MÉTODOS DE RESOLUÇÃO PARA O PROBLEMA DE PERCURSOS DE VEÍCULOS

RENATO LOURES BUENO FILHO

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Nelson Maculan Filho, D.Sc.
(Presidente)

Prof. Celso da Cruz Carneiro Ribeiro, D.ING.

Prof[a] Dina Feigenbaum Cleiman, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

ABRIL, 1989.

BUENO FILHO, RENATO LOURES

Métodos de Resolução para o Problema de Percursos de Veículos [Rio de Janeiro] 1989.

ix, 142 p. 29,5 cm (COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação, 1989).

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. Otimização de Percursos de Veículos I. COPPE/UFRJ II. Título (série).

A meus pais, Regina e Renato.

Agradecimentos

A Nelson Maculan Filho, pela orientação e apoio.

A Capes e CNPq, pela ajuda financeira.

A Celso Carneiro Ribeiro e Dina Feigenbaum Cleiman, pela participação na banca examinadora.

A Leu, pelo carinho e apoio incondicional durante todo o curso de mestrado.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M. Sc.).

MÉTODOS DE RESOLUÇÃO PARA O PROBLEMA DE PERCURSOS DE VEÍCULOS

RENATO LOURES BUENO FILHO

ABRIL, 1989

Orientador: Prof. Nelson Maculan Filho.
Programa: Engenharia de Sistema e Computação.

Neste trabalho apresentamos um estudo dos métodos de resolução para o Problema de Percursos de Veículos (PPV). Inicialmente, discutimos os principais métodos exatos e aproximados de resolução para o PPV básico. Em seguida, apresentamos uma implementação em microcomputador de dois algoritmos aproximados que solucionam o PPV com restrições adicionais relativas aos horários de atendimento dos clientes. Resultados computacionais são apresentados.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.).

METHODS OF SOLUTION FOR THE

VEHICLE ROUTING PROBLEM

RENATO LOURES BUENO FILHO

APRIL, 1989

Thesis Supervisor: Prof. Nelson Maculan Filho.

Department: System and Computer Engeneering.

We present in this work a study of solution methods for the Vehicle Routing Problem (VRP). In its first part, we consider the main exact and approximate methods for the basic VRP. In its second part, we present a microcomputer implementation of two approximate algorithms for the VRP with Time Windows constraints. Computational results are presented.

Índice

CAPÍTULO I

INTRODUÇÃO

O Problema de Percursos de Veículos (PPV) consiste em determinar percursos ótimos para uma frota de veículos estacionada em um domicílio (garagem ou depósito) de forma a atender um conjunto de clientes geograficamente dispersos.

Um exemplo clássico aparece nos problemas de distribuição de mercadorias (DANTZIG & RAMSER, 1959), onde cada cliente possui uma demanda específica e os veículos apresentam uma capacidade limitada. Porém, o problema também aparece em várias situações práticas não relacionadas com distribuição de produtos, como por exemplo, na entrega e/ou coleta de correspondências pelo correio, no transporte de crianças em ônibus escolares, na coleta de lixo, no planejamento de rotas aéreas, etc..

Os problemas de percursos de veículos são na sua grande maioria NP-árduos (LENSTRA & RINNOY KAN, 1981), portanto, pouco prováveis de serem resolvidos em tempo polinomial. O nosso interesse está em desenvolver um sistema para microcomputador que produza soluções de qualidade em um tempo computacional viável. Acreditamos que

isto é possível através da combinação de três fatores: uma modelagem cuidadosa do problema, a utilização de um método de solução eficiente e uma interface adequada com o usuário. Neste trabalho procuramos cumprir uma primeira etapa para o desenvolvimento deste sistema que consiste no estudo dos modelos e métodos de solução para o problema e a implementação de alguns destes métodos em um microcomputador.

Organizamos este trabalho em cinco capítulos. Ainda neste capítulo introdutório, apresentamos uma classificação para os problemas de percursos de veículos e definimos um problema básico. Este problema possui características que normalmente ocorrem nos vários problemas de percursos de veículos encontrados na prática e na literatura e geralmente os métodos que o solucionam podem ser facilmente estendidos ou adaptados para resolver problemas mais complexos. No capítulo II desenvolvemos os principais modelos e métodos exatos de solução e no capítulo III os métodos aproximados. No capítulo IV implementamos alguns destes métodos para resolver um problema de percursos de veículos um pouco mais complexo que aparece muito na vida real. Este problema, que considera restrições adicionais relativas aos horários de atendimento dos clientes, é conhecido na literatura como Problema de Percursos Veículos com restrições de Janela de Tempo. Finalmente, no capítulo V apresentamos algumas conclusões e indicamos futuros trabalhos e pesquisa que podem dar continuidade a este estudo.

I.1 - Classificação e Definição do PPV-básico

Devido a enorme variedade de problemas de percursos de veículos extraídos da vida real convém inicialmente classificá-los em três grandes grupos, a saber, Problemas de Percursos de Veículos (**PPV**), Problemas de Sequenciamento de Veículos (**PSV**) e Problemas de Percursos e Sequenciamento de Veículos (**PPSV**). Uma rota ou percurso de veículo é uma seqüência de pontos de entrega e/ou coleta que o veículo deve percorrer ordenadamente, iniciando e finalizando num depósito ou garagem; portanto, o **PPV** consiste em determinar as rotas que minimizam uma função objetivo, que normalmente é a distância percorrida pela frota. Um sequenciamento de veículos é uma rota onde associamos a cada ponto um tempo de chegada e um tempo de partida, ou de início e fim de serviço, portanto no **PSV** os veículos devem percorrer os pontos numa determinada ordem respeitando os tempos especificados. Quando os aspectos espacial e temporal se combinam de tal forma que um não prevalece sobre o outro temos o **PPSV**. Normalmente esta classe de problemas é caracterizada pela existência de um dos dois (ou ambos) seguintes tipos de restrição, relação de precedência entre dois clientes, isto é, quando é exigido que um determinado cliente seja servido antes que outro e Janela de Tempo ("Time Window") que ocorre quando os clientes exigem que o atendimento seja realizado dentro de um intervalo de tempo.

Naturalmente, esta classificação não é suficiente para definir completamente um problema de percursos de

veículos. Vários problemas podem ser encontrados dentro de cada uma das três classes definidas acima. BODIN & GOLDEN (1981) apresentam uma classificação para os problemas de percursos de veículos baseada nas características dos principais componentes sempre presentes nestes problemas, a saber: o depósito, os clientes, a frota de veículos e a rede. Por exemplo, o **PPV** que vamos definir em seguida possui um único depósito, os clientes possuem uma demanda conhecida, a frota de veículos é homogênea e a rede viária é não direcionada.

Vamos agora definir o **PPV**-básico para nos próximos capítulos apresentarmos os vários métodos de solução.

Considere um grafo $G = (N,A)$ onde $N = \{i = 1,\ldots,n\}$ é o conjunto de nós que representam os clientes, ou pontos de demanda, quando $i \neq 1$ e o depósito quando $i = 1$. Sejam $N' = N - \{1\}$ e $A = \{(i,j): i,j \in N\}$ o conjunto de arcos, cada um possuindo um custo c_{ij} que pode ser interpretado como a distância entre i e j ou como o tempo de viagem entre i e j. Cada cliente possui uma demanda d_i e a frota é composta por m veículos idênticos que possuem capacidade D.

O **PPV**-básico consiste em construir as rotas de custo mínimo de forma que:

(i) Cada cliente de N' deve ser visitado exatamente uma vez de forma a satisfazer sua demanda;

(ii) Cada veículo inicia e termina sua jornada no depósito;

(iii) A demanda total de cada rota de veículo deve ser menor ou igual que a capacidade do veículo, ou seja, $\sum_{i \in R} d_i \leq D$, onde $R \subset N'$;

(iv) O custo total (distância percorrida ou tempo de viajem) de cada rota deve ser menor ou igual que um limite dado L, i.é., $\sum_{i,j \in R} c_{ij} \leq L$.

Podemos classificar os problemas de percursos de veículos de acordo com sua matriz de custos $C = (c_{ij})$. O PPV é dito **Simétrico** quando a matriz C é simétrica, caso contrário o problema é **Assimétrico**. Além disto, o problema é dito **Euclidiano** quando C satisfaz à desigualdade triangular, ou seja, se $c_{ij} \leq c_{ik} + c_{kj}$ $(i,j,k \in N)$. Caso contrário o problema é **Não-euclidiano**.

Geralmente os algoritmos são desenvolvidos para resolver os problemas simétricos-euclidianos, porém são facilmente adaptáveis para resolver os demais casos.

I.2) Classificação dos métodos de solução

Os métodos de solução podem ser divididos em duas grandes classes: os métodos Exatos e os métodos

Aproximados.

Os métodos Exatos fornecem uma soluçao ótima para o problema e baseiam-se em técnicas conhecidas de Programação Matemática, porém sua aplicação é restrita a problemas de pequeno porte, já que a maioria dos problemas de percursos são NP-Árduos (LENSTRA & RINNOOY KAN, 1981].

Um algoritmo Aproximado é um procedimento que utiliza a estrutura do problema em questão aliado a uma heurística, muitas vezes intuitiva, para fornecer uma solução viável "quase-ótima" para o problema. Por solução viável "quase-ótima" entendemos uma solução que satisfaça as restrições do problema e forneça um valor para a função objetivo suficientemente bom ou próximo do valor ótimo. Naturalmente, uma solução aproximada pode ser comparada com um limite inferior para uma formulação exata, como por exemplo o dual de uma relaxação lagrangeana, para a verificação da sua qualidade.

CAPÍTULO II

MÉTODOS EXATOS

II.1 - Introdução

Neste capítulo apresentamos os principais algoritmos exatos para o **PPV-básico**, mencionamos também algumas extensões importantes quando algoritmos exatos foram desenvolvidos com sucesso.

Os métodos exatos de solução de problemas de percursos de veículos normalmente incluem procedimentos de enumeração do tipo **"Branch & Bound"**. Estes métodos estão intimamente relacionados com o modelo utilizado na formulação do problema, portanto, convém classificá-los de acordo com cada modelo. Utilizamos a seguinte classificação para os métodos de solução, sugerida em MAGNANTI (1981) e também seguida por LAPORTE & NOBERT (1985):

(i) Algoritmos de Busca Direta;
(ii) Algoritmos de Programação Dinâmica;
(iii) Algoritmos de Programação Inteira.

Estes últimos podem ser subdivididos em:

a) formulação por Particionamento de Conjuntos;
b) formulação por Fluxo de Veículos;

II.2 - Algoritmos de Busca Direta

Os métodos de busca direta consistem em construir rotas de veículos por meio de árvores de **"Branch & Bound"** (B-B). Métodos enumerativos do tipo B-B solucionam problemas de otimização discreta através do particionamento ("Branch") do conjunto das soluções viáveis em sucessivos subconjuntos de menor dimensão, calculando limitantes ("Bounds") sobre o valor da função objetivo em cada subconjunto. Estes "bounds" são usados para selecionar subconjuntos promissores e/ou para descartar outros que não têm possibilidade de conter a solução ótima. Os "bounds" são obtidos pela substituição do problema original por um problema ***relaxado***, mais fácil de ser resolvido, de tal forma que o valor da solução deste limita o valor da solução do primeiro. O procedimento termina quando cada subconjunto ou produziu uma solução viável ou demonstrou-se que não contém nenhuma solução melhor que aquela obtida até o momento. A melhor solução encontrada durante o processo é então uma solução ótima global.

Os ingredientes essenciais de qualquer algoritmo B-B para resolver um problema P de otimização discreta da forma Min $\{f(x) / x \in S\}$ são:

(i) **Uma relaxação de P**: um problema R da forma Min $\{g(x)/ x \in T\}$, tal que $S \subseteq T$ e para cada $x,y \in S$, $f(x) < f(y) \Rightarrow g(x) < g(y)$;

(ii) **Uma estratégia de "Branching" (separação):** uma regra que particione o conjunto viável S_i do problema corrente P_i em subconjuntos $S_{i_1}, \ldots, S_{i_k}$ tal que $\bigcup_{j=1,k} S_{i_j} = S_i$;

(iii) Uma estratégia de "Lower Bounding": um procedimento para determinar uma solução ótima (ou quase ótima) para cada relaxação R_i de cada subproblema P_i ;

(iv) Uma estratégia de Busca: uma regra para escolher o próximo subproblema a ser processado.

A seguir apresentamos duas estratégias de **"branching"**, que combinadas com diferentes estratégias de **"bounding"** produzem vários algoritmos.

II.2.1 - Branching nos Arcos

Nesta seção apresentamos uma estratégia de "branching" que foi definida por LITTLE et al. (1963) para resolver o Problema do Caixeiro Viajante (PCV) e mostramos como utilizá-la no caso do PPV.

Esta estratégia basea-se na escolha de um **arco** do grafo G para realizar a partição de um subproblema, representado por um nó na **"árvore de busca"** (figura II.1), em dois outros: o que contém todas as rotas que incluem o arco escolhido, e aquele que não inclue este arco nas suas rotas.

Seja x_{ij} a variável de decisão que possui os seguintes valores:

$$x_{ij} = \begin{cases} 1 & \text{se } (i,j) \text{ pertence à solução do problema;} \\ \emptyset & \text{caso contrário.} \end{cases}$$

Portanto cada partição definida acima corresponde à escolha de uma variável x_{ij} (variável de "branching") que se atribui o valor 1 ou Ø. Na figura II.1 abaixo pode-se visualizar este esquema.

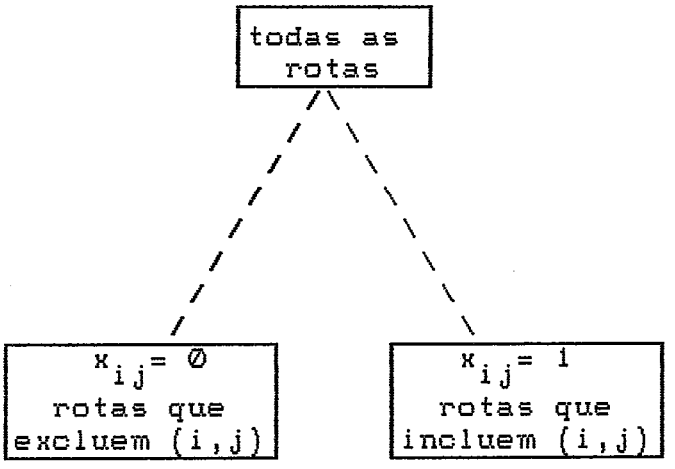

figura II.1 Árvore de Busca

O procedimento de "**branching**" é repetido até que algum nó da árvore de busca acima represente uma solução com rota única ótima.

Convém observar que em qualquer estágio deste processo, a união dos conjuntos representados pelos nós terminais da árvore é o conjunto de todas as rotas do problema.

No algoritmo de LITTLE et al. (1963) uma regra para a escolha da variável de branching é descrita da seguinte maneira.

> "Sejam X, Y e Y' nós da árvore de busca, onde X será o nó a ser particionado nos nós Y, quando $x_{kl} = 1$, e Y' quando $x_{kl} = \emptyset$, onde x_{kl} é a variável de branching escolhida.
>
> Inicialmente, a partir da matriz de custos determine a matriz de "**custos reduzidos**", isto é, uma matriz com elementos não negativos e com pelo menos um zero em cada coluna e linha. Esta matriz é obtida pela seguinte técnica de redução de linhas e colunas: subtrair cada linha (coluna) pelo menor elemento da linha (coluna). Note que os custos relativos de todas as rotas permanecem os mesmos, portanto, qualquer rota ótima na matriz original continua ótima na matriz reduzida.

O próximo passo é efetuar a **estratégia de "branching"**. Esta estratégia basea-se na escolha de um par de clientes de X, que será incluido em Y e excluído de Y', visando construir em Y a melhor rota contida em X. As rotas de menor custo a serem consideradas para Y são aquelas envolvendo um par de clientes (i,j) de Custo Reduzido $c_{ij} = \emptyset$.

Considere agora os custos das rotas que não contêm (i,j), i.é., rotas possíveis para Y'. Como o cliente **i** deve pertencer à rota do caixeiro viajante, os custos destas rotas devem pelo menos incluir o custo de menor valor entre os custos da linha **i**, excluindo-se c_{ij}. O mesmo ocorre com a cidade **j**, portanto, as rotas de Y' também devem incluir pelo menos o custo de menor valor entre os custos da coluna **j**, excluindo-se c_{ij}. Se definirmos como θ_{ij} a soma destes dois custos, o critério de escolha do par de cidades para o "branching" (variável de branching) será dado por:

$$\theta_{kl} = \text{máx} \{ \theta_{ij} \}.$$

Isto significa uma busca, na matriz reduzida, sobre os (i,j) que possuem $c_{ij} = \emptyset$ e $\theta_{ij} \neq \emptyset$, como mostra a figura II.2."

	1	2	3	4
1	∞	11	27	Ø (11)
2	1	∞	15	Ø (1)
3	Ø (0)	Ø (4)	∞	3
4	Ø (0)	4	Ø (15)	∞

Matriz de custos reduzidos. Os Nos. no canto superior direito são os valores de θ_{ij}.

$\theta_{k1} = \theta_{43} = 15$ portanto (4,3) será o par de cidades escolhido para o branching, e x_{43} a variável de branching.

figura II.2 - Exemplo de escolha de uma variável de "branching".

CHRISTOFIDES & EILON (1969) desenvolveram um algoritmo de B-B baseado na estratégia de "branching" descrita acima para resolver o PPV-básico.

O primeiro passo deste algoritmo consiste em transformar o PPV-básico em um Problema do Caixeiro Viajante (PCV). Isto pode ser feito da seguinte maneira:

> substituir o único depósito por m depósitos artificiais, todos localizados na mesma posição. Viagens entre os depósitos são proibidas atribuindo-se valores iguais a λ para as distâncias entre os depósitos, onde o valor que λ assume depende da F.O. do problema considerado (LENSTRA & RINNOOY KAN, 1975):

$\lambda = \infty$ fornece a distância mínima para m veículos;

$\lambda = \emptyset$ fornece a distância mínima para no máximo m veículos;

$\lambda = -\infty$ fornece a distância mínima para o número mínimo de veículos.

O número de depósitos artificiais m é igual ao número de veículos utilizados na solução ótima. A estrutura da matriz de distâncias do problema modificado, agora um PCV, é mostrada na figura II.3.

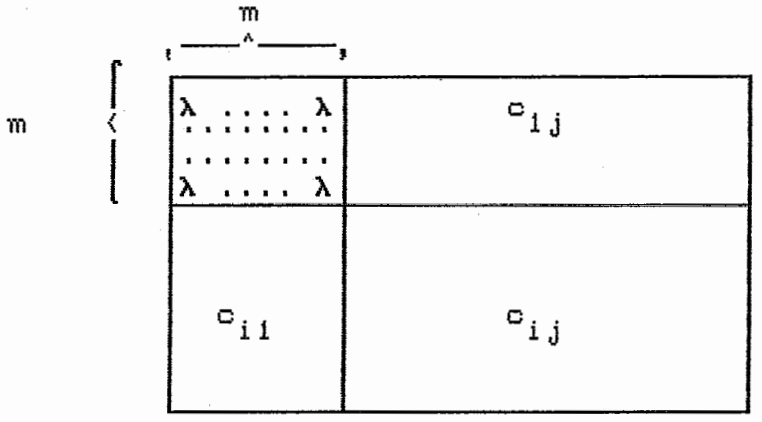

figura II.3 - Matriz de Distâncias para o PPV na forma do PCV.

Para resolver o PPV-básico o algoritmo abandona um "branch" quando :

(i) A carga total para um veículo excede sua capacidade;

(ii) A distância total acumulada por um veículo

excede o limite L;

(iii) A demanda total das cidades ainda não incluídas na rota excede a capacidade total dos veículos ainda não utilizados.

Além das questões acima, que descrevem as restrições de capacidade e distância, é necessáro determinar os "bounds" para os nós da árvore de B-B de maneira a reduzir a busca envolvida (estratégia de "bounding").

Já vimos que uma estratégia para o cálculo de "bounds" baseia-se numa relaxação do problema original. Para melhor visualizarmos as possíveis relaxações para o PCV , consideramos a seguinte formulação do problema em termos de Programação Linear Inteira (LAPORTE & NOBERT, 1985):

$$\text{(PCV)} \quad \textbf{Min} \sum_{i,j \in N} c_{ij}\, x_{ij} \qquad \text{(II.1)}$$

sujeito a

$$\sum_{i \in N,\ i \neq j} x_{ij} = 1 \quad (j \in N) \qquad \text{(II.2)}$$

$$\sum_{j \in N,\ i \neq j} x_{ij} = 1 \quad (i \in N) \qquad \text{(II.3)}$$

$$\sum_{i \in S} \sum_{j \in \overline{S}} x_{ij} \geq 1 \quad (2 \leq |S| \leq n-2,\ S \subset N) \qquad \text{(II.4)}$$

$$x_{ij} = \{ \emptyset, 1 \} \qquad (i \neq j \text{ e } i, j \in N) \qquad (II.5)$$

Nesta formulação, as restrições (II.2), (II.3) e (II.5) descrevem o Problema de Alocação (ou Designação). As restrições (II.4) eliminam as ocorrências de subrotas, obrigando a existência de pelo menos um arco interligando uma partição $(S,\bar{S})$ de N, i.é, evitam obter um grafo desconexo como solução.

LITTLE et al. (1963) apresentam uma estratégia de "bounding" que relaxa as restrições (II.4). O Problema de Alocação resultante é resolvido evitando-se a formação de subrotas impondo penalidades sobre os arcos que as formariam.

CHRISTOFIDES & EILON (1969) sugerem como limitante inferior uma **Árvore Geradora de Peso Mínimo**, descrevendo a sua justificativa da seguinte maneira:

> " A rota do caixeiro viajante através de n pontos é um ***Ciclo Hamiltoniano***, isto é, um ciclo onde cada nó possui grau dois. Uma **Árvore Geradora** é uma configuração de n-1 arcos passando por n pontos e uma **Árvore Geradora de Peso Mínimo**, ou simplesmente Árvore Geradora Mínima, é uma Árvore Geradora onde a soma de seus pesos é mínima. A figura II.4(a) mostra um exemplo de uma rota mínima de caixeiro viajante. A árvore geradora mostrada em (b) é obtida pela

remoção de um arco de (a), portanto (b) é um Limitante Inferior ("Lower Bound") para (a). A árvore de peso mínimo em (c) também é um Limitante Inferior para (a). Como a configuração em (c) consiste de n-1 ligações, um Limitante melhor para a rota do caixeiro viajante pode ser obtido pela adição de um arco conveniente, por exemplo, o menor arco do grafo, como está mostrado em (d). Alternativamente um ponto fictício pode ser adicionado, colocando-o na mesma posição de um já existente, porém, designando-se uma distância infinita entre eles (e). Desta forma converte-se o conjunto de n pontos em um de n+1 pontos. A árvore de peso mínimo dos n+1 pontos é um "Lower Bound" para a rota mínima do caixeiro viajante dos n pontos originais."

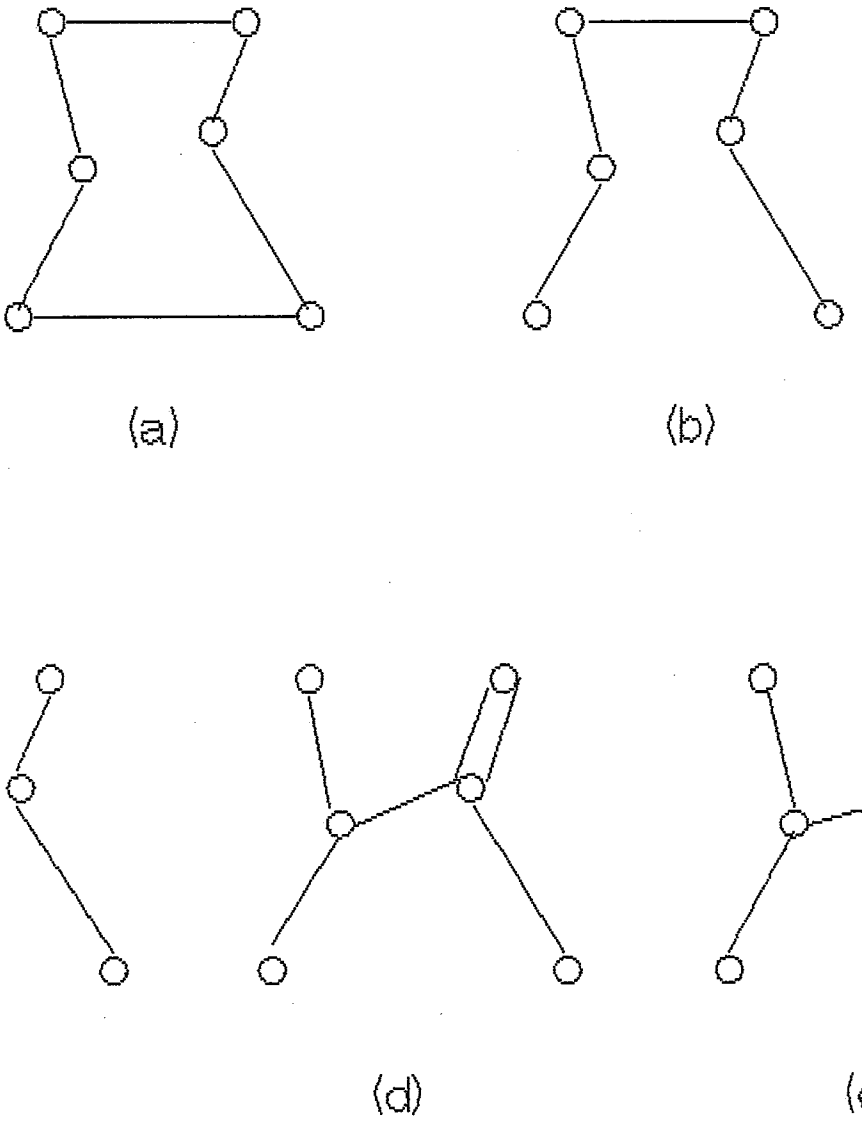

figura II.4 - A Árvore Geradora Mínima como um "bound" para o PCV.

II.2.2 - Branching nas Rotas

Em CHRISTOFIDES (1976) é descrito um algoritmo B-B onde as rotas viáveis são geradas como e quando desejadas. Portanto cada nó da árvore de busca corresponde à definição de uma nova rota de veículos.

O estado da busca em um estágio h de uma árvore de B-B pode ser representado por uma lista ordenada H, da seguinte maneira:

$$H = \{S_{j_1}(x_{i_1}),\ S_{j_2}(x_{i_2}),\ \ldots,\ S_{j_h}(x_{i_h})\}$$

onde $S_{j_r}(x_{i_r})$ é uma rota viável que inclui um cliente específico x_{i_r} e outros clientes que ainda não foram incluídos nas rotas anteriores $S_{j_1}(x_{i_1}),\ldots,\ S_{j_{r-1}}(x_{i_{r-1}})$.

Seja $F(h)$ o conjunto dos clientes livres, i.é., que não pertencem a nenhuma rota até o estágio h, e seja $\bar{F}(h) = N' - F(h)$ o conjunto dos clientes que já pertencem a uma rota em h.

Um "branching" de algum nó h envolve a escolha de um cliente $x_{i_{h+1}} \in F(h)$ e a construção de uma lista $P(x_{i_{h+1}})$ de todas as rotas viáveis passando por este cliente. Parece evidente que quanto menor for o número de ramificações ("branchs") possíveis em um nó h mais eficiente será a busca. Portanto, a escolha de $x_{i_{h+1}}$ deve

ser feita de forma a tornar a lista $P(x_{i_{h+1}})$ a menor possível. Este parece ser o caso em que $x_{i_{h+1}}$ é escolhido como o cliente mais afastado do depósito ou o que possui uma demanda muito alta.

Uma vez alcançado o fundo da árvore de busca, isto ocorre em um estágio m quando $F(m) = \phi$ (onde m é o número de rotas da solução ótima), a lista H contém uma solução para o problema que consiste de m rotas. Nesta altura um "***backtracking***" deve ser realizado para considerar as possibilidades ainda não analisadas.

Entretanto, nem todos os nós produzidos por $P(x_{i_{h+1}})$ necessitam ser explorados, em geral é possível mostrar que um nó específico pode ser abandonado. Isto pode ser feito em uma das seguintes circunstâncias:

(i) A rota que contém $x_{i_{h+1}}$ viola uma das restrições do problema;

(ii) Os veículos ainda não utilizados não podem suprir as demandas dos clientes de F(h+1), i.é, a capacidade total destes veículos é menor que o volume das demandas dos clientes livres;

(iii) Sejam Z^* o custo da melhor solução viável conhecida, C(S) o custo ótimo para suprir os clientes de $S \subseteq N'$, $C^-(S)$ um "lower bound" para

$C(S)$ e $C^{+}(S)$ um "upper bound". Então o nó $h+1$ da árvore de busca não precisa ser criado, a partir do nó h, se:

$$C(\bar{F}(H)) + C(\bar{F}(h+1) - \bar{F}(H)) + C^{-}(F(h+1)) \geq Z^{*} \ ;$$

(iv) Se $(h+1)$ é dominado por outro descendente imediato $(h'+1)$ de h, i.é.:

$$C(\bar{F}(h'+1)-\bar{F}(H))+C^{+}(F(h'+1)) \leq C(\bar{F}(h+1)-\bar{F}(H)+C^{-}(F(h+1))).$$

Nos ítens acima é assumido que limitantes superiores e inferiores para os subproblemas restantes, definidos pelos clientes livres, podem ser calculados em qualquer estágio. Limitantes inferiores podem, como já vimos, ser calculados pela relaxação de alguma das restrições e os limitantes superiores podem ser obtidos por alguma heurística.

Em CHRISTOFIDES et al. (1981) os autores apresentam duas maneiras de calcular "lower bounds" para o **PPV-básico**, que mostramos em seguida

II.2.3 - Bound a Partir da k-DCT

Uma **"k-Degree Centre Tree"** (**k-DCT**) é uma árvore onde o nó raiz possui grau **k**. Este "bound"

basea-se no fato de que o valor da solução ótima do Problema do Caixeiro Viajante Múltiplo (**m-PCV**) é limitado pelo peso total de uma **k-DCT**.

Podemos observar que o conjunto dos arcos pertencentes à solução ótima pode ser particionado em três subconjuntos, a saber:

(i) A_1: arcos que formam uma k-DCT, onde $k = 2m - y$ $y \leq m$;

(ii) A_2: y arcos incidentes ao nó 1 (depósito);

(iii) A_3: m - y arcos não incidentes ao nó 1.

Para derivarmos o "bound" a partir da **k-DCT** é necessário antes apresentarmos uma nova formulação para o problema.

Sejam:

.$\xi_I^t = \begin{cases} 1 & \text{se } I \in A_t \\ 0 & \text{c.c.} \end{cases}$, onde t= 1,2,3 e $I \in A$;

.y: número de arcos incidentes ao depósito;

.c_I: o custo do arco I;

.A^i: o conjunto dos arcos incidentes ao nó i;

.$(S,\bar{S})$: o conjunto de todos os arcos com um vértice em S e o outro em $\bar{S}$.

Então, podemos formular o problema da seguinte maneira:

$$(\text{m-PCV})\ \text{Min} \sum_{I \in A} c_I(\xi_I^1 + \xi_I^2 + \xi_I^3) \qquad (II.6)$$

sujeito a:

$$\sum_{I \in (S,\bar{S})} \xi_I^1 \geq 1 \qquad (S \subseteq A,\ S \neq \emptyset) \qquad (II.7)$$

$$\sum_{I \in A_1} \xi_I^1 = 2m - y \qquad (II.8)$$

$$\sum_{I \in A} \xi_I^1 = n - 1 \qquad (II.9)$$

$$\sum_{I \in A_1} \xi_I^2 = y \qquad (II.10)$$

$$\sum_{I \in A-A^1} \xi_I^3 = m - y \qquad (II.11)$$

$$\sum_{I \in A^i} (\xi_I^1 + \xi_I^2 + \xi_I^3) = 2 \qquad (i \in N') \qquad (II.12)$$

$$\xi_I^t \in \{0,1\} \ ; \ t=1,2,3 \ \text{e} \ I \in A \qquad (II.13)$$

$$y \geq 0 \ \text{e inteiro} \qquad (II.14)$$

A restrição (II.7) impõe a conectividade da **k-DCT**, (II.8) define o grau do vértice 1 (depósito), (II.9),(II.10) e (II.11) especificam o número de arcos necessários e (II.12) garantem que os nós de N' tenham grau igual a 2.

Finalmente, (II.13) e (II.14) impõe a integralidade às variáveis de decisão.

Na formulação acima o número de veículos foi considerado como uma variável de decisão **y**, porém se fixarmos o seu valor em m, estaremos considerando uma frota cujo tamanho é fixado a priori. Notamos também que a formulação acima não considera as restrições de capacidade e distância, sendo portanto o Problema do Caixeiro Viajante Múltiplo (**m-PCV**).

O "bound" é obtido através da Relaxação Lagrangeana das restrições (II.12), decompondo o problema em três subproblemas P_1, P_2 e P_3. A função objetivo do Problema Lagrangeano fica:

$$V^t(\lambda,y)= \Sigma_{I\in A}\ \bar{c}_I\omega_I - 2\ \Sigma_{i\in N'}\ \lambda_i$$

onde:

.$\lambda_i \geq \emptyset$, $i\in N'$ é o vetor de penalidades associadas com as restrições relaxadas, e $\lambda_1 = \emptyset$;

.$\bar{c}_I = c_I + \lambda_{i_I} + \lambda_{j_I}$ onde i_I e j_I são os dois nós adjacentes ao arco I;

.t = 1,2 ou 3;

.$\omega_I = \{\xi_I^t;\ t = 1,2\ 3\}$ de acordo com cada problema.

Para um dado valor de y e λ, sejam:

.$V^1(\lambda,y)$ o valor da solução ótima de P_1 dado pelas restrições (II.7), (II.8), (II.9) e (II.13);

.$V^2(\lambda,y)$ o valor da solução ótima de P_2 dado pelas restrições (II.10) e (II.13);

.$V^3(\lambda,y)$ o valor da solução ótima de P_3 dado pelas restições (II.11) e (II.13).

Então um "lower bound" para o (m-PCV) é

$$\beta = \text{Máx}_{m_1 \leq y \leq m} \{\text{Máx}_{\lambda \geq 0} \Sigma_{t=1,3} V^t(\lambda,y)\} \qquad (II.15)$$

onde m_1 representa um limitante sobre o número de rotas na solução ótima.

O "bound" acima naturalmente também vale para o PPV-básico. Mais que isto, podemos melhorá-lo quando consideramos as restrições de capacidade e distância (ou tempo) percorrida por cada veículo. Esta melhoria baseia-se na escolha de um melhor valor de m_1 para o PPV. Isto é feito da seguinte maneira (LAPORTE & NOBERT, 1985):

suponha os clientes ordenados em ordem decrescente de suas demandas d_i. Como m_1 rotas podem no máximo suprir uma quantidade $\Sigma_{i=1,m_1} d_i$ e os veículos restantes devem estar aptos a suprir as

demandas ainda não atendidas, temos que:

$$(m - m_1)\ D \geq \sum_{i=m_1+1,n} d_i \qquad (II.16)$$

Analogamente, cada cliente i contribui com uma quantidade de pelo menos $u_i = \delta_i + 1/2(c_{ii_1} + c_{ii_2})$ para o comprimento de uma rota, onde δ_i **é o tempo gasto no atendimento do cliente** i, i_1 e i_2 são as duas cidades mais próximas de i. Então, se as cidades forem ordenas em ordem decrescente de u_i, m_1 deve satisfazer a:

$$(m - m_1)\ L \geq \sum_{i=m_1+1,n} u_i \qquad (II.17)$$

Portanto, podemos tomar m_1 como sendo o maior valor que satisfaz (II.16) e (II.17).

A seguir descrevemos os procedimentos sugerido em CHRISTOFIDES et al. (1981) para o cálculo de cada $V^t(\lambda,y)$, t = 1,2 e 3, usado no cálculo do "bound" em (II.15).

- <u>Cálculo de $V^1(\lambda,y)$</u>

Para um dado valor de λ e de y, resolver P_1 significa determinar uma k-DCT de peso mínimo. Seja T_k^* a solução de P_1. T_k^* pode ser obtida através do cálculo da Árvore Geradora Mínima do grafo, verificando-se o grau g do vértice 1 (o depósito) da seguinte maneira:

(i) se $g = k$ então o problema está resolvido;

(ii) se $g < k$ então uma penalidade $\mu > 0$ é dada ao vértice i, e os custos c_{ij} são trocados para $c_{ij} - \mu$;

(iii) se $g > k$ então uma penalidade $\mu < 0$ é dada ao nó i e os custos c_{ij} tornam-se $c_{ij} - \mu$.

Nos casos (ii) e (iii) a Árvore Geradora Mínima é recalculada, e o procedimento é repetido até que $g = k$.

- <u>Cálculo de $V^2(\lambda,y)$</u>

A solução de P_2 é dada pelos y arcos incidentes ao depósito, isto é que pertencem a A_2, de menor comprimento $\bar{c}_{ij}$. Uma boa escolha destes y arcos pode ser obtida pelo seguinte procedimento:

.defina $\bar{c}_1^r$ e $\bar{c}_2^r$ como sendo os custos dos dois arcos que ligam uma rota r ao depósito; suponha que $\bar{c}_2^r \geq \bar{c}_1^r$ e note que quando $\bar{c}_1^r = \bar{c}_2^r$ a rota é formada por um único cliente;

.Ordene crescente e lexicograficamente as rotas pelo vetor $(\bar{c}_2^r, \bar{c}_1^r)$, renumerando-as de maneira que a

rota r se refira à r-ésima rota da ordenação. Como para cada uma das y rotas podemos escolher para elemento de A_2 o mais longo entre os dois arcos que ligam cada rota ao depósito, temos que o custo de A_2 será sempre menor ou igual a $\sum_{r=1,y} \bar{c}_2^r$;

.Construa uma outra lista dos arcos adjacentes ao depósito, ordenada crescentemente pelos custos $\bar{c}_{1j}$, repetindo os primeiros m_1 custos da lista de forma que estes apareçam duas vezes na lista. Defina $h(p)$ como o custo na p-ésima posição desta lista. Então, temos que para qualquer solução do **m-PCV**, acima formulado, $\bar{c}_2^1 \geq h(2)$, $\bar{c}_2^2 \geq h(4)$, $\bar{c}_2^3 \geq h(6)$, etc. e em geral $\bar{c}_2^r \geq h(2r)$ para qualquer $r = 1,\ldots,m$;

Então, temos que:

$$L^2(\lambda,y) = \sum_{r=1,y} h(2r) \leq \sum_{r=1,y} \bar{c}_2^r$$

é um limitante inferior para o custo de A_2 e pode ser usado em (II.15) no lugar de $V^2(\lambda,y)$.

- <u>Cálculo de $V^3(\lambda,y)$</u>

A solução de P_3 é dada pelo somatório dos custos $\bar{c}_I$ dos $(m-y)$ arcos de menor custo não incidentes ao depósito (i.é., arcos que pertencem a A_3).

Seja $\bar{c}_i[1]$ o custo do arco incidente ao nó i de menor custo. Não considerando o depósito, suponha que as quantidades $c_i[1]$ estejam ordenadas na ordem crescente e que os clientes tenham sido renumerados de forma que i se refira ao i-ésimo desta lista. Desta maneira, $V^3(\lambda,y)$ é calculado por:

$$V^3(\lambda,y) = \sum_{i=1,m-y} \bar{c}_i[1] \qquad (II.18)$$

- O Algoritmo para o cálculo do bound pela k-DCT

Para definir por completo a estratégia de "bound", baseada na relaxação da k-DCT, é apresentado em CHRISTOFIDES et al. (1981) um algoritmo onde:

.y é inicializado com o valor m_1 (dado por (II.16) e (II.17)) e sempre que necessário é incrementado de uma unidade;

.os valores λ_i, i=1,n , são calculados pela fórmula dada pelo **método do subgradiente** até um número de iterações fixado à priori.

A seguir apresentamos o algoritmo.

PØ. (inicialização)

defina Z_L^* como o melhor lower bound e Z_U^* o valor da melhor solução encontrada até o momento. Faça $Z_L^* = \emptyset$ e $y = m_1$;

P1. (Início passo iterativo)

Faça kount= 1, $\lambda_i = 0$ para i=1,...,n , k= 2m-y;

P2. (k-DCT)

Calcule T_k^* da maneira que mencionamos anteriormente;

P3.(Arcos adicionais)

Calcule $L^2(\lambda,y)$ e $V^3(\lambda,y)$;

P4. (Grafo PPV)

Forme o grafo G pela união dos arcos encontrados nos passos P2 e P3 e faça Z_L= custo total deste grafo, que é um "lower bound" para a solução do **PPV**.

Se $Z_L^* < Z_L$ faça:

.$Z_L^* = Z_L$;

.Se $Z_L^* \geq Z_U^*$ PARE (este nó deve ser abandonado, realize um backtracking no algoritmo principal)

.Senão (se $Z_L^* < Z_U^*$) vá para o passo P5;

Se $Z_L^* \geq Z_L$ e kount= No. máximo de iterações permitidas vá para o passo P7;

Senão faça kount = kount + 1 e vá para o passo P5.

P5. (Penalidades λ_i)

Se o grau g_i do vértice x_i do grafo G é igual a dois e $g_1 = 2m$, PARE (Z_L^* é o melhor "lower bound" que pode ser obtido por este procedimento);

caso contrário calcule as penalidades λ_i da seguinte maneira:

$$\lambda_i = \lambda_i + \alpha \frac{(Z_U^* - Z_L)}{[\ \Sigma_{j \in N'} (d_j - \sigma_j)^2\ ]^{1/2}} (d_i - \sigma_i)$$

onde $\sigma_i = 2$ se $i \neq 1$ e $\sigma_i = 2m$ se $i = 1$ e α é uma cte..

P6. (Atualize matriz de custos)

Faça $c_{ij} = c_{ij} + \lambda_i + \lambda_j \quad \forall\ i,j \in N$ e vá para o passo P2;

P7. (Atualize y)

Se y= m PARE, senão faça y= y + 1 e vá para o passo P1.

II.2.4 - Bound a Partir da q-Rota Mínima

O segundo "bound" desenvolvido em CHRISTOFIDES et al. (1981) é baseado em uma rota especial, denominada q-rota, que definimos abaixo.

Seja W o conjunto de todas as cargas (demandas) que podem existir em qualquer rota de veículo, ou seja:

$$W = \{q \;/\; \sum_{i \in N'} d_i \xi_i = q \leq D, \text{ para algum } \xi_i \in \{0,1\} \}$$

(II.19)

Considere que os elementos de W estão ordenados crescentemente e que $q(l)$ é o valor do l-ésimo elemento de W. Uma q-rota de menor custo, $\Psi_l(i)$, é uma rota que:

(i) passa pelo nó i;

(ii) inicia e termina no depósito;

(iii) não possui ciclos do tipo (i_1, i_2, i_1) ;

(iv) possui demanda total $q(l)$.

Para derivar um "bound" a partir desta **q-rota**, é necessário que formulemos o **PPV** da seguinte maneira:

$$\textbf{(PPV1)} \quad \textbf{Min} \sum_{r=1,r'} c_r y_r \qquad \text{(II.20)}$$

sujeito a:

$$\sum_{r \in R_i} y_r = 1 \qquad (i=1,n) \qquad \text{(II.21)}$$

$$\sum_{r=1,r'} y_r = m \qquad \text{(II.22)}$$

$$y_r \in \{0,1\} \qquad \text{(II.23)}$$

onde:

.r = 1,r' é o índice das r' rotas viáveis;

.c_r é o custo da rota r;

.R_i é o conjunto de índices de rotas que visitam o cliente i;

.$y_r = \begin{cases} 1 & \text{se a rota r pertence à solução;} \\ \emptyset & \text{c.c.} \end{cases}$

Podemos reformular o problema acima substituindo y_r por:

$$y_r = \frac{1}{K_r} \sum_{i \in M_r} \xi_{ir} d_i \quad ; \quad \xi_{ir} \in \{\emptyset, 1\} \ ,$$

onde M_r é o conjunto de índices dos clientes da rota r e $K_r = \sum_{i \in M_r} d_i$ é a carga total da rota r, e substituindo a restrição (II.21) pelas seguintes :

$$\sum_{r \in R_i} \xi_{ir} = 1 \qquad (i = 1,n) \qquad \text{(II.24)}$$

$$\xi_{ir} = \frac{1}{K_r} \sum_{j \in M_r} \xi_{jr} \, d_j \qquad (i \in M_r \ ; \ r = 1,r') \qquad \text{(II.25)}$$

A restrição (II.25) garante que $\xi_{ir} = 1$ se e só se $\xi_{jr} = 1 \ \forall \ j \in M_r$ e portanto $y_r = 1$.

Vamos relaxar este novo problema removendo as restrições (II.25) e trocando o conjunto M_r pelo conjunto completo de clientes N', para obter a seguinte formulação:

$$\text{(PPV2)}\quad \text{Min} \sum_{i\in N} \sum_{I=1,w} \bar{c}_{iI}\, \xi_{iI} \qquad \text{(II.26)}$$

sujeito a:

$$\sum_{I=1,w} \xi_{iI} = 1 \quad (i = 1,\ldots,n) \qquad \text{(II.27)}$$

$$\sum_{i\in N'} \sum_{I=1,w} \frac{d_i}{q(I)}\, \xi_{iI} = m \qquad \text{(II.28)}$$

$$\sum_{i\in N'} \sum_{I=1,w} d_i \xi_{iI} = D_T \qquad \text{(II.29)}$$

$$\xi_{iI} \in \{0,1\} \qquad \text{(II.30)}$$

onde $\bar{c}_{iI} = c_{iI}\, d_i/q(I)$, c_{iI} sendo o custo da rota que passa por i com carga $q(I)$; $D_T = \sum_{i\in N'} d_i$ e $w = |W|$.

É evidente que o custo $\psi_I(i)$, custo da rota mínima que passa por i com carga $q(I)$, é um "lower bound" para c_{iI}. Portanto a solução do problema

$$\text{Min} \sum_{i\in N'} \sum_{I\in W} b_{iI}\xi_{iI} \qquad \text{(II.31)}$$

sujeito a (II.27), (II.28), (II.29) e (II.30), onde $b_{iI} = \psi_I(i)\, d_i/q(I)$ é um lower bound para o PPV.

Obs.: Note que b_{iI} é um lower bound para $\bar{c}_{iI}$ obtido pela relaxação das restrições que garantem que em uma solução viável o grau de cada nó, diferente do depósito, é

igual a dois.

Finalmente, podemos facilmente calcular um "lower bound" para o **PPV** se ignorarmos as restrições (II.28) e (II.29) do problema acima definido, resultando em:

$$\beta = \sum_{i \in N'} \; \min_{l \in W} \; [b_{il}].$$

Uma alternativa mais refinada seria calcular o "lower bound" através de uma Relaxação Lagrangeana das restrições (II.28) e (II.29).

II.3 - Algoritmos de Programação Dinâmica

Os principais elementos de um algoritmo de Programação Dinâmica (**P.D.**) são **Estados**, **Transições** entre estados e **Equações de Recorrência** para determinar o valor da Função Objetivo em cada estado. A formulação de um problema de percursos de veículos através da Programação Dinâmica apresenta uma dificuldade comum a vários problemas de otimização combinatória: o número extremamente elevado de estados torna sua resolução inviável na prática.

Afim de contornar esta dificuldade CHRISTOFIDES et al. (1981b) introduziram um método, denominado "**Relaxação do Espaço de Estados**", para reduzir o número de estados do problema.

II.3.1 - Relaxação do Espaço de Estados

Relaxar o espaço de estados associado a uma dada recursão de programação dinâmica consiste em reduzir o número de estados de tal forma que a solução da recursão relaxada determine um limitante ("bound") para o valor ótimo do problema; este "bound" pode ser então utilizado em um procedimento B-B para solucionar o problema original.

A redução do número de estados é feita através de uma função $g(.)$ que mapeia valores de um espaço de estados S em um espaço de estados G de menor cardinalidade. Portanto, se existir uma transição de um estado S_1 para um estado S_2 no espaço de estados original, então deve existir uma transição de $g(S_1)$ para $g(S_2)$ no espaço de estados relaxado G.

A relaxação do espaço de estados pode ser bem entendida através do clássico **Problema do Caixeiro Viajante (PCV)**. Vamos considerar a seguinte formulação do **PCV** em termos de Programação Dinâmica **(P.D.)**.

Seja $f(S,i)$ o custo do caminho mais curto que inicia no depósito 1, visita cada nó de S e termina no nó $i \in S$. A recursão da **P.D.** é dada por:

$$f(S,i) = \min_{j \in S-\{i\}} \{f(S-\{i\},j) + c_{ji}\} \qquad (II.30)$$

para todo $S \subseteq N'$ e $\forall\, i \in S$.

A inicialização da recorrência é dada por

$$f(\{i\},i) = c_{1i} \quad \forall\, i.$$

Então, a solução ótima do problema é

$$\min_{i \in N'} \{f(N',i) + c_{i1}\}.$$

Seja $g(.)$ a função que mapeia o domínio de (S,x) para algum outro espaço $(g(S),x)$ de menor cardinalidade. Então, a relaxação da recursão (II.30)para o PCV torna-se:

$$f(g(S),x) = \min_{y \in S'} \{f(g(S-\{x\}),y) + c_{yx}\} \qquad (II.31)$$

onde $\Gamma^{-1}(x) \supseteq S' \supseteq S-\{x\}$. A inicialização é dada por:

$$f(w,y) = \begin{cases} c_{1y} & \text{se } w = g(\{y\}), \\ \infty & \text{caso contrário.} \end{cases}$$

Listamos abaixo cinco funções $g(.)$ examinadas em CHRISTOFIDES et al. (1981b), que podem ser utilizadas na relaxação descrita acima.

(i) $g(S) = |S|$;

(ii) $g(S) = \sum_{i \in S} d_i$;

(iii) $g(S) = (|S|, \sum_{i \in S} d_i)$;

(iv) $g(S) = (\sum_{i \in S} d_i^1 , \sum_{i \in S} d_i^2)$, onde d_i^1 e d_i^2 são inteiros tais que $d_i^1 + d_i^2 \geq 1$;

(v) $g(S) = (\sum_{i \in S_1 \cap S} d_i , \sum_{i \in S_2 \cap S} d_i , \ldots , \sum_{i \in S_r \cap S} d_i)$

onde $S_1, \ldots, S_r$ representam uma partição do conjunto N'.

II.3.2 - Formulações do PPV em Programação Dinâmica

É importante observar que a eficácia da relaxação do espaço de estados em produzir limitantes é relativa a uma dada formulação de P.D.. Nesta seção apresentamos três formulações para o PPV com restrições de capacidade. Na seção seguinte derivamos os limitantes a partir da relaxação do espaço de estados para cada uma destas formulações.

a) Formulação 1

Sejam $f(m',S)$ o menor custo para suprir um conjunto S de clientes usando apenas m' veículos; $v(S)$ a solução para o PCV definido pelos clientes pertencentes a S e o depósito 1. Então a fórmula de recursão da P.D. fica:

$$f(m',S) = \underset{U \subset S}{\text{Min}} \{f(m'-1,S-U) + v(U)\} \qquad (II.32)$$

sujeito a:

$$\sum_{i \in S} d_i - (m'-1)D \leq \sum_{i \in U} d_i \leq D \quad p/\ m' = 2,m \qquad (II.33)$$

onde $S \subseteq N'$ deve satisfazer a:

$$D_T - (m-m')D \leq \sum_{i \in S} d_i \leq m'D \qquad (II.34)$$

onde $D_T = \sum_{i \in N'} d_i$.

A recursão é inicializada por $f(1,S) = v(S)$.

b) Formulação 2

Vamos ver uma relaxação alternativa que contém alguma redundância de estados em relação a anterior. Esta redundância é útil quando realizamos a relaxação do espaço de estados (uma analogia à relaxação lagrangeana pode ser

feita para uma primeira justificativa deste fato).

Seja $f(m',S,k)$ o menor custo para suprir os clientes de S, usando m' veículos, com os últimos clientes das correspondentes m' rotas estando entre os clientes 2,...,k. E seja $v(S,k)$ a solução do PCV no cojunto S onde o último cliente da rota é k. Neste caso a recursão é dada por:

$$f(m',S,k) = \mathbf{Min}\{f(m',S,k-1),\ \min_{U \subset S} \{f(m'-1,S-U,k-1)+v(U,k)\}\} \qquad (II.35)$$

sujeito a (II.33).

c) **Formulação 3**

Seja $f(m',S,\bar{d},i)$ o menor custo para suprir os clientes de S usando m' veículos onde as primeiras (m'-1) rotas estão fechadas e a última rota está aberta, tem uma demanda de $\bar{d}$ e termina no cliente i. A recursão desta P.D. é:

$$f(m',S,\bar{d},i) = \min_{j \in S \cap \Gamma^{-1}(i)} \{f(m',S-\{i\},\bar{d}-d_i,i) + c_{ji}\} \qquad (II.36)$$

para $d_i \leq \bar{d} \leq D$ e $m'=1,\ldots,m$.

As funções são inicializadas por :

$$f(1,\{i\},\bar{d},i) = \begin{cases} c_{1i} & \text{para } \bar{d} = d_i \\ \infty & \text{c.c.} \end{cases} \qquad (II.37)$$

O valor ótimo do **PPV** é então dado por:

$$\min_{i\in\Gamma^{-1}(1),\,\bar{d}} \{f(m,N',\bar{d},i) + c_{i1}\} \qquad (II.38)$$

onde $D_T - (m-1)D \leq \bar{d} \leq D$.

II.3.3 - Relaxação do Espaço de Estados e Bounds para o PPV

As mesmas funções de mapeamento utilizadas no PCV podem ser usadas para relaxar as retrições dadas acima, no caso do **PPV**. As variaveis S, do vetor de estado em todas as formulações acima, podem ser relaxadas através das funções **g(S)** dadas na seção (II.2.1).

a) Relaxação da formulação 1

Considere $g(S) = q = \sum_{i\in S} d_i$, então a relaxação da recursão (II.32) fica:

$$f(m',q) = \min_{p} \{f(m'-1,q-p) + \bar{v}(p)\} \qquad (II.39)$$

onde $D_T-(m-1)D \leq p \leq \min\{q,D\}$ e $\bar{v}(p)$ é o custo mínimo do circuito C para o qual $\sum_{i\in C} d_i = p$. Como o cáloulo de $\bar{v}(p)$ é um problema difícil , define-se uma função $\bar{v}(p)$

para ser um Limitante Inferior para este custo:

$$\bar{v}(p) = \min_{i \in \Gamma^{-1}(1)} \{f(p,i) + c_{i1}\} \qquad (II.40)$$

onde:

$$f(p,i) = \min_{j \in E^{-1}(p,i)} \{f(p-p_i, j) + c_{ji}\} \qquad (II.41)$$

onde:

$$f(p,i) = \begin{cases} c_{1i} & \text{se } q=d_i \\ \infty & \text{caso contrário} \end{cases} ;$$

$$E^{-1}(p,i) = \{j \;/\; i \in \Gamma(j),\ p-d_i \geq d_j\}.$$

Um limitante inferior para o PPV é então dado por $f(m,D_T)$.

b) Relaxação da formulação 2

Um bound melhor pode ser obtido a partir da recursão (II.35). Usando-se a mesma função $g(S) = q$ chega-se à seguinte relaxação de (II.35):

$$f(m',q,k) = \min \{f(m',q,k-1),\ \min_{p} \{f(m'-1,q-p,k-1)+\bar{v}(p,k)\}\} \qquad (II.42)$$

para $D_T-(m-1)D \leq p \leq \min \{q,D\}$

onde $\bar{v}(p,k) = f(p,k) + c_{k1}$.

O limitante inferior será dado por

$f(m,D,n) \geq f(m,D_T)$.

c) Relaxação da formulação 3

Utilizando-se mais uma vez da função de mapeamento $g(s)= q$ para relaxar a recursão (II.38), obtemos:

$$f(m',q,\bar{d},i) = \min_{j\in\Gamma^{-1}(i)} \{f(m',q-d_i,\bar{d}-d_i,j) + c_{ji}\} \quad (II.43)$$

O "lower bound" obtido diretamente da recursão acima é:

$$\min_{i\in\Gamma^{-1}(i)} \{f(m,D_T,\bar{d},i) + c_{i1}\}$$

onde $D_T-(m-1)D \leq \bar{d} \leq D$.

II.4 - Algoritmos Baseados em Particionamento de Conjuntos

Podemos formular o **PPV** como uma generalização do Problema de Particionamento de Conjunto (Set Particioning), como sugerem BALINSKI & QUANDT (1964).

Consideremos $R = \{r \ / \ r = 1,\ldots,m\}$ como sendo o conjunto das rotas viáveis de um **PPV**. Sejam a_{ir} um coeficiente que toma o valor 1 se, e só se, a cidade i pertence à rota r, c_r^* o custo ótimo da rota r, e a variável x_r= 1 se a rota r pertence à

solução ótima ou $x_r = 0$ caso contrário. Podemos então escrever o **PPV** da seguinte maneira:

$$\textbf{(PPV3) Min} \quad \sum_{r \in R} c_r^* x_r \qquad \text{(II.44)}$$

sujeito a:

$$\sum_{r \in R} a_{ir} x_r = 1 \quad (i \in N') \qquad \text{(II.45)}$$

$$x_r \in \{0,1\} \qquad \text{(II.46)}$$

Existem duas dificuldades associadas a esta formulação:

(i) o excessivo número de variáveis binárias que um problema real pode alcançar;

(ii) a dificuldade para calcular todos os valores c_r^*. Por exemplo, no **PPV-básico**, cada rota r corresponde a um conjunto de cidades S_r que satisfaz a:

(a) $\sum_{i \in S_r} d_i \leq D$ (restrição de capacidade);

(b) $\sum_{i,r \in S_r} c_{ir} \leq L$ (restrição de distância).

Portanto para obter o valor de cada c_r^* é preciso resolver um Problema do Caixeiro Viajante sobre S_r.

Podemos enfrentar estas dificuldades através da

técnica de **Geração de Colunas**. No método Simplex padrão, a cada iteração devemos escolher uma variável não-básica para entrar na base, logo, para resolvermos o **PPV3** acima precisamos ter em mãos as colunas $a_{.r}$ relativas a todas as rotas viáveis possíveis, que como já vimos exige um esforço computacional muito elevado. Na técnica de geração de colunas nós só calculamos as colunas na medida em que precisamos, ou seja, uma nova coluna - relativa à variável entrante na base - de custo marginal mínimo é gerada pela solução de um subproblema apropriado em cada iteração do Simplex. Se o custo marginal é negativo então a coluna é adicionada ao problema de Programação Linear, o problema é reotimizado e a geração de coluna é novamente aplicada; se, ao contrário, o custo marginal é positivo siginifica que a solução corrente é a ótima.

Esta técnica é utilizada por vários autores para tratar de **PPV's** que podem ser formulados por Particionamento de Conjuntos. Vejamos alguns destes exemplos.

FOSTER & RYAN (1976) sugerem que um esquema de geração de colunas onde cada rota é obtida por Programação Dinâmica pode teoricamente ser desenvolvido. Porém este esquema possui uma taxa de convergência muito lenta, além de raramente fornecer uma solução inteira.

Tentando evitar estes problemas, FOSTER & RYAN (1976) apresentam uma alternativa de solução que se

baseia no seguinte procedimento:

(i) Definir uma nova região viável, mais restrita que a original, de maneira a diminuir o número de rotas a serem exploradas, consequentemente diminuindo o número de colunas a serem geradas;

(ii) Usando informações providas deste modelo, que passamos a chamar de "Problema Restrito", relaxar progressivamente as restrições adicionais sobre as novas rotas viáveis de maneira a encontrar uma solução ótima para o problema original.

Podemos restringir a região viável observando que:

(a) as rotas servem um setor da região com centro no depósito. Pontos de entrega dentro de um setor são raramente desviados; exceções resultam de restrições na estrutura das rotas (p.ex., capacidade e distância);

(b) rotas adjacentes raramente se cruzam;

(c) para algum subconjunto de pontos de entrega, a rota ótima é a solução do Problema do Caixeiro Viajante para aquele subconjunto de pontos de entrega.

Portanto, se restringirmos a composição de R a rotas com estes atributos, nós efetivamente temos que

resolver um modelo mais "restrito" do **PPV**. O problema de definir conjuntos restritos de rotas se reduz a designar pontos de entregas às rotas, baseadas em (i) e (ii) e então achar a solução do PCV para cada rota.

Existem várias maneiras de se realizar a designação acima. FOSTER & RYAN (1976) desenvolveram um algoritmo que segue o seguinte procedimento.

Os pontos de entrega são ordenados radialmente a partir do depósito e todos os subconjuntos contíguos desta ordenação são considerados candidatos para uma rota. Subconjuntos cuja capacidade total excedem o limite para uma rota são rejeitados e os remanescentes são submetidos a uma heurística que solucione o PCV. A rota resultante é então adotada como uma rota viável, contanto que não exceda o limite **L** adotado para a distância máxima permitida. A solução viável produzida desta maneira fornece rotas do tipo radial (semelhante a pétalas de uma flôr).

O procedimento acima visa obter boas soluções para o **PPV**-básico dentro de um tempo computacional razoável. Como resultado obtemos uma solução viável restrita a um conjunto de rotas promissoras. A mesma filosofia pode ser adotada no processo de relaxamento da região viável do Problema Restrito. Ou seja, o processo é direcionado de maneira que a busca seja realizada em áreas onde exista uma provável melhoria. Uma relaxação que obteve sucesso entre as investigadas pelos autores é a que troca um cliente entre

duas rotas de maneira a reduzir a distância percorrida.

Uma característica que merece destaque é a propriedade de "quase-integralidade" da estrutura da região viável do problema Restrito: em nenhum dos problemas examinados por FOSTER & RYAN (1976) foi necessário mais do que um corte simples para manter a integralidade da solução. Os planos de corte utilizado pelos autores são da forma:

$$\sum_{r \in R} a_{C_r} x_r \leq \left\lceil \frac{k}{2} \right\rceil$$

onde C é um conjunto de k pontos de demanda e

$$a_{C_r} = \begin{cases} 1 & \text{se a r-ésima rota contém pelo menos dois pontos de C;} \\ \emptyset & \text{c.c.} \end{cases}$$

Um outro algoritmo também baseado em **(PPV3)** é apresentado por RAO & ZIONTS (1968). Os autores utilizam um esquema de geração de colunas onde as rotas são geradas por meio do algoritmo **"Out-of-Kilter"** de fluxo de custo mínimo. O **PPV** tratado por eles é um problema de percursos de navios em um conjunto de portos.

A determinação da variável do Simplex que entra na base envolve a solução de m (um para cada navio) subproblemas **"Out-of-Kilter"**. Uma rota de custo mínimo para cada navio é encontrada e aquela que se mostrar mais

econômica em relação à solução corrente é utilizada para melhorar a solução básica (ou seja, a coluna que entra na base). A solução básica é considerada ótima se nenhum dos navios tem uma rota que torne a solução corrente mais econômica.

Um algoritmo mais recente que parece obter bons resultados é desenvolvido por DESROSIERS et al. (1984). Os autores resolvem um **m-PCV** com restrições de "Time Window", onde os subproblemas são Problemas de Caminho mais Curto com restrições de "Time Window". Estes subproblemas são eficientemente resolvidos por um algoritmo de Programação Dinâmica desenvolvido por DESROSIERS et al. (1983). Este algoritmo é uma extensão do clássico algoritmo de BELLMAN (1958).

II.5 - Algoritmos Baseados na Formulação de Fluxo de Veículos

Nesta formulação usamos variáveis binárias para indicar quando um veículo viaja entre dois nós (pontos de demanda) na solução ótima. Podemos distinguir duas famílias dentro desta formulação:

(i) a que utiliza três índices;

(ii) e a que utiliza dois índices.

No primeiro caso, os três índices dados a cada variável de fluxo indicam a origem do veículo, seu destino e o próprio veículo. No segundo caso o veículo não é identificado.

II.5.1 - Formulações utilizando três índices

LAPORTE & NOBERT (1985) apresentam uma formulação para o PPV-assimétrico involvendo no máximo m veículos e r depósitos. Esta formulação foi baseado no trabalho de GOLDEN et al. (1977) onde são desenvolvidos algoritmos heurísticoos de solução. Esta parece ser uma formulação clássica, apesar de não ter sido desenvolvido nenhum algoritmo exato baseado nela.

Afim de apresentarmos esta formulação, considere a seguinte notação, além da empregada até aqui:

D_k capacidade do veículo k;

L_k o comprimento máximo permitido para a rota k;

δ_i^k o tempo para o veículo k servir o cliente i;

t_{ij}^k o tempo de viajem entre i e j do veículo k;

c_{ij}^k o custo de se usar o veículo k no arco (i,j);

$$x_{ij}^{k} = \begin{cases} 1 & \text{se o veículo } k \text{ viaja de } i \text{ para } j \\ \emptyset & \text{c.c.} \end{cases}$$

Então, o PPV pode ser formulado como:

$$\textbf{(PPV4)} \quad \text{Min} \sum_{i\in N} \sum_{j\in N} \sum_{k=1,m} c_{ij} x_{ij}^{k} \qquad (II.47)$$

sujeito a:

$$\sum_{i\in N'} \sum_{k=1,m} x_{ij}^{k} = 1 \quad (j= r+1,\ldots,n) \qquad (II.48)$$

$$\sum_{j\in N'} \sum_{k=1,m} x_{ij}^{k} = 1 \quad (i= r+1,\ldots,n) \qquad (II.49)$$

$$\sum_{i\in N'} x_{i\ell}^{k} - \sum_{j\in N'} x_{\ell j}^{k} = \emptyset \quad (k= 1,m;\ \ell= 2,n) \qquad (II.50)$$

$$\sum_{i\in N'} d_i \left(\sum_{j\in N'} x_{ij}^{k} \right) \leq D_k \quad (k= 1,m) \qquad (II.51)$$

$$\sum_{i\in N'} \delta_i^{k} \sum_{j\in N'} x_{ij}^{k} + \sum_{i\in N} \sum_{j\in N} t_{ij} x_{ij}^{k} \leq L \quad (k= 1,m) \qquad (II.52)$$

$$\sum_{i=1,r} \sum_{j=r+1,n} x_{ij}^{k} \leq 1 \quad (k= 1,\ldots,m) \qquad (II.53)$$

$$\sum_{j=1,r} \sum_{i=r+1,n} x_{ij}^{k} \leq 1 \quad (k= 1,\ldots,m) \qquad (II.54)$$

$$\sum_{i\in S} \sum_{j\notin S} x_{ij}^{k} \geq 1 \quad (|S|\geq 1;\ S\subset N;\ k= 1,m) \qquad (II.55)$$

$$x_{ij}^{k} = \{\emptyset, 1\} \quad (i,j\in N';\ k= 1,m) \qquad (II.56)$$

Nesta formulação as restrições (II.48) e (II.49) especificam que cada cliente deve ser servido exatamente uma vez e por um, e somente um, veículo. As restrições (II.50) garantem a conservação do fluxo de veículos em cada nó da rede. (II.51) e (II.52) são as restrições de capacidade e distância, respectivamente. (II.53) e (II.54) garantem que não mais que m veículos partem do depósito e que existe no máximo um veículo por rota. Finalmente a restrição (II.55) elimina as subrotas ilegais, enquanto que (II.56) é a restrição de integralidade das variáveis.

FISHER & JAIKUMAR (1981) desenvolveram dois algoritmos baseados numa formulação diferente de **(PPV4)**. O primeiro garante uma solução exata enquanto que o segundo utiliza uma heurística eficiente para encontrar uma solução aproximada em um tempo computacional reduzido. Nesta seção trataremos do algoritmo exato, deixando o outro para o próximo capítulo. A formulação é a seguinte:

$$\textbf{(PPV5)}\ \textbf{Min} \sum_{i\in N}\sum_{j\in N}\sum_{k=1,m} c_{ij}\, x_{ij}^{k} \qquad \text{(II.57)}$$

sujeito a:

$$\sum_{i\in N'} d_i\, y_{ik} \le D_k \qquad (k= 1,\ldots,m) \qquad \text{(II.58)}$$

$$\sum_{k=1,m} y_{ik} = \begin{cases} m & \text{se } i= 1 \\ 1 & \text{se } i= 2,\ldots,n \end{cases} \qquad \text{(II.59)}$$

$$\sum_{i\in N'} x_{ij}^{k} = y_{jk} \qquad (j=1,\ldots,n;\ k=1,\ldots,m) \qquad \text{(II.60)}$$

$$\sum_{j \in N'} x_{ij}^{k} = y_{ik} \qquad (i=1,\ldots,n;\ k=1,\ldots,m) \qquad (II.61)$$

$$\sum_{i,j \in S} x_{ij}^{k} \leq |S|-1 \qquad (S \subseteq N;\ 2 \leq |S| \leq n-1;\ k=1,\ldots,m) \qquad (II.62)$$

$$y_{ik} \in \{0,1\} \qquad (i=1,\ldots,n;\ k=1,\ldots,m) \qquad (II.63)$$

$$x_{ij}^{k} \in \{0,1\} \qquad (i,j=1,\ldots,n;\ k=1,\ldots,m) \qquad (II.64)$$

onde: $y_{ik} = \begin{cases} 1 & \text{se a cidade } i \text{ é servida pelo veículo } k; \\ 0 & \text{caso contrário.} \end{cases}$

$x_{ij}^{k} = \begin{cases} 1 & \text{se o veículo } k \text{ trafega no arco } i,j; \\ 0 & \text{caso contrário.} \end{cases}$

Dois conhecidos problemas de otimização combinatória estão presentes na formulação acima:

(i) O problema de Designação Generalizado, dado pelas restrições (II.58), (II.59) e (II.63);

(ii) O Problema do Caixeiro Viajante: quando as variáveis y_{ik} são fixadas de maneira a satisfazer as restrições do Problema de Alocação, as restrições (II.60)-(II.62) e (II.64) definem um PCV para um dado veículo k.

Os autores desenvolvem um algoritmo baseado na

decomposição de Benders para resolver o **(PPV5)** dado pela formulação acima. A idéia básica do método pode ser descrita através da seguinte reformulação do **PPV** em termos do Problema de Alocação (não linear):

$$\text{Min} \quad \sum_{k} f(y_k) \tag{II.65}$$

sujeito a: (II.58),(II.59) e (II.63)

onde $f(y_k)$ é o custo de uma rota ótima do PCV dos clientes pertencentes a $N(y_k) = \{i : y_{ik} = 1\}$.

A função $f(y_k)$ pode ser definida matematicamente, para cada k, por:

$$f(y_k) = \text{Min} \sum_{i,j,k} c_{ij} x_{ij}^{k} \tag{II.66}$$

sujeito a: (II.60)-(II.62) e (II.64)

A cada passo o Problema de Alocação é resolvido, uma cota inferior para $f(y_k)$ é calculada a partir das variáveis duais do problema acima e adicionado ao Problema de Alocação. Para obter as variáveis duais, a integralidade em x_{ij}^{k} é relaxada ou é imposta através de cortes padrões de Programação Inteira.

II.5.2 - Formulação utilizando dois índices

Nesta formulação é assumido que todos os veículos da frota possuem as mesmas características.

Nos artigos de LAPORTE & NOBERT (1983) e LAPORTE & DESROCHERS (1984) são considerados os problemas de percursos de veículos apenas com restrição de capacidade e com restrição de distância, respectivamente. Neles os problemas são resolvidos por meio de um procedimento de relaxação de restrições. A integralidade é obtida por B-B nos dois casos e também pelo método de Planos de Corte de Gomory no segundo artigo. Em um artigo posterior, LAPORTE et al. (1985b) desenvolvem um algoritmo de B-B mais eficiente que os acima citados. Este resolve o problema englobando ambas as restrições, ou seja, o PPV-básico tal como definimos na seção I.1.

Na descrição deste algoritmo utilizamos a seguinte notação, além do que já foi definido:

.$S^{+} = S \cup \{1\}$ e $S^{-} = S - \{1\}$; $S \subseteq N$;

.$H_v(S)$ é igual ao valor ótimo da solução do PCV-múltiplo sobre todos os nós de S (não envolvendo os $i \notin S$) com v caixeiros viajantes fixados se $1 \in S$ e $S \subseteq N$; ou igual ao valor ótimo do PCV sobre todos os nós de S se $v = 1$; ou é indefinido se $1 \notin S$ e $v > 1$;

.V(S) é um Limitante Inferior para o No. de veículos necessários para visitar todos os nós de S^+ na solução ótima do PPV em N;

.P(i) é o comprimento do menor caminho entre os nós 1 e i ($i \in N'$);

.x_{ij} tem o valor 1 se um veículo é usado em uma viagem entre i e j ($i,j \in N$); assume o valor 2 se um veículo é usado em uma "viagem de retorno" entre i= 1 e j e assume o valor ∅ se nenhuna das alternativas acima ocorrem;

x_{ij} só é definido se:

a) $i < j$;

b) $d_i + d_j \leq D$;

c) $P(i) + P(j) + c_{ij} \leq L$.

A formulação do problema é então dada por:

$$\textbf{(PPV6) Min} \sum_{i,j \in N} c_{ij} x_{ij} \qquad \text{(II.67)}$$

sujeito a:

$$\sum_{j \in N'} x_{1,j} = 2m \qquad \text{(II.68)}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \quad (k \in N') \qquad \text{(II.69)}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - V(S) \qquad (S \subseteq N',\ |S| \geq 3) \qquad (II.70)$$

$$x_{ij} = \begin{cases} 0,1,2 & \text{se } i=1,\ j \in N',\ c_{1j} \leq 1/2\ L \\ 0,1 & \text{c.c.} \end{cases} \qquad (II.71)$$

As restrições (II.68) e (II.69) especificam o grau de cada nó, (II.71) garante a integralidade das variáveis e (II.70) são as restrições de eliminação de subrotas ilegais.

O Algoritmo

O algoritmo proposto por LAPORTE et al. (1985b) basicamente resolve o problema relaxando as restrições de integralidade e de eliminação de subrotas; a integralidade é obtida pelo branching nas variáveis enquanto que as subrotas ilegais vão sendo eliminadas na medida em que são encontradas. A ordem em que os vários tipos de inviabilidades são investigados parece ser importante no desempenho do algoritmo. O fluxograma do algoritmo é mostrado na figura II.5 abaixo, e por ser um algoritmo B-B padrão apenas os principais passos são numerados e explicados em seguida.

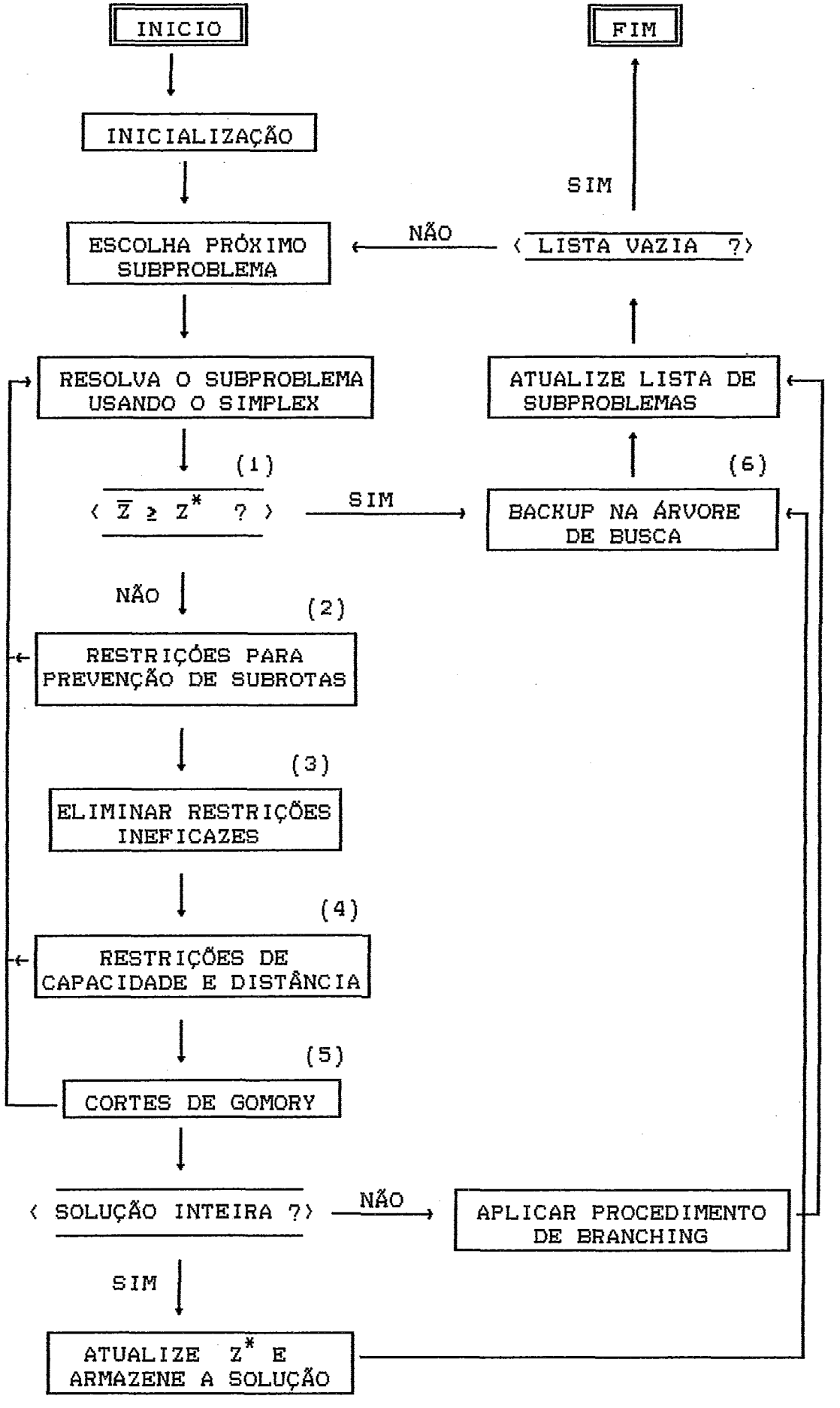

figura II.5 - Fluxograma do Algoritmo de LAPORTE (1985b).

Passo 1.

Z^* é o custo da melhor solução encontrada até o presente momento.

$\overline{Z}$ é o custo do subproblema corrente.

Passo 2. Previnir a ocorrência de subrotas ilegais.

Considere a solução do problema em um dado nó h da árvore de busca, esta solução contém:

(i) conjuntos de nós $\{i_1,\ldots,i_u\}$, $u>1$, correspondendo a cadeias $(i_1,\ldots,i_u)$ tal que $1\notin\{i_2,\ldots,i_{u-1}\}$ se $u>2$ e para o qual todas as variáveis $x_{i_1i_2}$, $x_{i_2i_3}$, $\ldots$, $x_{i_{u-1}i_u}$ estão fixadas no valor 1.

(ii) nós que não pertencem a estas cadeias (é definido para cada um destes nós i um conjunto $\{i\}$).

Nos referimos a estes conjuntos de nós de S_k , que correspondem às cadeias ou aos nós isolados, como "componentes". Cada componente S_k possui um peso definido por $\omega(S_k) = \sum_{i\in S_k} d_i$ e um comprimento dado por $L^h(S_k) = \sum_{t=1,u-1} c_{i_t i_{t+1}}$ no caso da cadeia $(i_1,\ldots,i_u)$ e zero no caso de um nó $\{i\}$.

Considere um componente S_r. Se ele corresponde a uma cadeia, sejam p_r e q_r os nós extremos desta cadeia, se S_r corresponde a um nó i seja $p_r = q_r = i$. No primeiro caso, a variável $x_{p_r q_r}$ pode ser forçada ao valor zero se:

(i) $p_r, q_r \in N'$ ou

(ii) $p_r = 1$ e $L^h(S_r) + c_{1q_r} > L$.

Considere agora dois componentes S_r e S_s e seja $i \in \{p_r, q_r\}$, $\{\bar{i}\} = \{p_r, q_r\} - \{i\}$, $j \in \{p_s, q_s\}$, $\{\bar{j}\} = \{p_s, q_s\} - \{j\}$. Então a variável x_{ij} pode ser forçada ao valor zero se:

(iii) $i, j \in N'$ e $\omega(S_r) + \omega(S_s) > D$, ou

(iv) $i, j \in N'$ e $P(\bar{i}) + P(\bar{j}) + L^h(S_r) + c_{ij} > L$, ou

(v) $i = 1$, $j \in N'$ e $P(\bar{j}) + L^h(S_s) + c_{ij} > L$.

Na prática, **"forçar"** as variáveis ao valor zero é feito pela introdução no subproblema corrente de uma **"restrição de prevenção de subrotas"**, estabelecendo que a soma de todas as variáveis não pode exceder o valor zero.

Os efeitos desta restrição são evitar a ocorrência

de subrotas ilegais ao invés de eliminá-las após terem sido identificadas, e melhorar o valor da F.O. em um estágio mais inicial da árvore de busca.

Passo 3. Eliminar restrições ineficazes.

As restrições geradas em um nível mais alto ou no mesmo nível da árvore de busca e que são atualmente ineficazes são eliminadas do programa. Em algumas raras instâncias estas restrições devem ser geradas novamente, mas em geral esta prática tem se mostrado eficiente computacionalmente. As restrições relativas ao grau dos nós não são removidas, pois elas tendem a se efetivar na solução ótima.

Passo 4. Impor restrições de capacidade e distância.

Em um nó h da árvore de busca, considere o grafo $G'=\{N',A'\}$ onde A' é o conjunto de todos os arcos (i,j) tais que $i,j \in N'$ e $x_{ij} > 0$. N' pode então ser particionado em k componentes conexos S_1, $S_2,\ldots,S_k$. Afim de impor as restrições de capacidade e distância, a seguinte restrição é acrescida:

$$\sum_{i,j \in S_k} x_{ij} \leq |S_k| - V^h(S_k) \qquad \text{(II.72)}$$

onde $V^h(S_k)$ pode ser calculado por :

$$V^h(S_k) = \text{Máx} \{ \lceil \omega(S_k)/D \rceil, \lceil L^h(S_k^+)/L \rceil \} \qquad \text{(II.73)}$$

Como as restrições (II.72) são às vezes impostas quando a solução corrente é não inteira, elas devem ser satisfeitas no momento de sua geração. LAPORTE et al. (1985) desenvolvem um algoritmo heurístico para esta situação.

Passo 5. Cortes de Gomory.

Cortes de Gomory são gerados no primeiro nó da árvore de busca afim de melhorar o lower bound inicial e ajudar a obter soluções inteiras mais rapidamente. Testes feito pelos autores confirmam esta estratégia.

Passo 6. Backtracking.

Quando é realizado um backtracking na árvore de busca do nível τ' para o nível τ ($\tau' > \tau$), é essencial remover todas as restrições de prevenção de subrotas e modificar todas as restrições de eliminação de subrotas cuja validade depende de quais variáveis estão fixadas nos níveis $\tau, \tau+1, \ldots, \tau'$ da árvore de busca.

As restrições de prevenção de subrotas dividem-se em

duas categorias (ver II.72 e II.73):

(i) aquelas que foram geradas com:

$$v^h(S_k) = \lceil \omega(S_k)/D \rceil \text{ e}$$

(ii) aquelas que foram geradas com:

$$v^h(S_k) = \lceil L^h(S_k^+)/L \rceil > \lceil \omega(S_k)/D \rceil .$$

As primeiras não precisam ser eliminadas, já que o valor de $\omega(S_k)$ não depende de h.

No segundo caso, o valor $L^h(S_k^+)$ depende de quais variáveis foram fixadas nos valores 0,1 ou 2 no nó h. Portanto é necessário modificar o lado direito destas restrições para

$$|S_k| - \lceil \omega(S_K)/D) \rceil .$$

Obs.: a formulação e o algoritmo para o problema não-euclidiano também é apresentado em (LAPORTE et al., 1985b].

Em dois outros artigos, LAPORTE et al. (1985c, 1986) desenvolvem algoritmos para o caso Assimétrico. Os autores utilizam a regra, já descrita na seção II.1, que transforma

o **PPV** no PCV para resolver o problema baseando-se numa adaptação do algoritmo de CARPANETO & TOTH (1980). Esta adaptação visa tratar as restrições de capacidade e distância que não estão presentes no PCV de CARPANETO & TOTH (1980). Os algoritmos são procedimentos de B-B onde são relaxadas as restrições de integralidade das variáveis de fluxo e de eliminação de subrotas. O tratamento das restrições de capacidade e distância é realizado na execução do "branching", onde regras especiais de particionamento são desenvolvidas.

II.6 - Conclusão

Neste capítulo vimos os principais algoritmos exatos para solucionar o **PPV**. Devido à complexidade destes problemas estes algoritmos só podem ser aplicados a problemas de porte relativamente pequeno. A dificuldade de implementação também é considerada por muitos profissionais um fator que depõe contra a utilização destes métodos.

Apesar disto, os métodos exatos de solução devem ser considerados devido a alguns fatores. O fator Custo-Benefício deve ser avaliado, em determinadas situações o custo de implementação e utilização de um algoritmo exato é justificado pela economia obtida com a solução ótima. Além disto, novas téonicas de solução em Programação Inteira podem propiciar bons resultados que em

muitos casos viabilizam a sua utilização. Isto vem acontecendo em muitos problemas, notadamente no Problema do Caixeiro Viajante. Para finalizar, podemos dizer que é através dos modelos de Programação Matemática que os problemas são perfeitamente entendidos, mais que isto, em muitos casos os métodos exatos servem para guiar o desenvolvimento de novas heurísticas.

CAPÍTULO III

MÉTODOS APROXIMADOS

III.1 - Introdução

A complexidade dos problemas de Percursos de Veículos, assim como vários outros problemas de otimização combinatória, inspirou os pesquisadores a desenvolverem algoritmos que produzam soluções boas, porém não necessáriamente ótimas, com esforço computacional pequeno. Estes algoritmos, denominados de "aproximados" por produzirem soluções aproximadas, baseiam-se em heurísticas que estão intimamente relacionadas com a estrutura do problema. Neste capítulo veremos os algoritmos pertencentes a esta classe que obtiveram bastante sucesso.

Podemos encontrar várias classificações para os algoritmos aproximados na literatura. Uma bastante completa, sob o ponto de vista do nível de detalhe na caracterização dos algoritmos, é dada por MAGNANTI (1981). Porém, para os propósitos deste trabalho, a seguinte classificação (BODIN et al.,1983) é suficiente.

(i) **Algoritmos Construtivos.** Onde as rotas vão sendo formadas na medida em que o algoritmo é executado. Estes métodos também podem ser classificados de acordo com o critério usado para expandir as rotas e/ou de acordo com o modo em que as rotas são

formadas: sequencialmente ou paralelamente.

(ii) **Algoritmos de Melhoria Iterativa.** Estes algoritmos partem de uma solução inicial viável e a cada iteração tentam melhorá-la através de modificações locais.

(iii) **Algoritmos de Duas Fases.** Nesta classe o problema é resolvido em duas etapas sempre presentes nos problemas de percursos, a fase de formação de grupos de clientes e a fase de construção das rotas

(iv) **Algoritmos de Otimização Incompleta.** Este método consiste em utilizar uma técnica qualquer de enumeração combinada com uma regra heurística para forçar uma parada prematura.

A seguir, vamos descrever algoritmos pertencentes as três primeiras classes, pois os algoritmos de Otimização Incompleta podem ser construídos a partir dos métodos já estudados no capítulo II. Um destes algoritmos está descrito em (CHRISTOFIDES et al., 1979).

III.2 - Algoritmos Construtivos

Dois tipos de algoritmos construtivos têm se destacado na solução de problemas de percursos, os algoritmos de **Economia** e os algoritmos de **Inserção**. Dividimos esta seção em duas partes para estudá-los separadamente.

III.2.1 - Algoritmos de Economia

Em 1964 CLARKE & WRIGHT (1964) introduziram o conceito de "economia" para solucionar problemas de percursos de veículos e desde então vários métodos têm sido desenvolvidos baseados nesta idéia.

Clarke & Wright (1964) **(C-W)** desenvolveram um algoritmo construtivo que parte de uma configuração inicial onde cada cliente é servido por um veículo. A configuração seguinte é atingida através da realização de uma **ligação** entre dois clientes de maneira que um veículo é eliminado e uma economia em termos da distância percorrida é alcançada. Esta economia é medida da seguinte maneira. Suponha que dois clientes i e j são atendidos por dois veículos, o custo, em termos de distância percorrida, deste atendimento é dado por:

$$c = c_{1i} + c_{ij} + c_{j1} \qquad \text{(III.1)}$$

Se os dois clientes forem atendidos por um só veículo economiza-se uma distância de :

$$s_{ij} = c_{ii} + c_{ij} - c_{ij} \qquad \text{(III.2)}$$

É bastante óbvio que quanto maior a economia s_{ij} mais desejável é a ligação entre os clientes i e j. Então C-W propuseram que as economias sejam ordenadas afim de realizar as ligações na ordem decrescente dos valores de s_{ij}.

O algoritmo de C-W pode ser resumido nos seguintes passos:

P1. Calcule as economias s_{ij} para todos os arcos $(i,j) \in A$.

P2. Construa uma lista onde as economias s_{ij} estejam decrescentemente ordenadas;

P3. Começando do topo da lista faça:

(i) Se a economia s_{ij} atual fornece uma "ligação viável" então realize-a. Caso contrário rejeite-a.

(ii) Pegue o próximo da lista e repita o passo P3.(i). Se nenhuma ligação da lista pode ser realizada, PARE: a configuração atual é a solução do problema.

Uma "ligação viável" é uma que fornece uma rota viável de acordo com as restrições do **PPV**, ou seja, onde a capacidade do veículo não é excedida e o comprimento da rota é inferior a **L**.

Apesar do grande sucesso alcançado pelo algoritmo, devido principalmente pela sua simplicidade e facilidade de implementação, existem algumas deficiências que devem ser consideradas:

- *A configuração inicial, conjunto de rotas simples* onde um veículo visita um cliente, é na maioria dos casos inviável, pois geralmente o número de veículos da frota é menor que o número de clientes. Isto significa que em nenhum estágio do algoritmo a viabilidade da solução está garantida. Portanto, em problemas com restrições "apertadas" o método pode falhar na produção de soluções viáveis.

- O esforço computacional necessário pode se tornar excessivo. O problema básico do método consiste em selecionar a melhor ligação. No algoritmo de **C-W** todas as possíveis ligações têm suas economias calculadas **a priori** e ordenadas em um arquivo. Naturalmente muitas destas ligações não serão realizadas e portanto tempo e memória computacionais podem ser economizados.

- A qualidade da solução encontrada também pode ser

afetada pela seguinte característica do método: uma ligação uma vez realizada jamais é removida. A qualidade é afetada porque a medida da conveniência de se efetuar uma ligação é realizada independentemente de outras ligações. Portanto, uma ligação que numa etapa intermediária foi preterida por outra poderia ter fornecido uma economia global melhor no final do procedimento.

Uma primeira modificação do algoritmo que pode ser usada para enfrentar algumas das dificuldades citadas é a versão sequencial do método.

Esta versão considera que as ligações são escolhidas de maneira a formar uma rota de cada vez, ao contrário da versão paralela de C-W onde várias rotas são consideradas em cada iteração. Podemos descrever este algoritmo substituindo o passo P3 do algoritmo de C-W por:

P3. **(Versão Sequencial)** Começando do topo da lista ordenada faça:

(i) Procure a primeira ligação viável da lista que pode ser usada para estender a rota atual através de um de seus dois nós terminais.

(ii) Se a rota atual não pode mais ser aumentada, escolha uma nova ligação viável da lista para

iniciar uma nova rota.

(iii) Repita os passos (i) e (ii) acima até que nenhuma ligação possa mais ser escolhida.

A seguir apresentamos algumas idéias que tentam melhorar as dificuldades citadas anteriormente.

Em um trabalho bastante interessante GASKELL (1967) investiga diferentes medidas de economia e após analizar algumas propriedades sugere duas diferentes medidas que fornecem resultados comparáveis aos obtidos por C-W. Estas medidas são:

$$\lambda_{ij} = s_{ij} \left(\bar{c} + |c_{1i} - c_{1j}| - c_{ij}\right) \qquad \text{(III.3)}$$

$$\pi_{ij} = s_{ij} - c_{ij}$$

onde $\bar{c}$ é a média de todos os c_{1i}.

Estas medidas colocam uma maior ênfase nas distâncias entre dois pontos, ao invés da posição relativa ao depósito. Como consequência obtem-se rotas mais radiais, em contraposição ao método de C-W que produz rotas circunferenciais.

Gaskell introduz estas medidas na versão pararela do

algoritmo de C-W e compara os resultados através do estudo de seis casos. Os resultados obtidos indicam que nenhum dos métodos é genericamente melhor que qualquer outro, e sugere uma generalização das medidas m_{ij} e s_{ij} dado por :

$$m_{ij} = s_{ij} - \alpha\, c_{ij} \qquad \text{(III.4)}$$

onde α é um parâmetro, denominado "parâmetro de formato de rota", definido pelo usuário.

Já vimos que uma das deficiências do algoritmo de C-W é a necessidade de calcular as economias de todas as ligações possíveis a priori, embora sabendo que apenas algumas delas serão realizadas. YELLOW (1970) propõe um método sequencial que não requer o cálculo prévio de todas as economias. O método baseia-se numa técnica de busca geométrica.

Para cada rota que está sendo construída faz-se uma lista apenas das economias relativas aos pontos adjacentes ao nó terminal. O cálculo das economias, no caso de considerarmos distâncias euclidianas, é dado em termos de coordenadas polares por:

$$s_{ij} = r_i + r_j - \alpha \sqrt{[r_i^2 + r_j^2 - 2 r_i r_j \cos(\theta_i - \theta_j)]} \qquad \text{(III.5)}$$

onde (r_i, θ_i) são coordenadas polares do ponto i, e α é o parâmetro de formato de rota.

Pode-se mostrar a partir de (III.5) que a região dos pontos que fornecem um mesmo valor de economia s quando ligados a um ponto i é dado por:

$$(\alpha^2-1)R^2-2R\{r_i[\alpha^2\cos(\theta_i-\Phi)+1]-s\}+r_i^2(\alpha^2-1)-s^2+2sr_i = 0 \qquad (III.6)$$

onde (R,Φ) são as coordenadas polares de tal ponto na região.

YELLOW (1970) define a partir de (III.6) uma **"região de busca"** para cada valor de s como sendo a área dentro da qual uma busca por ligações de melhores economias pode ser realizada. Esta busca se procede da seguinte maneira:

- Inicialmente calcule as economias relativas aos nós adjacentes ao nó terminal i da rota em questão. Ordene em $\bar{s}(k)$ estas economias na ordem crescente de seus desvios angulares a partir do nó terminal i.

- Faça $s = \bar{s}(1)$.

- Os demais pontos da lista ordenada são sucessivamente examinados. Um destes pontos j produzirá uma economia maior que a atual s se:

(i) para $\alpha>1$:

$$\frac{-b - \sqrt{b^2-4ac}}{2a} < r_j < \frac{-b + \sqrt{b^2-4ac}}{2a} \qquad \text{(III.7)}$$

onde $a = \alpha^2-1$;

$$b = -2r_i[\alpha^2\cos(\theta_j-\theta_i) + 1] + 2s;$$

$$c = r_i^2(\alpha^2-1) - s^2 + 2sr_i;$$

(ii) ou para $\alpha = 1$:

$$r_j > \frac{s^2 - 2sr_i}{2s - 2r_i[1 + \cos(\theta_j - \theta_i)]} \qquad \text{(III.8)}$$

Logo um novo s é calculado se (III.7) ou (III.8) é satisfeita e portanto uma nova região de busca é definida. Caso contrário um novo nó da lista $\bar{s}(k)$ é utilizado e o atual abandonado.

Como os pontos são examinados na ordem crescente de seus desvios angulares, a área de busca ao redor de i dimimue a cada novo valor de s encontrado, limitando cada vez mais a busca. Esta termina quando o próximo ponto k da lista a ser examinado situa-se fora da região de busca atual, mais especificamente quando:

$b^2 < 4ac$ para $\alpha > 1$

ou

$r_i[1 + \cos(\theta_i - \theta_k)] < s$ para $\alpha = 1$.

Nos testes realizados por YELLOW (1970) nota-se um grande ganho no tempo de processamento, porém em termos de otimalidade o algoritmo aparentemente não traz vantagens sobre o de Gaskell.

GOLDEN et al. (1977) utilizam técnicas computacionais eficientes para melhorar o algoritmo de **C-W**. Afim de melhorar o algoritmo no que se refere à otimalidade, memória utilizada e tempo de processamento, os autores modificam o algoritmo **C-W** em três aspectos, a saber:

(i) Utilizam o parâmetro de formato de rota α no cáloulo das economias, e a partir de variações no valor de α escolhem aquele que produz a melhor estrutura de rota;

(ii) Consideram apenas as economias entre nós que estão próximos um dos outros;

(iii) O armazenamento das economias s_{ij} é feito em uma estrutura de dados denominada **"heap"**, que permite reduzir o número de operações de comparação e acessar o s_{ij} desejado de maneira eficiente.

Quando o parâmetro α é aumentado a partir do valor zero é dada maior ênfase na distância entre os pontos i e j do que suas posições relativas ao depósito. Isto permite que o usuário varie α de acordo com as caracteristicas

geográficas do problema que está resolvendo, possibilitando uma maior flexibilidade na confecção das rotas.

Ao contrário de **C-W**, os autores não consideram todos os arcos no cálculo das economias mas somente aqueles de maior interesse ou conveniência. A topologia da rede é armazenada da seguinte maneira:

Em geral para cada arco é armazenado sua origem, destino e comprimento. Dadas as coordenadas de cada nó da rede, um "grid" artificial de comprimento h e largura w é colocado sobre a rede. O "grid" é então dividido em D^2 retângulos de tal forma que cada nó fica contido em um retângulo de comprimento h/D e largura w/D . Desta maneira cada nó i possui coordenadas BX(i) e BY(i), que são as coordenadas de cada retângulo (caixa) do "grid".
O conjunto dos arcos que serão explorados fica limitado a todos os arcos entre o depósito e os pontos de demanda e os arcos entre pontos de demanda cuja distância entre eles não é maior que uma "caixa". Ou seja, se

$$|BX(i) - BX(j)| > 1 \quad \text{ou} \quad |BY(i) - BY(j)| > 1$$

o arco (i,j) é ignorado.

Note que o valor de D influe na acuracidade do

algoritmo e pode ser alterado de acordo com o problema. Quanto menor o parâmetro D , para um particular problema, maior é o número de arcos que será considerado.

O problema básico encontrado nos métodos baseados na idéia de economias é a determinação da ligação de maior economia. Para isto é necessário que se realize comparações, estas podem ser realizadas rápida e convenientemente por dados ordenados parcialmente em uma estrutura de dados especial. Nela as economias $s_1,\ldots,s_n$ são arranjadas em uma árvore binária de k níves chamada "heap". A propriedade essencial de um "heap" é que $s_i \geq s_{2i}$ e $s_i \geq s_{2i+1}$.

III.2.2 - Algoritmos de Inserção

A técnica de inserção é bastante intuitiva quando pensamos nos algoritmos construtivos, pois, formar rotas pode ser visto como incluir clientes numa lista que será servida por um veículo. Naturalmente, esta lista deve estar ordenada de alguma maneira lógica. Afim de conseguir isto, os algoritmos de inserção possuem dois critérios básicos, a saber, o critério de **Seleção**, que indica qual é o cliente livre a ser inserido, e o critério de **Inserção** que fornece a posição que o cliente irá ocupar na solução corrente.

MOLE & JAMESON (1976) desenvolveram um algoritmo sequencial de inserção onde os dois critérios definidos

acima generalizam o conceito de economia de CLARKE & WRIGHT (1964). A idéia é calcular a economia alcançada quando um cliente k é incluido numa rota entre os clientes i e j. Este critério de **Inserção** é dado pela seguinte relação:

$$s_k(i,j) = 2c_{ik} - sd_k(i,j) \qquad (III.9)$$

onde $sd_k(i,j)$ pode ser interpretado como o custo associado ao desvio da rota para o cliente k, se o custo for medido em termos de distância será a **"distância extra"** necessária para atender o novo cliente k. Este custo é dado por:

$$sd_k(i,j) = c_{ik} + c_{jk} - c_{ij} \qquad (III.10)$$

Assim, a melhor posição para incluir k na rota é dado por:

$$sd_k(i,j) = \min_{a,b \in \Gamma_k} \{sd_k(a,b)\} \qquad (III.11)$$

onde Γ_k é o conjunto dos nós adjacentes ao nó k.

Além do critério acima precisamos saber qual é o melhor cliente h a ser incluído na rota de maneira a não violar as restrições (capacidade e distância) do problema. Fazemos isto utilizando o seguinte critério de seleção:

$$s_h(l,m) = \text{Máx} \{sd_k(i,j)\} \qquad (III.12)$$

sujeito a:

k possui uma demanda viável;

$sd_k(i,j)$ satisfaz a restrição de distância do problema.

Podemos ainda generalizar a equação (III.10) da mesma maneira que em (GASKELL, 1967):

$$ms_k(i,j) = \lambda c_{1k} - msd_k(i,j) \qquad \text{(III.13)}$$

onde $msd_k(i,j) = c_{ik} + c_{jk} - \mu c_{ij}$.

Assim o critério de inserção anteriormente definido passa a ser:

$$msd_k(i,j) = \underset{a,b \in \Gamma_k}{\text{Min}} \{msd_k(a,b)\} \qquad \text{(III.14)}$$

onde a inserção do cliente k deve manter a viabilidade da rota com relação à distância máxima permitida. O critério de seleção é dado por:

$$ms_k(l,m) = \text{Máx}\ \{ms_k(i,j)\} \qquad \text{(III.15)}$$

Os autores introduzem os critérios acima em um algoritmo sequencial que pode ser resumido nos seguintes passos:

P1. Para cada cliente ainda não atendido, determine a melhor posição de inserção na presente rota através

do critério de Inserção (III.14).

P2. Utilize o critério de Seleção (III.15) para determinar o próximo cliente a ser incluído na rota atual.

P3. Tente melhorar a rota atual reordenando os clientes através de uma heurísica 2-ótima (veja seção III.3).

O passo P3 é outra novidade do algoritmo, pois até então o uso de heurísticas do PCV para obter melhorias só tinha sido feito depois de encontrada uma solução final. Note que se no passo P3 houver um novo sequenciamento dos clientes então o passo P1 deverá ser repetido totalmente.

Um algoritmo de inserção bastante conhecido é o denominado de **"Vizinho Mais Próximo"**. Neste caso os critérios de seleção e inserção se confundem, pois, é escolhido para entrar na rota o cliente mais próximo do último cliente inserido. O primeiro cliente escolhido é aquele que está mais próximo do depósito.

Existem outros critérios de inserção utilizados por vários autores. Muitos destes foram desenvolvidos para realizar agrupamentos ("clusters") de clientes, estratégia que consiste a primeira fase dos algoritmos de Duas Fases do tipo "cluster-first-route-second" que veremos na seção III.4.

III.3 - Métodos de Melhoria Iterativa

Os algoritmos de melhoria iterativa, ao contrário dos algoritmos construtivos, solucionam um problema de percursos de veículos a partir de uma solução viável inicial. A cada iteração a solução viável corrente é modificada localmente de maneira a produzir uma nova solução viável de menor custo.

Este tipo de algoritmo foi proposto por LIN (1965) para resolver o **PCV**. A seguir descrevemos este algoritmo e em seguida mostramos como utilizá-lo para resolver o **PPV**.

III.3.1 - O Algoritmo k-ótimo

O algoritmo **k-ótimo** para solucionar o **PCV** inicia com uma rota de caixeiro viajante viável, ou seja, um **Ciclo Hamiltoniano** **H**. Em cada iteração **k** arcos de **H** são removidos formando-se **k** caminhos desconexos, onde alguns destes caminhos podem ser constituidos de um nó isolado. Em seguida estes **k** caminhos são novamente ligados de maneira a formar uma outra rota viável **H'**, naturalmente, usando arcos diferentes daqueles que foram removidos. A figura III.1 ilustra este procedimento de troca para **k = 2** e **k = 3**. Se o custo de **H'** for menor que o custo de **H**, então a rota **H** é abandonada e o procedimento se repete sobre **H'**; caso contrário, escolha outro conjunto de **k** arcos de **H** para realizar a troca. Esta troca iterativa de

arcos de **H** é realizada até que nenhuma melhoria adicional é conseguida.

(a)

(b)

figura III.1 - Procedimento de troca dos Algoritmos k-ótimos: em (a) k = 2 e em (b) k = 3.

A solução final, que não pode ser melhorada pela troca de quaisquer **k** arcos, é chamada de "**rota k-ótima**". É fácil notar que em geral quanto maior **k** melhor será a solução e maior o esforço computacional. Para valores superiores a quatro o desempenho do algoritmo já começa a ficar comprometido. Alguns estudos (GOLDEN et al., 1980 e LIN & KERNIGHAN, 1973) revelam que a heurística 3-ótima produz resultados bastante superiores aos produzidos pela 2-ótima, porém, quando comparamos a 4-ótima com a 3-ótima o ganho não é suficiente para justificar o aumento do custo computacional exigido. Portanto, as heurísticas 2-ótima e 3-ótima são as mais usadas. Uma boa implementação em Pascal destas duas Heurísticas pode ser encontrada em SYSLO et al. (1983).

III.3.2 - Solucionando o PPV com as Heurísticas k-Ótimas

Podemos usar as heurísticas k-ótimas de duas maneiras para resolver o **PPV**.

A primeira delas consiste em aplicar a heurística k-ótima em cada rota de uma solução viável inicial do **PPV**. A solução inicial pode ser obtida por qualquer um dos métodos construtivos descritos na seção anterior. CHRISTOFIDES et al. (1979) utilizam esta idéia para desenvolver um algoritmo onde em cada iteração a heurística 3-ótima é aplicada sobre a solução da heurística 2-ótima. Esta estratégia mostrou-se bastante eficiente para

problemas de até 120 clientes. As heurísticas k-ótimas também são muito usadas nos algoritmos de Duas Fases (veja seção III.4) para minimizar as rotas obtidas na primeira fase de "cluster".

A outra maneira consiste em transformar o **PPV** em um **PCV**, como já vimos em seções anteriores isto pode ser facilmente conseguido através da adição de depósitos artificiais de maneira a formar uma "rota gigante" única. Podemos utilizar esta estratégia para aplicar as heurísticas k-ótimas sobre esta rota gigante. RUSSEL (1976) utiliza muito bem esta idéia para solucionar um **PPV** com restrições de "schedule" sobre alguns clientes. LAPALME et al. (1986) apresentam um algoritmo para o **m-PCV** que também utiliza rota gigante. Porém, afim de aproveitar a estrutura do problema original, os autores generalizam o procedimento para realizar trocas que não produzem solução viável para o **PCV** mas que são aproveitadas para o **PPV**. Estas trocas são aquelas que produzem subrotas onde pelo menos um depósito artificial está presente.

III.4 - Métodos de Duas Fases

Os métodos de Duas Fases solucionam um **PPV** separando-o em dois subproblemas, a saber, Agrupamento ("cluster") e Otimização de Percursos ("routing"). Se na primeira fase os clientes são agrupados e na segunda os percursos são construídos temos um algoritmo

"cluster-first-route-second", se esta ordem for invertida temos um "route-first-cluster-second". A seguir veremos alguns algoritmos que pertencem à primeira categoria.

III.4.1 - O Algoritmo de Varredura ("Sweep")

Neste método (GILLETT & MILLER, 1974) os clientes usados na formação de cada rota são determinados de acordo com sua posição relativa ao depósito. O algoritmo considera que os clientes estão ordenados lexicograficamente pelas suas coordenadas polares (θ_i, r_i). As rotas são formadas à medida que uma varredura angular é realizada, com o depósito como pivot. Ou seja,os clientes vão sendo incluidos na rota de acordo com a ordem em que aparecem numa lista ordenada pelas coordenadas polares. Logo, na primeira fase é realizado um "cluster" dos pontos de demanda através da heurística "sweep" e em seguida, para cada "cluster" formado, as rotas são construídas mediante uma heurística de Caixeiro Viajante. O algoritmo é mostrado abaixo.

Considere um **PPV** euclidiano onde os clientes (pontos de demanda) são dados pelas suas coordenadas polares (θ_i, r_i), com o depósito situado em $r_1 = 0$ e um cliente arbitrário i_s (semente) em $\theta_{i_s} = 0$. Reordene os clientes de maneira que $\theta_2 \leq \ldots \leq \theta_n$. O algoritmo procede da seguinte maneira:

P1. Escolha um veículo disponível;

P2. Iniciando do cliente i, de menor ângulo θ_i entre os clientes ainda não atendidos, inclua na rota clientes consecutivos i+1,i+2,... até atingir a capacidade máxima do veículo;

P3. Se todos os clientes estão varridos ou se todos os veículos da frota estão ocupados vá para o passo P4, senão retorne ao passo P1;

P4. Resolva um PCV para cada conjunto de clientes alocados aos veículos para formar as rotas ótimas.

Devemos observar que o algoritmo acima inicia a partir de um cliente particular i_s, o cliente semente, e portanto o resultado pode variar de acordo com o cliente semente escolhido. Uma opção seria aplicar o algoritmo para vários clientes sementes e escolher o melhor resultado.

Resultados computacionais revelam que o algoritmo é bastante eficiente quanto a otimalidade e requer um tempo de processamento um pouco maior que o algoritmo de GASKELL (1967). As seguintes características são observadas pelos autores:

(i) O tempo computacional aumenta com o número total de clientes, se o número de clientes por rota permanecer relativamente constante;

(ii) O tempo computacional aumenta quadraticamente com o número médio de clientes por rota, se o número total de clientes permanecer relativamente constante.

Portanto o algoritmo pode ser útil para problemas de grande porte onde se tem em média poucos clientes por rota.

III.4.2 - O Algoritmo de Christofides, Mingozzi e Toth

Na primeira fase deste algoritmo, desenvolvido por CHRISTOFIDES et al. (1979), os clientes são agrupados de acordo com duas heurísticas de inserção. Na primeira delas, passos P1 a P4 do algoritmo descrito em seguida, os clientes são agrupados sequencialmente através de um critério de inserção relativo ao cliente semente, isto é, o cliente escolhido para inicializar a rota. Feito isto, é realizado um procedimento de inserção paralelo, passos P5-P8, onde o número de rotas utilizado e os clientes sementes são aqueles obtidos no procedimento sequencial. Na fase dois as rotas obtidas são melhoradas através da resolução de Problemas do Caixeiro Viajante.

FASE 1.

P1. Faça h= 1;

P2. Escolha um cliente livre i_h para inicializar uma rota R_h. Para todo cliente livre i_l calcule o seguinte custo de inserção relativo ao cliente semente i_h:

$$\delta_l = c_{il} + \lambda c_{li_h} \quad (\lambda \geq 1);$$

P3. Insira em R_h o cliente viável I tal que:

$$\delta_I = \min \{\delta_l\}.$$

Repita P3 até que nenhum cliente possa mais entrar em R_h;

P4. Faça h= h + 1 e retorne ao passo P2 para começar uma nova rota R_h até que todos os clientes estejam agrupados ou nenhum cliente possa mais ser agrupado.

Nos próximos passos, h rotas são consideradas onde a r-ésima rota é $\bar{R}_r = (1, i_r, 1)$ e i_r é o mesmo cliente escolhido no passo P2 para inicializar a rota R_r. Seja S o conjunto de rotas $\bar{R}_r$, r= 1,...,h.

P5. Para cada cliente livre l e para cada rota $\bar{R}_r \in S$ calcule:

$$s_{rl} = c_{il} + \mu\, c_{li_r} - c_{li_r} \qquad (\mu \geq 1),$$

associe l com a rota $\bar{R}_r$ que satisfaz a

$$s_{rl} = \min_{R_r \in S} \{s_{rl}\}.$$

P6. Escolha uma rota $\bar{R}_{\bar{r}} \in S$, faça $S = S - \bar{R}_{\bar{r}}$ e calcule para cada cliente l associado com esta rota o seguinte valor:

$$\delta_l = s_{r'l} - s_{\bar{r}l} \qquad \forall\, l \in \bar{R}_{\bar{r}},$$

onde r' é dado por:

$$s_{r'l} = \min_{R_r \in S} \{s_{rl}\}$$

P7. Insira em $\bar{R}_{\bar{r}}$ o cliente viável I associado com $\bar{R}_{\bar{r}}$ que satisfaz a:

$$\delta_I = \text{Max}\ \{\delta_l\}.$$

Repita o passo P7 até que não mais exista cliente viável associado a $\bar{R}_{\bar{r}}$.

P8. Se $S \neq \phi$ vá para P6. Caso contrário, se todos os clientes estão agrupados **PARE**; ou se ainda existe clientes livres vá para P1.

FASE 2.

P1. Para cada grupo de clientes formado na **FASE I** resolva um Problema do Caixeiro Viajante para obter as rotas finais.

III.4.3 - O Algoritmo de Fisher e Jaikumar

O método de duas fases de FISHER & JAIKUMAR (1981) é fortemente baseado na formulação **(PPV5)** vista no capítulo II.

A primeira fase deste método realiza um agrupamento paralelo resolvendo otimamente um problema de "Alocação Generalizado", e na segunda fase resolvem-se vários **PCV's**.

FASE 1.

P1. Escolha m clientes como "sementes" de grupo e designe um veículo para cada grupo.

P2. Para cada cliente i e para cada grupo k, calcule um custo de inserção c_{ik} relativo à semente do grupo.

P3. Solucione o seguinte problema de alocação generalizado:

$$\text{Min} \sum_{i,k} c_{ik}\, y_{ik} \qquad \text{(III.16)}$$

sujeito a:

$$\sum_{k} y_{ik} = \begin{cases} 1 & \text{se } i=2,\ldots,n \\ m & \text{se } i=1 \end{cases} \qquad \text{(III.17)}$$

$$\sum_{i} d_i y_{ik} \leq Q_k \qquad (k = 1\ldots m) \qquad \text{(III.18)}$$

$$y_{ik} \in \{0,1\} \qquad \text{(III.19)}$$

onde:

$$y_{ik} = \begin{cases} 1 & \text{se o cliente } i \text{ é visitado pelo veículo } k; \\ 0 & \text{c.c.} \end{cases}$$

FASE 2.

P4. Para cada *grupo* k *formado na Fase 1* solucione o PCV para obter a rota de distância mínima.

Os "clientes sementes" podem ser selecionados por uma regra automática ou pelo usuário. Neste último caso a vantagem é evidente: a experiência de um bom planejador de rotas pode indicar um conjunto de sementes que forneça uma

boa solução para o problema. Os autores fornecem uma procedimento para determinar automaticamente as sementes para um problema onde os veículos são homogeneos.

Na implementação do método os autores usam para resolver os PCV's um algoritmo exato similar ao de MILIOTIS (1976). O Problema de Alocação Generalizado é resolvido por um método baseado em Relaxação Lagrangeana onde os multiplicadores são determinados por um método de ajustamento do tipo descrito em (FISHER et al., 1981b).

III.5 - Conclusão

A maioria dos algoritmos aproximados vistos neste capítulo possuem um desempenho computacional suficientemente bom para resolver problemas de porte encontrados na vida real.

FISHER & JAIKUMAR (1981) e CHRISTOFIDES et al. (1979) apresentam estudos comparativos para problemas que possuem até 120 clientes. Nestes artigos são comparados cinco dos algoritmos aqui apresentados, os algoritmos de economias de CLARKE & WRIGTH (1964), de inserção de MOLE & JAMENSON (1976), "sweep" de GILLETT & MILLER (1974), Duas Fases de CHRISTOFIDES et al. (1979) e o de FISHER & JAIKUMAR (1981). Os problemas testes foram classificados em duas classes, na primeira os clientes estão uniformemente distribuidos e na segunda estão distribuidos de forma

primeiro caso Christofides revela que o algoritmo "sweep" obteve melhores soluções seguido dos algoritmos de Duas Fases, Economias e Inserção. Para os problemas onde os clientes formam grupos o algoritmo "sweep" produz os piores resultados, ficando o melhor desempenho para os algoritmos construtivos e Duas Fases. Fisher apresenta resultados onde o seu algoritmo supera os demais. Em termos de tempo de processamento, em média o algoritmo de economias apresenta o melhor desempenho seguido pelos algoritmos de Fisher e Jaikumar, Duas Fases, Inserção e "Sweep".

Quanto à facilidade de implementação os algoritmos construtivos levam uma vantagem sobre os demais, principalmente sobre o algoritmo de Fisher & Jaikumar.

Os algoritmos construtivos também permitem a fácil inclusão de restrições adicionais que não foram aqui consideradas mas que ocorrem com freqüência na prática. No capítulo seguinte apresentamos a implementação de alguns algoritmos construtivos onde uma destas restrições é considerada.

CAPÍTULO IV

O PROBLEMA DE PERCURSOS DE VEÍCULOS COM RESTRIÇÕES NO HORÁRIO DE ATENDIMENTO DOS CLIENTES

IV.1 - Introdução

Nos capítulos anteriores estudamos o Problema de Percursos de Veículos básico, onde apenas algumas restrições são consideradas. Os algoritmos que tratam este problema muitas vezes não são capazes de lidar com todas as restrições encontradas em problemas reais. Neste capítulo mostramos como podemos modificar os **algoritmos aproximados** para um problema mais complexo onde restrições adicionais são impostas. Estas restrições dizem respeito ao horário de atendimento dos clientes da rede e ocorrem com muita freqüência em problemas de distribuição e serviços urbanos.

A seguir definimos e formulamos o problema em questão para na seção IV.3 apresentarmos os algoritmos implementados. Na seção IV.4 apresentamos os resultados computacionais sobre um conjunto de problemas testes e concluímos este capítulo na seção IV.5.

IV.2 - Definição e Formulação

Neste problema de percursos de veículos os clientes devem ser servidos dentro de um intervalo de tempo pré-definido, denominado na literatura de **Janela de Tempo**

("Time Window"). A imposição deste tipo de restrição aumenta consideravelmente a complexidade (computacional) do problema. Agora, além dos aspectos espaciais ou geográficos, também temos que tratar com os aspectos temporais. Isto levou os especialistas a criarem uma nova classe de problemas de percursos de veículos denominada de Problemas de Percursos e "Sequenciamento" de Veículos (**PPSV**), onde o termo Sequenciamento ("Scheduling" em Inglês) inclue o planejamento de cada rota levando-se em consideração os horários envolvidos. Dentro desta classe dois problemas se destacam, o **PPSV** com restrições de precedência, onde existem clientes da rede que devem ser atendidos antes que outros e o **PPSV** com restrições de Janela de Tempo (**PPSV-JT**), que é o que estamos aqui considerando. Eventualmente, as duas restrições mencionadas ocorrem em um mesmo problema, aumentando ainda mais a sua complexidade. Um "survey" bastante recente do **PPSV-JT** é apresentado por DESROCHERS et al. (1988).

Podemos formular o **PPSV-JT** através da Programação Matemática da seguinte maneira. Seja $G = (V,A)$ o grafo que representa a rede onde V é o conjunto de vértices e A o conjunto de arcos. Seja $\{0\}$ o vértice que representa o depósito e $N = \{1,\ldots,n\}$ os clientes. Então, $V = \{0\} \cup N$ e $A = \{(\{0\} \times N) \cup I \cup (N \times \{0\})\}$, onde $I \subset N \times N$ é o conjunto de arcos que ligam os clientes entre si, $\{0\} \times N$ contém os arcos que ligam o depósito aos clientes e $N \times \{0\}$ os que ligam os clientes ao depósito. Para cada cliente i existe uma demanda d_i e

uma Janela de Tempo $[a_i, b_i]$. Para cada arco $(i,j) \in A$ associamos um custo c_{ij} e um tempo de viagem t_{ij}. Finalmente, cada veículo possui uma capacidade máxima D. Se associarmos a cada cliente uma tarefa a ser realizada (por ex., entrega, coleta, etc.), o grafo G definido acima pode ser visto como um grafo de tarefas e a formulação matemática passa a considerar as seguintes variáveis:

- TI_i: Tempo de Início da tarefa do cliente i;
- TF_i: Tempo de Término da tarefa do cliente i;
- y_i: especifica a carga do veículo antes de iniciar a tarefa de i;
- $x_{ij} = \begin{cases} 1 & \text{se o arco } (i,j) \text{ é percorrido por um veículo;} \\ \emptyset & \text{caso contrário.} \end{cases}$

Podemos agora formular o PPSV-JT da seguinte maneira:

$$\textbf{(PPSV-JT)} \qquad \textbf{Min} \sum_{(i,j)\in A} c_{ij}\, x_{ij} \qquad \text{(IV.1)}$$

sujeito a

$$\sum_{j\in N} x_{ij} = 1 \qquad (i \in N) \qquad \text{(IV.2)}$$

$$\sum_{j\in N} x_{ji} = 1 \qquad (i \in N) \qquad \text{(IV.3)}$$

$$x_{ij} = 1 \Rightarrow TF_i + t_{ij} \leq TI_j \quad ((i,j)\in I) \quad (IV.4)$$

$$a_i \leq TI_i \leq b_i \quad (i\in N) \quad (IV.5)$$

$$x_{ij} = 1 \Rightarrow y_i + d_i \leq y_j \quad ((i,j)\in I) \quad (IV.6)$$

$$0 \leq y_i \leq D \quad (i\in N) \quad (IV.7)$$

$$x_{ij} \in \{0,1\} \quad ((i,j)\in A) \quad (IV.8)$$

As restrições (IV.2) e (IV.3) garantem os graus nos vértices da rede, (IV.4) e (IV.5) são as restrições de sequenciamento ("scheduling") enquanto que (IV.6) e (IV.7) mantém a viabilidade de carga. É interessante observar que (IV.4)-(IV.5) ou (IV.6)-(IV.7) também evitam a formação de subrotas e podem ser vistas como uma generalização da clássica restrição de eliminação de subrotas proposta para o PCV por MILLER et al. (1960).

IV.3 - Descrição dos Algoritmos de Solução

Os métodos de solução para o Problema de Percursos de Veículos com Restrição de Janela de Tempo têm sido recentemente estudados por alguns autores. Os métodos exatos foram estudados por DESROSIERS et al. (1984, 1986), SWERSEY & BALLARD (1984) e KOLEN et al. (1987). Os dois primeiros reportam experiências sobre o Problema de Percursos de Ônibus Escolares, que essencialmente é um

m-PCV com restrições de "Time Window", restando o último como o único trabalho de nosso conhecimento até agora publicado que resolve otimamente o **PPSV-JT** tal como definimos anteriormente. Para se ter uma idéia da complexidade do problema, neste artigo os autores apresentam resultados para uma rede de apenas quinze clientes.

Os métodos de solução aproximada normalmente baseiam-se nas heurísticas desenvolvidas para o **PPV-básico**. SOLOMON (1986, 1987) foi um dos primeiros autores a estender estas heurísticas para o caso do **PPSV-JT**. SAVELSBERGH (1985) e VAN LANDEGHEM (1988) também realizaram estudos neste campo.

Neste trabalho realizamos uma implementação para dois algoritmos baseados nos conceitos desenvolvidos por Solomon. Na próxima seção apresentamos os principais conceitos que garantem a viabilidade relativa às restrições de "scheduling" quando usamos os algoritmos do **PPV** para resolver o **PPSV-JT**, para em seguida passarmos a descrever os principais detalhes de cada implementação.

IV.3.1 - Mantendo a Viabilidade nos Horários de Atendimento dos Clientes

No cap. III vimos que entre os algoritmos que alcançaram maior sucesso estão os algoritmos construtivos.

Vamos agora apresentar os conceitos que garantem a viabilidade de "Schedule" quando usamos estes algoritmos para resolver o **PPSV-JT**. Estes conceitos, desenvolvidos por SOLOMON (1987), podem ser aplicado em todos os algoritmos construtivos vistos no capítulo III e também adaptados para os algoritmos k-ótimos (SOLOMON et al., 1988).

Seja $(0,1,2,\ldots,n,n+1)$ uma seqüência que representa uma rota onde a saída do depósito é o nó 0 e a chegada ao depósito é o nó $n+1$. Vamos supor que desejamos inserir um cliente i entre os clientes p e u de uma rota em formação. Naturalmente, a primeira condição que deve ser verificada é se o novo cliente será atendido dentro de sua Janela de Tempo. Isto pode ser feito pela seguinte relação:

$$a_i \leq TI_i = TF_p + t_{pi} \leq b_i \qquad (IV.9)$$

Se esta condição for satisfeita, devemos olhar para os clientes pertencentes à subsequência $(u,u+1,\ldots,n)$, pois, estes clientes podem sofrer uma defasagem nos seus tempos de ínicio de tarefa após a inserção de i que inviabiliza as suas restrições de Janela de Tempo. A figura IV.1 esquematiza esta situação.

Na figura IV.1(a) temos um esquema de uma rota parcial onde os retângulos representam os clientes, com seus comprimentos indicando a Janela de Tempo do cliente.

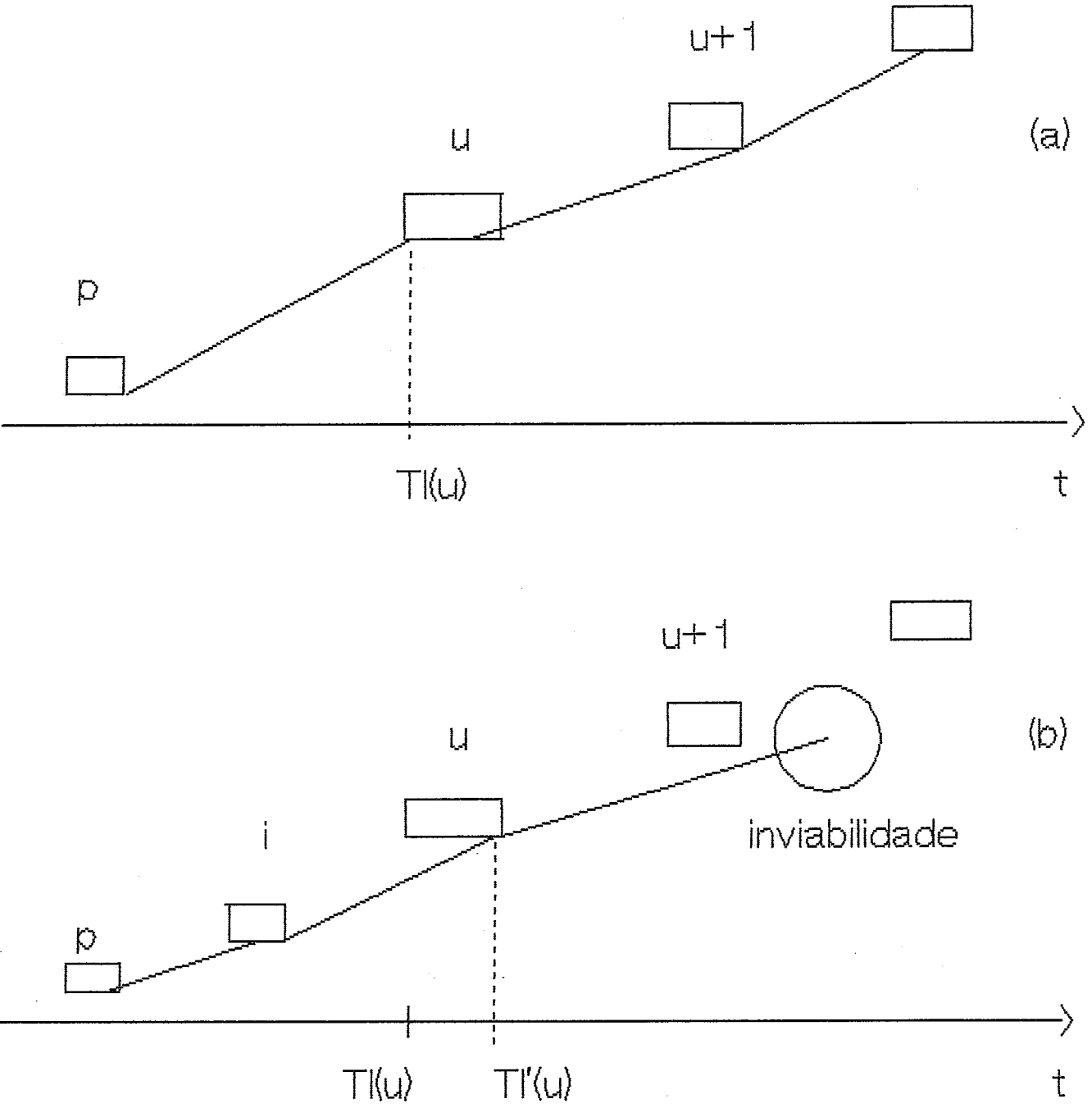

figura IV.1 - O esquema (a) mostra uma rota parcial antes da inserção do oliente i. Em (b), após a inserção de i, ocorre a inviabilidade no horário de atendimento do oliente u+1.

Na figura IV.1(b) o cliente i foi inserido causando uma defasagem no tempo de início de tarefa de **u** igual a $TI'_u - TI_u$, onde TI'_u representa o novo tempo de início da tarefa de **u**. Esta defasem causa a inviabilidade na restrição de "schedule" do cliente **u+1**.

Afim de evitarmos esta situação definimos a grandeza $\varphi(u,n)$ como sendo a maior defasagem (ou incremento) no tempo de início de tarefa do cliente **u** que não causa violação nas Janelas de Tempo ao longo do caminho **(u,u+1,...,n)**. Então é fácil deduzir que:

$$\varphi(u,n) = \underset{u \le j \le n}{\mathbf{Min}} \{b_j - (TF_u + \sum_{u \le i \le n} t_{i\ i+1})\} \qquad (IV.10)$$

Esta grandeza expressa a flexibilidade que temos quando deslocamos no tempo os clientes do caminho (u,u+1,...,n).

Podemos agora especificar a segunda condição de viabilidade para incluir um cliente **i** entre os clientes **p** e **u** de uma rota parcial:

$$\varphi_u \le \varphi(u,n) \qquad (IV.11)$$

onde $\varphi_u = TI'_u - TI_u$ é a defasagem em **u** após a inserção do cliente **i**.

IV.3.2 - O algoritmo de Economia para o PPSV-JT

Quando usamos a heurística de economia de CLARKE e WRIGHT (1964) para resolver o **PPV**-básico a direção do arco que une as duas rotas normalmente não é considerada. Entretanto, no caso do **PPSV-JT** só podemos unir duas rotas quando a primeira termina num horário que permite ao veículo se deslocar até a segunda rota e atender o seu primeiro cliente sem infringir suas restrições de "schedule". Portanto, quando combinamos duas rotas, os tempos relativos aos clientes da primeira rota não mudam enquanto que os clientes da segunda rota poderão sofrer mudanças em seus tempos.

Podemos facilmente adaptar a primeira estratégia descrita na seção anterior para resolver o **PPSV-JT** pelo algoritmo de **C-W**. A condição de viabilidade dada em (IV.9) deve ser verificada no primeiro cliente **p** da segunda rota e (IV.11) deve ser válida para os clientes da segunda rota, portanto, **u** e **n** representam agora o primeiro e o último cliente da segunda rota, respectivamente.

Já vimos no cap.III que a implementação do algoritmo de **C-W** merece uma atenção especial em dois aspectos: na ordenação das economias e no armazenamento das economias. O primeiro aspecto afeta o desempenho do algoritmo relativo ao tempo de execução e o segundo diz respeito à memória computacional requerida pelo algoritmo. Para ordenar eficientemente as economias calculadas utilizamos uma

estrutura de dados "heap" e o algoritmo "Heap Sort" (HOROWITZ & SAHNI, 1978). Para diminuir a memória utilizada usamos a estratégia de calcular somente as economias entre clientes próximos entre si, onde a "proximidade" é medida pelo seguinte critério:

$$dis(i,j) \leq k * \underset{j \in N}{Max} \{ dis(0,j) \} \qquad (IV.12)$$

onde $dis(i,j)$ representa o custo (distância) entre dois clientes, $dis(0,j)$ entre o depósito e o cliente j e o parâmetro $k \in [0,1]$. Este critério reduz consideravelmente a memória utilizada sem afetar a otimalidade da solução.

Outra característica de implementação que merece destaque diz respeito às restrições de Janela de Tempo. Afim de facilitar a resolução dos problemas com restrição no horário de atendimento dos clientes, relaxamos um pouco estas restrições da seguinte maneira, permitimos que o veículo chegue antes da Janela de Tempo do cliente. Na prática isto corresponde a ocorrência de um tempo de espera, ou seja, a situação em que o veículo chega no local especificado antes da hora estipulada pelo cliente, tendo que esperar pela "abertura da Janela de Tempo". No entanto, devemos limitar este tempo de espera para que não ocorram ligações entre clientes próximos geograficamente porém "distantes temporalmente".

A seguir, apresentamos o pseudo-código simplificado do algoritmo.

```
ALGORITMO ECONOMIA-JT;
(1) Leia Dados de Entrada;
(2) Inicialize Variáveis;
(3) Calcule o parâmetro de redução de rede max;
(4) Faça l:= 1;
(5) PARA cada arco  (i,j)  da rede FAÇA
        SE  dis(i,j) < max  ENTÃO
            Calcule a economia
            s[l]:=dis(i,Ø) + dis(j,Ø) - μ*dis(i,j);
            l:= l+1;
        FIM SE;
    FIM PARA;
(6) Ordene  s[l]  através da subrotina Heap;
(7) PARA  i:= 1 até l  FAÇA
        (7.1) Verifique as viabilidades de carga e
              de rede da ligação relativa a  s[i];
              SE  a ligação é viável  ENTÃO
               (7.2) Verifique restrições de
                     "Schedule";
               (7.3) SE estas estão satisfeitas
                     ENTÃO
                         Realize a ligação;
                      FIM SE;
              FIM SE;
        (7.4) Ajuste Heap;
    FIM PARA;
(8) Construir rotas;
(9) Imprimir Solução.
```

Nos passos (3)-(5) as economias são calculadas utilizando-se o critério de redução de rede especificado em (IV.12). A ordenação no passo (6) utiliza a já mencionada estrutura "heap"; esta estrutura de dados garante que em s[1] sempre estará armazenada a ligação de maior economia. Depois de processada esta ligação o "heap" deve ser atualizado, isto é feito no passo (7.4). A viabilidade da ligação é verificada em duas etapas, o passo (7.1) verifica se a rota resultante da ligação atual possui demanda compatível com a capacidade do veículo e se um dos clientes da ligação já foi ligado anteriormente a outra rota, fato que inviabiliza esta ligação. Depois disto, em (7.2) as restrições de "schedule", dadas pelas relações (IV.9)-(IV.11), são avaliadas sobre os clientes da segunda rota tal como descrito na seção IV.3.1. Se a viabilidade de "schedule" é verificada, a ligação é então realizada em (7.3); isto é feito atualizando-se duas listas, uma que indica o predecessor de cada cliente e outra o sucessor. Além disto, os tempos relativos aos clientes da segunda rota devem ser adequadamente atualizados. Ao final do passo (7), o que temos são as rotas implicitamente armazenadas nas listas de sucessor e predecessor. A partir destas listas as rotas são construidas no passo (8) onde as totalizações desejadas também são realizadas.

IV.3.3 - O Algoritmo de Inserção para o PPSV-JT

Em (SOLOMON, 1987) vários critérios de inserção são estudados. Nós utilizamos aquele que generaliza o critério de MOLE & JAMESON (1976), descrito no capítulo III, para o caso do PPSV-JT.

Este algoritmo pertence à classe de algoritmos sequenciais e necessita de um critério para inicializar uma nova rota. Após inicializar a rota corrente, a cada iteração temos que escolher a melhor posição de inserção para cada cliente livre afim de estabelecer o melhor cliente a ser inserido na rota.

O critério de inserção, que determina a melhor posição de inserção no caso do PPSV-JT, pode ser estabelecido da seguinte maneira. Seja ℓ um cliente livre a ser incluido entre os clientes p e u da rota corrente $(\emptyset,1,\ldots,n)$, onde n e $\emptyset$ representam o depósito, então para todo ℓ calcule:

$$ci(p,\ell,u) = \min_{i\in(\emptyset,1,..,n)} \{\alpha_1 c_1(i,\ell,i+1) + \alpha_2 c_2(i,\ell,i+1)\}; \qquad (IV.13)$$

onde

$$c_1(i,\ell,i+1) = c_{i\ell} + c_{\ell i+1} - \mu c_{i\,i+1};$$

$$c_2(i,\ell,i+1) = TI'_{i+1} - TI_{i+1};$$

$$\alpha_1 + \alpha_2 = 1.$$

O critério de seleção, que determina o cliente ℓ^* a ser inserido na rota atual, é dado por:

$$cs(p,\ell^*,u) = \max_{\ell \text{ viável}} \{\lambda\, c_{0\ell} - ci(p,\ell,u)\} \qquad (IV.14)$$

Portanto, neste critério, o melhor local viável de inserção para um cliente livre é aquele que minimiza a combinação ponderada de suas distância e tempo de inserção, i.é., aquele que minimiza uma medida da distância extra e do tempo extra necessário para visitar o cliente ℓ. A seguir, apresentamos o algoritmo.

```
ALGORITMO INSERÇÃO-JT;
(1) Leia Dados de Entrada;
(2) Inicialize Variáveis;
(3) Faça r:= 1;
(4) ENQUANTO  Existir clientes livres  FAÇA
      (4.1) Inicie rota r;
      (4.2) REPETIR
            (4.3) PARA  cada cliente livre  FAÇA
                        Verifique viabilidade de
                        Carga;
                        SE  Carga é viável  ENTÃO
                          Determine  melhor  local
                          viável de inserção (critéro
                          (IV.13));
                        FIM SE
                  FIM PARA
            (4.4) SE existir clientes viáveis para
```

inserir na rota atual r ENTÃO

Determine o melhor cliente utilizando o critério de seleção dado por (IV.14);

Inserir o cliente escolhido;

Deletar o cliente escolhido da lista de clientes livres;

FIM SE;

ATÉ não existir clientes viáveis para incluir na rota r;

Faça r:= r+1;

FIM ENQUANTO;

(5) Construir Rotas;

(6) Imprimir Solução.

No algoritmo sequencial acima o passo (4.1) inicializa uma nova rota r escolhendo um cliente semente. Atualmente, o critério utilizado escolhe o cliente com o mais cedo horário de fechamento de Janela de Tempo, isto é, aquele que possui o menor b_i. No passo (4.2) a rota corrente é formada pela inserção de um novo cliente viável a cada iteração até que não mais exista cliente livre que possa ser incluido na rota. Isto é realizado em duas etapas. No passo (4.3) escolhe-se o melhor local viável de inserção para cada cliente livre de acordo com o critério de inserção definido por (IV.13). As posições viáveis para os clientes que possuem viabilidade de carga são aquelas que satisfazem as restrições de "schedule", que são verificadas através da estratégia definida na secão IV.3.1

pelas relações (IV.9)-(IV.11). No passo seguinte, se existir clientes viáveis para a rota atual, determina-se qual é o melhor cliente segundo o critério dado por (IV.14). Escolhido o melhor cliente viável, ele é inserido na rota atualizando-se todas as variáveis pertinentes e deletado da lista de clientes livres. Terminado o passo (4), tal como no algoritmo Economia-JT, as rotas, com todos os dados de saída desejados, são construídas a partir das listas de predecessor e sucessor.

IV.4 - Resultados Computacionais

Para avaliar o desempenho computacional dos algoritmos apresentados nas seções anteriores nós desenvolvemos um conjunto de problemas testes, pois, na literatura não encontramos dados disponíveis para o **PPSV-JT**. Estes problemas possuem características que tentam cobrir todos os aspectos que podem afetar o comportamento das heurísticas de otimização de percursos e sequenciamento de veículos. Estas características incluem dados geográficos, o número de clientes servidos por um veículo e características das Janelas de Tempo tais como percentagem de clientes com restrição no horário de atendimento, posição e comprimento das Janelas de Tempo.

No primeiro conjunto de problemas teste os clientes estão distribuidos de maneira a formar grupos. Esta característica é bastante encontrada na vida real e pode

ser caracterizada por regiões ou bairros de um centro urbano. Os dados para as coordenadas e demandas dos clientes foram extraídos da literatura (CHRISTOFIDES et al., 1979), este conjunto de problemas testes para o **PPV** se tornou padrão e é utilizado por vários autores. Este problema possui 120 clientes e uma capacidade para os veículos e um horizonte de planejamento que permite, no caso do **PPV**, a formação de rotas com uma média de aproximadamente onze clientes por rota. No segundo conjunto de problemas, que também possue 120 clientes, os dados geográficos dos clientes foram gerados aleatoriamente através de uma distribuição uniforme. Além destes, construímos um terceiro conjunto de problemas que tenta distribuir 100 clientes de maneira a formar um problema intermediário que poderíamos classificar de "semi-agrupado".

Para cada conjunto de dados geográficos criamos inicialmente quatro problemas **PPSV-JT's** gerando Janelas de Tempo para 100%, 75%, 50% e 25% dos clientes. As Janelas de Tempo foram geradas da seguinte maneira: inicialmente foram escolhidos aleatóriamente os clientes com restrição de horário, em seguida gerados aleatoriamente os horários de abertura das Janelas a_i dentro do intervalo de tempo $[t_{0i}, Tmáx - t_{i0}]$, onde Tmáx representa o horizonte de planejamento, em ambos os casos o gerador possui distribuição de probabilidade uniforme. O comprimento das Janelas de Tempo foram fixadas em um valor que corresponde a 15% do horizonte de planejamento e

representa problemas com restrições "apertadas", ou seja, onde as restrições temporais se sobrepõe às espaciais na construção das rotas. Em seguida, para cada um dos problemas já definidos, nós relaxamos o comprimento das Janelas de Tempo atribuindo um valor que corresponde a 30% de Tmáx, representando problemas onde os aspectos geográficos se destacam na construção das rotas e o aspecto temporal é relaxado. Naturalmente, esta característica deve ser combinada com a densidade de clientes que possuem Janelas de Tempo para melhor caracterizar o problema.

Os Algoritmos descritos nas seções anteriores foram implementados em TURBO-PASCAL versão 4.0 e os testes foram realizados em um computador IBM-PC-AT compatível. A qualidade da solução é medida em termos do número de veículos, distância percorrida e tempo total de espera, nesta ordem, ou seja, nós usamos uma ordenação lexicográfica da solução. Portanto, uma solução, por exemplo, com poucos veículos e com uma distância total alta é melhor do que uma solução que utiliza mais veículos porém percorre uma distância total menor. Na tabela IV.1 apresentamos os problemas teste gerados cararacterizados pelo número de clientes (n), a maneira como estão distribuídos (**distribuição**), a densidade de clientes que possuem Janelas de Tempo (**% JT**) e o comprimento das Janelas de Tempo (**Comprimento JT**). Nas tabelas que seguem apresentamos para cada problema as soluções dos métodos implementados.

Por uma consulta às tabelas podemos verificar que o algoritmo Inserção-JT apresenta soluções de melhor qualidade em relação ao algoritmo Economia-JT na grande maioria dos problemas testes. O algoritmo Economia-JT só consegue melhor desempenho em quatro problemas, sendo que em três deles a densidade de clientes com Janela de Tempo é de 25%. Estes resultados foram conseguidos variando-se os parâmetros μ,, α_1 e α_2 do algorito Inserção-JT e os parâmetro μ e k do algoritmo Economia-JT de maneira a produzir o melhor resultado em cada problema.

Quanto ao desempenho dos algoritmos em relação ao tempo de processamento verificamos, como era esperado (ver Anexo), uma melhor performance do algoritmo Economia-JT. No algoritmo Inserção-JT o tempo de processamento aumenta quando relaxamos as restrições, isto se deve ao aumento do número de clientes viáveis que devem ser considerados em cada iteração.

Prob	n	Distribuição	% JT	Largura JT
AA1	120	agrupados	100	15% de Tmax
AA2	120	agrupados	75	15% ''
AA3	120	agrupados	50	15% ''
AA4	120	agrupados	25	15% ''
AB1	120	agrupados	100	30% ''
AB2	120	agrupados	75	30% ''
AB3	120	agrupados	50	30% ''
AB4	120	agrupados	25	30% ''
UA1	120	uniforme	100	15% ''
UA2	120	uniforme	75	15% ''
UA3	120	uniforme	50	15% ''
UA4	120	uniforme	25	15% ''
UB1	120	uniforme	100	30% ''
UB2	120	uniforme	75	30% ''
UB3	120	uniforme	50	30% ''
UB4	120	uniforme	25	30% ''
SA1	100	semi-agrup.	100	15% ''
SA2	100	semi-agrup.	75	15% ''
SA3	100	semi-agrup.	50	15% ''
SA4	100	semi-agrup.	25	15% ''
SB1	100	semi-agrup.	100	30% ''
SB2	100	semi-agrup.	75	30% ''
SB3	100	semi-agrup.	50	30% ''
SB4	100	semi-agrup.	25	30% ''

Tabela IV.1 - Descrição dos problemas testes.

Problema	Economia		Inserção	
AA1	16	2483	15	2306
	231	2:07,04	347	3:24,87
AA2	14	2004	14	2291
	268	2:06,21	106	4:19,72
AA3	13	2062	12	2044
	197	2:06,17	18	5:48,18
AA4	13	1815	12	1844
	86	2:05,73	41	7:29,05
AB1	14	2171	13	2114
	290	2:06,43	222	4:52,75
AB2	13	1784	12	1996
	118	2:05,73	121	6:01,85
AB3	12	1953	11	1843
	50	2:06,39	12	6:56,33
AB4	13	1726	11	1570
	14	2:39,34	85	7:30,61

Tabela IV.2 - Resultados Computacionais: no cantosuperior esquerdo está o número de rotas, no direito a distância total percorrida pela frota, no canto inferior esquerdo está o tempo total de espera e no direito o tempo computacional dado em minutos,segundos e centésimos de segundo.

Problema	Economia		Inserção	
UA1	17	2752	16	2794
	280	2:20,17	269	3:08,56
UA2	16	2593	15	2953
	285	2:18,90	79	4:16,77
UA3	14	2381	13	2620
	163	2:18,96	46	5:52,23
UA4	14	1980	14	2340
	79	2:18,80	6	7:09,74
UB1	15	2387	14	2758
	148	3:31,59	175	4:28,25
UB2	15	2460	13	2606
	308	2:40,60	109	5:06,59
UB3	13	2223	12	2177
	156	2:17,97	99	5:55,36
UB4	13	2097	13	2150
	56	2:18,25	0	7:30,17

Tabela IV.2a - Resultados Computacionais (Continuação).

Problema	Economia		Inserção	
SA1	16	2092	15	2262
	144	1:16,24	82	1:52,26
SA2	15	1819	14	2026
	125	1:15,97	68	2:44,72
SA3	13	1777	12	1830
	61	1:29,53	55	3:11,91
SA4	12	1490	11	1556
	41	1:28,76	19	4:17,93
SB1	15	1812	13	1984
	81	1:42,93	112	3:00,27
SB2	13	1635	12	1917
	183	1:29,25	51	2:59,71
SB3	12	1487	11	1698
	19	1:29,03	24	3:44,75
SB4	11	1432	11	1441
	19	1:29,20	19	4:31,28

Tabela IV.2b - Resultados Computacionais (Continuação).

V.5 - Conclusão

Neste capítulo nós definimos o Problema de Percursos e Sequenciamento de Veículos com Restrições no Horário de Atendimento dos Clientes (**Janelas de Tempo**) e apresentamos a implementação de dois algoritmos aproximados que são extensões dos conhecidos algoritmos construtivos de CLARKE & WRIGHT (1964) e de MOLE & JAMESON (1976).

Os testes computacionais revelam que a heurística de inserção de MOLE & JAMESON (1976) adaptada para o **PPSV-JT** produz melhores resultados que o algoritmo de CLARKE & WRIGTH (1964). Podemos dizer que no **PPSV-JT** o aspecto temporal prevalece sobre o espacial e o algoritmo Inserção-JT consegue explorar eficientemente este fato através de seus oritérios de inserção e seleção.

Com relação ao tempo computacional verificamos que ambos os algoritmos fornecem a solução em tempo hábil. Achamos que o algoritmo Inserção-JT pode sofrer pequenas modificações para melhorar o seu desempenho, como por exemplo, considerar em cada iteração apenas um subconjunto dos clientes livres na escolha do próximo cliente a ser inserido na rota corrente. Com isto torna-se viável uma pós-otimização através de uma heurística k-ótima.

Para finalizar, gostaríamos de salientar que ambos os algoritmos dependem de ajustes em seus parâmetros para produzir resultados "otimizados". No caso do algoritmo Inserção-JT, os parâmetros α_1 e α_2 devem ser devidamente ajustados dependendo se o problema possui restrições de "schedule" (comprimento das Janelas de Tempo e densidade de clientes com JT) fortes ou não.

CAPÍTULO V

CONCLUSÃO

Na primeira parte deste trabalho, capítulos II e III, estudamos os principais métodos de solução para o **PPV**. Os métodos de solução exatos mostraram-se incapazes de resolver problemas de porte elevado. A natureza combinatória desta classe de problemas nos leva a utilizar métodos aproximados de solução. As várias heurísticas desenvolvidas para o **PPV** fornecem soluções satisfatórias, sob o ponto de vista de otimalidade, com um esforço computacional que permite uma implementação em micro-computador.

Na segunda parte, capítulo IV, desenvolvemos a implementação de dois algoritmos aproximados para resolver o **PPSV-JT**. Os testes computacionais revelam que o algoritmo Inserção-JT é superior ao algoritmo Economia-JT, no que diz respeito à qualidade da solução, na maioria dos problemas testados. Sob o ponto de vista da velocidade de processamento o algoritmo Economia-JT se revelou mais rápido. Notamos também que ambos os algoritmos dependem do ajuste de seus parâmetros para produzir soluções de qualidade. Acreditamos que cada particular problema de percursos de veículos merece um estudo preliminar para decidir o método de solução a ser adotado.

Os dois algoritmos implementados são extensões de conhecidos métodos para o PPV. Achamos que ambos podem ser melhorados em suas deficiências. No caso do algoritmo Economia-JT pode-se estabelecer um critério adicional para realizar as ligações que considere os aspectos relativos às restrições de "schedule". Isto pode melhorar o desempenho do algoritmo nos problemas com restrições mais "apertadas". No algoritmo Inserção-JT uma redução no tempo de processamento pode ser alcançada se a busca pelo melhor cliente a ser inserido for realizada sobre um **subconjunto** dos clientes livres. Naturalmente, isto deve ser feito sem comprometer a qualidade da solução.

Além destas melhorias, uma pós-otimização através de uma heurística k-ótima que considere as restrições de "schedule" pode ser realizada.

REFERÊNCIAS

BALINSKI, M. & QUANDT, R.(1964)

"On an Integer Program for a Delivery Problem", Op. Res. 12, pp.300-304.

BELLMAN, R. (1958)

"On A Routing Problem", Quart. Appl. Math. 16, 87-90.

BODIN, L. D. & GOLDEN, B. L. (1981)

"Classification In Vehicle Routing And Scheduling", Networks 11(2), 97-108.

BODIN, L.D., GOLDEN, B.L., ASSAD, A. & BALL,M. (1983)

"Routing and Scheduling of Vehicles and Crews, The State of the Art", Computers and Operations Research 10, pp.69-211.

CARPANETO, G & TOTH, P. (1970)

"Some New Braching and Bounding Criteria for the Asymmetrical Travelling Salesman Problem", Management Science 26, pp.736-743.

CHRISTOFIDES, N. (1976)

"The Vehicle Routing Problem", RAIRO (Recherche Opérationnelle) 10 , pp.55-70.

CHRISTOFIDES, N. & EILON, S. (1969)

"An Algorithm for the Vehicle Dispatching Problem", Op. Res. Quart. 20, pp.309-318.

CHRISTOFIDES, N, MINGOZZI, A. & TOTH, P. (1979)

"The Vehicle Routing Problem", Combinatorial Optimization, Chap. 11, pp.315-338. Wiley, New York.

CHRISTOFIDES, N., MINGOZZI, A. & TOTH, P. (1981)

"Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations", Math. Prog. 20, pp.255-282.

CHRISTOFIDES, N., MINGOZZI, A. & TOTH, P. (1981b)

"State Relaxation Procedures for the Computation of Bounds to Routing Problems", Networks 11, pp.145-164.

CLARKE, G. & WRIGHT, J.W. (1964)

"Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", Op. Res. 12 , pp.568-581.

DANTZIG, G.B. & RAMSER, J (1959)

"The Truck Dispatching Problem", Management Science 6, pp.81-91.

DESROCHERS, M., LENSTRA, J. K., SAVELSBERGH, M. W. P. & SOUMIS, F. (1988)

"Vehicle Routing With Time Window: Optimization and Aproximation", on Vehicle Routing: Methods and

Studies, B. L. Golden and A. A. Assad editors (North-Holland).

DESROSIERS, J., PELLETIER, P. & SOUMIS, F. (1983)
"Plus Court Chemin Avec Contraites D' Horaries", RAIRO Rech. Opér. 17(4), 357-377.

DESROSIERS, J., SOUMIS, F. & DESROCHERS, M (1984)
"Routing With Time-Windows by Column Generation", Networks 14, pp.545-565.

DESROSIERS, J., SOUMIS, F. DESROCHERS, M & SAUVE, M. (1986)
"Methods For Routing With Time Windows", European Journal Of Oper. Res. 23, 236-245.

FISHER, M. & JAIKUMAR, R. (1981)
"A Generalized Assignment heuristic for Vehicle Routing", Networks 11, pp.109-124.

FISHER, M, JAIKUMAR, R. & VAN WASSENHOVE, L. (1981b)
"Multiplier Adjustment Method for the Generalized Assignment Problem", Decision Sciences Working Paper, University of Pennsylvania, March, 1981.

FOSTER, B.A. & RYAN, D.M. (1976)
"An Integer Program Approach to the Vehicle Scheduling Problem", Op. Res. Quart. 27, pp.367-384.

GASKELL, T.J. (1967)

"Bases for Vehicle Fleet Scheduling", Op. Res. Quart. 18, pp.281-295.

GILLETT, B.E. & MILLER, L.R. (1974)

"A Heuristic Algorithm for the Vehicle-Dispatch Problem", Op. Res. 22, pp.341-349.

GOLDEN, B., MAGNANTI, T.L. & NGUYEN, H.Q. (1977)

"Implementing Vehicle Routing Algorithms", Networks 7, pp.113-148.

GOLDEN, B., BODIN, L., DOYLE, T. & STEWART, W. (1980)

"Aproximate Traveling Salesman Algorithms", Oper. Res. 28, 694-711.

HOROWITZ, E. & SAHNI, S. (1978)

"Fundamentas of Computer Algorithms", Computer Science Press, Inc.

KOLEN, A. W. J., RINNOOY KAN, A.H.G. & TRIENEKENS, H. W. (1987)

"Vehicle Routing With Time Window", Oper. Res. 35, 266-273.

LAPALME, G., POTVIN, J., ROUSSEAU, J. (1986)

"A General Exchange Procedure for MTSP", Centre de Recherche sur les transports - Université de Montréal - Publication #489, September 1986.

LAPORTE, G., DESROCHERS, M. & NOBERT, Y (1984)

"Two Exact Algorithm for the Distance-Constrained Vehicle Routong Problem", Networks 14, pp.161-172.

LAPORTE, G., MERCURE, H. & NOBERT, Y. (1986)

"An Exact Algotithm for the Asymmetrical Capacitated Vehicle Routing Problem", Networks 16, 33-46.

LAPORTE, G. & NOBERT, Y. (1983)

"A Branch and Bound Algorithm for the Capacitated Vehicle Routing Problem", Operations Res. Spekt. 5, pp.77-85.

LAPORTE, G. & NOBERT, Y. (1985)

"Exact Algorithm for the Vehicle Routing Problem", Annal of Disorete Mathematics, vol. 31, 1987.

LAPORTE, G., NOBERT, Y. & DESROCHERS, M. (1985b)

"Optimal Routing Under Capacity and Distance Restrictions", Op. Res. 33, pp.1050-1073.

LAPORTE, G., NGUYEN, T. & NOBERT, Y. (1985o)

"A Branch and Bound Algorithm for the Asymmetrical Distance Constrained Vehicle Routing Problems", Cahiers du GERAD, G-85-05, Ecole des Hautes Etudes Commerciales de Montreal, 18 pp., 1985.

LENSTRA, J. K., LAWLER, E. L. & RINNOOY KAN, A. H.G. (1983)
"The traveling Salesman Problem", Wiley, New York.

LENSTRA, J.K. & RINNOOY KAN, A. H. G. (1975)
"Some Simple Application of the Travelling Salesman Problem", Opl. Res. Q. 26, pp.717-734.

LENSTRA, J.K. & RINNOOY KAN, A. H. G. (1981)
"Complexity of Vehicle Routing and Soheduling Problem", Networks 11, pp.221-228.

LIN, S. (1965)
"Computer Solution of The Traveling Salesman Problem", Bell System Tech. J. 44, 2245-2269.

LIN, S. & KERNIGHAN, B. W. (1973)
"An Effective Heuristic Algorithm For The Traveling Salesman Problem", Oper. Res. 21, 498-516.

LITTLE, D.C., MURTY, K.G, SWEENEY, D.W. & KAREL,C. (1963)
"An Algorithm for the Travelling Salesman Problem", Oper. Res. 11, pp.972-989.

MAGNANTI, T.L. (1981)
"Combinatorial Optimization and Vehicle Fleet Planning: Perspectives and Prospects", Networks 11, pp.179-214.

MILIOTS, P. (1976)

"Integer Programming Aproaches to the Traveling Salesman Problem", Math. Program. 10, pp.367-378.

MILLER, C., TUCKER, A. & ZEMLIN, R. (1960)

"Integer Programming Formulation Of Traveling Salesman Problems", J. Ass. Comput. Mach. 7, 326-332.

MOLE, R.H. & JAMESON, S (1976)

"A Sequencial Route-building Algorithm Employing a Generalized Savings Criterion", Op. Res. Quart, 27, pp.503-511.

RAO, M.R. & ZIONTS, S (1968)

"Allocation of Transportation Units to Alternative Trips - A Column Generation Scheme with Out-of-Kilter Subproblems", Oper. Res. 16, pp.52-63.

RUSSEL, R. A. (1976)

"An Effective Heuristic For The M-Tour Traveling Salesman Problem With Some Side Contraints", Oper. Res. 25, 517-524.

SAVELSBERGH, S. (1985)

"Local Search For Routing Problems With Time Window", Ann. Oper. Res. 4, 285-305.

SOLOMON, M. M. (1986)

"On The Worst-Case Performance Of Some Heuristos For

The Vehicle Routing And Scheduling Problem With Time Window Constraints", Network 16, 161-174.

SOLOMON, M. M. (1987)

"Algorithms For The Vehicle Routing And Scheduling Problems With Time Window Contraints", Oper. Res. 35(2).

SOLOMON, M. M., BAKER, E. K. & SCHAFFER, J. R. (1988)

"Vehicle Routing And Scheduling Problems With Time Windows Constraints: Efficient Implementation Of Solution Improvement Procedures", on Vehicle Routing: Methods and Studies, B. L. Golden & A. A. Assad editors (North-Holland).

SYSLO, M. M., DEO, N. & KOWALIK, J. S. (1983)

"Discrete Optimization Algorithms With Pascal Programs", Prentice-Hall, Inc., New Jersey.

SWERSEY, A. J. S. & BALLARD, W. (1984)

"Scheduling School Buses", Management Sci. 30, 844-853.

VAN LANDEGHEN, H. R. G. (1988)

"A Bi-Criteria Heuristic For The Vehicle Routing Problem With Time Window", European Journal Of Oper. Res. 36, 217-226.

YELLOW, P. (1970)

"A Computational Modification to the Savings Method of Vehicle Scheduling", Opl. Res. Q. 21, pp.281-283.

ANEXO

COMPLEXIDADE DOS ALGORITMOS ECONOMIA-JT E INSERÇÃO-JT.

1. Introdução

Podemos definir **algoritmo** como sendo uma descrição passo a passo de como um problema é solucionável. A descrição deve ser finita e os passos devem ser bem definidos, sem ambiguidades e executáveis computacionalmente. Diante da necessidade de resolver um problema frequentemente devemos escolher o algoritmo mais adequado. Para isto devemos estabelecer **critérios de avaliação.**

Quando estamos tratando de problemas afetos à Otimização Combinatória muitas vezes, devido à natureza do problema, torna-se impossível encontrar a solução ótima de uma determinada instância do problema em tempo hábil. A alternativa então utilizada consiste em solucionar o problema de forma aproximada através de um **algoritmo aproximado.** Na avaliação de um algoritmo aproximado dois critérios de eficiência devem ser considerados:

(i) A qualidade da solução fornecida pelo algoritmo;

(ii) A Complexidade Computacional do algoritmo;

A avaliação destes dois critérios pode ser feita de maneira empírica ou analítica. No procedimento empírico um programa implementando o algoritmo é executado em um computador para algumas diferentes instâncias do problema. Para cada execução armazena-se o valor da solução e mede-se o tempo computacional correspondente. Ao final da experiência obtém-se informações sobre o desempenho do algoritmo para aquelas instâncias, porém, podemos apenas conjecturar com base em estatísticas qual será o comportamento do algoritmo para uma classe de problemas. Utilizamos este procedimento no capítulo IV para avaliar o desempenho de dois algoritmos que solucionam o PPSV-JT.

Apesar das informações obtidas através do procedimento empírico acima descrito serem bastante úteis, elas não permitem aferir o comportamento real do algoritmo sobre qualquer instância do problema. Além disso, essas medidas são dependentes de uma implementação particular (portanto, da habilidade do programador), do compilador utilizado, do computador empregado e até das condições locais de processamento no instante da realização das medidas. Esses fatos justificam a necessidade de adoção de algum procedimento analítico para avaliação da eficiência do algoritmo.

Uma avaliação analítica do desempenho destes algoritmos em relação à **qualidade da solução** foi realizada por SOLOMON (1986) que fez uma "**Análise da Performance do**

Pior Caso". Dada uma heurística H e uma instância I_n de tamanho n para o problema P_n, o resultado no pior caso descreve o desvio máximo do valor da solução $(Z_H(I_n))$, dada pela heurística H, do valor ótimo $(Z(I_n))$, através da **Razão de Performance do Pior Caso**, r_n, dada por:

$$r_n(H) = \text{Sup} \{ Z_H(I_n) / Z(i_n) \} \text{ para todo } I_n \in P_n.$$

Solomon concluiu que, $r_n = \Omega(n)$ $(f(n) = \Omega(g(n))$ se $\exists$ c>0 t.q. $f(n) \geq g(n)$ $\forall$ n), em termos da distância total viajada e do número de veículos utilizado. Isto basicamente significa que a Razão de Performance do Pior Caso para os algoritmos considerados não pode ser limitada por uma constante.

Passaremos a descrever um procedimento analítico que nos permite avaliar o desempenho dos algoritmos ECONOMIA-JT e INSERÇÃO-JT relativo ao tempo de processamento.

A tarefa de definir um critério analítico torna-se mais simples se a eficiência a ser avaliada for relativa a alguma máquina específica. Para isto, é conveniente utilizar um modelo matemático de um computador. Utilizaremos um modelo conhecido como RAM (Randon Access Machine - Máquina de Acesso Aleatório). Trata-se de um computador hipotético elementar composto de uma unidade de entrada, unidade de saída, memória e controle/processador. O processador dispõe de instruções que podem ser executadas. A memória armazena os dados e o programa. Cada

instrução i do modelo possui um tempo de instrução t(i). Deste modo, se para a execução de um programa P, para uma certa entrada fixa, são processadas r_1 instruções do tipo i_1, r_2 instruções do tipo i_2,..., r_n instruções do tipo i_n, então o **tempo de execução** T(n) do programa P é dado por:

$$T(N) = \sum_{i=1}^{n} r_i \; t(i).$$

O nosso interesse está em avaliar este somátorio. Para isto, vamos supor que t(i) = 1 para toda intrução i, isto é, que o tempo de execução de cada instrução seja constante e igual a 1. Com isto, o valor do tempo de execução de um programa torna-se igual ao número total de instruções computadas.

Denomina-se **passo de um algoritmo** α à computação de uma instrução do programa P que o implementa. A **Complexidade Local** do algoritmo α é o número total de passos necessários para a computação completa de P, para uma certa entrada E. Desejamos calcular o número total de passos para entradas suficientemente grandes. Para facilitar a avaliação da complexidade de um algoritmo define-se **Complexidade Assintótica** de um algoritmo como sendo um limite superior de sua complexidade local, para uma certa entrada suficientemente grande. A complexidade assintótica deve ser descrita em relação às variáveis que descrevem o tamanho da entrada do algoritmo. Para o caso específico dos algoritmos aproximados que solucionam o

PPSV-JT, o tamanho da entrada pode ser dada pelo número de nós **n** (clientes e depósito) do Grafo.

Para exprimir analiticamente a complexidade assintótica de um algoritmo utiliza-se a notação O(.). Dizemos que T(n) é de ordem O(f(n)) se existe constantes positivas k e $n_{\emptyset}$ tais que $T(n) \leq k.f(n)$, para $n \geq n_{\emptyset}$.

A complexidade assintótica de um algoritmo obviamente não é única, pois a diferentes entradas podem corresponder números de passos diferentes. Como estamos interessados em um procedimento que forneça uma medida de eficiência do algoritmo para qualquer instância do problema, defini-se a **Complexidade de Pior Caso**, ou simplesmente complexidade de um algoritmo, como sendo o valor máximo de todas as suas complexidades assintóticas, para entradas suficientemente grandes. Portanto, a complexidade de pior caso traduz um limite superior do número de passos necessários à computação da entrada mais desfavorável.

Na avaliação de O(.) duas regras se aplicam. A primeira delas é: se determinado problema P se divide em partes independentes, digamos P_1 e P_2 , e sendo $T_1(n)$ e $T_2(n)$ respectivamente de ordem O(f(n)) e O(g(n)), então $T(n) = T_1(n) + T_2(n)$ e P será de ordem O(máx {f(n), G(n)}). A segunda é: Se $T_1(n)$ e $T_2(n)$ são de ordem O(f(n)) e O(g(n)), respectivamente, e

$T(n) = T_1(n).T_2(n)$ então P é de ordem $O(f(n).g(n))$.

2. Complexidade do Algoritmo ECONOMIA-JT

Faremos a análise da complexidade de tempo do algoritmo ECONOMIA-JT a partir da descrição apresentada na seção IV.3.2 detalhando algum passo sempre que for necessário. De acordo com a primeira regra acima descrita para avaliação da complexidade de um algoritmo vamos realizar esta análise verificando a complexidade de cada um dos passos que compõe o algoritmo.

Passo 1:

A leitura dos dados de entrada é dominada pela leitura dos dados relativos a cada cliente e ao depósito, portanto, o tamanho da entrada para este algoritmo pode ser convenientemente aproximado para o número de nós do Grafo G(N,A). Assim, a complexidade deste passo é de ordem O(n). Neste ponto convém observar que utilizamos a distância euclidiana entre dois pontos (clientes) i e j como sendo o custo associado ao arco (i,j). Este custo é calculado a partir das coordenadas dos pontos i e j , sempre que necessário, por uma função **dis(i,j)**. Naturalmente, se os custos são fornecidos a priori a complexidade para ler os dados seria então dominada pela leitura dos custos, ou seja, de ordem $O(I)$, onde $I = |A|$.

Passo 2:

As inicializações feitas neste passo são dominadas pelas inicializações das variáveis relativas aos nós da rede, portanto a complexidade é de ordem O(n).

Passo 3:

O parâmetro de redução de rede é calculado por Máx { dis(∅,i), ∀ i∈N' }, onde ∅ é o nó que representa o depósito, como |N'| = n-1 a complexidade é de ordem O(n).

Passo 5:

Este passo realiza uma instrução de comparação para cada arco (i,j) ∈ A. Portanto, a complexidade é de ordem $O(I)$, onde novamente $I = |A|$. No caso, simétrico, isto é, quando dis(i,j) = dis(j,i) ∀ (i,j) ∈ A, $I = n(n-1)$. Portanto, a complexidade deste passo é de ordem $O(n^2)$.

Passo 6:

As economias calculadas no passo anterior são aqui ordenadas através de um estrutura de dados "heap". O "heap" de m elementos pode ser construído por um procedimente de complexidade de ordem O(m) (HOROWITZ & SAHNI, 1978). Neste caso, m é o número de arcos para os quais as economias foram calculdas. No pior caso $m = I = n(n-1)$, logo a complexidade é de ordem $O(n^2)$.

Passo 7:

No passo 7 do algoritmo para cada arco *I* quatro procedimentos são executados.

O procedimento (7.1) é realizado em tempo constante. Em (7.2) as restrições de "schedule" são verificadas de acordo com o procedimento descrito na seção IV.3.1 pelas relações (IV.10) e (IV.11), onde os clientes do caminho {u,u+1,...,n} são neste caso os clientes pertencentes à segunda rota que será ligada a uma primeira rota pela ligação (p,u). Portanto, a avaliação de (IV.10) é realizada por um procedimento de complexidade de ordem de $O(\bar{n})$, onde $\bar{n}$ é o número de clientes da segunda rota. No pior caso, onde o cliente p estaria sendo ligado a uma segunda rota possuindo todos os demais clientes, $\bar{n} = n-2$, logo a complexidade deste pocedimento é de ordem O(n).

No passo 7.3 o procedimento "Realize a ligação" envolve a atualização de variáveis relativas aos clientes das duas rotas ligadas e a atualização de "arrays" de tamanho n que armazenam as listas de predecessor e sucessor de um nó i. A complexidade da primeira atualização é dominada pela segunda que possui complexidade de ordem O(n).

No passo 7.4 o "heap" é ajustado de forma a excluir a ligação já considerada. Isto pode ser realizado por um algoritmo de complexidade de ordem O(log m), onde no pior caso m = n(n-1) portanto

$O(\log m) = O(\log n)$;

Finalmente, como os passos 7.2 e 7.3 dominam os passos 7.1 e 7.4 e são repetidos $I = n(n-1)$ vezes, temos que a complexidade do passo 7 é de ordem $O(I.n) = O(n^3)$.

Passo 8:

O procedimento "Construir rotas" consiste em formar a matriz Rota[r,i] $r = 1,..,\bar{r}$ e $i = 1,...,nor[r]$, onde nor[r] é o número de clientes da rota r, a partir dos "arrays" pd[i] e sc[i] que fornecem o predecessor e sucessor de cada i, respectivamente. Além disto, todas as totalizações desejadas são realizadas neste passo. Este procedimento pode ser sintetizado pelos seguintes passos relevantes:

```
r:= 1;
PARA  i:= 1  até  n  FAÇA
     SE  pd[i] = ∅  ENTÃO
          Rota[r,1]:= i;
          indice:= 2;
          sucessor:= sc[i];
          ENQUANTO  sucessor ≠ ∅  FAÇA
                    Rota[r,indice]:= sucessor;
                    indice:= indice + 1;
                    sucessor:= sc[sucessor];
          FIM ENQUANTO;
     FIM SE;
FIM PARA;
```

É fácil verificar que este procedimento possui complexidade de ordem $O(n^2)$.

Finalmente, no passo 9 a solução pode ser impressa, por um procedimento análogo ao acima descrito, em $O(n^2)$. Desta forma, baseados na primeira propriedade citada na seção anterior, concluímos que a complexidade deste algoritmo é de ordem $O(n^3)$.

3. Complexidade do Algoritmo INSERÇÃO-JT

O algoritmo INSERÇÃO-JT está descrito na seção IV.3.3. O tamanho da entrada deste algoritmo também pode ser aproximado, para fins analíticos, pelo número de nós n do grafo G(N,A).

Nesta seção vamos analisar a complexidade de tempo apenas do passo 4 do algoritmo INSERÇÃO-JT, pois, os demais passos são idênticos aos correspondentes passos do algoritmo ECONOMIA-JT, já analisados na seção anterior deste anexo.

No "loop" do passo 4 são realizados quatro procedimentos principais. Vamos analizá-los separadamente.

Passo 4.1:

O procedimento "Inicie rota" basicamente consiste em escolher um cliente $i \in S \subseteq N'$ (onde S é o conjunto dos clientes livres, isto é, que não pertencem a nenhuma rota já construída) através de um critério específico. Os critérios normalmente utilizados são escolher o cliente que possui a maior demanda, escolher o cliente que possui o menor prazo de atendimento (menor b_i) ou o cliente mais afastado do depósito. Para qualquer um destes critérios a escolha pode ser realizada por um procedimento com complexidade de ordem de $O(|S|)$, como $S \subseteq N$, temos uma complexidade de pior caso de ordem $O(n)$.

Passo 4.3

Neste passo temos dois procedimentos: "Verifique viabilidade de carga", que é realizado a tempo constante e "Determine melhor local de inserção". É conveniente descrever este último para realizarmos a análise de sua complexidade.

```
Procedimento Determine melhor local de inserção;
FAÇA  i:= índice cliente a ser inserido;
      p:= índice do depósito;
      s:= primeiro cliente da rota atual;
ENQUANTO  s ≠ índice do depósito FAÇA
          Verifique viabilidade de "schedule" para
          inserir  i  entre  p e s;
          SE  viável  ENTÃO
```

```
                Calcule o custo de inserção c(p,i,s);
                SE  c(p,i,s) < menor  ENTÃO
                    menor:= c(p,i,s);
                    p_escolhido:= p;
                    s_escolhido:= s;
                FIM SE;
            FIM SE;
            p:= s;
            s:= sucessor[p];
        FIM ENQUANTO;
```

No procedimento acima o comando de repetição ENQUANTO será repetido $\bar{n}$ vezes, onde $\bar{n}$ é o número de clientes da rota parcialmente construída. A verificação da viabilidade de "schedule" é realizada segundo o procedimento descrito na seção IV.3.1 que possui complexidade de ordem $O(\bar{n})$. Como os demais procedimentos são realizados a tempo constante concluímos que a complexidade deste procedimento é de ordem $O(\bar{n}^2)$. Como no passo 4.3 este procedimento é repetido para cada cliente livre temos que a complexidade deste passo é de ordem $O(n.\bar{n}^2)$. Como $\bar{n}$ é limitado por n, podemos escrever a complexidade deste passo como sendo de ordem $O(n^3)$.

Passo 4.4:

Neste passo deve ser escolhido o melhor cliente

viável para ser inserido na rota atual pelo critério especificado pela relação (IV.14). É fácil verificar que este procedimento pode ser realizado em no máximo n passos. Portanto a complexidade deste passo é de ordem O(n).

Os passos (4.3) e (4.4) estão inseridos dentro do passo (4.2). Como este passo é repetido enquanto existir clientes viáveis, ele não é repetido mais do que n vezes. E como (4.3) domina (4.4) temos que a complexidade de (4.2) é de ordem $O(n^4)$. Finalmente, como este passo esta dentro do passo (4) que é repetido r^* vezes, onde r^* é o número de rotas obtidas na solução, e se assumirmos que $r^* \ll n$ concluímos que a complexidade de (4) ou do algoritmo INSERÇÃO-JT é de ordem $O(n^4)$.