} - x_{ik} \geq 0, \quad \forall i,k \in V, \quad (2.4)$$

$$x_{ik} \in \{0,1\} \quad \forall i,k \in V. \quad (2.5)$$

Em que:

- V é conjunto de nós da rede;
- p é o número de concentradores a serem localizados;
- d_{ij} é a distância entre os nós i e j;
- w_{ij} é a quantidade de fluxo transferido entre os nós i e j;
- λ, α, δ são, respectivamente, os custos de coleta, transferência e distribuição;
- x_{ik} é uma variável de decisão, tal que $x_{ik} = 1$ se o nó i está alocado ao concentrador k ($x_{ik} = 0$ caso contrário). Deve-se observar que se $x_{kk} = 1$ então o nó k é um concentrador; Caso contrário, $x_{kk} = 0$.

Nessa formulação, a função-objetivo (2.1) estabelece o custo total a ser minimizado que corresponde a soma dos custos de coleta transferência e distribuição de uma rede, a restrição (2.2) fixa o número de concentradores igual a p, as restrições (2.3) garantem

que cada nó de demanda será alocado a um único concentrador, as restrições (2.4) asseguram que as alocações serão feitas apenas para nós que são concentradores, e as restrições (2.5) correspondem às condições de integralidade das variáveis de decisão.

Os problemas de localização de concentradores podem também ser classificados como de alocação única e de alocação múltipla. No problema de localização de concentradores com alocação única cada nó de demanda deve estar alocado a um único concentrador, enquanto que no problema de localização de concentradores com alocação múltipla, cada nó de demanda pode ser alocado a mais de um concentrador, ou seja, cada nó da rede pode enviar (ou receber) fluxo para (ou de) múltiplos concentradores. Neste trabalho pretende-se restringir o estudo a problemas de localização de concentradores com alocação única.

O primeiro modelo de Programação Linear Inteira para o USApHMP foi proposto por Campbell (1994). Este modelo, no entanto, exige um número muito grande de variáveis e restrições. Em seguida, vários outros modelos foram propostos, dentre os quais se destaca o de Ernst e Krishnamoorthy (1996) com o qual os autores relatam resultados considerados melhores que os obtidos por outros modelos. Ernst e Krishnamoorthy (1996) formularam o problema de localização de p-concentradores como:

$$f(x) = Min \sum_i \sum_k d_{ik} x_{ik} (\lambda E_i + \delta S_i) + \sum_i \sum_k \sum_l \alpha d_{kl} Y^i{}_{kl} \qquad (2.6)$$

sujeito a: (2.2)-(2.5) e

$$\sum_l Y_{ikl} - \sum_l Y_{ilk} = E_i x_{ik} - \sum_j w_{ij} x_{jk} \ \forall i, k \in V \qquad (2.7)$$

$$Y_{ikl} \geq 0 \ \forall i, k, l \in V \qquad (2.8)$$

em que: os símbolos V, p, $d_{ij,}$ $w_{ij,}$ λ, α, δ, x_{ik} têm os mesmos significados já apresentados para o modelo (2.1)-(2.5), Y_{ikl} significa a quantidade de fluxo transferido entre os

concentradores k e l originado a partir do nó i, E_i é a quantidade de fluxo que chega no de demanda i ($E_i = \sum_j w_{ij}$) e S_i a quantidade de fluxo que deixa o nó i ($S_i = \sum_j w_{ji}$).

A restrição (2.7) faz o balanceamento do nó i para o nó k em que a quantidade de fluxo que entra e o fluxo que sai são determinados pela variável de decisão $x_{ik.}$ A partir deste modelo, Ernst e Krishinamoorthy (1996) propuseram um algoritmo s*imulated annealing* para o USApHMP capaz de determinar limites superiores para um método do tipo *branch-and-bound*. Com esse algoritmo, os autores encontraram soluções ótimas para exemplares do problema definidos em redes de até 50 nós.

Ernst e Krishinamoorthy (1998) propuseram um algoritmo do tipo *branch-and-bound* para o USApHMP capaz de resolver problemas de caminho mais curto para a obtenção de limites inferiores para o PLC. Com essa proposta os autores conseguiram obter soluções exatas para exemplares do problema definidos em redes de até 100 nós.

Pirkul e Schilling (1998) desenvolveram um método de relaxação lagrangeana, com o uso do método de otimização por subgradientes e obtiveram em seus testes computacionais resultados com erro (*gap*) variando entre 0.048% e 1%, para problemas do conjunto CAB (ver Seção 2.2) do USApHMP.

Kratica et al (2007) apresentam dois algoritmos genéticos com estratégias diferentes para a representação dos indivíduos. Estes algoritmos trabalham apenas com soluções factíveis, ou seja, caso uma solução inviável apareça durante a busca, o algoritmo se encarrega em corrigi-la. Para o trabalho em questão os autores apresentaram um conceito denominado bits congelados (nome dado aos bits de um indivíduo que não se alteram devido à convergência prematura do método), uma vez identificados esses bits o método permite uma probabilidade maior de mutação a estes, aumentando a diversidade do material genético. Com essa estratégia os autores obtiveram soluções ótimas para exemplares do conjunto de dados AP com até 50 nós, para o problema USApHMP. Nos casos em que $n \geq 100$ os autores obtiveram soluções com um custo menor que as

apresentadas na literatura. Por essa razão, esses resultados são usados neste trabalho como referência para melhores soluções no Capítulo 5.

Chen (2008) propõe uma heurística híbrida semelhante à proposta por Chen (2007), aplicada ao USApHMP. Tal método é composto por um algoritmo *simulated annealing* e uma lista tabu. Com este trabalho o autor obteve soluções ótimas para problemas do conjunto AP (ver Seção 2.2) de até 50 nós, e boas soluções em um tempo razoável para problemas com mais de 100 nós.

O'Kelly (1992) apresentou um modelo para o problema de localização de concentradores, no qual cada instalação possui um custo fixo associado. Duas possibilidades foram consideradas: o problema de localização de concentradores capacitado, em que cada concentrador pode ter uma capacidade máxima de atendimento (alocação) e o problema não-capacitado, em que não existem limites para a capacidade de atendimento dos concentradores.

O problema não-capacitado foi formulado como:

$$f(x) = Min. \sum_i \sum_j w_{ij} \sum_k \lambda\ d_{ik} x_{ik} + \sum_i \sum_j w_{ji} \sum_l \delta\ d_{jl} x_{jl} + \sum_i \sum_k x_{ik} \sum_j \sum_l \alpha\ x_{jl} d_{kl} w_{ij} + \sum_k f_k x_{kk} \quad (2.9)$$

sujeito às restrições: (2.3)-(2.5).

Deve-se observar que a função-objetivo (2.9) diferencia-se da função (2.1) apenas quanto ao acréscimo do somatório dos custos fixos de localização dos concentradores.

Existem vários estudos relacionados a problemas de localização de concentradores não capacitados, entre esses o de Topcuoglu et al (2005) que apresentam um algoritmo genético e comparam seus resultados com um algoritmo *simulated annealing*, implementado pelos mesmos autores, e com um algoritmo genético com busca tabu

(ABDINNOUR-HELM, 1998) para conjuntos CAB e AP. Este trabalho resultou em um método eficiente e robusto, capaz de obter soluções boas em tempos computacionais reduzidos.

Cunha e Silva (2007) propuseram um algoritmo genético híbrido agregado a um método *simulated annealing* com a finalidade de melhorar a função de adaptação de cada indivíduo. Os autores testaram o método para problemas CAB e aplicaram o método a um problema real da companhia de caminhões LTL do Brasil, em que os custos não são simétricos, ou seja, os custos de ida são diferentes dos custos de volta. O algoritmo foi aplicado a uma rede de 46 nós, onde cada nó representa uma determinada cidade de uma filial da empresa. Tal trabalho gerou resultados muito bons para a companhia de caminhões. Algumas das melhores soluções ajudaram a empresa brasileira a considerar e analisar vários processos com finalidade de reduzir os custos.

Chen (2007) apresenta um algoritmo híbrido composto por um *simulated annealing* e uma lista tabu. Além disso, acrescentou-se um procedimento heurístico para o cálculo do limite superior para o número de concentradores. Com essa idéia, a metaheurística implementada gerou soluções muito boas em termos de tempo de execução e qualidade de solução. Chen (2007) conclui que a qualidade das soluções pode ser devido à inclusão dos limitantes superiores para o número de concentradores.

De acordo com Alumur e Kara (2008) as melhores heurísticas até o momento para o problema de localização de concentradores não-capacitado são as propostas por Chen (2007) e Cunha e Silva (2007). As melhores soluções relatadas no Capítulo 5 são apresentadas no trabalho de Chen (2007).

No caso do problema de localização de concentradores capacitado, o modelo matemático pode ser descrito pela mesma função-objetivo (2.9) sujeito às restrições (2.3), (2.4) e (2.5), porém com a inclusão das restrições de capacidade, dadas pela equação (2.10):

$$\sum_i E_i x_{ik} \leq Q_k x_{kk} \quad \forall\, k \in V \tag{2.10}$$

em que Q_i é a capacidade do concentrador *i.* Deve-se observar que as restrições (2.10) restringem a alocação dos nós de demanda à capacidade máxima de cada *h* concentrador do problema.

Ernst e Krishnamoorthy (1999), através de algumas alterações no modelo proposto pelos mesmos autores para o USApHMP, apresentaram um modelo matemático com menos variáveis e restrições para o problema capacitado:

$$f(x) = Min \sum_i \sum_k d_{ik} x_{ik} (\lambda E_i + \delta S_i) + \sum_i \sum_k \sum_l \alpha\, d_{kl} Y^i{}_{kl} + f_k x_{kk} \tag{2.11}$$

sujeito a: (2.3), (2.4), (2.5), (2.7), (2.8) e (2.10).

Ernst e Krishnamoorthy (1999) propõem dois algoritmos heurísticos para o problema capacitado: o primeiro baseia-se no método *simulated annealing* e o segundo em um método de descida randômico. Tal método consiste em gerar soluções vizinhas aleatoriamente e só aceitar movimentos de melhora. Com o apoio de um método do tipo *branch-and-bound* e com limitantes superiores providos pelas heurísticas os autores conseguiram obter soluções ótimas para problemas AP definidos em redes de até 50 nós, com exceção do teste em que n = 50 e os custos fixos e a capacidade são do tipo apertados.

Neste trabalho são propostos métodos heurísticos para a solução do problema de localização de concentradores não-capacitado (USAHLP) e para o problema de localização de *p* concentradores (USApHMP).

2.2 – Exemplares do Problema Disponíveis na Literatura

Os conjuntos de dados de teste CAB e AP para problemas de localização de concentradores têm sido referenciados em diversos trabalhos (SASAKI E FUKUSHIMA, 2002; SILVA, 2004; TOPCUOGLU ET AL, 2005; e CHEN, 2007, 2008). A utilização destes conjuntos de dados de teste tem facilitado a comparação de diferentes abordagens propostas para o problema. A verificação da eficácia e eficiência dos métodos heurísticos propostos neste trabalho faz uso destes conjuntos de dados de teste.

O conjunto CAB (*Civil Aeronautics Board*) foi apresentado por O'Kelly (1987). Neste conjunto, os problemas estão baseados no fluxo aéreo de passageiros entre as 25 maiores cidades dos Estados Unidos no ano de 1970. Os dados de teste disponíveis neste conjunto se agrupam em problemas definidos em redes de tamanhos $n = \{10, 15, 20, 25\}$.

Ernst e Krishnamoorthy (1996) tornaram disponível o conjunto AP (*Australian Post*), derivado do fluxo de *e-mails* de Sydney, Austrália. Este conjunto consiste de problemas definidos em redes de 200 nós que representam distritos postais.

Uma diferença importante entre estes dois conjuntos de dados de teste está no fato de a matriz de fluxos do conjunto AP não ser simétrica, como a do conjunto CAB.

3. A BUSCA POR AGRUPAMENTOS

O método de busca evolutiva por agrupamentos (ou ECS - *Evolutionary Clustering Search*), proposto por Oliveira e Lorena (2004, 2007), pode ser definida como uma metaheurística que se baseia no agrupamento (*clusters*) de soluções geradas por um determinado algoritmo e na busca local dentro dos *clusters* mais promissores.

Esse método surgiu com a idéia de que um algoritmo evolutivo, após algumas iterações, sempre tende a encontrar uma concentração maior de indivíduos com maior função de adaptação em determinadas áreas do espaço de soluções do problema. Devido a isso, um processo de agrupamento de indivíduos é realizado pelo algoritmo evolutivo. Com isso, pode-se analisar os grupos de maior incidência para que sejam melhor explorados através de uma busca local.

Chaves (2009) propôs uma generalização do método ECS e, devido a isto, o nome da técnica foi simplificado para busca por agrupamentos (CS, *Clustering Search*). Um *cluster c* é caracterizado por uma tripla $c = (C, \gamma, r)$, em que: C é a solução que representa o centro do *cluster c*, γ representa a quantidade de soluções pertencentes ao *cluster c* e r é uma variável de controle que armazena o número de vezes consecutivas que a busca local foi aplicada ao *cluster C* e não melhorou a solução.

O método de busca por agrupamentos consiste basicamente de três componentes: um gerador de soluções factíveis, um processo de agrupamento e um método de busca local.

O gerador de soluções pode ser qualquer heurística ou metaheurística capaz de gerar soluções com diversidade. Sua execução não depende dos outros componentes. No entanto, o algoritmo gerador de soluções deve garantir que as soluções serão geradas continuamente para o processo de agrupamento.

Para este trabalho foram escolhidas duas metaheurísticas para a geração de soluções:

- um algoritmo genético, devido a seus mecanismos evolutivos permitirem uma exploração maior do espaço de busca fazendo com que o método gere soluções variadas para a busca por agrupamento;
- um método de híbrido de busca em lista tabu com *simulated annealing*, devido aos bons resultados obtidos em Chen (2007), para o problema em questão.

O processo de agrupamento consiste em agrupar soluções similares dentro de um mesmo *cluster* e criar novos *clusters*, caso não exista algum similar a uma determinada solução. Com o objetivo de limitar o número de *clusters* a serem criados, definiu-se um limitante superior *Nc* para este número. Tal processo também é responsável por uma perturbação (assimilação) no centro de um *cluster* toda vez que uma nova solução é incluída neste agrupamento.

Para que o processo de agrupamento funcione adequadamente, é necessário estabelecer uma métrica de distância entre as soluções. A métrica estabelecida para os problemas de localização de concentradores corresponde ao número de alocações diferentes para os concentradores, entre duas soluções. Assim, pode-se medir a distância entre uma dada solução e o centro (que também corresponde a uma solução) de um *cluster*.

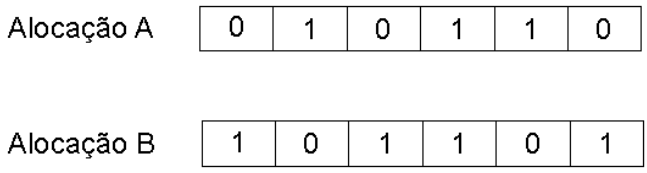

Figura 3.1- Comparação entre alocações de 2 soluções diferentes

Como apresentado na Figura 3.1, o cálculo da distância entre duas soluções distintas é realizado comparando-se o número de alocações diferentes. Pelo exemplo mostrado na figura é possível verificar que a distância entre as alocações das soluções A e B é igual a 5, pois a alocação A se diferencia da solução B em 5 posições do vetor.

No processo de assimilação utiliza-se um método de busca local. Neste trabalho, utilizou-se o método *path-relinking* (GLOVER, 1996), que realiza movimentos

exploratórios na trajetória que interconecta uma solução gerada pela metaheurística e o centro de um *cluster*.

Na Figura 3.2, ilustra-se o funcionamento do método *path-relinking* utilizado. Neste caso, tem-se uma rede com 4 nós. Cada solução é representada por um vetor *v* tal que v_i = 0, se o nó *i* corresponde a um nó de demanda e v_i = 1, se o nó *i* corresponde a um concentrador. Para a aplicação do método deve-se, inicialmente, gerar um conjunto de soluções vizinhas à solução inicial. Para o *path-relinking*, as soluções vizinhas foram obtidas trocando-se um valor de v_i da solução inicial pelo correspondente v_i da solução guia, obtida pela metaheurística. Com isto, foram obtidas 4 novas soluções, mostradas no primeiro nível da Figura 3.2. O método escolhe então uma dessas novas soluções, neste caso, a de menor custo, que corresponde à solução (1, 1, 0, 1). Para esta solução escolhida, aplica-se o mesmo procedimento de troca de elementos com a solução guia, gerando novas soluções. Para este exemplo, foram geradas 3 novas soluções, mostradas no segundo nível da Figura 3.2. Este procedimento prossegue até que a solução guia seja encontrada.

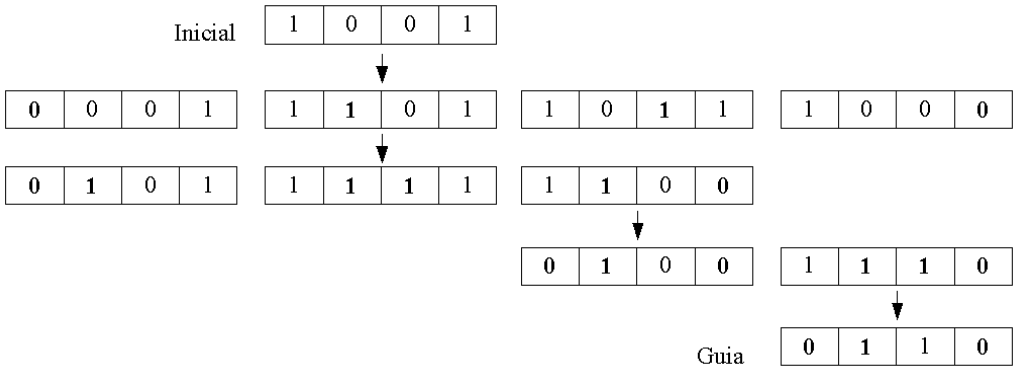

Figura 3.2 – Exemplo de *path-relinking* aplicado ao USAHLP

Com este método, define-se um “caminho” entre uma solução inicial e uma solução guia. A melhor solução encontrada em qualquer nível deste caminho é utilizada, para atualizar o centro do *cluster*.

Considerando γ_i como o número de soluções pertencentes ao cluster *i* e μ como o limite para que um *cluster* torne-se promissor, após a identificação de um *cluster* promissor, ou seja, um *cluster* em que $\gamma_i > \mu$ (para um dado valor de μ), deve-se verificar se o índice de ineficácia *r* excedeu o limite (r_{max}) de execuções sem melhora. Caso isto tenha ocorrido, o centro *C* do *cluster* deve sofrer um processo de perturbação. Essa perturbação tem como principal objetivo fazer com que o centro *C* escape de um mínimo local. Caso o valor de *r* não exceder seu limite de execuções sem melhora, o centro *C* é passado para o método de busca local que se encarrega de explorar a melhor vizinhança do agrupamento, procurando encontrar a melhor solução.

Na Figura 3.3, ilustra-se um resumo do funcionamento do método de busca por agrupamentos, descrito neste capítulo (mais detalhes sobre o funcionamento do CS podem ser vistos nos Apêndices A e B).

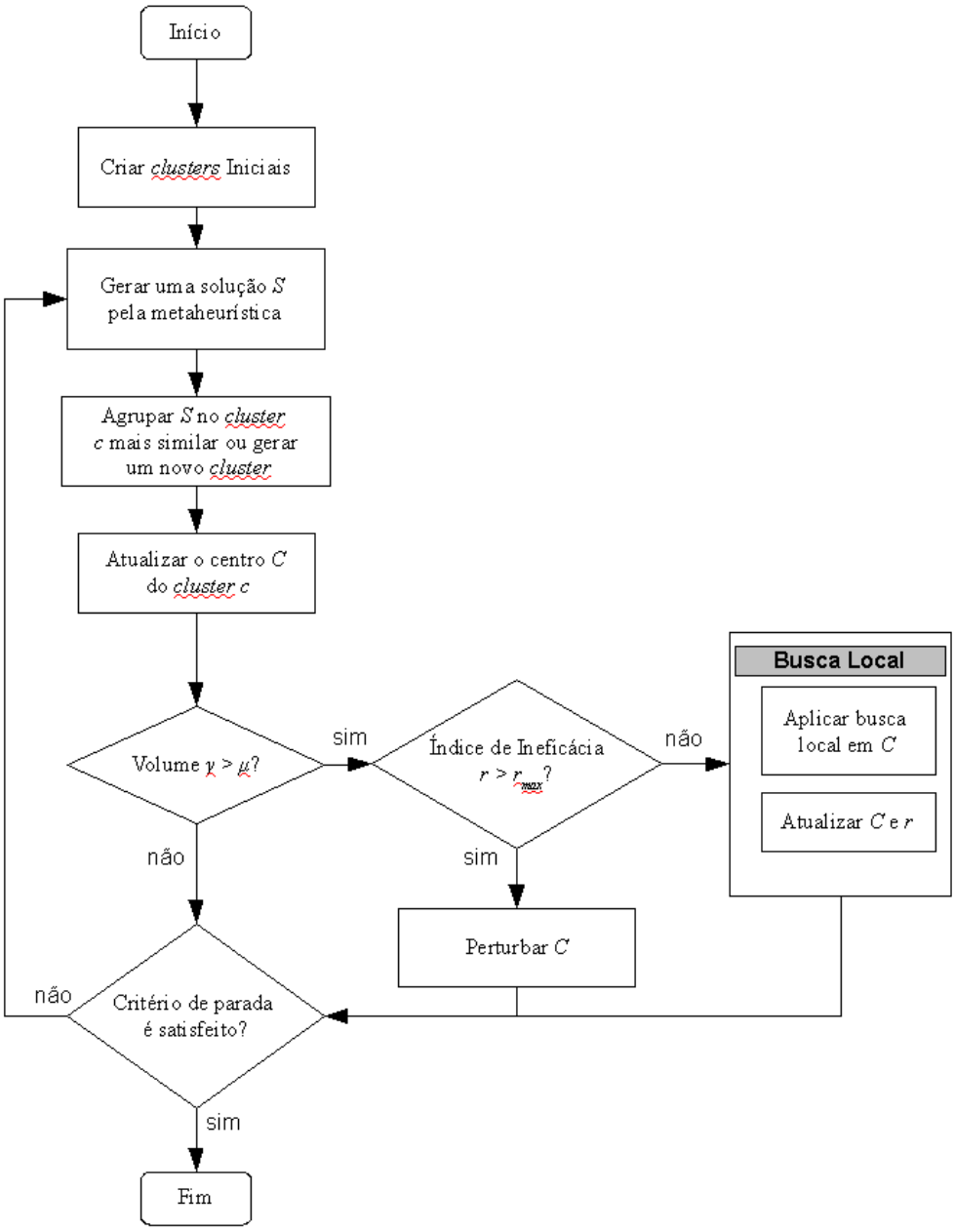

Figura 3.3 – Diagrama conceitual do CS

As Figuras 3.4, 3.5, 3.6 e 3.7 mostram um exemplo de como acontece o processo de criação de *clusters* e de assimilação de uma solução a um *cluster* já existente. Deve-se observar que, nestas figuras, cada ponto corresponde a uma solução, os *clusters* são representados por círculos e que no processo de agrupamento o mais importante é o centro do agrupamento (solução de menor custo do *cluster*), por isso, as soluções que

não são aproveitadas para a atualização do cluster são perdidas durante o processo de agrupamento.

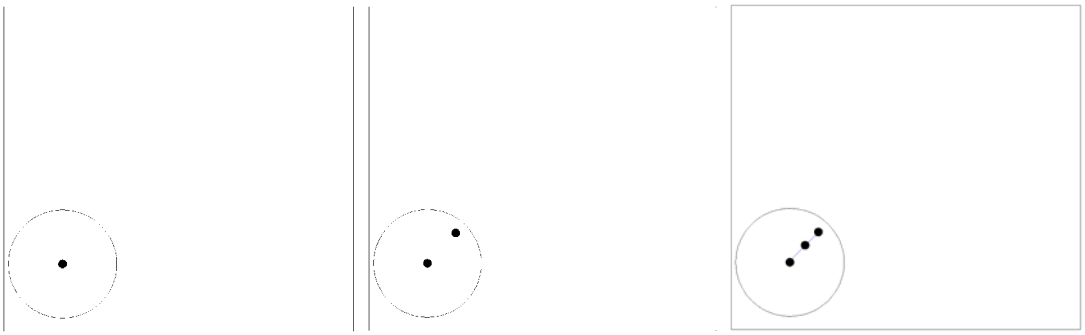

Figura 3.4 – Formação dos *clusters* nos instantes 1 a 3

Pelo exemplo da Figura 3.4, podemos perceber que, quando o método inicia, a primeira solução torna-se um *cluster*. Em seguida, com a próxima solução gerada, determinam-se as distâncias entre essa solução e os *clusters* já existentes, com o objetivo de incluir esta solução no *cluster* mais similar, caso exista. Neste caso, como a distância entre a nova solução e o centro do *cluster* 1 é menor do que o raio do cluster, considera-se que esta nova solução faz parte do *cluster* 1. O processo de assimilação desta nova solução corresponde a aplicar o algoritmo *path-relinking*. Como no caminho que interconecta a nova solução e o centro do *cluster* 1 foi descoberta uma solução com função-objetivo menor, o centro do *cluster* 1 é atualizado.

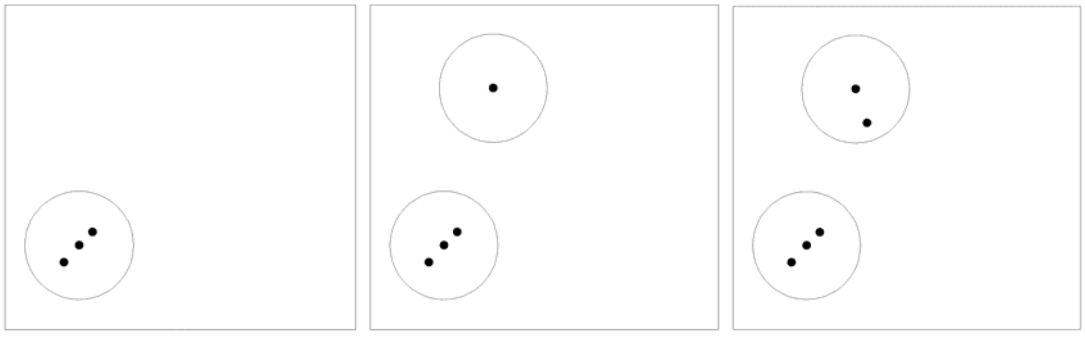

Figura 3.5 – Formação dos *clusters* nos instantes 4 a 6

Quando uma nova solução não encontra um *cluster* similar, devido à distância entre essa nova solução e o centro dos *clusters* já existentes ser maior do que o raio dos *clusters*,

essa nova solução transforma-se no centro de um novo *cluster*, como mostra a Figura 3.5. Note que no instante 5 um novo *cluster* é criado e no instante 6, uma nova solução é assimilada ao *cluster* 2.

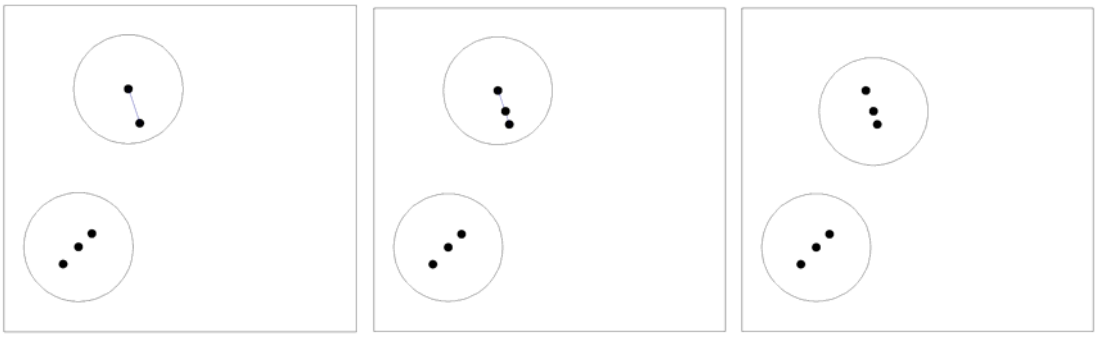

Figura 3.6 – Formação dos *clusters* nos instantes 7 a 9

A Figura 3.6 mostra a atualização do *cluster* 2 devido à assimilação de novas soluções e à atualização do centro do agrupamento.

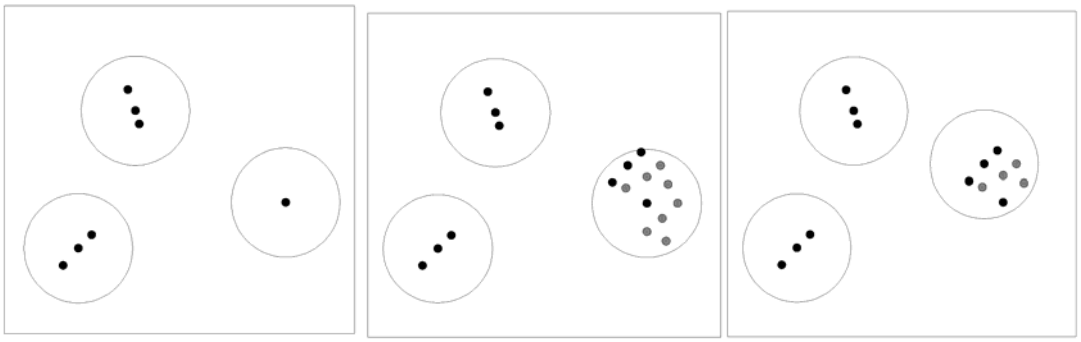

Figura 3.7 – Formação dos *clusters* nos instantes 10, 22 e 23

Na Figura 3.7 mostra-se a criação de um terceiro *cluster*, porque uma nova solução não se assimilou a nenhum dos *clusters* existentes, e à medida que novas soluções foram geradas, o volume do *cluster* 3 aumentou a ponto deste *cluster* ser considerado promissor. Com isso, o método de busca local é aplicado a este *cluster* e o centro do *cluster* é atualizado com base na melhor solução encontrada.

4. ABORDAGENS PROPOSTAS

Como foi discutido no Capítulo 3, a aplicação da busca por agrupamentos requer um algoritmo capaz de gerar soluções factíveis para o processo de agrupamento do CS. Neste capítulo, são apresentados os algoritmos propostos para a geração de soluções para o problema de localização de concentradores e sua aplicação à metaheurística CS.

4.1 – O Algoritmo Genético Proposto

Os algoritmos genéticos (AGs) podem ser definidos como heurísticas de busca inspiradas na teoria da evolução, capazes de encontrar boas soluções para um problema de Otimização Combinatória. Tal método, introduzido por Holland (1975), baseia-se em determinar em uma população de indivíduos (possíveis soluções para o problema), aqueles que, por serem mais adaptados, irão se reproduzir e gerar descendentes para novas gerações (ALMEIDA, 2006).

Algoritmos genéticos tradicionais podem ser descritos, resumidamente, pelos seguintes passos:

1. Crie uma população inicial aleatória de indivíduos (ou cromossomos) com tamanho N_p.
2. Aplique a função de avaliação (*fitness*) a cada indivíduo, obtendo o valor da função objetivo para cada um.
3. Gere novos indivíduos a partir da população atual, através de operadores evolutivos como: *crossover*, mutação e elitismo.
4. Aplique a função de avaliação aos novos indivíduos.
5. Selecione os indivíduos mais adaptados, ou seja, os que geram menor custo para a função objetivo, e elimine os indivíduos de menor adaptação na população.
6. Se o número de gerações atingiu o critério de parada, retorne a melhor solução obtida. Caso contrário, volte ao passo 3.

Para o problema considerado neste trabalho, a representação dos indivíduos baseia-se na proposta de Topcuoglu et al (2005), e corresponde a dois vetores de tamanho *n*: um para armazenamento dos concentradores e outro em que são guardadas as associações dos nós nós de demanda aos concentradores. Nestes vetores, denominados *HubArray* e *AssignArray*, cada posição corresponde a um nó da rede. O *HubArray* corresponde a um vetor binário em que cada posição armazena o valor 0, no caso do nó correspondente a esta posição ser um nó de demanda, ou 1, no caso deste nó ser um concentrador. O *AssignArray* equivale a um vetor em que cada posição armazena o índice do concentrador, ao qual o nó correspondente está associado, como mostra a Figura 4.1.

HubArray	0	0	1	0	1	0	0	1	0	0
AssignArray	8	5	3	3	5	5	3	8	3	8

Figura 4.1 – Exemplo de representação de um indivíduo

No exemplo da Figura 4.1 tem-se uma solução que considera os nós 3, 5 e 8 de uma rede de 10 nós como concentradores, sendo que alocados ao nó 3 estão os nós 4, 7 e 9, alocados ao nó 5 estão os nós 2 e 6, e alocados ao nó 8 estão os nós 1 e 10.

A população inicial é gerada aleatoriamente, sendo que o número de concentradores deve respeitar um limitante superior, como no trabalho de Chen (2007).

O método de cruzamento (*crossover*) escolhido para este trabalho foi o de dois pontos, porque sua natureza de funcionamento favorece a exploração do espaço de soluções, tornando a convergência mais promissora (SILVA, 2004).

O operador de *crossover* consiste no sorteio de dois pontos de cruzamento e na troca dos materiais genéticos de dois indivíduos, como mostrado na Figura 4.2. Os genes que estão entre os dois pontos de corte do primeiro cromossomo, unem-se com os que estão antes e depois do primeiro e segundo pontos de corte no outro cromossomo. Isto é

realizado tanto para o *HubArray* quanto para o *AssignArray*. Após a troca dos genes, um operador se responsabiliza em verificar se o nó de demanda *i*, no *AssignArray*, está associado a um concentrador válido do *HubArray*, caso contrário, este nó de demanda deve ser realocado ao seu concentrador mais próximo.

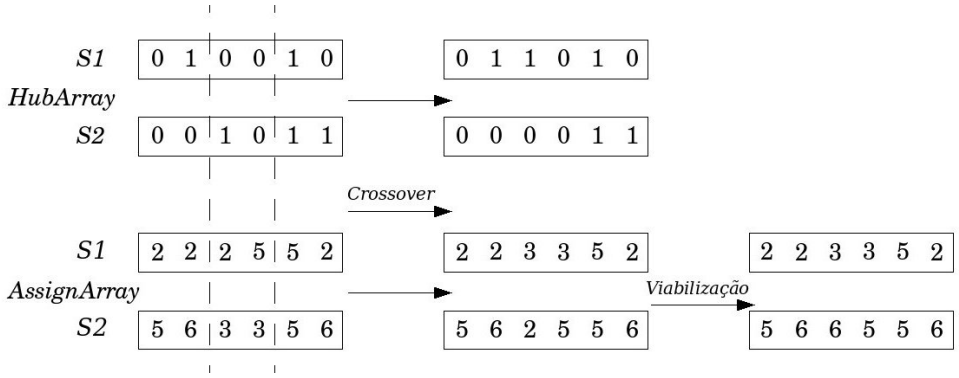

Figura 4.2 – *Crossover* de 2 pontos

O método de seleção utilizado foi o de seleção por *Ranking* Linear, que ordena os indivíduos de acordo com seu grau de adaptação (*fitness*), deixando o indivíduo de menor custo na posição 1 e o de maior custo na posição *n*. A probabilidade de seleção é designada a cada indivíduo linearmente, de acordo com sua posição no *ranking* (SILVA, 2004).

O operador de mutação implementado neste trabalho é composto por três fases, denominadas *shift*, *exchange* e uma busca local. Os dois primeiros são uma versão estendida de parte do movimento de busca tabu usada por Abdinnour-Helm (1998), onde:

- O *shift* sorteia um nó de demanda (associado a um determinado concentrador) e o associa a um outro concentrador. Se o cromossomo possuir apenas um concentrador, essa função não é aplicada.
- A função *exchange* seleciona dois nós de demanda aleatoriamente e troca suas associações. Um pré-requisito para esse operador é a existência de pelo menos dois concentradores e dois nós de demanda. Caso houver apenas um concentrador, ou um único nó de demanda, esse método não é executado.

Para o método de busca local utilizou-se uma metaheurística *simulated annealing* (SA). A idéia desta metaheurística consiste em simular o recozimento de metais. O processo começa com a temperatura inicial elevada e aos poucos a temperatura vai diminuindo de acordo com uma taxa de resfriamento qualquer. Essa característica permite que o algoritmo aceite diversos movimentos de piora no espaço de busca. À medida que a temperatura diminui, o método tende a se estabilizar em uma determinada região do espaço de busca.

Esta metaheurística também foi utilizada como algoritmo de busca local pelo CS. Um pseudocódigo desta metaheurística é apresentado na Figura 4.3.

```
Algoritmo Busca Local
    S* ← x {Melhor Solução obtida}
    Temperatura ← To {Temperatura corrente}
    Enquanto o critério de parada não for satisfeito faça
        Gere uma solução x' aleatoriamente na vizinhança definida por N(x).
        d ← f(x') – f(x)
        Se d < 0 então
            x ← x'
            Se (f(x') < f(x)) então
                S* ← x'
        Senão
            Tome um t qualquer no intervalo [0, 1]
            Se ( t < e −d/Temperatura) então
                S* ← x'
        Fim-se
        Temperatura ← Temperatura * TaxaResfriamento
    Fim-enquanto
Fim-algorimo
```

Figura 4.3 – Método de busca local utilizado pelo CS

As vizinhanças definidas por N(*x*) são:

- Trocar duas alocações;
- Alocar um outro concentrador a um determinado nó de demanda *j*;
- Sortear um nó de demanda *i* e um concentrador *j*, aleatoriamente, fazendo com que o concentrador *j* passe a ser um nó de demanda e o nó de demanda *i* um concentrador. Além disso, todos o nós que estavam associados ao antigo concentrador *j* passam a se conectar ao atual concentrador *i*.
- Transformar um nó de demanda em concentrador, adicionando um novo concentrador à solução corrente.
- Excluir um concentrador da solução, fazendo com que todos os nós associados ao concentrador eliminado passem a se conectar a um dos concentradores restantes, escolhidos aleatoriamente.

O método de busca local, simplesmente escolhe de maneira aleatória uma destas vizinhanças, como mostrado na Figura 4.3. O principal objetivo do método de busca local consiste tentar encontrar uma alocação que gere menor custo para a solução, pois em problemas de localização de concentradores, a alocação dos nós de demanda aos concentradores mais próximos nem sempre levam à solução ótima.

A geração resultante da aplicação dos operadores genéticos, como mostra a Figura 4.4, utiliza o elitismo, mantendo os indivíduos mais adaptados a cada geração. Após esse processo, define-se uma porcentagem de corte, em que todos os cromossomos com função-objetivo abaixo desse valor são eliminados. O processo de eliminação simplesmente sorteia os indivíduos com *fitness* acima do ponto de corte para sofrer *crossover* e mutação, gerando novos descendentes.

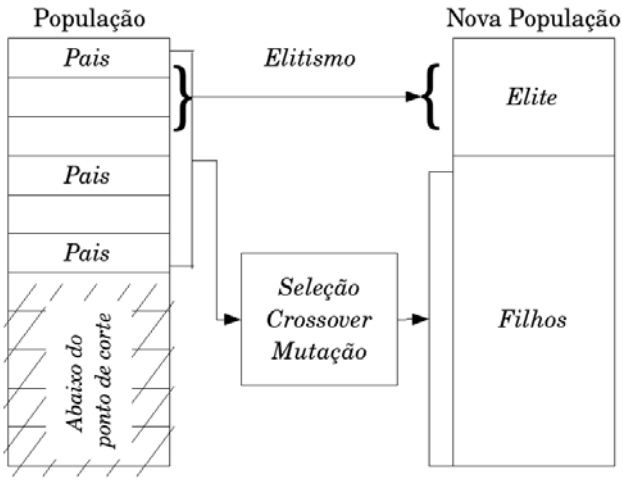

Figura 4.4 – Geração da nova população

O Apêndice A mostra o pseudocódigo completo do CSAG implementado neste trabalho.

4.2 – *Simulated Annealing* com Lista Tabu

O método *simulated annealing* com lista tabu (SATL, do inglês, *Simulated Annealing Tabu List*) consiste de três passos: o primeiro, determina o número de concentradores , o segundo, seleciona a localização desses concentradores, e o terceiro, aloca os nós de demanda aos concentradores escolhidos.

Neste trabalho a solução gerada pelo SATL foi representada da mesma forma que no AG, ou seja, pelo vetor *HubArray*, para armazenamento dos concentradores escolhidos e pelo vetor *AssignArray*, que guarda as associações dos nós de demanda aos seus respectivos concentradores.

Para determinar o número de concentradores, definiu-se um algoritmo para calcular uma estimativa inicial para o número de concentradores. O funcionamento desse algoritmo consiste em gerar uma solução viável qualquer com apenas 1 concentrador inicialmente, calcular os custos fixos e variáveis separadamente, e se a soma dos custos fixos f_j for maior que a soma dos custos variáveis (coleta, transferência e distribuição) tem-se a

estimativa igual a 1. Caso contrário, aumenta-se o número de concentradores de uma unidade, fazendo com que uma nova solução seja gerada e todo o procedimento se repita até que a soma dos custos fixos seja maior que a soma dos custos variáveis, ou o número de concentradores atingir n (número de nós da rede). A Tabela 4.1 mostra a aplicação deste algoritmo para um exemplar do problema com 15 nós, sendo α o custo de transferência entre concentradores e f_k o custo fixo para que o nó k se torne um concentrador.

Tabela 4.1 – Cálculo da estimativa inicial para p (n = 15, α = 0.6, f_k = 150)

P	Custos variáveis	Custo fixo	Custo total
1	1368,42	150,00	1518,42
2	1344,55	300,00	1644,55
3	1059,63	450,00	1509,63
4	972,06	600,00	1572,06
5	1208,76	750,00	1958,76
6	764,86	900,00	1664,86

De acordo com o exemplo da Tabela 4.1, o algoritmo começou os cálculos com uma solução simples com exatamente 1 concentrador. A soma dos custos fixos e variáveis são calculados separadamente, como mostrado na primeira linha da tabela. Como os custos fixos são menores que os custos variáveis, então o método aumenta o número de concentradores de uma unidade e os cálculos são realizados novamente. O processo se repete aumentando o número de concentradores, enquanto os custos variáveis forem maiores que os custos fixos. No caso mostrado na Tabela 4.1, o algoritmo terminou quando p atingiu 6, devido aos custos variáveis (764,86) tornarem menores que os custos fixos (900,00). (Mais detalhes podem ser vistos no pseudo código no Apêndice B).

Após o cálculo da estimativa inicial para o número de concentradores, o SATL procura encontrar a localização dos concentradores que traz o menor custo para a rede. Esse processo funciona basicamente da seguinte forma: primeiro gera-se uma localização

inicial para os *p* concentradores e, em seguida, executam-se diversos movimentos tentando encontrar a melhor alocação.

Com a finalidade de escolher uma boa localização inicial para os concentradores, alguns elementos que aparecem na função objetivo foram incluídos no algoritmo. Considere o fluxo total que entra e sai do nó *i* como sendo $w_i = \sum_j (w_{ij} + w_{ji})$ e o custo total do nó *i* $D_i = \sum_j (d_{ij} + d_{ji})$. Devido aos bons resultados obtidos por Chen (2007), a escolha dos concentradores iniciais é realizada dando-se prioridade aos nós com maior índice I_i ($I_i = w_i / \sum_j w_{ij} - D_i / \sum_j d_{ij}$).

Os movimentos usados pelo SATL, também conhecidos como procedimento SLEP (do inglês, *single location exchange procedure*), consistem basicamente, na troca de exatamente um nó de demanda por um dos concentradores atuais.

Após a definição do número *p* de concentradores e da localização destes, o método procura definir as alocações dos nós de demanda aos concentradores. De acordo com Chen (2007), existe uma tendência que nós de demanda com maior fluxo total (w_i) devem ser alocados aos concentradores mais próximos enquanto que a alocação dos que possuem menor fluxo total é incerto. Neste trabalho, os nós de demanda são inicialmente alocados aos concentradores mais próximos, e em seguida é realizado o seguinte procedimento de melhora:

1. A partir dos nós de demanda com menor fluxo total, realoque-os ao segundo concentrador mais próximo. Se a solução é melhorada, aceite a alocação. Caso contrário, aceite a realocação com uma probabilidade *pr*.
2. Se a melhor solução é melhorada, retorne ao passo 1. Caso contrário, pare o procedimento de melhora.

A lista tabu implementada no método consiste em não aceitar qualquer tipo de movimento que seja um movimento tabu. Neste trabalho a lista tabu contém uma lista com as sete soluções mais atuais. Quando uma solução gerada pela metaheurística é igual a uma das soluções contidas na lista, este movimento é considerado um movimento tabu.

O Apêndice B mostra o pseudocódigo completo do algoritmo CSSATL implementado.

4.3 – Aplicação da Busca por Agrupamentos

Para combinar as metaheurísticas AG e SATL implementadas com a técnica de busca por agrupamentos, após a geração de uma solução por qualquer das metaheurísticas, esta solução deve passar por um processo de assimilação por um dado *cluster*, caso este *cluster* exista, ou seja, caso exista um *cluster* que seja similar à esta solução. Se não existir tal *cluster*, torna-se necessária a criação de um novo *cluster*.

O gráfico da Figura 4.5 mostra a formação dos *clusters* à medida que uma nova solução é encontrada. O eixo horizontal deste gráfico representa as iterações do algoritmo AG. Cada linha horizontal deste gráfico representa um determinado *cluster*. As soluções foram plotadas de acordo com a distância até o centro do seu *cluster* correspondente. Neste caso, considerou-se o raio dos *clusters* como sendo igual a 5.

Pelo gráfico mostrado na Figura 4.5 observa-se que, inicialmente, o AG gerou soluções em diversos locais do espaço de busca, e após um determinado tempo o algoritmo CS optou por explorar os *clusters* 1, 2 e 3. Note que a partir da iteração 400 a maioria das soluções se estabilizaram no *cluster* 1, considerado o mais promissor.

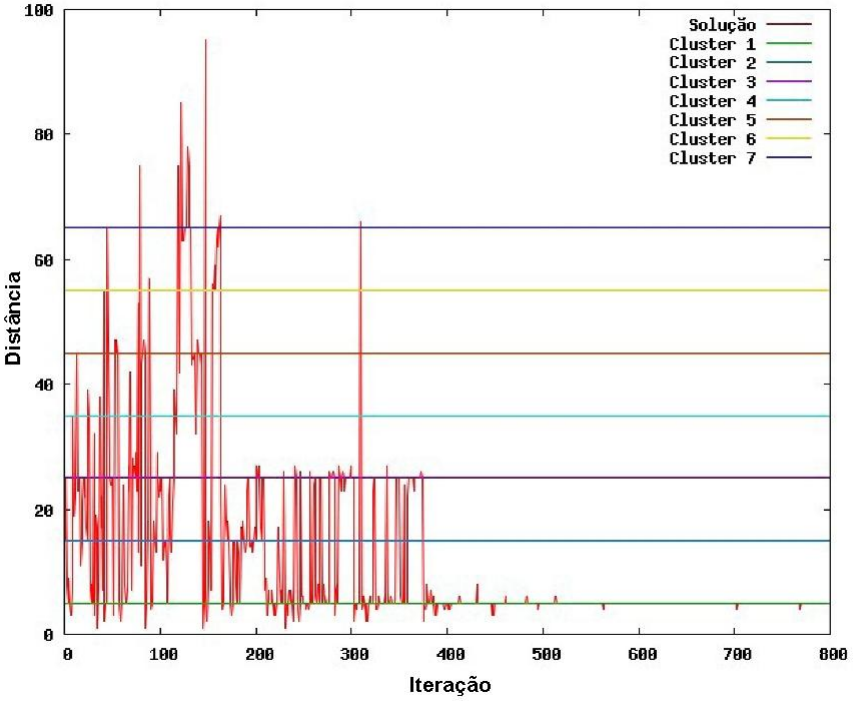

Figura 4.5 – Formação dos *clusters*

Como discutido anteriormente, se o *cluster* é promissor, verifica-se se o índice de ineficácia *r* é maior que o número máximo de vezes que o *cluster* pode ser explorado sem melhora (r_{max}). Caso esta condição se verifique, uma perturbação no centro do *cluster* é realizada.

O processo de perturbação escolhe dentre os *n* nós da rede, *p* nós aleatoriamente e, em seguida, os $n - p$ nós restantes são alocados randomicamente a um dos concentradores escolhidos, com isso, uma nova solução e gerada.

Quando um agrupamento é considerado promissor, neste *cluster* deve ser feito uma busca local, para que a melhor solução naquela vizinhança seja encontrada. Neste trabalho foi utilizado um algoritmo *simulated annealing*, o mesmo descrito interiormente pelo pseudocódigo apresentado na Figura 4.3.

5. RESULTADOS COMPUTACIONAIS

Os algoritmos apresentados no Capítulo 4 foram escritos na linguagem C e aplicados aos problemas de localização de concentradores não-capacitados USAHLP e USApHMP. Nas próximas sessões serão apresentados os resultados obtidos pelas metaheurísticas implementadas e os parâmetros utilizados por cada uma delas para solucionar o problema.

Para ambas as metaheurísticas implementadas, os testes foram executados em um computador com processador Core2-Duo 2.0 GHz, com 1 GB de memória RAM, sob o sistema operacional Linux.

Os parâmetros de entrada do CS foram os mesmos para os dois métodos testados:

- Raio do cluster: $r = 5$;
- Quantidade de *clusters*: $NC = 20$;
- Quantidade máxima de buscas locais consecutivas sem melhora no agrupamento: $r_{max} = 30$;
- Limite superior para que um *cluster* torne-se promissor: $\mu = 7$.

Devido ao método de busca local realizar exploração apenas nos *clusters* mais promissores o critério de parada utilizado consiste em 100 execuções consecutivas sem melhora.

Para o algoritmo SATL, na busca local do CS utilizou-se a temperatura inicial = 100, submetida a uma taxa de resfriamento igual à 0.9.

As tabelas apresentadas neste capítulo são divididas em duas classes, uma formada pelo conjunto de dados CAB e outra pelo conjunto AP. Para o conjunto AP consideram-se os custos fixos divididos em duas categorias: frouxos e apertados. Para o USAHLP cada algoritmo foi executado 10 vezes, para as duas classes de problemas. Para o

USApHMP, os testes foram realizados para o conjunto AP com custos fixos apertados, para exemplares pequenos e grandes. Nestas tabelas são utilizadas as seguintes legendas para as colunas:

- **Ótimo** representa a solução ótima conhecida para o problema;
- **Melhor** representa a melhor solução encontrada na literatura;
- **Sol** representa a melhor solução encontrada pelo método implementado;
- **TC** representa a média dos tempos gastos (em segundos) para que a metaheurística (AG, SATL) com ou sem busca por agrupamentos encontre a melhor solução;
- **TE** representa o tempo médio de execução total do algoritmo, até que o critério de parada seja alcançado;
- **Desv** corresponde à variação das soluções encontradas em relação à média;
- ***Gap*** corresponde ao erro da melhor solução encontrada pela metaheurística em relação à melhor solução encontrada na literatura;
- **Var** corresponde ao erro das soluções encontradas pelas metaheurísticas (AG ou SATL) em relação às soluções encontradas pelo método CS.
- **QU** define qual método encontrou a solução. (BL – Busca Loca, PR – *Path-relinking*, MH – Metaheurística, CS – *Clustering Search* ou AL – Procedimento de alocação da metaheurística).

Os valores de **Desv**, ***Gap*** e **Var** foram calculados com o objetivo de analisar a robustez e eficácia dos métodos implementados. Os valores de Desv, *Gap* e Var são calculados pelas equações (5.2), (5.3) e (5.4), em que $Sol_{média}$ corresponde à solução média, Sol_{MH} corresponde à solução da metaheurística (AG ou SATL), e Sol_{CS} corresponde à solução encontrada pelo algoritmo CS (combinado com a metaheurística correspondente).

$$Desv = 100 \times \frac{(Sol_{media} - Sol)}{Sol} \tag{5.2}$$

$$Gap = 100 \times \frac{(Sol - Melhor)}{Melhor} \tag{5.3}$$

$$Var = 100 \times \frac{(Sol_{MH} - Sol_{CS})}{Sol_{CS}} \quad (5.4)$$

Nas tabelas relativas ao conjunto de dados CAB, α é o custo de transferência entre concentradores, e f o custo fixo de instalação de um determinado concentrador.

5.1 – Soluções para o USAHLP

5.1.1 - Resultados obtidos pelo AG

Inicialmente, os parâmetros do algoritmo foram definidos de acordo com o trabalho de Topcuoglu et al (2005), porém no decorrer do trabalho observou-se que os parâmetros pré-definidos não eram suficientes para fazer com que o AG alcançasse boas soluções, por isso, com base em experimentos computacionais, alguns parâmetros foram modificados. Os valores adotados atualmente são:

- O tamanho da população: 100;
- A probabilidade de *crossover*: 0,80;
- A probabilidade de mutação: 0,10;
- A porcentagem de elite: 0,20;
- O número de gerações: 100.

Para o método de busca local do AG, foram utilizados os seguintes parâmetros:

- Critério de parada igual a 10 execuções consecutivas sem melhora;
- Temperatura inicial igual a 100;
- Taxa de resfriamento igual a 0,9.

Nas Tabelas 5.1 a 5.4 são apresentados os resultados obtidos pelo algoritmo genético com e sem a busca por agrupamentos. As Tabelas 5.1 e 5.2 apresentam os resultados

encontrados pelo AG para o conjunto de dados CAB com n = 20 e n = 25 nós, respectivamente . As Tabelas 5.3 e 5.4 mostram os resultados obtidos pelo AG para o conjunto de dados AP. Na Tabela 5.3 tem-se os resultados para custos fixos frouxos e na Tabela 5.4, os resultados para custos fixos apertados.

Tabela 5.1 – Conjunto CAB: CS × AG

***n* = 20**														
α	**F**	**Ótimo**	**CSAG**					**AG**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
0,20	100	967,74	967,74	0,00	0,00	0,16	BL	967,74	0,00	0,00	0,15	MH	0,23	0,00
	150	1174,53	1174,54	0,00	0,00	0,06	BL	1174,54	0,00	0,00	0,08	BL	0,16	0,00
	200	1324,53	1324,54	0,00	0,00	0,08	BL	1324,54	0,00	0,00	0,07	MH	0,15	0,00
	250	1474,53	1479,09	0,34	0,31	0,08	BL	1495,74	0,00	1,44	0,04	MH	0,12	1,13
0,40	100	1127,09	1131,41	0,42	0,38	0,11	BL	1131,41	0,47	0,38	0,09	MH	0,16	0,00
	150	1297,76	1297,77	0,00	0,00	0,08	BL	1297,77	0,00	0,00	0,08	BL	0,17	0,00
	200	1442,56	1442,57	0,00	0,00	0,14	BL	1442,57	0,00	0,00	0,13	BL	0,17	0,00
	250	1542,56	1542,57	0,37	0,00	0,05	BL	1542,57	0,37	0,00	0,02	BL	0,10	0,00
0,60	100	1269,15	1271,00	0,00	0,15	0,08	BL	1271,00	0,00	0,15	0,07	BL	0,15	0,00
	150	1406,04	1406,04	0,75	0,00	0,09	BL	1406,04	0,75	0,00	0,05	BL	0,12	0,00
	200	1506,04	1520,92	0,00	0,99	0,00	PR	1520,92	0,00	0,99	0,00	BL	0,08	0,00
	250	1570,91	1570,92	0,00	0,00	0,00	PR	1570,92	0,00	0,00	0,00	BL	0,08	0,00
0,80	100	1369,52	1369,52	0,44	0,00	0,14	BL	1369,52	0,46	0,00	0,11	BL	0,20	0,00
	150	1469,52	1470,92	0,00	0,10	0,00	PR	1470,92	0,00	0,10	0,00	BL	0,09	0,00
	200	1520,91	1520,92	0,00	0,00	0,00	PR	1520,92	0,00	0,00	0,00	BL	0,09	0,00
	250	1570,91	1570,92	0,00	0,00	0,00	PR	1570,92	0,00	0,00	0,00	BL	0,08	0,00
1,00	100	1410,07	1420,92	0,00	0,77	0,00	PR	1420,92	0,00	0,77	0,00	BL	0,09	0,00
	150	1470,91	1470,92	0,00	0,00	0,00	PR	1470,92	0,00	0,00	0,00	BL	0,09	0,00
	200	1520,91	1520,92	0,00	0,00	0,00	PR	1520,92	0,00	0,00	0,00	BL	0,08	0,00
	250	1570,91	1570,92	0,00	0,00	0,00	PR	1570,92	0,00	0,00	0,00	BL	0,08	0,00
Média				0,12	0,13	0,05			0,10	0,19	0,04		0,12	0,06

Tabela 5.2 – Conjunto CAB: CS × AG

***n* = 25**														
α	**F**	**Ótimo**	**CSAG**					**AG**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
0,20	100	1029,63	1029,63	0,00	0,00	0,25	BL	1029,63	0,21	0,00	0,23	MH	0,33	0,00
	150	1217,34	1217,35	1,40	0,00	0,16	BL	1217,35	3,43	0,00	0,09	BL	0,24	0,00
	200	1367,34	1367,35	0,34	0,00	0,13	BL	1367,35	1,47	0,00	0,11	MH	0,23	0,00
	250	1500,90	1500,91	0,00	0,00	0,15	BL	1500,91	0,00	0,00	0,06	MH	0,17	0,00
0,40	100	1187,51	1201,70	0,04	1,19	0,21	BL	1201,70	0,26	1,19	0,18	MH	0,31	0,00
	150	1351,69	1351,70	1,14	0,00	0,18	BL	1351,70	1,48	0,00	0,15	BL	0,26	0,00
	200	1501,62	1501,63	0,00	0,00	0,12	BL	1501,63	0,00	0,00	0,06	BL	0,20	0,00
	250	1601,62	1601,63	2,60	0,00	0,06	BL	1601,63	2,60	0,00	0,05	BL	0,15	0,00
0,60	100	1333,56	1333,56	0,01	0,00	0,27	BL	1333,56	0,02	0,00	0,23	BL	0,35	0,00
	150	1483,56	1483,56	0,71	0,00	0,17	BL	1483,56	0,71	0,00	0,11	BL	0,23	0,00
	200	1601,20	1690,58	0,00	5,58	0,00	MH	1690,58	0,00	5,58	0,00	BL	0,12	0,00
	250	1701,20	1740,58	0,00	2,31	0,01	MH	1740,58	0,00	2,31	0,00	BL	0,12	0,00
0,80	100	1458,83	1458,83	0,72	0,00	0,15	BL	1458,83	0,72	0,00	0,10	BL	0,23	0,00
	150	1594,08	1594,08	2,35	0,00	0,01	PR	1594,08	2,62	0,00	0,01	BL	0,14	0,00
	200	1690,57	1690,58	0,00	0,00	0,00	PR	1690,58	0,00	0,00	0,00	BL	0,13	0,00
	250	1740,57	1740,58	0,00	0,00	0,01	PR	1740,58	0,00	0,00	0,00	BL	0,13	0,00
1,00	100	1556,63	1562,16	1,64	0,36	0,02	PR	1562,16	1,64	0,36	0,01	BL	0,15	0,00
	150	1640,57	1640,58	0,00	0,00	0,00	PR	1640,58	0,00	0,00	0,00	BL	0,13	0,00
	200	1690,57	1690,58	0,00	0,00	0,00	PR	1690,58	0,00	0,00	0,00	BL	0,14	0,00
	250	1740,57	1740,58	0,00	0,00	0,00	PR	1740,58	0,00	0,00	0,00	BL	0,12	0,00
Média				0,55	0,47	0,10			0,76	0,47	0,07		0,19	0,00

Tabela 5.3 – Conjunto AP, custos fixos frouxos: CS × AG

n	Melhor	CSAG					AG					TE	Var
		Sol	Desv	Gap	TC	QU	Sol	Desv	Gap	TC	QU		
10	224250,05	224250,05	0,00	0,00	0,04	BL	224250,05	0,00	0,00	0,03	MH	0,07	0,00
20	234690,96	234690,96	0,00	0,00	0,12	BL	234690,96	0,00	0,00	0,12	MH	0,19	0,00
25	236650,63	236650,63	0,00	0,00	0,25	BL	236650,63	0,00	0,00	0,27	MH	0,38	0,00
40	240986,24	240986,23	0,12	0,00	1,08	BL	240986,23	0,12	0,00	1,02	MH	1,55	0,00
50	237421,99	237421,99	0,00	0,00	1,84	CS	237421,99	0,16	0,00	1,88	BL	2,28	0,00
100	238015,38	238016,28	1,21	0,00	18,92	PR	238016,28	1,21	0,00	19,05	BL	21,32	0,00
200	228944,18	228947,08	1,49	0,00	213,63	CS	228947,08	1,49	0,00	214,25	BL	230,30	0,00
Média			0,40	0,00	33,70			0,43	0,00	33,80		36,58	0,00

Tabela 5.4 – Conjunto AP, custos fixos apertados: CS × AG

n	Melhor	CSAG					AG					TE	Var
		Sol	Desv	Gap	TC	QU	Sol	Desv	Gap	TC	QU		
10	263399,95	263399,94	0,00	0,00	0,03	BL	263399,94	0,00	0,00	0,01	BL	0,05	0,00
20	271128,18	271128,18	0,78	0,00	0,06	CS	271128,18	2,60	0,00	0,00	BL	0,12	0,00
25	295667,84	295667,84	0,00	0,00	0,06	BL	295667,84	0,00	0,00	0,00	BL	0,16	0,00
40	293164,83	293164,84	0,00	0,00	0,04	PR	293164,84	0,00	0,00	0,01	BL	0,43	0,00
50	300420,98	300420,99	0,00	0,00	0,12	BL	300420,99	0,00	0,00	0,03	BL	0,79	0,00
100	305096,76	305097,95	0,00	0,00	1,21	BL	305097,95	0,00	0,00	0,11	BL	4,19	0,00
200	233537,33	233540,24	10,14	0,00	59,24	CS	233540,24	10,14	0,00	56,53	BL	78,63	0,00
Média			1,56	0,00	8,68			1,82	0,00	8,10		12,05	0,00

Analisando os resultados da Tabela 5.1 pela coluna **Var** é possível verificar que em dois casos o método CS conseguiu melhorar a solução gerada pelo AG. No caso das Tabelas 5.2, 5.3 e 5.4, o método CS encontra as mesmas soluções geradas pelo AG, mas pela coluna **Desv** nota-se que o CSAG mostrou-se mais robusto em relação ao AG na maioria dos casos testados. Tal fato foi comprovado através do cálculo do desvio médio total em que o CSAG obteve um desvio médio total igual a 0,94 enquanto que para o AG sem busca por agrupamentos o desvio médio é de 0,98.

Tabela 5.5 – Resumo CS x AG (USAHLP)

	CSAG	AG
Desvio médio total	0,50	0,61
% das soluções ótimas obtidas	75,00	75,00

Como pode ser verificado na Tabela 5.5, o CSAG obteve um desvio total médio menor que o AG, o que torna o CSAG mais robusto em relação ao AG.

5.1.2 – Resultados obtidos pelo SATL

Nas Tabelas de 5.6 a 5.9 são apresentados os resultados obtidos pelo algoritmo SATL. As Tabelas 5.6 e 5.7 apresentam as soluções encontradas pelo SATL para o conjunto de dados CAB para $n = 20$ e $n = 25$ nós respectivamente . As Tabelas 5.8 e 5.9 mostram os resultados obtidos pelo SATL para o conjunto de dados AP, com custos fixos frouxos e apertados, respectivamente.

Tabela 5.6 – Conjunto CAB: CS × SATL

***n* = 20**														
α	**F**	**Ótimo**	**CSSATL**					**SATL**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
0,20	100	967,74	977,62	0,53	1,02	1,25	BL	977,62	2,54	1,02	0,10	AL	2,85	0,00
	150	1174,53	1174,53	0,00	0,00	0,12	BL	1174,53	1,54	0,00	0,07	AL	1,07	0,00
	200	1324,53	1324,53	1,35	0,00	0,10	CS	1324,53	1,63	0,00	0,07	AL	0,91	0,00
	250	1474,53	1474,53	0,58	0,00	0,16	CS	1474,53	0,68	0,00	0,04	AL	0,65	0,00
0,40	100	1127,09	1127,09	0,52	0,00	1,00	CS	1136,74	1,72	0,86	0,10	AL	3,01	0,86
	150	1297,76	1297,76	0,46	0,00	0,48	BL	1297,76	1,89	0,00	0,06	AL	1,13	0,00
	200	1442,56	1442,56	0,22	0,00	0,37	BL	1442,56	0,38	0,00	0,04	AL	1,81	0,00
	250	1542,56	1542,56	0,07	0,00	0,06	CS	1542,56	0,15	0,00	0,04	AL	0,41	0,00
0,60	100	1269,15	1269,15	0,61	0,00	0,15	CS	1270,99	1,52	0,14	0,08	AL	12,97	0,14
	150	1406,04	1406,04	0,00	0,00	1,45	CS	1406,04	0,05	0,00	0,04	AL	7,90	0,00
	200	1506,04	1506,04	0,10	0,00	0,11	BL	1506,04	0,10	0,00	0,04	AL	1,18	0,00
	250	1570,91	1570,91	0,00	0,00	0,07	CS	1570,91	0,00	0,00	0,00	AL	0,87	0,00
0,80	100	1369,52	1369,52	0,00	0,00	0,07	CS	1369,52	0,03	0,00	0,04	AL	26,21	0,00
	150	1469,52	1469,52	0,04	0,00	0,07	BL	1469,52	0,04	0,00	0,03	AL	9,77	0,00
	200	1520,91	1520,91	0,00	0,00	0,08	BL	1520,91	0,00	0,00	0,00	AL	10,23	0,00
	250	1570,91	1570,91	0,00	0,00	0,09	CS	1570,91	0,00	0,00	0,00	AL	4,14	0,00
1,00	100	1410,07	1410,07	0,00	0,00	0,14	BL	1410,07	0,28	0,00	0,04	AL	189,22	0,00
	150	1470,91	1470,91	0,00	0,00	0,12	BL	1470,91	0,00	0,00	0,01	AL	13,53	0,00
	200	1520,91	1520,91	0,00	0,00	0,10	CS	1520,91	0,00	0,00	0,00	AL	12,02	0,00
	250	1570,91	1570,91	0,00	0,00	0,07	BL	1570,91	0,00	0,00	0,00	AL	2,95	0,00
Média				0,22	0,05	0,30			0,63	0,10	0,04		15,14	0,05

Tabela 5.7 – Conjunto CAB: CS × SATL

***n* = 25**														
α	**F**	**Ótimo**	**CSSATL**					**SATL**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
0,20	100	1029,63	1029,63	1,19	0,00	2,01	CS	1029,63	2,76	0,00	0,15	AL	3,60	0,00
	150	1217,34	1217,34	0,25	0,00	0,44	CS	1220,55	0,68	0,26	0,11	AL	2,15	0,26
	200	1367,34	1367,34	0,24	0,00	0,23	CS	1367,34	0,80	0,00	0,12	AL	1,22	0,00
	250	1500,90	1500,90	0,00	0,00	0,22	BL	1500,90	0,31	0,00	0,09	AL	0,94	0,00
0,40	100	1187,51	1187,51	0,85	0,00	0,27	BL	1195,99	1,44	0,71	0,16	AL	5,86	0,71
	150	1351,69	1351,69	0,69	0,00	0,22	CS	1361,24	0,63	0,71	0,12	AL	1,61	0,71
	200	1501,62	1501,62	0,07	0,00	0,16	CS	1501,62	0,35	0,00	0,09	AL	1,76	0,00
	250	1601,62	1601,62	0,00	0,00	0,38	CS	1601,62	0,42	0,00	0,07	AL	0,97	0,00
0,60	100	1333,56	1333,99	0,49	0,03	0,27	BL	1341,87	0,52	0,62	0,12	AL	16,95	0,59
	150	1483,56	1483,99	0,39	0,03	1,51	CS	1491,87	0,35	0,56	0,11	AL	2,61	0,53
	200	1601,20	1601,20	0,01	0,00	0,19	BL	1601,20	0,15	0,00	0,08	AL	2,64	0,00
	250	1701,20	1701,20	0,02	0,00	0,15	CS	1701,20	0,95	0,00	0,07	AL	1,24	0,00
0,80	100	1458,83	1458,83	0,66	0,00	0,33	BL	1458,83	1,78	0,00	0,13	AL	51,17	0,00
	150	1594,08	1594,08	0,21	0,00	0,24	BL	1597,51	0,48	0,22	0,08	AL	9,85	0,22
	200	1690,57	1690,57	0,00	0,00	0,16	BL	1690,57	0,00	0,00	0,00	AL	8,65	0,00
	250	1740,57	1740,57	0,00	0,00	0,12	CS	1740,57	0,00	0,00	0,01	AL	4,10	0,00
1,00	100	1556,63	1559,19	0,15	0,16	0,29	BL	1559,19	0,54	0,16	0,08	AL	224,40	0,00
	150	1640,57	1640,57	0,00	0,00	0,14	BL	1640,57	0,00	0,00	0,01	AL	17,86	0,00
	200	1690,57	1690,57	0,00	0,00	0,10	CS	1690,57	0,00	0,00	0,01	AL	10,08	0,00
	250	1740,57	1740,57	0,00	0,00	0,19	BL	1740,57	0,00	0,00	0,00	AL	8,82	0,00
Média				0,26	0,01	0,38			0,61	0,16	0,08		18,82	0,15

Tabela 5.8 – Conjunto AP, custos fixos frouxos: CS × SATL

n	Melhor	CSSATL					SATL					TE	Var
		Sol	Desv	Gap	TC	QU	Sol	Desv	Gap	TC	QU		
10	224250,05	224250,05	0,00	0,00	0,46	BL	224250,05	0,00	0,00	0,03	AL	0,55	0,00
20	234690,96	234690,97	0,00	0,00	1,55	BL	234690,97	0,00	0,00	0,04	AL	2,45	0,00
25	236650,63	236650,63	0,00	0,00	2,66	BL	236650,63	0,08	0,00	0,94	AL	3,55	0,00
40	240986,24	240986,23	0,00	0,00	9,17	BL	240986,23	0,02	0,00	2,81	AL	16,54	0,00
50	237421,99	237421,99	0,00	0,00	17,15	BL	237421,99	0,08	0,00	6,38	AL	30,91	0,00
100	238015,38	238016,28	0,00	0,00	187,71	BL	238016,28	0,63	0,00	54,67	AL	265,46	0,00
200	228944,18	228947,08	0,16	0,00	1007,35	BL	228947,08	1,56	0,00	49,21	AL	1269,47	0,00
Média			0,02	0,00	175,15			0,34	0,00	16,30		226,99	0,00

Tabela 5.9 – Conjunto AP, custos fixos apertados: CS × SATL

n	Melhor	CSSATL					SATL					TE	Var
		Sol	Desv	Gap	TC	QU	Sol	Desv	Gap	TC	QU		
10	263399,95	263399,95	0,00	0,00	0,02	CS	263399,95	0,00	0,00	0,01	AL	0,71	0,00
20	271128,18	271128,18	0,00	0,00	1,56	CS	271128,18	0,00	0,00	0,30	AL	2,55	0,00
25	295667,84	295667,83	0,00	0,00	3,41	BL	295667,83	0,00	0,00	0,01	AL	4,11	0,00
40	293164,83	293164,84	0,00	0,00	13,46	BL	293164,84	0,00	0,00	0,06	AL	19,74	0,00
50	300420,98	300420,99	0,00	0,00	17,12	BL	300420,99	0,00	0,00	0,09	AL	34,64	0,00
100	305096,76	305097,93	0,00	0,00	38,63	CS	305097,93	0,36	0,00	75,23	AL	238,63	0,00
200	233537,33	233540,24	0,13	0,00	321,07	CS	233540,24	3,27	0,00	48,14	AL	673,45	0,00
Média			0,02	0,00	56,47			0,52	0,00	17,69		139,12	0,00

Pelos resultados da Tabela 5.6, é possível notar que o método CS conseguiu melhorar a solução gerada pelo SATL em um dos casos. Além disso, o método algoritmo CSSATL mostrou-se mais robusto que o algoritmo SATL (simplesmente), como se nota pelos valores das colunas **Desv**.

No caso da Tabela 5.7, em quatro casos o CSSATL melhorou as soluções geradas pelo SATL e mostrou-se mais robusto que o SATL (simplesmente) para a maioria dos casos.

Para as Tabelas 5.8 e 5.9, o CS encontrou as mesmas soluções que o SATL, porém como pode ser visto pelas colunas **Desv**, o CS mostrou-se mais robusto na maioria dos casos.

Tabela 5.10 – Resumo CS x SATL (USAHLP)

	CSSATL	SATL
Desvio médio total	0,19	0,61
% das soluções ótimas obtidas	90,00	75,00

Como mostrado na Tabela 5.10, os resultados gerados pelo CSSATL comprovam a robustez do método, uma vez que o desvio total obtido pelo CSSATL foi menor que o gerado pelo SATL. Além disso, a tabela mostra que o CSSATL foi mais eficiente, pelo fato que ter obtido 90% das soluções ótimas enquanto que o SATL obteve 75% destas.

5.2 – Solução para o USApHMP

5.2.1 – Resultados obtidos pelo AG

Os parâmetros utilizados pelo AG desta sessão são os mesmos utilizados para o USAHLP. No entanto, como os exemplares grandes dos problemas AP exigem um tempo computacional maior, cada teste foi executado 5 vezes.

Pela Tabela 5.11, percebe-se que em todos os casos tanto o CSAG quanto o AG, encontraram as soluções ótimas ($Gap = 0$) e, praticamente para todas as execuções ($Desv \cong 0$).

Pela Tabela 5.12 observa-se que o método CSAG obteve soluções bem próximas das melhores soluções conhecidas atualmente. Um caso interessante pode ser notado para o exemplar com $n = 200$ e $p = 5$, que corresponde a uma solução inédita em relação às melhores encontradas na literatura.

Tabela 5.11 – Conjunto AP: Exemplares pequenos

AP com custos fixos apertados														
n	**p**	**Ótimo**	**CSAG**					**AG**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
10	2	167493,06	167493,06	0,00	0,00	0,09	PR	167493,06	0,00	0,00	0,00	BL	0,44	0,00
	3	136008,13	136008,12	0,00	0,00	0,09	PR	136008,12	0,00	0,00	0,01	BL	0,42	0,00
	4	112396,07	112396,07	0,00	0,00	0,10	PR	112396,07	0,00	0,00	0,02	BL	0,39	0,00
	5	91105,37	91105,37	0,00	0,00	0,08	PR	91105,37	0,00	0,00	0,00	BL	0,34	0,00
20	2	172816,69	172816,70	0,00	0,00	0,34	PR	172816,70	0,00	0,00	0,00	BL	1,39	0,00
	3	151533,08	151533,08	0,00	0,00	0,37	PR	151533,08	0,00	0,00	0,01	BL	1,56	0,00
	4	135624,88	135624,88	0,00	0,00	0,60	PR	135624,88	0,00	0,00	0,32	BL	1,66	0,00
	5	123130,09	123130,09	0,00	0,00	0,44	PR	123130,09	0,00	0,00	0,23	BL	1,53	0,00
25	2	175541,98	175541,98	0,00	0,00	0,57	PR	175541,98	0,00	0,00	0,01	BL	2,44	0,00
	3	155256,32	155256,32	0,00	0,00	0,65	PR	155256,32	0,00	0,00	0,05	BL	2,62	0,00
	4	139197,17	139197,17	0,00	0,00	0,71	PR	139197,17	0,00	0,00	0,24	BL	2,90	0,00
	5	123574,29	123574,29	0,00	0,00	0,89	PR	123574,29	0,00	0,00	0,39	BL	2,72	0,00
40	2	177471,67	177471,67	0,00	0,00	1,76	PR	177471,67	0,00	0,00	0,02	BL	6,06	0,00
	3	158830,54	158830,54	0,00	0,00	2,12	PR	158830,54	0,00	0,00	0,16	BL	7,72	0,00
	4	143968,88	143968,88	0,00	0,00	2,66	CS	143968,88	0,00	0,00	0,44	BL	8,63	0,00
	5	134264,95	134264,96	0,00	0,00	3,98	PR	134264,96	0,00	0,00	2,16	BL	9,28	0,00
50	2	178484,29	178484,28	0,00	0,00	3,26	PR	178484,28	0,00	0,00	0,05	BL	10,78	0,00
	3	158569,93	158569,94	0,00	0,00	4,21	CS	158569,94	0,00	0,00	0,17	BL	13,98	0,00
	4	143378,05	143378,04	0,00	0,00	4,45	CS	143378,04	0,00	0,00	0,57	BL	14,97	0,00
	5	132366,95	132366,95	0,00	0,00	9,28	CS	132366,95	0,00	0,00	7,30	BL	17,54	0,00
Média				0,00	0,00	1,83			0,00	0,00	0,61		5,37	0,00

Tabela 5.12 – Conjunto AP: Exemplares grandes

AP com custos fixos apertados														
n	**p**	**Melhor**	**CSAG**					**AG**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
100	5	136929,40	136929,44	0,00	0,00	78,45	CS	136929,44	0,00	0,00	64,18	BL	130,01	0,00
	10	106469,60	106547,95	0,12	0,07	160,68	CS	106547,95	0,12	0,07	152,05	BL	179,38	0,00
	15	90533,52	91159,58	0,52	0,69	142,07	CS	91159,58	0,52	0,69	130,40	BL	153,30	0,00
	20	80270,96	81725,00	0,61	1,81	136,21	CS	81725,00	0,61	1,81	124,25	BL	152,37	0,00
200	5	140175,60	140062,65	0,04	-0,08	945,70	CS	140062,65	0,04	-0,08	877,79	BL	974,92	0,00
	10	110147,70	110286,18	0,14	0,13	1372,79	CS	110286,18	0,14	0,13	1294,88	BL	1415,11	0,00
	15	94496,41	94927,79	0,71	0,46	169,20	CS	94927,79	0,71	0,46	868,34	BL	1234,08	0,00
	20	85129,34	86299,47	0,60	1,37	1442,42	CS	86299,47	0,60	1,37	1268,24	BL	1668,20	0,00
Média				0,34	0,56	555,94			0,34	0,56	597,52		429,90	0,00

Outro fato interessante a ser observado em dois dos testes em que *p*=15, o CSAG encontrou a melhor solução muito mais rápido que o AG, acredita-se que isso aconteceu devido ao método de busca por agrupamentos conter o *path-relinking* responsável por atualizar o cluster caso encontre uma de menor custo entre o caminho que interliga o centro do cluster a uma dada solução.

Tabela 5.13– Resumo CS x AG (USApHMP)

	CSAG	AG
Desvio médio total	0,10	0,10
% das soluções ótimas obtidas	100	100

Pela tabela 5.13, verifica-se que tanto CSAG quanto AG obtiveram 100% das soluções ótimas.

5.2.2 – Resultados obtidos pelo SATL

Os parâmetros utilizados nesta sessão são os mesmos utilizados pelo SATL para o USAHLP. Para cada um dos testes mostrados nas tabelas a seguir, os métodos foram executados 10 vezes para cada teste.

Tabela 5.14 – Conjunto AP: Exemplares pequenos

AP com custos fixos apertados														
n	**p**	**Ótimo**	**CSSATL**					**SATL**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
10	2	167493,06	167493,06	0,00	0,00	0,01	PR	167493,06	0,00	0,00	0,00	AL	0,03	0,00
	3	136008,13	136008,12	0,00	0,00	0,03	CS	136008,12	0,49	0,00	0,00	AL	0,03	0,00
	4	112396,07	112396,07	0,00	0,00	0,58	CS	112396,07	0,84	0,00	0,01	AL	0,58	0,00
	5	91105,37	91105,37	0,37	0,00	0,02	CS	91937,31	0,00	0,91	0,00	AL	0,03	0,91
20	2	172816,69	172816,70	0,00	0,00	0,16	PR	172816,70	0,06	0,00	0,09	AL	0,17	0,00
	3	151533,08	151533,08	0,00	0,00	0,05	PR	151533,08	0,00	0,00	0,03	AL	0,17	0,00
	4	135624,88	135624,88	0,00	0,00	0,04	CS	135624,88	0,00	0,00	0,02	AL	0,14	0,00
	5	123130,09	123130,09	0,00	0,00	0,04	CS	123130,09	0,00	0,00	0,01	AL	0,13	0,00
25	2	175541,98	175541,98	0,00	0,00	0,31	PR	175541,98	0,07	0,00	0,15	AL	0,40	0,00
	3	155256,32	155256,32	0,00	0,00	0,15	PR	155256,32	0,00	0,00	0,07	AL	0,28	0,00
	4	139197,17	139197,17	0,00	0,00	0,12	CS	139197,17	0,00	0,00	0,07	AL	0,26	0,00
	5	123574,29	123574,29	0,01	0,00	0,11	CS	123574,29	0,01	0,00	0,06	AL	0,25	0,00
40	2	177471,67	177471,67	0,00	0,00	0,67	PR	177471,67	0,00	0,00	0,45	AL	1,17	0,00
	3	158830,54	158830,54	0,08	0,00	0,57	CS	158830,54	0,08	0,00	0,33	AL	1,05	0,00
	4	143968,88	143968,88	0,00	0,00	0,49	CS	143968,88	0,00	0,00	0,27	AL	1,06	0,00
	5	134264,95	134428,65	0,00	0,12	0,69	CS	134428,65	0,00	0,12	0,47	AL	1,12	0,00
50	2	178484,29	178484,28	0,00	0,00	1,26	PR	178484,28	0,01	0,00	0,53	AL	2,08	0,00
	3	158569,93	158569,94	0,02	0,00	1,18	CS	158616,06	0,00	0,03	0,66	AL	2,02	0,03
	4	143378,05	143661,99	0,00	0,20	0,95	CS	143661,99	0,00	0,20	0,72	AL	1,96	0,00
	5	132366,95	133077,61	0,23	0,54	1,41	CS	133077,61	0,23	0,54	1,32	AL	2,08	0,00
Média				0,04	0,04	0,44			0,09	0,09	0,26		0,75	0,05

Como mostrado na Tabela 5.14, em um dos testes computacionais a solução gerada pelo CSSATL não só melhorou a melhor solução encontrada pelo SATL, mas também encontrou a solução ótima para este teste.

Tabela 5.15 – Conjunto AP: Exemplares grandes

AP com custos fixos apertados														
n	**p**	**Melhor**	**CSSATL**					**SATL**					**TE**	**Var**
			Sol	**Desv**	**Gap**	**TC**	**QU**	**Sol**	**Desv**	**Gap**	**TC**	**QU**		
100	5	136929,44	137297,71	0,37	0,27	11,63	CS	137297,71	0,37	0,27	10,23	AL	15,54	0,00
	10	106469,60	109668,79	0,98	3,00	7,98	CS	109668,79	0,98	3,00	6,63	AL	15,49	0,00
	15	90533,52	92130,41	1,42	1,76	10,18	CS	92130,41	1,42	1,76	9,70	AL	15,15	0,00
	20	80270,96	83381,86	0,47	3,88	9,85	CS	83381,86	0,47	3,88	8,85	AL	15,18	0,00
200	5	140175,60	140966,61	0,23	0,56	110,77	CS	140966,61	0,23	0,56	97,24	AL	135,08	0,00
	10	110147,70	113114,82	1,43	2,69	117,52	CS	113114,82	1,43	2,69	110,27	AL	125,18	0,00
	15	94496,41	98684,88	1,24	4,43	101,68	CS	98684,88	1,24	4,43	96,91	AL	125,27	0,00
	20	85129,34	90093,53	0,88	5,83	66,51	CS	90093,53	0,88	5,83	63,61	AL	118,93	0,00
Média				0,88	2,80	54,52			0,88	2,80	50,43		70,73	0,00

Pelos resultados mostrados na Tabela 5.15, nota-se que o método CSSATL mostrou ser eficiente para obter boas soluções para o problema. Um acontecimento interessante pode ser visto em 2 casos em que $n = 200$: nestes testes o método proposto foi capaz de obter soluções até o momento não conhecidas.

Tabela 5.16 – Resumo CS x SATL (USApHMP)

	CSSATL	SATL
Desvio médio total	0,28	0,31
% das soluções ótimas obtidas	85,00	75,00

Através dos resultados apresentados pela Tabela 5.16 é perceptível que o CSSATL mostrou-se mais robusto e mais eficiente em relação ao SATL.

Os algoritmos desenvolvidos neste trabalho encontram-se integrados a uma plataforma computacional para solução de problemas logísticos (PCLOG) (TROFINO E SENNE, 2008). A Figura 5.1 mostra a solução obtida para o exemplar do USApHMP com $n = 200$ e $p = 5$ do conjunto AP.

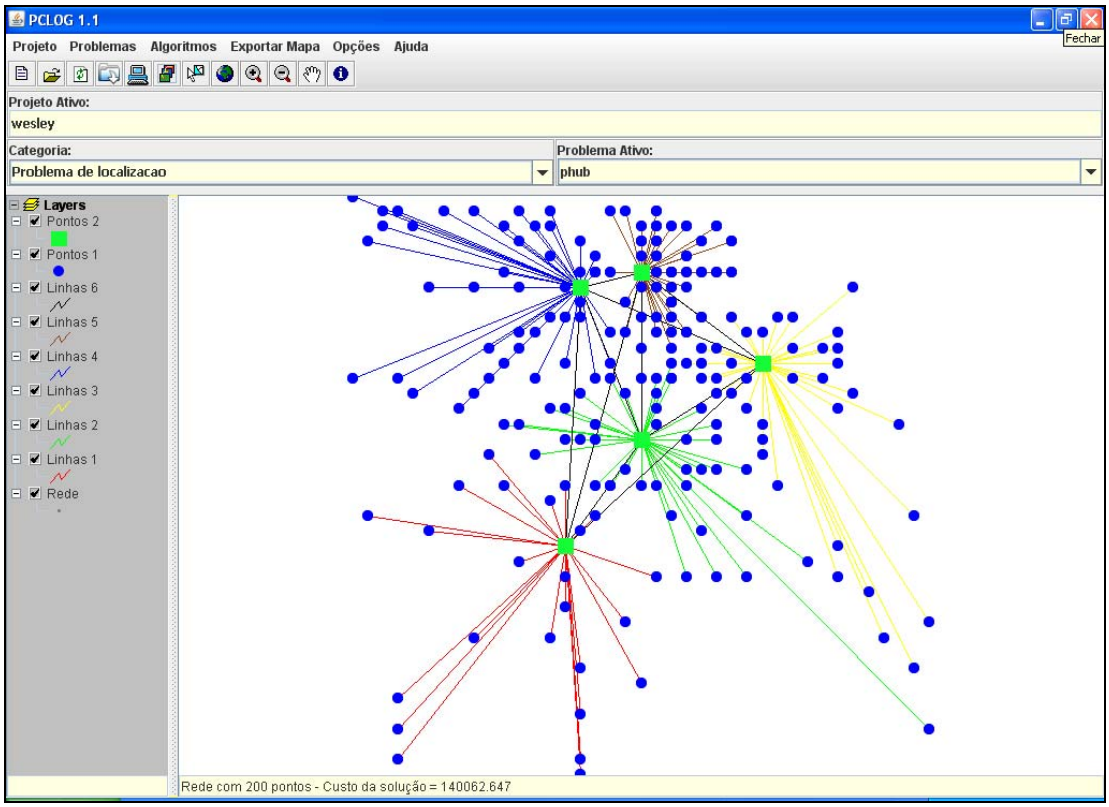

Figura 5.1 – Solução do exemplar do conjunto AP

6. CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

O problema de localização de concentradores é um problema de Otimização Combinatória relevante, pois ocorre em diversas situações práticas em que o transporte de alguma entidade (pessoas, dados, produtos, etc) precisa passar por um processo de agregação antes de ser distribuída ao seu destino. Boas soluções para o problema podem representar ganhos econômicos significativos para muitos setores empresariais. A solução do problema consiste na determinação da quantidade necessária e na localização dos concentradores, além da alocação das demais instalações a estes concentradores, minimizando os custos envolvidos.

O problema, no entanto, é conhecido ser da classe NP-difícil e algoritmos exatos, para determinar a solução ótima do problema, precisam utilizar métodos enumerativos que exigem grande esforço computacional e podem ser impraticáveis para exemplares do problema de grandes dimensões. Os métodos propostos neste trabalho, por serem heurísticos, não garantem que a solução obtida seja ótima, mas, como observado pelos testes realizados, é capaz de gerar boas soluções em um tempo computacional razoável.

A construção de algoritmos heurísticos eficientes requer bons mecanismos de intensificação de busca. Sem regiões promissoras de busca, um algoritmo heurístico, para escapar dos ótimos locais, precisa fazer buscas locais indiscriminadas, o que aumenta o tempo computacional. Os resultados obtidos neste trabalho comprovam que a busca por agrupamentos (*clustering search*) é uma técnica de intensificação de busca que permite identificar regiões do espaço de soluões que merecem especial atenção e para as quais uma busca local deve ser intensificada.

Pelos testes realizados pode-se concluir que aplicação da busca por agrupamentos permite obter soluções de melhor qualidade, em relação às soluções obtidas pelos algoritmos AG e SATL considerados isoladamente. Além disso, o método CS mostrou-se robusto independentemente do algoritmo gerador de soluções utilizado neste trabalho para o processo de agrupamentos.

Com relação ao tempo computacional para encontrar a melhor solução, os testes mostram que o CS leva mais tempo para encontrar a melhor solução. Isso ocorre porque na busca por agrupamentos muitas das boas soluções (isto, evidentemente, depende das soluções geradas pelo algoritmo gerador de soluções) são obtidas durante o processo de assimilação das soluções a um *cluster*, em que um procedimento de busca local (neste trabalho, *path-relinking*) também é aplicado para atualizar o centro do *cluster*.

Como sugestão para trabalhos futuros seria interessante um estudo comparativo com outros algoritmos de geração de soluções e com outras técnicas de busca, como *Scatter Search*, VNS (*Variable Neighbourhood Search*), GRASP (*Greedy Randomized Adaptive Search Procedure*), dentre outras.

Outra proposta poderia ser um estudo mais aprofundado para o problema USApHMP, na tentativa de encontrar soluções melhores para o problema. Sugere-se também um estudo mais aprofundado para os outros problemas de localização de concentradores citados neste trabalho, em particular, o problema em que cada concentrador tem uma capacidade de atendimento, pois trata-se de um problema mais próximo de aplicações reais.

Uma possibilidade interessante seria a geração de um conjunto de dados para as cidades brasileiras, na tentativa de determinar uma rede com o menor custo possível para aplicações de linhas aéreas ou em redes de transporte terrestre. No caso do transporte terrestre, seria interessante incluir no modelo, pesos nos caminhos entre as cidades, de maneira a aproximar o modelo da realidade das estradas nacionais.

REFERÊNCIAS BIBLIOGRÁFICAS

ABDINNOUR-HELM, S.; VENKATARAMANAN, M.A. **Using simulated annealing to solve the p-hub location problem**. In: IRMIS Working paper, 9304, Decision and Information Systems Department, School of Business, Indiana University, Bloomington, Indiana, 1993.

ABDINNOUR-HELM, S. A hybrid heuristic for the uncapacitated hub location problem. **European Journal of Operational Research**, v. 106, p. 489-499, 1998.

ALMEIDA, W.G. **Algoritmos genéticos para projeto de estruturas metálicas sob condições de incêndio**. Monografia, Faculdades Integradas de Caratinga, MG, 2006.

ALUMUR, S.; KARA, B. Y. Network hub location problems: the state of the art. **European Journal of Operational Research**, v. 190, n. 1, p. 1-21, Oct., 2008.

AYKIN, T. Lagrangian relaxation based approaches to capacitated hub-and-spoke network design problem. **European Journal of Operational Research**, v. 79, n. 33, p. 501-523, 1994.

CHAVES, A. A. **Meta-heurística híbrida com busca por agrupamentos aplicada a problemas de otimização combinatória**. Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 197 p., 2009.

CAMPBELL, J.F. Integer programming formulations of discrete hub location problems. **European Journal of Operational Research**, v. 72, p. 387-405, 1994.

CHEN, J.F. A hybrid heuristic for the uncapacited hub location problem. *Omega, The* **International Journal of Management Science**, v. 35, p. 211-220, 2007.

CHEN, J.F. A note on solution of the uncapacited single allocation p-hub median problem. **Journal of the Chinese Institute of Industrial Engineer***s*, v. 25, n. 1, p. 11-17, 2008.

CUNHA, C.B.; SILVA, M.R. A genetic algorithm for the problem of configuring a hub-and-spoke network for a LTL trucking company in Brazil. **European Journal of Operational Research**, v. 179, p. 747-758, 2007.

EBERY, J. Solving a large Single allocation p-hub problems with two or three hubs. **European Journal of Operational Research**, v. 128, n. 2, p. 447-458, 2001.

ERNST, A.T.; KRISHINAMOORTHY, M. Efficient algorithms for the uncapacited single allocation p-hub median problem. **Location Science**, v. 4, n. 3, p. 139-154, 1996.

ERNST, A.T.; KRISHNAMOORTHY, M. An Exact Solution Approach Based on Shortest-Paths for P-Hub Median Problems, **INFORMS Journal on Computing**, v.10 n. 2, p. 149-162, Feb., 1998.

ERNST, A.T.; KRISHNAMOORTHY, M. Solution Algorithms for the Capacitated Single Allocation Hub Location Problem, **Annals of OR**, v. 86, p. 141-159, 1999.

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. Tabu Search and adaptive memory programming: Advances, applications and challenges. In: BARR, R.S.; HELGASON, R.V.; KENNINGTON, J.L. (eds) **Interfaces in Computer Science and Operational Research**. Kluwer, 1996. p. 1-75.

GOLDMAN, A.J. Optimal location for centers in a network. **Transportation Science**, v. 3, p. 352-360, 1969.

HOLLAND, J. **Adaptation in natural and artificial systems**. Ann Arbor, Michigan: University of Michigan Press, 1975.

KLINCEWICZ, J. Heuristics for the p-hub median problem. **European Journal of Operational Research**, v. 79, p. 25-37, 1991.

KRATICA, J.; STANIMIROCIC, Z.; TOSIC, D.; FILIPOVIC, V.. Two genetic algorithms for solving the uncapacted sigle allocation p-hub median problem. **European Journal of Operational Research**, v. 182, p. 15-28, 2007.

OLIVEIRA, A. C. M.; LORENA, L. A. N. Detecting promising areas by evolutionary clustering search. **Lecture Notes in Artificial Intelligence**, v. 3171, p. 385-394, 2004.

OLIVEIRA, A. C. M.; LORENA, L. A. N. Hybrid Evolutionary Algorithms and Clustering Search. In: GROSAN, Crina; ABRAHAM, Ajith; ISHIBUCHI, Hisao (Ed.). **Hybrid Evolutionary Systems - Studies in Computational Intelligence**. Springer, 2007. v. 75, p. 81-102. (INPE--/).

O'KELLY, M. A quadratic integer program for the location of interacting hub facilities. **European Journal of Operational Research**, v. 32, p. 393-404, 1987.

O'KELLY, M. Hub facility location with fixed costs. **Papers in Regional Science**, DOI : 10.1007/BF01434269 p. 293-306, 1992.

PIRKUL, H.; SCHILLING, D.A. An efficient procedure for designing single allocation hub and spoke systems. **Management Science**, v. 44, n. 12S, p. 235-242, 1998.

SASAKI, M.; FUKUSHIMA, M. On the hub-and-spoke model with arc capacity constraints. **Journal of the Operational Research Society of Japan**, v. 46, n. 4, p. 409-428, 2002.

SILVA, M.R. **Uma contribuição ao problema de localização de terminais de consolidação no transporte de carga parcelada**. Dissertação de Mestrado. Escola Politécnica da Universidade de São Paulo, 2004.

SMITH, K.; KRISHNAMOORTHY, M.; PALANISWAMI, M. Neural versus traditional approaches to the location of interacting hub facilities. **Location Science** (Special Issue on Hub Location), v. 4, n. 3, p. 155-171, 1996.

TOPCUOGLU, H.; CORUT, F.; ERMIS, M.; YLMAZ, G. Solving the uncapacitated hub location problem using genetic algorithms. **Computers and Operations Reseach**, v. 32, p. 967-984, 2005.

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.

ZIVIANI, N. **Projeto de Algoritmos com Implementações em Pascal e C.** 2a.. ed. São Paulo: Thompson Learning, 2004. v. 1. 572 p.

Apêndice A: Pseudocódigo do algoritmo CSAG implementado

```
Algoritmo
    leia os dados de entrada
    gere uma populaçao inicial Pop
    Enquanto critério de parada não for satisfeito faça
        x1 = randomico entre 0 e ne
        x2 = randomico entre 0 e POP
        Para i = 0 até ne faça
            Nov[i] = Pop[i]
        Para i = ne até POP faça
            aplique o operador de crossover(P[x1], P[x2], S1, S2)
            aplique o operador de mutação (S1)
            aplique o operador de mutação (S2)
            inclua S2 na nova polução Nov
            Nov[i] = S1
            i = i + 1
            Se i < POP faça
                Nov[i] = S2
            fim-se
            Assimile S1 aos clusters
            Assimile S2 aos clusters
        fim-para
        Para i = ne até POP faça
            Pop[i] = Nov[i]
        fim-para
        coloque a população Pop em ordem crescente de acordo o fitness de cada solução
        Se Pop[0] < best então
            best = Pop[0]
        fim-se
        Assimile best aos clusters
```

```
        Para i = 0 até no nc faça
            Se act[i] > 7 então
                act[i] = 0
                aplique busca local em C[i]
            fim-se
            Se inineficacia[i] > 30 então
                aplique uma perturbação em C[i]
                ineficacia[i] = 0
            fim-se
            Se bestC[i] < bestCenter então
                bestCenter = bestC[i]
            fim-se
            Se C[i] < bestC[i] então
                bestC[i] = C[i]
            senão
                ineficacia[i] = ineficacia[i] + 1
            fim-se
        fim-para
    fim-enquanto
    escreva a soloução best encontrada pelo AG
    escreva a soloução bestCenter encontrada pelo CSAG
fim-algoritmo

procedimento gere uma populaçao inicial Pop
    Para i = 0 até Pop faça
        Pop[i].P = randomico entre 0 e n
        Pop[i].valor = 0
        Para j= 0 até Pop[i].P faça
            Pop[i].ass_arr[j] = 0
            Pop[i].hub_arr[j] = 0
        fim-para
```

```
        Para j = 0 até Pop[i].P faça
            x = randomico de 1 a n
            Pop[i].hub_arr[x] = 1
            Pop[i].ass_arr[x] = x
            Pop[i].ind_hub[j] = x
        fim-para
        Para j = 0 até n faça
            Se Pop[i].hub_arr[j] = 0 faça
                x = randomico entre 0 e Pop[i].P
                Pop[i].ass_arr[j] = Pop[i].hub_arr[x]
            fim-se
        fim-para
    fim-para
fim-procedimento

procedimento crossover(x1, x2, s1, s2)
    x= (randomico entre 0 e 1000)/1000
    Se x < Pc faça
        faça
            p1 = randomico entre 0 e n
            p2 = randomico entre 0 e n
        enquanto p1 = p2
        Para j = 0 até p1 faça
            s1.hub_arr[j] = x1.hub_arr[j]
            s1.ass_arr[j] = x1.ass_arr[j]
            a = a + x1.hub_arr[j]
            b = b + x2.hub_arr[j]
            s2.hub_arr[j] = x2.hub_arr[j]
            s2.ass_arr[j] = x2.ass_arr[j]
        fim-para
        Para j = p1 até p2 faça
```

```
            s1.hub_arr[j] = x2.hub_arr[j]
            s1.ass_arr[j] = x2.ass_arr[j]
            a = a + x2.hub_arr[j]
            b = b + x1.hub_arr[j]
            s2.hub_arr[j] = x1.hub_arr[j]
            s2.ass_arr[j] = x1.ass_arr[j]
        fim-para
        Para j = p2 até n faça
            s1.hub_arr[j] = x1.hub_arr[j]
            s1.ass_arr[j] = x1.ass_arr[j]
            a = a + x1.hub_arr[j]
            b = b + x2.hub_arr[j]
            s2.hub_arr[j] = x2.hub_arr[j]
            s2.ass_arr[j] = x2.ass_arr[j]
        fim-para
        s1.P = a
        s2.P = b;
        viabilização(s1);
        viabilização(s2);
    senão
        s1 = x1
        s2 = x2
    fim-se
fim-procedimento

procedimento viabilização(aux)
    p=0
    Para i = 0 até n faça
        Se (aux.hub_arr[i] =1) faça
            aux.ind_hub[p] = i
            p = p+1
```

```
        fim-se
    fim-para
    aux.P = p
    Para j = 0 até n faça
        Se (aux.hub_arr[j] = 0) então
            Se (aux.hub_arr[aux.ass_arr[j]] = 0) então
                menor = aux.ind_hub[0]
                Para w =1 até aux.P faça
                    indice = aux.ind_hub[w];
                    Se (c[j][indice] < c[j][menor]) então
                        menor = indice;
                    fim-se
                fim-para
                aux.ass_arr[j] = menor;
            fim-se
        fim-se
    fim-para
    fitness(aux)
fim-procedimento

procedimento fitness(p)
    soma1 = 0.0000000
    soma2 = 0.0000000
    Para i = 0 até n faça
        soma1 = soma1 + c[i][p.ass_arr[i]]*wn[i]
    fim-para
    Para i = 0 até n faça
        Para j = 0 até n faça
            soma1 = soma1 + w[i][j]* c_trans * c[p.ass_arr[i]][p.ass_arr[j]];
        fim-para
    fim-para
```

```
    Se (isAP) então
        soma3 = 0.0
        Para i = 0 até p.P faça
            soma1 = soma1 + Fix[p.ind_hub[i]];
        fim-para
        p.valor = soma1;
    senão
        soma3 = p.P*c_fix;
        p.valor = (soma1 + soma2)/somaf+ soma3;
    fim-se
fim-procedimento

procedimento mutação (aux)
    x = (randomico entre 0 e 1000)/1000
    Se (x<P_m) faça
        p1 = aux
        p2 = aux
        p3 = aux
        n_hubs = aux.P
        //shift, se não tiver apenas um hub
        //Aqui troca-se apenas o hub que um não está ligado
        Se (n_hubs != 1) faça
            a = 1
            faça
                //sorteia um spoke
                no = randomico entre 0 e n
            enquanto (p1.hub_arr[no] != 0)
            //sorteia um hub, e associa ao spoke sorteado
            hub = randomico entre 0 e p1.P
            indice = p1.ind_hub[hub]
            p1.ass_arr[no] = indice
```

```
        fim-se
        Se ((n_hubs != 1)&&((n-n_hubs) != 1)) faça
            b = 1
            faça
                no = randomico entre 0 e n
            enquanto (p2.hub_arr[no] != 0)
            faça
                no2 = randomico entre 0 e n
            enquanto (p2.hub_arr[no2] != 0)
            indice = p2.ass_arr[no]
            p2.ass_arr[no] = p2.ass_arr[no2]
            p2.ass_arr[no2] = indice
        fim-se
        aplique a busca local em (p3, 100, ebest, 10)
        Se ((a = 1)&&(b = 1)) faça
            fitness(p1)
            fitness(p2)
            Se (p1.valor < p2.valor) então
                aux = p1
            senão
                aux = p2
            fim-se
        fim-se
        Se (p3.valor < aux.valor) faça
            aux = p3
        fim-se
        viabilização(aux)
fim-procedimento

procedimento busca local(p)
    sem_mel = 0
```

```
enquanto (sem_mel<parada) faça
    x = randômico entre 0 e 4
    Se (x = 0) então
        k = randômico entre 0 e n
        j = randômico entre 0 e n
        Se ((p.hub_arr[j] = 0)&&(p.hub_arr[k] = 0)) então
            //troca duas associações
            Se (p.ass_arr[k] != p.ass_arr[j]) então
                aux = p
                aux1 = p.ass_arr[k]
                p.ass_arr[k] = p.ass_arr[j]
                p.ass_arr[j] = aux1
                temp = p.valor
                fitness(p)
                Se (p.valor >= aux.valor) então
                    p.valor = temp
                    aux1 = p.ass_arr[k]
                    p.ass_arr[k] = p.ass_arr[j]
                    p.ass_arr[j] = aux1
                    sem_mel++
                    d = p.valor - aux.valor
                    pr = exp(-1*d/temperatura)
                    r = (randômico entre 0 e 1000)/1000.0
                    Se (r > pr) então
                        //não aceita então recupera os dados antigos
                        p = aux;
                    fim-se
                senão
                    Se (p.valor < ebest.valor) então
                        best = p;
                    sem_mel = 0
```

```
            fim-se
        fim-se
    fim-se
//associa um outro hub ao spoke j
senão se (x = 1) então
    k = randômico entre 0 e p.P
    j = randômico entre 0 e n
    Se (p.hub_arr[j] = 0) então
        aux = p
        aux1 = p.ass_arr[j]
        p.ass_arr[j] = p.ind_hub[k]
        temp = p.valor
        fitness(p)
        Se (p.valor >= aux.valor) então
            p.valor = temp
            p.ass_arr[j] = aux1
            sem_mel++
            d = p.valor - aux.valor
            pr = exp(-1*d/temperatura)
            r = (randômico entre 0 e 1000)/1000.0
            Se (r > pr) então
                //não aceita então recupera os dados antigos
                p = aux
            fim-se
        senão
            Se (p.valor < ebest.valor) então
                best = p
            sem_mel = 0
        fim-se
    fim-se
senão se (x = 2) então
```

```
aux1 = randômico entre 0 e n
se (p.hub_arr[aux1] = 0) então
    aux = p
    k = p.ass_arr[aux1]
    p.hub_arr[k] = 0
    p.hub_arr[aux1] = 1
    p.ass_arr[aux1] = aux1
    Para j = 0 até n faça
        Se (p.ass_arr[j] = k) então
            p.ass_arr[j] = aux1
        fim-se
    fim-para
    fim-para
    Para (j = 0 j<p.P j++)
        se (p.ind_hub[j] = k) então
            p.ind_hub[j] = aux1
            break
        fim-se
    fim-para
    fitness(p)
    Se (p.valor >= aux.valor) então
        p = aux
        sem_mel++
        d = p.valor - aux.valor
        pr = exp(-1*d/temperatura)
        r = (randômico entre 0 e 1000)/1000.0
        Se (r > pr) então
            //não aceita então recupera os dados antigos
            p = aux
        fim-se
    senão
```

```
            Se (p.valor < ebest.valor) então
                best = p
            fim-se
            sem_mel = 0
        fim-se
    fim-se
senão se (x = 3) então
    se (p.P < (n-1)) então
        k = randômico entre 0 e n
        Se (p.hub_arr[k] = 0) então
            p.hub_arr[k] = 1
            p.ass_arr[k] = k
            p.ind_hub[p.P] = k
            p.P++
        fim-se
        fitness(p)
        Se (p.valor >= aux.valor) então
            p = aux
            sem_mel++
            d = p.valor - aux.valor
            pr = exp(-1*d/temperatura)
            r = (randômico entre 0 e 1000)/1000.0
            Se (r > pr) então
                //não aceita então recupera os dados antigos
                p = aux
            fim-se
        senão
            Se (p.valor < ebest.valor) então
                best = p
            sem_mel = 0
        fim-se
```

```
        fim-se
    senão se (x = 4) então
        Se (p.P > 1) então
            aux = p
            k = randomico entre 0 e p.P
            j = p.ind_hub[k]
            Para t = k até p.P-1 faça
                p.ind_hub[t] = p.ind_hub[t+1]
            p.P--
            k = j
            p.hub_arr[k] = 0
            j = randomico entre 0 e p.P
            j = p.ind_hub[j]
            Para t = 0 até n faça
                Se (p.ass_arr[t] = k) faça
                    p.ass_arr[t] = j
            fim-para
            fitness(p)
            Se (p.valor >= aux.valor) então
                p = aux
                sem_mel++
                d = p.valor - aux.valor
                pr = exp(-1*d/temperatura)
                r = (randômico entre 0 e 1000)/1000.0
                Se (r > pr) então
                    //não aceita então recupera os dados antigos
                    p = aux
                fim-se
            senão
                Se (p.valor < ebest.valor) então
                    best = p
```

```
                    sem_mel = 0
                fim-se
            fim-se
        fim-se
        temperatura = temperatura*0.9
    fim-enquanto
fim-procedimento

função CalcDist(x, C): inteiro
    Para i de 0 a n faça
        Se (x->hub_arr[i] != C->hub_arr[i]) então
            d = d + 1
        fim-se
    fim-para
    retorne d
fim-função

função newCenter(ind *p): inteiro
    Se (nc < NC) faça
        C[nc] = p
        act[nc] = 0
        nc++
        resp = 1
    fim-se
    retorne resp
fim-função

procedimento assmilate (p, ci)
    j = 0;
    Se (p.valor < C[ci].valor) então
        C[ci] = p
```

```
fim-se
p3 = p
solCam = C[ci]
Para i = 0 até n faça
    menor = 99999
    p.valor = 99999
    Para j = 0 até n faça
        Se (p.hub_arr[j] != C[ci].hub_arr[j]) então
            ant = p
            Se (p.hub_arr[j] = 1) então
                p.hub_arr[j] = 0
                cont = 0
                enquanto (p.ind_hub[cont] != j) faça
                    cont++
                fim-enquanto
                Para k = cont até p.P-1 faça
                    p.ind_hub[k] = p.ind_hub[k+1]
                fim-para
                p.P = p.P -1
                Se (p.P > 0) então
                    Para k = 0 até n faça
                        Se (p.ass_arr[k] = j) então
                            t = randômico entre 0 e p.P
                            p.ass_arr[k] = p.ind_hub[t]
                        fim-se
                    fim-para
                    fitness(p)
                senão
                    Para k = 0 até n faça
                        Se (p.ass_arr[k] = j) então
                            t = randômico entre 0 e C[ci].P
```

```
                        p.ass_arr[k] = C[ci].ind_hub[t]
                    fim-se
                fim-para
                fitness(p)
            fim-se
        senão
            p.hub_arr[j] = 1
            p.ass_arr[j] = j
            p.ind_hub[p.P] = j
            p.P++
            Se (p.P = 1) então
                Para k = 0 até n faça
                    p.ass_arr[k] = j
                fim-para
            fim-se
            fitness(p)
        fim-se
        Se (p.valor<menor & p.P > 0) então
            menor = p.valor
            aux = p
        fim-se
        p = ant
    fim-se
  fim-para
  p = aux
  Se (aux.valor < solCam.valor) então
    solCam = aux
  fim-se
fim-para
p = p3
Se (solCam.valor < C[ci].valor) então
```

```
        C[ci] = solCam
    senão
        r = (randômico entre 0 e 1000)/1000.0
        se (r < 0.2) então
            C[ci] = solCam
        fim-se
    fim-se
fim-procedimento

procedimento assimile(p)
    cont = 1
    minDist = 9999
    minC = -1
    Para i = 0 até nc faça
        d = CalcDist(p, &C[i])
        se ((d < raio) & (d < minDist)) então
            minDist = d
            minC = i
        fim-se
    fim-para
    Se (minC < 0) então
        newCenter(p)
    senão
        assimilate(p, minC)
        act[minC]++
    fim-se
fim-procedimento

procedimento pertubação (x)
    Para j = 0 até n faça
        x.hub_arr[j] = 0;
```

```
        x.ass_arr[j] = 0;
        saiu[j] = 0;
    fim-para
    Para j = 0 até x.P faça
        faça
            t = randômico entre 0 e n;
        enquanto (saiu[t])
        saiu[t] = 1
        x.hub_arr[t] = 1
        x.ind_hub[j] = t
        x.ass_arr[t] = t
    fim-para
    Para j = 0 até n faça
        Se (x.hub_arr[j] = 0) então
            t = randômico entre 0 e x.P
            x.ass_arr[j] = x.ind_hub[t];
        fim-se
    fim-para
    fitness(x);
fim-procedimento
```

Apêndice B: Pseudocódigo do algoritmo CSSATL implementado

```
Algoritmo
    criaLista(listaTabu, 7);
    enquanto critério de parada não satisfeito faça
        Se (PILOTO = 1) faça
            sol.P = 1
            faça
                InitialAlocation(p, sol)
                Allocation(sol, temperatura)
                inserir(&listaTabu, sol)
                SLEP(sol, 100, ebest)
                inserir(&listaTabu, sol)
                AI(ebest, C)
                soma1 = 0.0
                soma2 = 0.0
                Para i = 0 até n faça
                    soma1 = soma1 + c[i][sol.ass_arr[i]]*wn[i]
                fim-para
                Para i = 0 até n faça
                    Para j = 0 até n faça
                        soma2 = soma2 + w[i][j]* c_trans * c[sol.ass_arr[i]][sol.ass_arr[j]]
                    fim-para
                fim-para
                Se (isAP) então
                    ct_trans = soma1 + soma2
                    ct_fix = 0.0
                    Para i = 0 até sol.P faça
                        ct_fix = ct_fix + Fix[sol.ind_hub[i]]
                senão
                    ct_fix = c_fix*sol.P
```

```
            ct_trans = (soma1 + c_trans*soma2)/somaf
        fim-se
        Se ct_fix < ct_trans) então
            sol.P++
        senão
            PILOTO = 0
        fim-se
        cont++
    enquanto ((ct_fix < ct_trans) & (cont < 100))
    Se (sol.valor < ebest.valor) então
        ebest = sol;
        semMelhora = 0
    senão
        semMelhora++
    fim-se
fim-se
faça
    SLEP(sol, temperatura, ebest)
enquanto(MovIsTabu(&listaTabu, sol))
inserir(&listaTabu, sol)
temperatura = temperatura*0.9
Se (sol.valor < ebest.valor) então
    ebest = sol
    semMelhora = 0
senão
    semMelhora++
fim-se
Assimile best aos clusters
Para i = 0 até no nc faça
    Se (act[i] > 7) então
        act[i] = 0
```

```
            aplique busca local em C[i]
        fim-se
        Se (ineficacia[i] > 30) então
            aplique uma perturbação em C[i]
            ineficacia[i] = 0
        fim-se
        Se (bestC[i] < bestCenter) então
            bestCenter = bestC[i]
        fim-se
        Se (C[i] < bestC[i]) então
            bestC[i] = C[i]
        senão
            ineficacia[i] = ineficacia[i] + 1
        fim-se
    fim-para
  fim-enquanto
  DestroyLista(&listaTabu);
fim-algoritmo

procedimento InitialAlocation(int p, ind *x)
  Para j = 0 até n faça
    x.hub_arr[j] = 0
    x.ass_arr[j] = 0
    saiu[j] = 0
  fim-para
  Se (x.P >= n) então
    x.P = randômico entre 1 e 5
  fim-se
  Para j = 0 até x.P faça
    t = getIndiceDeMaiorI()
    x.hub_arr[t] = 1
```

```
        x.ind_hub[j] = t
        x.ass_arr[t] = t
    fim-para
    Para j = 0 até n faça
        Se (x.hub_arr[j] = 0) então
            menor = x.ind_hub[0]
            Para w =1 até x.P faça
                indice = x.ind_hub[w]
                Se (c[j][indice]<c[j][menor]) então
                    menor = indice
            fim-para
            x.ass_arr[j] = menor
        fim-se
    fim-para
    fitness(x)
fim-procedimento

procedimento Allocation(ind *x, float temperatura)
    Para j = 0 até n faça
        //todos são de ordem 1, ou seja estão alocados ao primeiro mais proximo
        //evita que pegue nós hubs
        se (x.hub_arr[j] = 0) então
            menor = x.ind_hub[0]
            Para w =1 até x.P faça
                indice = x.ind_hub[w]
                Se (c[j][indice] < c[j][menor]) então
                    menor = indice
                fim-se
            fim-para
            x->ass_arr[j] = menor
        fim-se
```

```
fitness(x)
Se (x->valor < ebest->valor) então
    ebest = x
fim-se
semMelhora = 0
faça
    Para j = 0 até n faça
        ocorreu[j] = x.hub_arr[j]
    //atualiza a solução global
    Para i = 0 até n faça
        indice = -1
        faça
            indice = indice + 1
            menor = wf[indice]
        enquanto (ocorreu[indice])
        Para j = 0 até n faça
            //pega os não-hubs de menor fluxo que ainda não foram escolhidos
            Se ((wf[j] < menor) & (!ocorreu[j])) faça
                menor = wf[j]
                indice = j
            fim-se
        fim-para
        Se (indice < n) faça
            ocorreu[indice] = 1
            Se (ordem[indice] < x.P) faça
                ordem[indice] = ordem[indice] +1
            fim-se
            temp = x.ass_arr[indice]
            f = x.valor
            x.ass_arr[indice] = getIesimoMaisProximo(x, indice, ordem[indice])
            fitness(x)
```

```
            d = x.valor - f
            Se (x.valor < ebest.valor) então
                ebest = x
                semMelhora = 0
            senão
                semMelhora = semMelhora + 1
            fim-se
            Se (x.valor >= f) então
                //verifica se vai aceitar o movimento de piora
                pr = exp(-1*d/temperatura)
                r = (randômico entre 0 e 1000)/1000.0
                Se (r > pr) então
                    //não aceita então recupera os dados antigos
                    x.valor = f
                    x.ass_arr[indice] = temp
                fim-se
            fim-se
        fim-se
    fim-para
  enquanto (x.valor < ebest.valor)
fim-procedimento

procedimento SLEP(ind *x, float temperatura, ind *ebest)
  semMelhora = 0
  enquanto (semMelhora < 200) faça
    aux = x
    cont = 0
    t = -1
    faça
      faça
        t = randômico entre 0 e n
```

```
    enquanto(x.hub_arr[t] != 0)
    h = randômico entre 0 e x.P
    hi = x.ind_hub[h]
    x.hub_arr[t] = 1
    x.ass_arr[t] = t
    x.hub_arr[hi] = 0
    x.ind_hub[h] = t
    Para i = 0 até n faça
        Se (x.ass_arr[i] = hi) então
            x.ass_arr[i] = t
    Allocation(x, temperatura)
    Se (x.valor < ebest.valor) então
        copia(ebest, x)
        cont = 0
    senão
        cont = cont+1
    fim-se
enquanto ((MovIsTabu(&listaTabu, x)) & (cont < 1000))
inserir(&listaTabu, x)
fitness(x)
d = x.valor - aux.valor
Se (x.valor < ebest.valor) então
    ebest = x
    semMelhora = 0
senão
    semMelhora++
fim-se
Se (x.valor >= aux.valor) então
    //verifica se vai aceitar o movimento de piora
    pr = exp(-1*d/temperatura)
    r = (randômico entre 0 e 1000)/1000.0
```

```
            Se (r > pr) então
                //não aceita então recupera os dados antigos
                copia(x, aux)
            fim-se
        senão
            //aceita
            //semMelhora = 0
        fim-se
    fim-enquanto
fim-procedimento

função getIndiceDeMaiorI() : inteiro
    maior = 0;
    Para i = 1 até n faça
        Se ((I[i]>I[maior]) & (!saiu[i])) então
            maior = i;
            saiu[i] = 1;
        fim-se
    fim-para
    retorne maior;
fim-função

função getIesimoMaisProximo(ind *x, int j, int ordem) : inteiro
    Para w = 0 até x->P faça
        ocorreu[w] = 0
    fim-para
    menor = x.ind_hub[0]
    Se (ordem <= x.P) então
        Para i = 0 até ordem faça
            Para w = 1 até x.P faça
                indice = x.ind_hub[w];
```

```
            Se ((c[j][indice] < c[j][menor]) & (!ocorreu[indice])) então
                menor = indice;
            fim-se
        fim-para
        ocorreu[menor] = 1;
    fim-para
    x.ass_arr[j] = menor;
  fim-se
  retorne menor;
fim-função

procedimento criaLista(TipoLista *l, int max)
  l.MAX = max;
  l.quantidade = 0;
  l.v = faz alocações de memória;
fim-procedimento

procedimento DestroyLista(TipoLista *l)
  Desaloca espaço de memória utilizado pela lista
fim-procedimento

procededimento inserir(TipoLista *l, ind *x)
  l.v[l->indiceUltimo] = x
  l.indiceUltimo = (l->indiceUltimo + 1) % l->MAX;
  Se (l.quantidade < l.MAX) então
    l.quantidade++;
fim-procedimento

função IsEqual(ind *x, ind *y) : inteiro
  Para i = 0 até n faça
    Se (x.ass_arr[i] != y.ass_arr[i]) então
```

```
            retorne 0;
    fim-para
    retorne 1;
fim-função

função MovIsTabu(TipoLista *l, ind *x) : inteiro
    Para i = 0 até l->quantidade faça
        Se (x = l->v[i])
            retorne 1;
    fim-para
    retorne 0;
fim-função

procedimento busca local(p)
    sem_mel = 0
    enquanto (sem_mel < parada) faça
        x = randômico entre 0 e 4
        Se (x = 0) então
            k = randômico entre 0 e n
            j = randômico entre 0 e n
            Se ((p.hub_arr[j] = 0) & (p.hub_arr[k] = 0)) então
                //troca duas associações
                Se (p.ass_arr[k] != p.ass_arr[j]) então
                    aux = p
                    aux1 = p.ass_arr[k]
                    p.ass_arr[k] = p.ass_arr[j]
                    p.ass_arr[j] = aux1
                    temp = p.valor
                    fitness(p)
                    Se (p.valor >= aux.valor) então
                        p.valor = temp
```

```
                aux1 = p.ass_arr[k]
                p.ass_arr[k] = p.ass_arr[j]
                p.ass_arr[j] = aux1
                sem_mel++
                d = p.valor - aux.valor
                pr = exp(-1*d/temperatura)
                r = (randômico entre 0 e 1000)/1000.0
                Se (r > pr) então
                    //não aceita então recupera os dados antigos
                    p = aux;
                fim-se
            senão
                Se (p.valor < ebest.valor) então
                    best = p
                sem_mel = 0
            fim-se
        fim-se
    fim-se
//associa um outro hub ao spoke j
senão se (x = 1) então
    k = randômico entre 0 e p.P
    j = randômico entre 0 e n
    Se (p.hub_arr[j] = 0) então
        aux = p
        aux1 = p.ass_arr[j]
        p.ass_arr[j] = p.ind_hub[k]
        temp = p.valor
        fitness(p)
        Se (p.valor >= aux.valor) então
            p.valor = temp
            p.ass_arr[j] = aux1
```

```
            sem_mel++
            d = p.valor - aux.valor
            pr = exp(-1*d/temperatura)
            r = (randômico entre 0 e 1000)/1000.0
            Se (r > pr) então
                //não aceita então recupera os dados antigos
                p = aux
            fim-se
        senão
            Se (p.valor < ebest.valor) então
            best = p
            sem_mel = 0
        fim-se
    fim-se
senão se (x = 2) então
    aux1 = randômico entre 0 e n
    se (p.hub_arr[aux1] = 0) então
        aux = p
        k = p.ass_arr[aux1]
        p.hub_arr[k] = 0
        p.hub_arr[aux1] = 1
        p.ass_arr[aux1] = aux1
        Para j = 0 até n faça
            Se (p.ass_arr[j] = k) então
                p.ass_arr[j] = aux1
            fim-se
        fim-para
        fim-para
        Para j = 0 até p.P faça
            se (p.ind_hub[j] = k) então
                p.ind_hub[j] = aux1
```

```
                break
            fim-se
        fim-para
        fitness(p)
        Se (p.valor >= aux.valor) então
            p = aux
            sem_mel++
            d = p.valor - aux.valor
            pr = exp(-1*d/temperatura)
            r = (randômico entre 0 e 1000)/1000.0
            Se (r > pr) então
                //não aceita então recupera os dados antigos
                p = aux
            fim-se
        senão
            Se (p.valor < ebest.valor) então
                best = p
            fim-se
            sem_mel = 0
        fim-se
    fim-se
senão se (x = 3) então
    se (p.P < (n-1)) então
        k = randômico entre 0 e n
        Se (p.hub_arr[k] = 0) então
            p.hub_arr[k] = 1
            p.ass_arr[k] = k
            p.ind_hub[p.P] = k
            p.P++
        fim-se
        fitness(p)
```

```
            Se (p.valor >= aux.valor) então
                p = aux
                sem_mel++
                d = p.valor - aux.valor
                pr = exp(-1*d/temperatura)
                r = (randômico entre 0 e 1000)/1000.0
                Se (r > pr) então
                    //não aceita então recupera os dados antigos
                    p = aux
                fim-se
            senão
                Se (p.valor < ebest.valor) faça
                    best = p
                    sem_mel = 0
                fim-se
        fim-se
    senão se (x = 4) então
        Se (p.P > 1)então
            aux = p
            k = randômico entre 0 e p.P
            j = p.ind_hub[k]
            Para t = k até p.P-1 faça
                p.ind_hub[t]=p.ind_hub[t+1]
            p.P--
            k=j
            p.hub_arr[k]=0
            j = randômico entre 0 e p.P
            j = p.ind_hub[j]
            Para t = 0 até n faça
                Se (p.ass_arr[t] = k) faça
                    p.ass_arr[t] = j
```

```
            fim-para
            fitness(p)
            Se (p.valor >= aux.valor) então
                p = aux
                sem_mel++
                d = p.valor - aux.valor
                pr = exp(-1*d/temperatura)
                r = (randômico entre 0 e 1000)/1000.0
                Se (r > pr) então
                    //não aceita então recupera os dados antigos
                    p = aux
                fim-se
            senão
                Se (p.valor < ebest.valor) então
                    best = p
                sem_mel = 0
            fim-se
        fim-se
        temperatura = temperatura*0.9
    fim-enquanto
fim-procedimento

função CalcDist(x, C) : inteiro
    Para i = 0 até n faça
        Se (x->hub_arr[i] != C->hub_arr[i]) então
            d = d + 1
        fim-se
    fim-para
    returne d
fim-função
```

```
função newCenter(ind *p) : inteiro
    Se (nc < NC) faça
        C[nc] = p
        act[nc] = 0
        nc++
        resp = 1
    fim-se
    returne resp
fim-função

procedimento assmilate (p, ci)
    j = 0;
    Se (p.valor < C[ci].valor) então
        C[ci] = p
    fim-se
    p3 = p
    solCam = C[ci]
    Para i = 0 até n faça
        menor = 99999
        p.valor = 99999
        Para j = 0 até n faça
            Se (p.hub_arr[j] != C[ci].hub_arr[j]) então
                ant = p
                Se (p.hub_arr[j] = 1) então
                    p.hub_arr[j] = 0
                    cont = 0
                    enquanto (p.ind_hub[cont] != j) faça
                        cont++
                    fim-enquanto
                    Para k = cont até p.P-1 faça
```

```
            p.ind_hub[k] = p.ind_hub[k+1]
        fim-para
        p.P = p.P -1
        Se (p.P > 0) então
            Para k = 0 até n faça
                Se (p.ass_arr[k] = j) então
                    t = randômico entre 0 e p.P
                    p.ass_arr[k] = p.ind_hub[t]
                fim-se
            fim-para
            fitness(p)
        senão
            Para k = 0 até n faça
                Se (p.ass_arr[k] = j) então
                    t = randômico entre 0 e C[ci].P
                    p.ass_arr[k] = C[ci].ind_hub[t]
                fim-se
            fim-para
            fitness(p)
        fim-se
    senão
        p.hub_arr[j] = 1
        p.ass_arr[j] = j
        p.ind_hub[p.P] = j
        p.P++
        Se (p.P = 1) então
            Para k = 0 até n faça
                p.ass_arr[k] = j
            fim-para
        fim-se
        fitness(p)
```

```
            fim-se
            Se (p.valor<menor & p.P > 0) então
                menor = p.valor
                aux = p
            fim-se
            p = ant
        fim-se
    fim-para
    p =aux
    Se (aux.valor < solCam.valor) então
        solCam = aux
    fim-se
  fim-para
  p = p3
  Se (solCam.valor < C[ci].valor) então
    C[ci] = solCam
  senão
    r = (randômico entre 0 e 1000)/1000.0
    se (r < 0.2) então
      C[ci] = solCam
    fim-se
  fim-se
fim-procedimento

procedimento assimile(p)
  cont = 1
  minDist = 9999
  minC = -1
  Para i = 0 até nc faça
    d = CalcDist(p, &C[i])
    se ((d < raio) & (d<minDist)) então
```

```
            minDist = d
            minC = i
        fim-se
    fim-para
    Se (minC < 0) então
        newCenter(p)
    senão
        assimilate(p, minC)
        act[minC]++
    fim-se
fim-procedimento

procedimento pertubação (x)
    Para j = 0 até n faça
        x.hub_arr[j] = 0;
        x.ass_arr[j] = 0;
        saiu[j] = 0;
    fim-para
    Para j =0 até x.P faça
        faça
            t = randômico de 0 a n;
        enquanto (saiu[t])
        saiu[t] = 1
        x.hub_arr[t] = 1
        x.ind_hub[j] = t
        x.ass_arr[t] = t
    fim-para
    Para j = 0 até n faça
        Se (x.hub_arr[j] = 0) então
            t = randômico entre 0 e x.P
            x.ass_arr[j] = x.ind_hub[t];
```

```
        fim-se
    fim-para
    fitness(x);
fim-procedimento
```