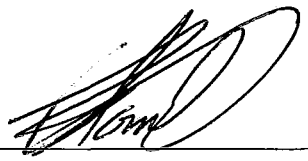
TESTES DE REDUÇÃO E ALGORITMO *BRANCH AND BOUND* PARA O PROBLEMA DE LOCALIZAÇÃO CAPACITADO

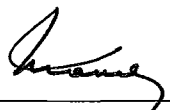
Douglas Valiati

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.

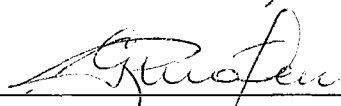
Aprovada por:



Prof. Claudio Thomas Bornstein, Dr. Rer. Nat.



Prof. Nelson Maculan Filho, D.Sc.



Prof. Geraldo Robson Mateus, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
NOVEMBRO DE 1999

VALIATI, DOUGLAS

Testes de Redução e Algoritmo *Branch and Bound* para o Problema de Localização Capacitado [Rio de Janeiro] 1999

XII, 111 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1999)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Problema de Localização Capacitado

2 - Otimização Combinatória

3 - Relaxação Lagrangeana

I. COPPE/UFRJ II. Título (série)

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

TESTES DE REDUÇÃO E ALGORITMO *BRANCH AND BOUND* PARA O PROBLEMA DE LOCALIZAÇÃO CAPACITADO

Douglas Valiati

Novembro/1999

Orientador: Cláudio Thomás Bornstein

Programa: Engenharia de Sistemas e Computação

O problema de localização capacitado com custos lineares de transporte 70 e fixos de instalação é considerado. Primeiro serão examinados testes de redução baseados em técnicas ADD/DROP. Uma função $\Delta(.)$ que permite o balanço do aumento/decréscimo dos custos fixos e variáveis devido à abertura/fechamento de uma facilidade é definida. Existe intervalos para $\Delta(.)$ onde uma decisão ótima de abertura ou fechamento de facilidades deve ser tomada. Na maioria dos casos existe um *gap* em $\Delta(.)$ onde não existe decisão ótima. A dinâmica de abertura/fechamento de facilidades realimenta a redução do gap, fortalecendo o potencial de novas decisões ótimas. Ocasionalmente o gap fecha, levando assim a uma solução completa. Um exemplo para este caso é apresentado. De qualquer modo esta situação é muito especial para problemas grandes. Um grande número de facilidades indefinidas reduz o impacto da abertura/fechamento de uma determinada facilidade enfraquecendo o potencial dos procedimentos de ADD/DROP. Esta é a principal ideia para embutir as técnicas de ADD/DROP em um algoritmo *branch and bound.* A abertura ou fechamento de facilidades obtida pela ramificação na árvore reforça o potencial dos procedimentos de ADD/DROP permitindo que novas decisões possam ser tomadas. De outro lado o uso de ADD/DROP diminui a árvore do *branch and bound.* O limite inferior é obtido utilizando relaxação lagrangeana.

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

REDUCTION TESTS AND BRANCH AND BOUND ALGORITHM FOR THE CAPACITATED PLANT LOCATION PROBLEM

Douglas Valiati

November, 1999

Advisors: Cláudio Thomás Bornstein

Department: Computing and Systems Engineering

The capacitated plant location problem with linear transpotation and fixed location costs is considered. First we examine reduction tests based on ADD/DROP procedures. A function $\Delta(.)$ allowing the balance of the increase/decrease of fixed and variable costs due to the opening/closing of plants is defined. There are intervals for $\Delta(.)$ where an optimal decision of opening or closing facilities can be made. In most cases however there is a gap of $\Delta(.)$ where no optimal decision exist. The dynamics of opening/closing facilities feed back to the gap reducing it, thus strengthening the potencial for further optimal decisions. Occasionallly the gap closes thus leading to a complete solution. An example for such a case is presented. However this situation is very rare specially for large problems. A great number of unsettled facilities reduces the impact of opening/closing a certain facility i weakening the potential of ADD/DROP procedures. This is the main idea for embedding ADD/DROP techniques in a branch and bound algorithm. The opening or closing of facilities obtained by branching the tree reinforces the potential strength of the ADD/DROP procedures allowing further decisions to be made. On the other hand the use of ADD/DROP slims the branch and bound tree. A lower bound based on lagrangean relaxation prunes the branches allowing further slimming of the tree. Computational results are presented.

Agradecimentos

Aos meus pais, Danilo e Helena, pelo incentivo, motivação, carinho, apoio financeiro e tudo de bom que eles me deram durante esta caminhada. A minha irmã (Sandra Mara), parentes e amigos do Paraná que embora não me ajudaram em aspectos técnicos, me ajudaram muito emocionalmente.

Ao meu orientador pela ótima orientação, pela compreensão, paciência, pelos conhecimentos passados, por aturar meus erros de português, por me dar "bronca" quando era necessário, por elogiar meu esforço, etc. Em fim, agradecer pelo excelente papel de orientador e amigo que desempenhou.

Aos professores Nelson Maculan e Adilson Xavier por estarem sempre do meu lado quando precisei de ajuda e conselhos. Devo muito a eles e ao meu orientador o que conquistei neste período. Também quero agradecer por não serem apenas orientadores e professores, mas por serem amigos.

Ao Marcos Castilho, Marcos Sunye e Andre Guedes pelo incentivo a fazer mestrado e ajuda para chegar até a COPPE/Sistemas. Foram estas pessoas que me informaram o que era o mestrado e quais seus benefícios.

Ao Andre Guedes e sua Família pelo apoio e hospitalidade quando cheguei ao Rio de Janeiro. Foram eles que estiveram do meu lado, talvez no período mais difícil, a fase de adaptação ao Rio de Janeiro e mestrado.

Ao meu grande amigo Gabriel Paillard por ter vivido quase tudo o que eu passei bem de perto, compartilhando das horas de felicidade e de tristeza. Este é um amigo que fiz dentro do programa e que está guardado do lado esquerdo do peito.

A Casa dos Guerreiros, Paulo André (Pará), Cesar e Magnos Martinello, que considero minha familia aqui no Rio, por terem me ajudado e me aturado. Quero destacar o Magnos por ter mais tempo de convivência e por ter passado comigo as maiores dificuldades.

Aos professores e amigos que conviveram comigo durante todo esse tempo, em especial, Claudio Prata, Rafael Castro de Andrade, Grasiela Andreola, Vania Dias, Melise Maria Veiga de Paula e Lucyana de Oliveira Cantanhede.

As secretarias Claudia, Solange, Sueli, Dona Gersina, Lurdes e o restante dos funcionários da COPPE/Sistemas.

Índice

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Para muitas organizações, a decisão de instalação de facilidades é de grande importância. A instalação das facilidades, como depósitos ou fábricas, afetam o custo de distribuição de mercadorias aos clientes e podem levar a diferentes custos de transporte e custos de instalação das facilidades. Esses custos variam conforme a localização e o tamanho das facilidades. Várias questões associadas com o problema de localização de facilidades têm surgido na literatura. O leitor pode tomar conhecimento sobre o assundo através dos artigos: Francis, McGinnis e White [17], Aikens [2], Domschke e Drexl [16] e Sridharan [26].

O problema de localização capacitado (PLC) consiste em, dado um conjunto de facilidades I em que cada facilidade tem uma capacidade a_i e um conjunto de clientes J, em que cada um tem necessidade b_j, encontrar um conjunto $P \subseteq I$ que atenda toda as necessidades dos clientes de forma a minimizar o custo total. O custo é composto pelo custo fixo fi, que é o custo de instalação da facilidade i, e o custo variável c_{ij}, que é o custo de transporte da facilidade i ao cliente j.

O problema de localização capacitado pode ser formulado como

$$minimize \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i \tag{1.1}$$

$$sujeito\ a:$$

$$\sum_{j \in J} x_{ij} \leq a_i y_i, \quad \forall i \in I \tag{1.2}$$

$$\sum_{i \in I} x_{ij} = b_j, \quad \forall j \in J \tag{1.3}$$

$$x_{ij} \geq 0, \quad \forall i \in I, \forall j \in J \tag{1.4}$$

$$y_i \in \{0, 1\}, \quad \forall i \in I \tag{1.5}$$

onde x_{ij} é a quantidade enviada de i para j, e y_i representa a instalação ou não da facilidade i, ou seja, se $y_i = 1$ então a facilidade i será instalada, caso contrário $y_i = 0$. A equação 1.2 assegura que nenhum cliente seja atendido por uma facilidade fechada e que o total de demanda atendida pela facilidade não ultrapasse a sua capacidade. A equação 1.3 assegura que a demanda de cada cliente seja satisfeita. A equação 1.4 assegura que as quantidades transportadas não sejam negativas, e a equação 1.5 é a restrição de integralidade. O PLC é um problema combinatório que pertence a classe de problemas NP-Completos.

O PLC tem sido amplamente considerado na literatura. Algoritmos ótimos podem ser encontrados em Bartezzaghi, Colorni and Palermo [7], Baker [6], Beasley [8], Christofides and Beasley [13] e Van Roy [25]; algoritmos heurísticos podem

ser vistos em Bornstein and Azlan [10], Domschke and Drexl [15], Bornstein and Mateus [11], Jacobsen [20] e Beasley [9]. Além destes, outros trabalhos podem ser vistos em Baker [5], Cornuejols, Sridharan and Thizy [14] e Guignard-Spielberg and Kim [18].

No Capítulo 2 apresentaremos dois testes de redução, o O-Teste e o C-Teste, que através de critérios de dominância de custos fixos e variáveis identificam a possibilidade de abertura e fechamento de uma facilidade *a priori*. Os custos fixos incluem os custos de construção das facilidades, por outro lado, os custos variáveis englobam os custos de transporte. A supermodularidade do Problema do Transporte garante que o sucesso de um teste para uma determinada facilidade só tende a ficar mais significativo, informando que uma decisão tomada por um teste é ótima. A seção 2.1 mostrará os testes de redução. Pelo fato do problema de transporte ter custo computacional caro, desenvolvemos aproximações para os testes, as quais serão vistas na seção 2.2. Na seção 2.3 apresentaremos os resultados computacionais das aproximações e comparamos com outras aproximações e com os testes exatos. Apesar dos testes não definirem o *status* de todas as facilidades, na maioria dos casos, quando aplicados podem reduzir as dimensões dos problemas.

No capítulo 3 realizaremos um estudo sobre o fechamento do gap de indefinição, que trata-se de um intervalo deixado em casos onde os testes não coseguem definir o *status* de todas as facilidades. Apresentaremos nas seções 3.1 e 3.2 as dificuldades do fechamento do gap. Na seção 3.3 mostraremos como um teste

de redução (O e C-Teste) influencia o outro na busca do fechamento do gap. Na seção 3.4 apresentaremos dois algoritmos que verificam o fechamento do gap através dos testes de redução, e exemplos onde estes conseguem definir o *status* de todas as facilidades. Finalmente, na seção 3.5 apresentaremos um estudo de classes de problemas bem restritas, para as quais os testes não conseguem garantir o fechamento do gap.

Os testes de redução abrem/fecham facilidades *a priori*, contudo para o bom desempenho, eles necessitam da abertura/fechamento de facilidades. Este possível ciclo vicioso será quebrado utilizando um algoritmo *Branch and Bound* (algoritmo exato).

Embora o PLC pertença a classe dos problemas NP-Completo, desenvolvemos um método exato para resolução do problema. A principal motivação para o desenvolvimento de um método exato é que geralmente os PLC não requerem uma solução imediata (em segundos ou minutos). Como os PLC frequentemente envolvem custos muito altos, um erro de 1% gerado por um método heurístico pode representar um custo relativamente grande.

No capítulo 4 apresentaremos o método *Branch and Bound* desenvolvido, que utiliza, além dos testes de redução, a relaxação lagrangeana para o cálculo do limite inferior, procurando fazer um grande número de podas na árvore de busca, as quais serão vistas na seção 4.2. A busca na árvore é realizada em profundidade, o *backtracking* vai em busca do último *branching* realizado que não foi totalmente explorado e a condição de parada do método é identificada quando ocorrer *Ba-*

cktracking no nó raiz da árvore. A maneira como é realizada a Busca na árvore, o *backtracking* e a condição de parada do método serão vistos na seção 4.3. As regras de *branching* serão apresentadas na seção 4.4. O *branching* ocorre se todos os testes falharem, a escolha da variável a ser ramificada e do primeiro valor a ser fixado (0 ou 1) a ela é realizada através de heurísticas. As heurísticas são facilmente calculadas, pois elas usam os resultados dos testes que falharam, ou seja, resultados já calculados. A utilização de heurísticas fornece uma solução próxima da ótima na primeira busca em profundidade, isto é, fornece um bom limite superior para o problema no início da exploração da árvore. Logo o método além de garantir a otimalidade da solução, fornece geralmente uma solução próxima da ótima logo no início. O método também proporciona uma medida de qualidade da solução. Em um determinado ponto do algoritmo ele pode informar qual é a máxima distância que a solução encontrada está da solução ótima. Este cálculo é realizado com base no limite inferior calculado, que será apresentado na seção 4.5. Isso quer dizer que o usuário pode parar o algoritmo em um determinado momento, mesmo sem saber se a solução obtida é a ótima, mas tendo um bom parâmetro de sua qualidade. Na seção 4.6 apresentaremos o algoritmo do método desenvolvido e na seção 4.7 os resultados computacionais obtidos, comparando-os com os obtidos por outros métodos.

Capítulo 2

Testes de Redução

Neste capítulo apresentaremos testes de redução cujo objetivo é diminuir a dimensão do PLC. Os testes de redução aplicados ao problema determinam a abertura ou o fechamento de uma facilidade *a priori*. Com isso, se reduz a dimensão do problema original. Estes testes de abertura de uma determinada facilidade (O-Teste) e de fechamento (C-Teste), são baseados em critérios que levam em conta tanto o custo fixo como o custo variável do PLC.

Inicialmente apresentaremos os testes de redução exatos na seção 2.1. Para a realização destes testes é necessário a resolução de Problemas de Transporte. Esta resolução em problemas grandes, exige um tempo computacional considerável, deste modo na seção 2.2 desenvolveremos aproximações para os testes. Os testes aproximados permitem maior rapidez do que os testes exatos, contudo são menos poderosos, pois enquadram menos facilidades.

Além dos testes exatos e do desenvolvimento das aproximações, apresentare-

mos neste capítulo, na seção 2.2, um exemplo onde somente com as aproximações consegue-se definir o *status* de todas as facilidades. Na seção 2.3 realizaremos uma comparação, em termos de rapidez e qualidade, com os testes desenvolvidos por Alzán [4] e Mateus [23]. Compara-se também os tempos requeridos pelos testes aproximados com os requeridos pelos testes exatos.

2.1 O-Teste e C-Teste

Seja $w(K) = min\{\sum_{k\in K}\sum_{j\in J} c_{kj}x_{kj} | x \in X(K)\}$ o Problema de Transporte associado ao PLC, onde $K \subseteq I$ é o conjunto de facilidades abertas e $X(K) = \{x \geq 0 | \sum_{j\in J} x_{kj} \leq a_k, \forall k \in K; \sum_{k\in K} x_{kj} = b_j, \forall j \in J\}$, o conjunto de soluções viáveis. É considerado que, se $X(K) = \emptyset$ então $w(K) = \infty$. A função $w(K)$ tem uma importante propriedade, a supermodularidade. Uma função $w(K)$ é supermodular se $w(K) - w(K \cup i) \leq w(K') - w(K' \cup i)$ $\forall K' \subseteq K$ e $\forall i \notin K$. A supermodularidade é uma espécie de convexidade. Mais detalhes sobre o assunto podem ser encontrados em Wolsey [27].

Para $i \notin K$, $\delta_i(K) = w(K) - w(K \cup i) \geq 0$ é o cálculo do aumento/decréscimo do custo no Problema de Transporte se for fechada/aberta a facilidade i. $\Delta_i(K) = f_i - \delta_i(K)$ é o balanço entre o custo fixo e o custo variável com relação à facilidade i.

Sejam K_0 e K_1 respectivamente o conjunto de facilidades fechadas e abertas, isto é, $K_0 = \{i \in I | y_i = 0\}$ e $K_1 = \{i \in I | y_i = 1\}$. Facilidades cujo valor não

esteja definido pertencem a $K_2 = I - K_0 - K_1$. Inicialmente tem-se $K_0 = K_1 = \emptyset$ e $K_2 = I$. Os testes de redução são realizados da seguinte forma:

- O-Teste: Se $\Delta_i(K_1 \cup K_2 - i) \leq 0$ então $y_i = 1$.
- C-Teste: Se $\Delta_i(K_1) \geq 0$ então $y_i = 0$.

Para maiores detalhes sobre os testes veja Bornstein and Azlan [10].

Os testes fornecem critérios para alocar facilidades aos conjuntos K_0 e K_1. A supermodularidade garante que o aumento de K_0 e K_1, isto é, o decréscimo de $K_1 \cup K_2$ não aumentará $\Delta_i(K_1 \cup K_2 - i)$. Da mesma maneira a supermodularidade garante que, o aumento de K_1 não decrescerá o valor de $\Delta_i(K_1)$. Assim, o balanço entre o custo fixo e o variável detectado pelos O e C-Testes nos estágios iniciais do algoritmo tendem a ficar mais significativos, ou seja, mais negativo para o O-Teste e mais positivo para o C-Teste. Este fato indica que se alocada uma facilidade em K_0 ou em K_1 ela permanecerá até o final do algoritmo, garantindo assim a otimalidade.

A figura 2.1 ilustra como os testes trabalham. As setas no topo da figura mostram como $\Delta_i(K_1)$ e $\Delta_i(K_1 \cup K_2 - i)$ movem-se à medida que aumentam o número de iterações de um algoritmo baseados nos O e C-Testes. Uma definição mais precisa deste algoritmo será dada na seção 3.4. Se na aplicação dos testes $K_2 = \{i\}$ então $\Delta_i(K_1) = \Delta_i(K_1 \cup K_2 - i)$, significando que $y_i = 1$ se $\Delta_i(K_1) < 0$ ou $y_i = 0$ se $\Delta_i(K_1) > 0$. Se $\Delta_i(K_1) = 0$ a facilidade i pode ser aberta ou fechada. Neste caso o algoritmo conseguiu gerar uma solução ótima, isto é, o O-Teste e

o C-teste conseguiram fixar o *status* (definir a situação) de todas as facilidades, fazendo $K_2 = \emptyset$.

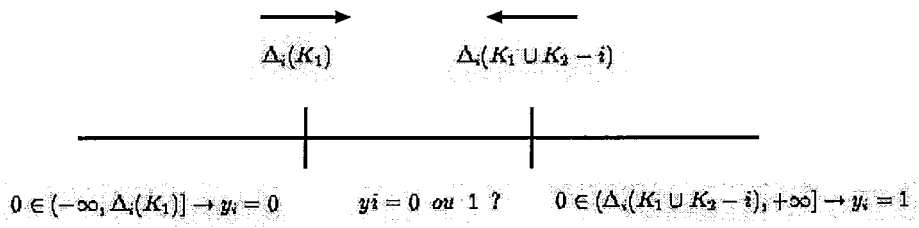


Figura 2.1: Funcionamento dos testes

Os testes alocam facilidades aos conjuntos K_0 e K_1 diminuindo o conjunto K_2. Se na aplicação dos O e C-Testes, todos os elementos de K_2 forem alocados aos conjuntos K_0 e K_1, a solução gerada é uma solução ótima. Contudo na maioria dos problemas estes testes não conseguem fazer $K_2 = \emptyset$. Conforme mostrado na figura 2.1, quando o algoritmo não consegue fazer $K_2 = \emptyset$ fica um intervalo, chamado de gap de indefinição, que representa a região de indecisão, ou seja, representa as facilidades $i \in K_2$ para as quais $\Delta_i(K_1) < 0 < \Delta_i(K_1 \cup K_2 - i)$. No capítulo 3 será discutido o fechamento do gap de indefinição.

É importante observar que para a realização do C-Teste é necessário que K_1 seja viável, ou seja, $X(K_1) \neq \emptyset$. Quando $X(K_1) = \emptyset$ tem-se que $w(k_1) = \infty$, implicando que $\delta_i(K_1)$ não pode ser calculado. Logo, é permitido aplicar o C-Teste em casos onde existem elementos em K_1 tal que $X(K_1) \neq \emptyset$, ou seja, se existir um número de facilidades abertas tal que estas atendam as demandas dos clientes.

2.2 Aproximações para o O-Teste e C-Teste

Para obter-se aproximações referentes aos testes apresentados é necessário encontrar um limite superior para o O-Teste e um limite inferior para o C-teste. Seja $\Delta_i^{ls}(K_1 \cup K_2 - i)$ o limite superior de $\Delta_i(K_1 \cup K_2 - i)$, então, se $\Delta_i^{ls}(K_1 \cup K_2 - i) \leq 0$ tem-se $\Delta_i(K_1 \cup K_2 - i) \leq \Delta_i^{ls}(K_1 \cup K_2 - i) \leq 0$, o que implica em $y_i = 1$. Da mesma forma seja $\Delta_i^{li}(K_1)$ o limite inferior de $\Delta_i(K_1)$, então, se $\Delta_i^{li}(K_1) \geq 0$ tem-se $\Delta_i(K_1) \geq \Delta_i^{li}(K_1) \geq 0$, o que implica em $y_i = 0$.

2.2.1 Aproximações para o O-Teste

Tem-se $\Delta_i(K_1 \cup K_2 - i) = f_i - \delta_i(K_1 \cup K_2 - i)$. Para maior eficiência computacional, uma vez que o cálculo de $\Delta_i(K_1 \cup K_2 - i)$ é computacionalmente caro, calcula-se um limite superior de $\Delta_i(K_1 \cup K_2 - i)$, adotando o limite inferior $\tilde{\delta}_i(K_1 \cup K_2 - i)$. Para isso utiliza-se como hipótese que $w(K_1 \cup K_2 - i)$ seja viável, caso contrário, fica evidente que i deve ser aberto. Tem-se que:

$$\delta_i(K_1 \cup K_2 - i) = w(K_1 \cup K_2 - i) - w(K_1 \cup K_2)$$

Sejam $\hat{x} \in X(K_1 \cup K_2)$ e $x^* \in X(K_1 \cup K_2 - i)$ soluções dos Problemas de Transporte que fornecem respectivamente os valores ótimos para $w(K_1 \cup K_2)$ e $w(K_1 \cup K_2 - i)$. Calcula-se o limite inferior $\tilde{\delta}_i(K_1 \cup K_2 - i)$ tomando uma solução viável $\overline{x} \in X(K_1 \cup K_2)$ ao invés de $\hat{x}$. $\overline{x}$ é obtido a partir de x^* retirando as quantidades atendidas pelas facilidades $g \in (K_1 \cup K_2 - i)$ que podem ser

atendidas por i de modo a fornecer um custo menor. Isto é, $\overline{x}$ é obtido fazendo

$$\overline{x}_{gj} = x^*_{gj} - x^+_{gj} \quad \forall g \in K_1 \cup K_2 - i \;\; \forall j \in J \tag{2.1}$$

onde x^+_{gj} é um valor não negativo que representa uma parcela que será atendida pela facilidade i, logo a quantidade de cada cliente atendida por i é dada por

$$\overline{x}_{ij} = \sum_{g \in K_1 \cup K_2 - i} x^+_{gj} \quad \forall j \in J \tag{2.2}$$

onde $\overline{x}_{ij}$ obedece à seguinte condição:

$$\sum_{j \in J} \overline{x}_{ij} \leq a_i \tag{2.3}$$

A condição 2.3 garante o atendimento da restrição de oferta de i, enquanto que 2.1 garante o atendimento das restrições de oferta das demais facilidades $g \in K_1 \cup K_2 - i$, pois:

$$a_g \geq \sum_{j \in J} x^*_{gj} \geq \sum_{j \in J} x^*_{gj} - \sum_{j \in J} x^+_{gj} = \sum_{j \in J} \overline{x}_{gj}$$

As restrições de demanda também são atendidas por $\overline{x}$, pois:

$$\sum_{g \in K_1 \cup K_2} \overline{x}_{gj} = \sum_{g \in K_1 \cup K_2 - i} \overline{x}_{gj} + \overline{x}_{ij} \quad \forall j \in J$$

Usando as expressões 2.1 e 2.2 tem-se:

$$\sum_{g \in K_1 \cup K_2} \overline{x}_{gj} = \sum_{g \in K_1 \cup K_2 - i} x^*_{gj} - \sum_{g \in K_1 \cup K_2 - i} x^+_{gj} + \sum_{g \in K_1 \cup K_2 - i} x^+_{gj} \quad \forall j \in J$$

Simplificando a expressão obtem-se:

$$\sum_{g \in K_1 \cup K_2} \overline{x}_{gj} = \sum_{g \in K_1 \cup K_2 - i} x^*_{gj} = b_j \quad \forall j \in J$$

Isso demonstra que $\overline{x}$ atende às restrições de demanda. Pode-se agora calcular uma expressão para $\tilde{\delta_i}$:

$\delta_i(K_1 \cup K_2 - i) \geq \sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{gj} x^*_{gj} - \sum_{g \in K1 \cup K2} \sum_{j \in J} c_{gj} \overline{x}_{gj} =$
$\sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{gj} x^*_{gj} - \sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{gj} \overline{x}_{gj} - \sum_{j \in J} c_{ij} \overline{x}_{ij}$

Utilizando a expressão 2.1 tem-se:

$\delta_i(K_1 \cup K_2 - i) \geq \sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{gj} x^*_{gj} - (\sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{gj} x^*_{gj} -$
$\sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{gj} x^+_{gj}) - \sum_{j \in J} c_{ij} \overline{x}_{ij}$

Simplificando a expressão obtem-se:

$$\delta_i(K_1 \cup K_2 - i) \geq \sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{gj} x^+_{gj} - \sum_{j \in J} c_{ij} \overline{x}_{ij}$$

Seja $c_{min_j} = min_{g \in K_1 \cup K_2 - i} c_{gj} \quad \forall j \in J$. Então pode-se escrever:

$\delta_i(K_1 \cup K_2 - i) \geq \sum_{g \in K1 \cup K2-i} \sum_{j \in J} c_{min_j} x^+_{gj} - \sum_{j \in J} c_{ij} \overline{x}_{ij} = \sum_{j \in J} \sum_{g \in K1 \cup K2-i}$
$c_{min_j} x^+_{gj} - \sum_{j \in J} c_{ij} \overline{x}_{ij} = \sum_{j \in J} c_{min_j} \sum_{g \in K1 \cup K2-i} x^+_{gj} - \sum_{j \in J} c_{ij} \overline{x}_{ij}$

Utilizando a expressão 2.2 tem-se:

$$\delta_i(K_1 \cup K_2 - i) \geq \sum_{j \in J} (c_{min_j} - c_{ij}) \overline{x}_{ij}$$

Seja $\overline{x}_i$ um conjunto de valores $\overline{x}_{ij}$ $\forall j \in J$. Como $\overline{x}_{ij}$ deve satisfazer a uma série de condições, define-se o conjunto $\overline{X}_i = \{\overline{x}_i | \sum_{j \in J} \overline{x}_{ij} \leq a_i,\ \overline{x}_{ij} \leq b_j\ , \forall j \in J,\ \overline{x}_{ij} \geq 0\ , \forall j \in J\}$.

Como o interesse é encontrar o maior limite inferior de $\delta_i(K_1 \cup K_2 - i)$ que favorece o menor limite superior de $\Delta_i(K_1 \cup K_2 - i)$, tem-se:

$$\delta_i(K_1 \cup K_2 - i) \geq \tilde{\delta}_i(K_1 \cup K_2 - i) = \max_{\overline{x}_i \in \overline{X}_i} \sum_{j \in J} (c_{min_j} - c_{ij}) \overline{x}_{ij}$$

Tal problema equivale a resolver um problema da mochila. Para resolver o problema basta ordenar crescentemente os coeficientes $(c_{min_j} - c_{ij}) > 0$ de modo que:

$$(c_{min_{j_1}} - c_{ij_1}) \geq (c_{min_{j_2}} - c_{ij_2}) \geq ... \geq (c_{min_{j_n}} - c_{ij_n})$$

Tem-se

$$\delta_i(K_1 \cup K_2 - i) \geq \tilde{\delta}_i(K_1 \cup K_2 - i) = \sum_{k=1...n} (c_{min_{j_k}} - c_{ij_k}) \tilde{x}_{ij_k}$$

onde,

$$\tilde{x}_{ij_1} = min(a_i, b_{j_1})$$

$$\tilde{x}_{ij_2} = min(a_i - \tilde{x}_{ij_1}, b_{j_2})$$

$$\tilde{x}_{ij_n} = min(a_i - \tilde{x}_{ij_1} - \tilde{x}_{ij_2} - ... - \tilde{x}_{ij_{n-1}}, b_{j_n})$$

Não é necessário determinar todos $\tilde{x}_{ij_k}$, basta determinar até o primeiro $\tilde{x}_{ij_k} = 0$, os demais terão valor zero.

Como trata-se do cálculo de um limite inferior pode-se cosiderar os p primeiros coeficientes para obter o valor de $\tilde{\delta}_i(K_1 \cup K_2 - i)$ $(1 \leq p \leq n)$.

Exemplo 2.1 *Aplicação do* $\tilde{\delta}_i(K_1 \cup K_2 - i)$.

A seguir apresentaremos um exemplo que permite aplicar $\tilde{\delta}_i(K_1 \cup K_2 - i)$ para reduzir a dimensão do problema. Os dados relativos aos custos variáveis, custos fixos, capacidades e demandas são fornecidos na tabela 2.1, onde F_i $i = 1, ..., 4$, representa as facilidades, C_j $j = 1, ..., 6$, os clientes e f_i $i = 1, ..., 4$, os custos fixos de instalação das facilidades. O custo de transporte dado na tabela é unitário, representando o custo de transporte de uma unidade da facilidade i para o cliente j. A demanda de cada cliente é representada por b_j $j = 1, ..., 6$, e a capacidade de cada facilidade, por a_i $i = 1, ..., 4$. A tabela 2.2 mostra a realização da aproximação para O-Teste $\forall i \in K_2$.

	C_1	C_2	C_3	C_4	C_5	C_6	f_i	a_i
F_1	20	20	20	10	10	20	250	30
F_2	25	25	30	25	25	30	200	30
F_3	30	30	30	25	25	25	160	30
F_4	7	7	30	30	30	30	250	30
b_j	10	10	10	10	10	10		

Tabela 2.1: Dados do problema (F_1, F_4 - ótimo)

A aplicação da aproximação tem sucesso para F_1 e F_4, fornecendo $K_0 = \emptyset$, $K_1 = \{F_1, F_4\}$ e $K_2 = \{F_2, F_3\}$. •

F_i	c_{min_j}	δ_i	$\tilde{\delta}_i$	Δ_i^{ls}	**Resultado**
F_1	7,7,30,25,25,25	400	400	-150	sucesso
F_2	7,7,20,10,10,20	0	0	200	falha
F_3	7,7,20,10,10,20	50	0	160	falha
F_4	20,20,20,10,10,20	360	260	-10	sucesso

Tabela 2.2: Aplicação da aproximação para O-Teste

2.2.2 Aproximação para o C-Teste

Tem-se $\Delta_i(K_1) = f_i - \delta_i(K_1)$. Para maior eficiência computacional, uma vez que o cálculo de $\Delta_i(K_1)$ é computacionalmente caro, calcula-se um limite inferior de $\Delta_i(K_1)$, adotando o limite superior $\overline{\delta_i}(K_1)$. Tem-se que:

$$\delta_i(K_1) = w(K_1) - w(K_1 \cup i)$$

Sejam $\hat{x} \in X(K_1)$ e $x^* \in X(K_1 \cup i)$ soluções dos Problemas de Transporte que fornecem respectivamente os valores ótimos para $w(K_1)$ e $w(K_1 \cup i)$. Calcula-se o limite superior $\overline{\delta_i}(K_1)$ tomando uma solução viável $\overline{x} \in X(K_1)$ ao invés de $\hat{x}$. $\overline{x}$ é obtido a partir de x^* realocando as quantidades x^*_{ij} fornecidas pela facilidade i, de modo que estas sejam atendidas pelas demais facilidades $g \in K_1$. Isto é, $\overline{x}$ é obtido fazendo

$$\overline{x}_{gj} = x^*_{gj} + x^+_{gj} \quad \forall g \in K_1 \ \ \forall j \in J$$

onde x^+_{gj} obedece às seguintes condições:

$$x^+_{gj} \geq 0 \quad \forall g \in K_1 \ \ \forall j \in J \tag{2.4}$$

$$\sum_{j \in J} x^+_{gj} \leq a_g - \sum_{j \in J} x^*_{gj} \quad \forall g \in K_1 \tag{2.5}$$

$$\sum_{g \in K_1} x_{gj}^{+} = x_{ij}^{*} \quad \forall j \in J \tag{2.6}$$

Como x^* satisfaz as restrições de demanda, $\overline{x}$ também o faz, isto é:

$$\sum_{g \in K_1} \overline{x}_{gj} = \sum_{g \in K_1} x_{gj}^{*} + \sum_{g \in K_1} x_{gj}^{+} = \sum_{g \in K_1 \cup i} x_{gj}^{*} = b_j \quad \forall j \in J$$

Além disso 2.5 garante que as restrições de oferta são atendidas, pois:

$$\sum_{j \in J} \overline{x}_{gj} = \sum_{j \in J} x_{gj}^{*} + \sum_{j \in J} x_{gj}^{+} \leq \sum_{j \in J} x_{gj}^{*} + a_g - \sum_{j \in J} x_{gj}^{*} = a_g \quad \forall g \in K_1$$

Finalmente, para demonstrar a viabilidade de $\overline{x}$, basta mostrar que existem valores de x^+ que satisfazem a 2.4, 2.5 e 2.6. Examinando 2.4, 2.5 e 2.6 verifica-se que as mesmas representam as restrições de um problema de transporte onde $g \in K_1$ e $j \in J$ são respectivamente os centros de oferta e de demanda. Sabe-se que o Problema de Transporte admite solução se a oferta total é maior que a demanda total, isto é:

$$\sum_{g \in K_1} (a_g - \sum_{j \in J} x_{gj}^{*}) \geq \sum_{j \in J} x_{ij}*$$

Da equação acima obtem-se:

$$\sum_{g \in K_1} a_g \geq \sum_{g \in K_1} \sum_{j \in J} x_{gj}^{*} + \sum_{j \in J} x_{ij}^{*} = \sum_{g \in K_1 \cup i} \sum_{j \in J} x_{gj}^{*} = \sum_{j \in J} b_j$$

Tem-se portanto a viabilidade de $\overline{x}$. Com isso pode-se calcular uma expressão para $\overline{\delta}_i$.

$\delta_i(K_1) \leq \sum_{g \in K_1} \sum_{j \in J} c_{gj} \overline{x}_{gj} - \sum_{g \in K_1 \cup i} \sum_{j \in J} c_{gj} x_{gj}^{*} = \sum_{g \in K_1} \sum_{j \in J} c_{gj} x_{gj}^{*} +$
$\sum_{g \in K_1} \sum_{j \in J} c_{gj} x_{gj}^{+} - \sum_{g \in K_1} \sum_{j \in J} c_{gj} x_{gj}^{*} - \sum_{j \in J} c_{ij} x_{ij}^{*} = \sum_{g \in K_1} \sum_{j \in J} c_{gj} x_{gj}^{+} -$
$\sum_{j \in J} c_{ij} x_{ij}^{*}$

Seja $c_{max_j} = max_{g \in K_1} c_{gj} \quad \forall j \in J$. Tem-se portanto um limite inferior para $\Delta_i(K_1)$ fazendo:

$\delta_i(K_1) \leq \sum_{g \in K_1} \sum_{j \in J} c_{max_j} x^+_{gj} - \sum_{j \in J} c_{ij} x^*_{ij} = \sum_{j \in J} \sum_{g \in K_1} c_{max_j} x^+_{gj} - \sum_{j \in J} c_{ij} x^*_{ij} = \sum_{j \in J} c_{max_j} \sum_{g \in K_1} x^+_{gj} - \sum_{j \in J} c_{ij} x^*_{ij}$

Utilizando 2.6 tem-se:

$$\delta_i(K_1) \leq \sum_{j \in J} c_{max_j} x^*_{ij} - \sum_{j \in J} c_{ij} x^*_{ij} = \sum_{j \in J} (c_{max_j} - c_{ij}) x^*_{ij}$$

Como

$$\sum_{j \in J} x^*_{ij} \leq a_i$$

$$x^*_{ij} \leq b_j \quad \forall j \in J$$

$$x^*_{ij} \geq 0$$

tem-se que x^* é uma solução viável do seguinte problema da mochila

$$\max_{x_i \in X_i} \sum_{j \in J} (c_{max_j} - c_{ij}) x_{ij}$$

onde x_i representa um conjunto de valores x_{ij} $\forall j \in J$, $X_i = \{x_i | \sum_{j \in J} x_{ij} \leq a_i,\ x_{ij} \leq b_j\ \forall j \in J,\ x_{ij} \geq 0\ \forall j \in J\}$. Seja $\tilde{x}$ a solução ótima do problema da mochila acima. Então:

$$\delta_i(K_1) \leq \max \sum_{j \in J} (c_{max_j} - c_{ij}) x^*_{ij} \leq \sum_{j \in J} (c_{max_j} - c_{ij}) \tilde{x}_{ij}$$

Ao invés de determinar x^* calculando $w(K_1 \cup i)$, o que implica na resolução de um problema de transporte, basta resolver o problema da mochila (contínuo) acima.

Para resolver o problema ordenar-se crescentemente os coeficientes $(c_{max_j} - c_{ij}) > 0$ de modo que:

$$(c_{max_{j_1}} - c_{ij_1}) \geq (c_{max_{j_2}} - c_{ij_2}) \geq ... \geq (c_{max_{j_n}} - c_{ij_n})$$

Tem-se

$$\delta_i(K_1) \leq \overline{\delta_i}(K_1) = \sum_{k=1...n} (c_{max_{j_k}} - c_{ij_k})\tilde{x}_{ij_k}$$

onde:

$$\tilde{x}_{ij_1} = min(a_i, b_{j_1})$$

$$\tilde{x}_{ij_2} = min(a_i - \tilde{x}_{ij_1}, b_{j_2})$$

$$\tilde{x}_{ij_n} = min(a_i - \tilde{x}_{ij_1} - \tilde{x}_{ij_2} - ... - \tilde{x}_{ij_{n-1}}, b_{j_n})$$

Não é necessário determinar todos $\tilde{x}_{ij_k}$, basta determinar até o primeiro $\tilde{x}_{ij_k} = 0$, os demais terão valor zero.

Exemplo 2.2 *Aplicação do* $\overline{\delta_i}(K_1)$.

Neste exemplo, são utilizados os dados do exemplo 2.1 mostrados na tabela 2.1 para verificar a funcionalidade da aproximação do C-Teste. A tabela 2.3 mostra a realização da aproximação para C-Teste $\forall i \in K_2$, depois da aplicação da aproximação do O-Teste visto na tabela 2.2.

A aplicação da aproximação tem sucesso para F_2 e F_3, fornecendo $K_0 = \{F_2, F_3\}$, $K_1 = \{F_1, F_4\}$ e $K_2 = \emptyset$. Com isso, somente com a aplicação das aproximações consegue-se definir o *status* de todas as facilidades. •

F_i	c_{max_j}	δ_i	$\tilde{\delta}_i$	Δ_i^{li}	**Resultado**
F_2	20,20,30,30,30,30	0	100	100	sucesso
F_3	20,20,30,30,30,30	50	150	10	sucesso

Tabela 2.3: Aplicação da aproximação para C-Teste

2.3 Avaliação Comparativa das Aproximações

Foram desenvolvidas aproximação para o O e C-Testes por Akinc and Khumawala [3], Bornstein and Mateus [11] e Azlán [4].

A tabela 2.4 mostra os resultados computacionais das aproximações para o O-Teste, e a tabela 2.5 mostra os resultados das aproximações para o C-Teste. As aproximações para o C-Teste são utilizadas somente para os problemas onde a aplicação das aproximações para o O-Teste tornaram K_1 viável. Utilizou-se os problemas (41-131) de Kühn & Hamburger [22] e (A1,B1,C1) de Beasley [8] para obtenção dos resultados. Para comparação do O-Teste, utilizou-se a aproximação desenvolvida por Azlán [4], por ser a melhor em termos de qualidade da solução gerada e praticamente tão eficiente em termos computacionais quanto as outras que podem ser vistas em Azlán [4], Mateus [23] e Campêlo Neto [24]. Referenciamos o O-Teste exato, a aproximação de Azlán [4] e a aproximação proposta respectivamente por $\Delta_i(K_1 \cup K_2 - i)$, $\overline{\Delta_i}(K_1 \cup K_2 - i)$ e $\Delta_i^{ls}(K_1 \cup K_2 - i)$. Para as aproximações $\overline{\Delta_i}(K_1 \cup K_2 - i)$ e $\Delta_i^{ls}(K_1 \cup K_2 - i)$ a tabela 2.4 apresenta os seguintes resultados: o total em segundos para a realização dos n testes, a percentagem de testes aproximados que obtiveram resultados iguais aos testes

exatos representado por $\xi(\%)$ e o erro médio relativo($\frac{\Delta_{aproximado}-\Delta_{exato}}{\Delta_{exato}} * 100$) dos n testes com relação aos testes exatos representado por $\epsilon(\%)$.

Prob.	**nxm**	Δ_i	$\overline{\Delta_i}$			Δ_i^{ls}		
		sec.	**sec.**	$\xi(\%)$	$\epsilon(\%)$	**sec.**	$\xi(\%)$	$\epsilon(\%)$
41	16x50	0.15	0.06	25	20.95	0.0006	18	56.39
51	16x50	0.06	0.04	68	3.79	0.0006	56	13.79
61	16x50	0.05	0.04	81	2.53	0.0006	50	11.87
71	16x50	0.05	0.03	100	0.0	0.0006	62	9.8
81	25x50	0.15	0.11	68	7.34	0.0010	52	20.25
91	25x50	0.11	0.08	92	0.99	0.0010	80	3.89
101	25x50	0.10	0.07	100	0.0	0.0010	88	2.9
111	50x50	0.33	0.27	88	4.87	0.0020	82	8.80
121	50x50	0.28	0.23	96	0.42	0.0021	96	0.42
131	50x50	0.27	0.23	100	0.0	0.0020	100	0.0
A1	100x1000	125.59	78.66	100	0.0	0.1093	100	0.0
B1	100x1000	134.41	86.20	100	0.0	0.1079	100	0.0
C1	100x1000	135.92	88.81	100	0.0	0.1114	100	0.0
média		30.57	19.6	86	3.15	0.0262	75.69	9.85

Tabela 2.4: Resultados computacionais das aproximações para O-Teste

Para comparação do C-Teste utilizou-se a aproximação desenvolvida por Bornstein and Mateus [11], por ser uma das melhores na qualidade da solução e eficiência computacional. Mais detalhes podem ser visto em Azlán [4], Mateus [23] e Campêlo Neto [24]. Referenciando o C-Teste exato, a aproximação de Bornstein and Mateus [11] e a aproximação proposta respectivamente por $\Delta_i(K_1)$, $\tilde{\Delta}_i(K_1)$ e $\Delta_i^{li}(K_1)$, a tabela 2.5 mostra, para os problemas 61, 91 e 121, o tempo total em segundos para obtenção respectivamente de 7, 15 e 43 ($\forall i \in K_2$) testes realizados, calculando-se $\Delta_i(K_1)$. Para as aproximações $\tilde{\Delta}_i(K_1)$ e $\Delta_i^{ls}(K_1)$, a tabela 2.5 apresenta os seguintes resultados: o total em segundos para a realização dos testes,

a percentagem de testes aproximados que obtiveram resultados iguais aos testes exatos representado por $\xi(\%)$ e o erro médio relativo ($\frac{\Delta_{exato}-\Delta_{aproximado}}{\Delta_{exato}} * 100$) dos testes com relação aos testes exatos representado por $\epsilon(\%)$.

Prob.	**nxm**	Δ_i	$\tilde{\Delta}_i$			Δ_i^{li}		
		sec.	**sec.**	$\xi(\%)$	$\epsilon(\%)$	**sec.**	$\xi(\%)$	$\epsilon(\%)$
61	16x50	0.0158	0.0055	100	0.0	0.0004	0	604.17
91	25x50	0.0354	0.0086	100	0.0	0.0008	0	823.34
121	50x50	0.1342	0.0138	46	32.23	0.0022	0	563.19
média		0.0618	0.0093	82	10.74	0.0011	0	663.57

Tabela 2.5: Resultados computacionais das aproximações para C-Teste

Os algoritmos de aproximação foram implementados em C e o problema de transporte foi implementado em C++. Os testes foram rodados em uma estação Sun Sparc 4. O método de transporte utilizado foi uma combinação dos métodos propostos por Jacobsen [19], Branman [12] e Ahrens and Finke [1], utilizando uma adaptação para o PLC. Esta adptação consiste em utilizar a base ótima de um problema de transporte para a resolução do outro, pois na resolução dos testes $\forall i \in K_2$, os problemas de transporte diferem em apenas um centro de oferta.

Concluindo, pode-se dizer que Δ_i^{ls} fornece resultados piores com relação à qualidade da solução, contudo em termos de tempo computacional os resultados são obtidos em um tempo menor. Os resultados de Δ_i^{li} deixam muito a desejar com relação a qualidade da solução encontrada. A vantagem em termos de tempos computacionais não justifica a piora em termos da qualidade de Δ_i^{li}, de modo que esta aproximação deve ser descartada.

Como pode ser visto na tabela 2.4 o tempo para resolver os testes exatos não chega em média a ser duas vezes maior que a aproximação desenvolvida por Azlán [4]. Pode-se concluir que pelo fato de ser usado um algoritmo de transporte muito eficiente que calcula novas soluções ótimas a partir de soluções ótimas antigas fazendo-se pequenas modificações, onde são necesárias poucas iterações adicionais, a solução dos testes exatos é muito rápida não se justificando portanto as aproximações. Portanto, o pequeno ganho em termos computacionais não justifica a piora na qualidade do desempenho dos testes. Com relação à aproximação Δ_i^{ls} para o O-Teste, houve um ganho computacional razoável em relação a solução exata que pode vir a justificar a piora na qualidade da solução obtida.

Capítulo 3

Fechamento do gap de indefinição

Primeiramente mostraremos, nas seções 3.1 e 3.2, porque os testes não são suficientemente fortes para garantir que o gap feche, ou seja, identificaremos quais são os principais problemas que levam a falha de ambos os testes na tentativa de definir a que conjunto pertence uma certa facilidade. Depois mostraremos, na seção 3.3, como o sucesso de um teste, potencializa o método, isto é, por exemplo, o sucesso do O-Teste, ou seja, a abertura de uma facilidade, aumenta K_1, aumentando a chance do C-Teste funcionar, fechando novas facilidades. Na seção 3.4 apresentaremos dois algoritmos constituidos dos testes de redução. Pode-se verificar que em alguns casos o uso destes algoritmos, ou seja, a aplicação sucessiva do O-Teste e do C-Teste, pode definir o *status* de todas as facilidades. Por último discutiremos vários casos especiais, para os quais não se consegue garantir o fechamento do gap de indefinição aplicando somente os testes de redução. Nestes casos verificaremos como é o comportamento dos testes, ou seja, verificaremos qual é a influência do O-Teste e do C-Teste no esforço do fechamento do gap.

Estes casos serão vistos na seção 3.5.

3.1 Dificuldades no fechamento do gap

Examinaremos a seguir um caso completo de aplicação do O-Teste e C-Teste verificando razões para falhas destes.

Exemplo 3.1 *falhas no fechamento do gap.*

Apresentaremos um exemplo com 4 facilidades e 6 clientes sem restrições de capacidade para as facilidades. Com a eliminação das restrições de capacidade tem-se que a demanda de um cliente j é atendida somente por uma facilidade i (veja Jones, Lower, Muller. et al [21]), com isso, os dados relativos ao custo de transporte da facilidade i para o cliente j podem ser representados na forma $c_{ij}x_{ij}$, onde o valor de x_{ij} é toda a demanda do cliente j, logo $c_{ij}x_{ij} = c_{ij}b_j$. Os dados relativos a $c_{ij}b_j$ e f_i são apresentados na tabela 3.1, onde $F_i \;\; i = 1, .., 4$ representa as facilidades, $C_j \;\; j = 1, ..., 6$ os clientes e $f_i \;\; i = 1, ..., 4$ representa os custos fixos. Por exemplo, a linha 2 coluna 2 cujo valor é 20 representa o custo da facilidade F_1 atender toda a demanda do cliente C_1.

Na resolução do problema será primeiro aplicado o O-Teste para todas as facilidades e depois o C-Teste. Para realização dos testes precisa-se do cálculo do Problema de Transporte associado a cada teste. Para que se possa ter uma visualização mais precisa do que está acontecendo quando se aplicam os testes,

	C_1	C_2	C_3	C_4	C_5	C_6	f_i
F_1	20	20	20	10	10	20	10
F_2	10	10	10	20	20	10	20
F_3	30	30	15	30	20	5	10
F_4	5	5	30	30	30	30	20

Tabela 3.1: Dados do problema (F_1, F_2 - ótimo)

apresenta-se na figura 3.1 as resoluções dos Problemas de Transporte necessários aos testes. Os círculos brancos representam as facilidades, os hachurados os clientes e o valor da linha que liga uma facilidade a um cliente representa o custo de atendimento do cliente pela facilidade.

K_0 e K_1 são inicializados com vazio, enquanto K_2 é inicializado com todas as facilidades do problema. Primeiro realiza-se o O-Teste $\forall i \in K_2$, mostrados na Tabela 3.2. Para facilidade de notação denota-se no texto F_i por i.

Como visto na tabela 3.2, com a aplicação do O-Teste, obteve-se sucesso somente para facilidade 1, deixando $K_0 = \emptyset$, $K_1 = \{1\}$ e $K_2 = \{2, 3, 4\}$. Como o problema não tem restrição de capacidade K_1 é viável, logo pode-se aplicar o C-Teste $\forall i \in K_2$, mostrados na tabela 3.3.

A tabela 3.3 mostra que nenhum C-teste obteve sucesso, indicando a falha dos testes $\forall i \in K_2$. •

O Exemplo 3.1 informa que mesmo para o problema sem restrições de capacidade, os testes de redução frequentemente não conseguem fechar o gap. Isso acontece por causa da distribuição de clientes a facilidades. Cada cliente procura

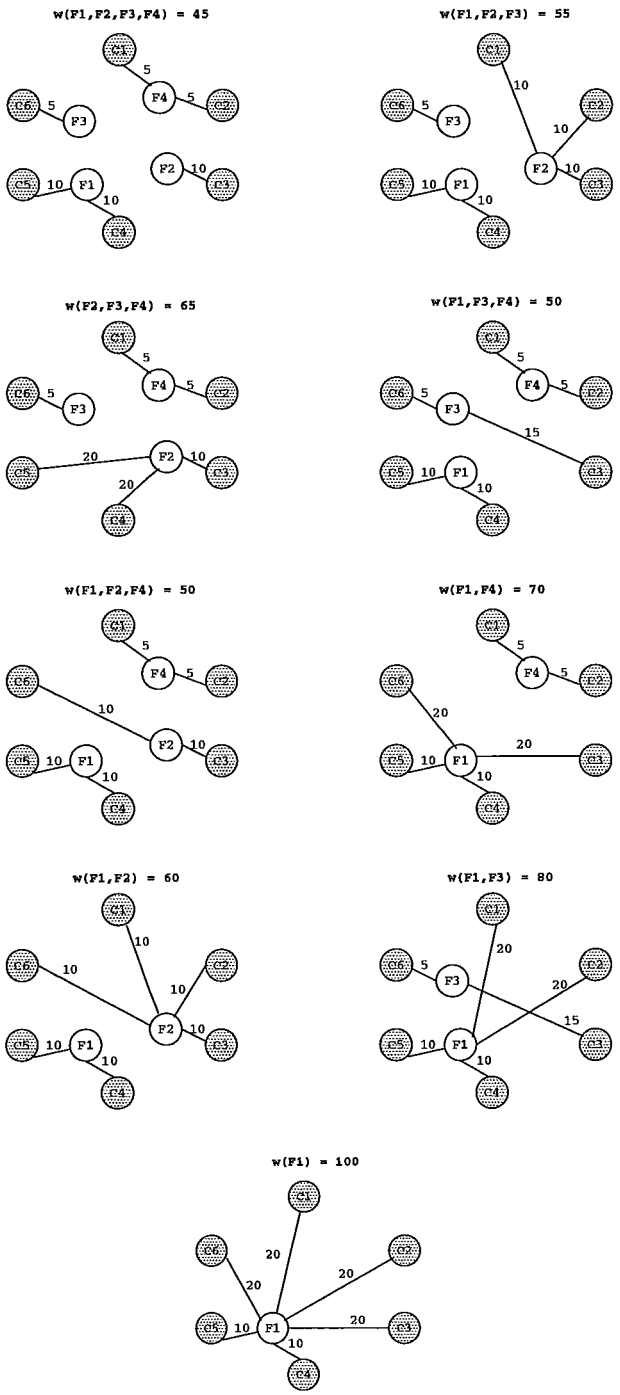


Figura 3.1: Resolução de algum w(K)

O-Teste para i = 1	$K_0 = K_1 = \emptyset K_2 = I = \{1,2,3,4\}$
$\delta_1(K_1 \cup K_2 - 1) = 65 - 45 = 20$	
$\Delta_1(K_1 \cup K_2 - 1) = 10 - 20 = -10$	
Logo O-Teste de 1 funcionou	
O-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_2(K_1 \cup K_2 - 2) = 50 - 45 = 5$	
$\Delta_2(K_1 \cup K_2 - 2) = 20 - 5 = 15$	
Logo O-Teste de 2 falhou	
O-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_3(K_1 \cup K_2 - 3) = 50 - 45 = 5$	
$\Delta_3(K_1 \cup K_2 - 3) = 10 - 5 = 5$	
Logo O-Teste de 3 falhou	
O-Teste para i = 4	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_4(K_1 \cup K_2 - 4) = 55 - 45 = 10$	
$\Delta_4(K_1 \cup K_2 - 4) = 20 - 10 = 10$	
Logo O-Teste de 4 falhou	

Tabela 3.2: Aplicação do O-Teste

C-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_2(K_1) = 100 - 60 = 40$	
$\Delta_2(K_1) = 20 - 40 = -20$	
Logo C-Teste de 2 falhou	
C-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_3(K_1) = 100 - 80 = 20$	
$\Delta_3(K_1) = 10 - 20 = -10$	
Logo C-Teste de 3 falhou	
C-Teste para i = 4	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_4(K_1) = 100 - 70 = 30$	
$\Delta_4(K_1) = 20 - 30 = -10$	
Logo C-Teste de 4 falhou	

Tabela 3.3: Aplicação do C-Teste

ser atendido pela facilidade mais interessante, ou seja, a facilidade que forneça o custo variável menor. Com isso, a indefinição do O-Teste costuma ocorrer para problemas em que $K_1 \cup K_2$ é muito grande, isto é, quando K_0 é pequeno. Neste caso clientes são atendidos por facilidades que na verdade deveriam estar fechadas na solução ótima. Assim, ao se retirar uma facilidade i, a falta que esta faz, não é sentida em toda sua extensão, pois os clientes que deveriam ser atendidos por ela são atendidos por uma das muitas facilidades que possivelmente (na solução ótima) deveriam estar fechadas. Por exemplo para $K_1 = \{1\}$ e $K_2 = \{2, 3, 4\}$ no O-Teste para $i = 2$, a qual é aberta na solução ótima atendendo os clientes C_1, C_2, C_3 e C_6, as facilidades 3 e 4 $\in K_2$ que devem estar fechadas na solução ótima roubam clientes de 2 e amenizam a falta que esta faz. 4 atende C_1 e C_2 e 3 atende C_6 e C_3. Com isto, o aumento nos custos de transporte ocasionado pela retirada de 2 é pequeno e consequentemente o O-Teste deixa de funcionar para 2. Caso qualquer uma das facilidades 3 ou 4 fossem fechadas o O-Teste para 2 teria sucesso.

O problema inverso ocorre com o C-Teste. Neste caso um K_1 muito pequeno provoca uma sensibilidade muito grande do custo de transporte quando é inserido uma facilidade, dificultando o fechamento desta. Por exemplo quando com $K_1 = \{1\}$ e $K_2 = \{2, 3, 4\}$ se aplica o C-Teste para $i = 4$ o ganho com a abertura de 4 em termos de redução do custo de transporte é de 30, superando o custo fixo 20 (indicando a falha do teste), enquanto que o ganho real é de somente 10 se a facilidade 2 estivesse aberta, a qual deve estar na solução ótima.

Resumindo, pode-se dizer que o O-Teste funciona bem quando existe grande sensibilidade do custo de transporte à retirada/adição de uma facilidade $i \in K_2$. Por sensibilidade entende-se a variação dos custos de transporte. Ora, a sensibilidade é grande quando o conjunto $(K_1 \cup K_2)$ de facilidades onde atua i é pequeno. Já o C-Teste funciona bem quando existe pequena sensibilidade dos custos de transporte à retirada/adição de uma facilidade $i \in K_2$. A sensibilidade é pequena quando o conjuto (K_1) de facilidades onde atua i é grande.

O que enfraquece os testes e evita o fechamento do gap é que, na etapa inicial, o O- e C-Teste são aplicados respectivamente nos conjuntos K_0 e K_1 muito pequenos.

3.2 Generalização das dificuldades do fechamento do gap para um PLC qualquer

Os problemas identificados na seção anterior não necessariamente acontecem para todas as facilidades. Podem existir facilidades e clientes agrupados de tal maneira que a decisão associada aos testes seja fácil. Por outro lado, podem existir facilidades e clientes que constituem uma *região de indecisão*, onde problemas como os vistos em 3.1 ocorrem. É o que será visto no exemplo a seguir.

A figura 3.2 apresenta um problema onde as facilidades são representadas por círculos e os clientes por quadrados, consideram-se aplicados os testes de redução. Os círculos pretos representam as facilidades que os testes conseguiram abrir, e os círculos hachurados representam as facilidades que os testes conseguiram fechar.

As regiões R_1, R_2 e R_3 representam *regiões de indecisão*. Uma decisão tomada com objetivo de solucionar a indecisão na região R_1 não influenciará em nada a resolução das regiões R_2 e R_3. Já uma decisão tomada com o objetivo de solucionar a indecisão de uma das regiões R_2 ou R_3 pode influenciar na resolução da outra. Isso porque ambas tem uma facilidade em comum, e decidindo o resultado desta pode influenciar positivamente o sucesso dos testes nas duas regiões, conforme será visto na seção 3.3. Isso mostra que existe um grau de dependência entre as *regiões de indecisão*, o qual depende do número de facilidades em comum que as regiões têm.

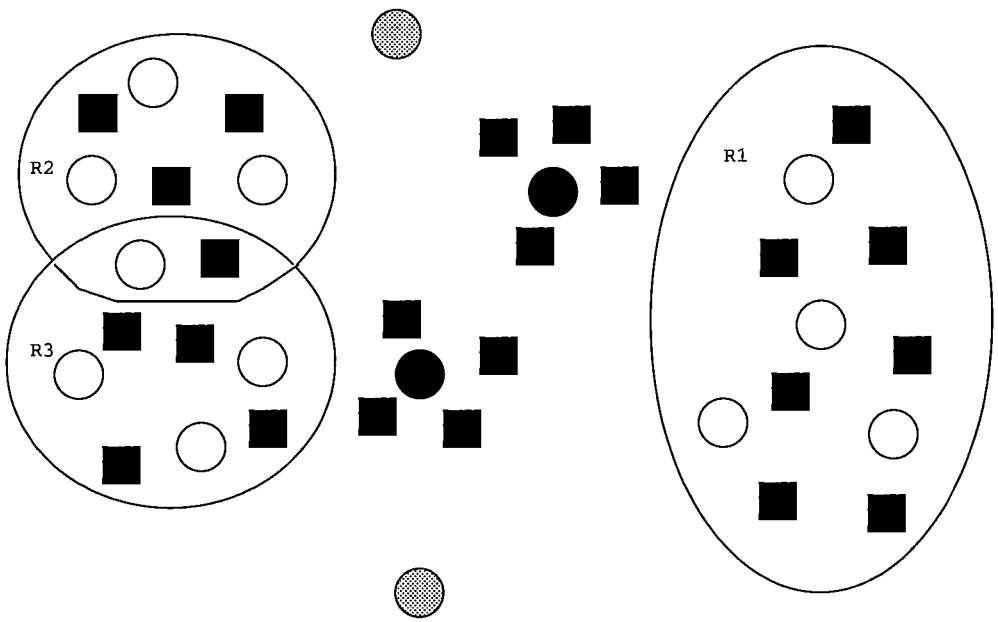


Figura 3.2: Panorama geral da execução dos testes de redução

Verificou-se que a capacidade não influencia muito no problema do fechamento do gap, ou seja, encontra-se o mesmo problema para o caso capacitado.

A decisão relativa à abertura/fechamento de uma facilidade dentro de uma

região de indecisão, mesmo que temporária, isto é, a decisão pode ser revertida em etapas posteriores do algoritmo, aumenta o potencial dos testes com relação a outras facilidades, pois caso uma facilidade seja fechada o conjunto $K_2 \cup K_1$ ficará menor, aumentando assim as possibilidades de sucesso do O-Teste. Da mesma forma caso uma facilidade seja aberta aumentará o conjunto K_1 aumentando juntamente as possibilidades de sucesso do C-Teste. Esta característica favorece o uso dos testes em conexão com um algoritmo *Branch and Bound*, que será visto no capítulo 4.

3.3 Influência de um teste no outro

Nesta seção apresentaremos o relacionamento entre os testes, ou seja, como o sucesso de um influencia a resolução do outro.

O sucesso de um teste, que é a tomada de decisão de abrir ou fechar uma facilidade, influencia positivamente o sucesso do outro, ou seja, quando um teste obtem sucesso ele aumenta as chances do outro obter sucesso também. O sucesso do C-Teste diminui $K_1 \cup K_2$, fazendo com que no O-Teste a retirada de uma facilidade seja melhor sentida, ou seja, a falta da facilidade é mais representativa. Por outro lado, o sucesso do O-Teste aumenta K_1, implicando que no C-Teste a inserção de um facilidade seja avaliada de forma mais realista.

Exemplificando, suponha que aplicando-se o O-Teste $\forall i \in K_2$ nenhum tenha sucesso, e aplicando-se o C-Teste este tenha sucesso para pelo menos um $i \in K_2$.

Como o sucesso do C-Teste pode habilitar o sucesso do O-Teste, então aplicando o O-Teste, pode ser que consiga-se sucesso. O exemplo número a seguir reflete bem a influência de um teste no outro.

Exemplo 3.2 *Influência de um teste no outro.*

Os dados do exemplo são fornecidos na Tabela 3.4. Como no exemplo 3.1 F_i $i = 1, ..., 4$ representa as facilidades, C_j $j = 1, ..., 6$ os clientes, e f_i $i = 1, ..., 4$ os custos fixos de instalação das facilidades. Como agora o problema é capacitado, o custo de transporte de uma facilidade para um cliente é unitário, ou seja, representa o custo de transporte da facilidade i para o cliente j de uma unidade. A demanda de cada cliente é representada por b_j $j = 1, ..., 6$, e a capacidade de cada facilidade por a_i $i = 1, ..., 4$.

	C_1	C_2	C_3	C_4	C_5	C_6	f_i	a_i
F_1	4	4	2.5	2.5	2	5	10	31
F_2	2	2	1.25	5	4	2.5	20	22
F_3	4.2	4.2	2.375	5.25	4	2.75	10	20
F_4	1	1	3.75	7.5	6	7.5	20	20
b_j	5	5	8	4	5	4		

Tabela 3.4: Dados do problema (F_1, F_2 - ótimo)

A tabela 3.5 apresenta somente as soluções dos Problemas de Transporte necessários para a realização dos testes. Os conjuntos são inicializados da seguinte forma: $K_0 = K_1 = \emptyset$ e $K_2 = I = \{1, 2, 3, 4\}$. Primeiro realiza-se o O-Teste até que ele não tenha mais sucesso, a tabela 3.6 mostra a realização do O-Teste $\forall i \in K_2$.

Problema de Transporte	**Solução**
w(F1,F2,F3,F4)	50
w(F1,F2,F3)	60
w(F2,F3,F4)	70
w(F1,F3,F4)	60
w(F1,F2,F4)	50
w(F1, F4)	70
w(F1, F2)	60
w (F1,F3)	90
w (F1)	100

Tabela 3.5: Resolução de alguns $w(K)$

O-Teste para i = 1	$K_0 = K_1 = \emptyset\ K_2 = I = \{1,2,3,4\}$
$\delta_1(K_1 \cup K_2 - 1) = 70 - 50 = 20$	
$\Delta_1(K_1 \cup K_2 - 1) = 10 - 20 = -10$	
Logo O-Teste de 1 funcionou	
O-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_2(K_1 \cup K_2 - 2) = 60 - 50 = 10$	
$\Delta_2(K_1 \cup K_2 - 2) = 20 - 10 = 10$	
Logo O-Teste de 2 falhou	
O-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_3(K_1 \cup K_2 - 3) = 50 - 50 = 0$	
$\Delta_3(K_1 \cup K_2 - 3) = 10 - 0 = 10$	
Logo O-Teste de 3 falhou	
O-Teste para i = 4	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\delta_4(K_1 \cup K_2 - 4) = 60 - 50 = 10$	
$\Delta_4(K_1 \cup K_2 - 4) = 20 - 10 = 10$	
Logo O-Teste de 4 falhou	

Tabela 3.6: Aplicação do O-Teste

Com a realização dos O-Testes obteve-se $K_0 = \emptyset$, $K_1 = \{1\}$ e $K_2 = \{2, 3, 4\}$. Agora realiza-se o C-Teste até que este não tenha mais sucesso, a tabela 3.7 mostra os resultados da aplicação dos C-Testes.

C-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2, 3, 4\}$
$\delta_2(K_1) = 100 - 60 = 40$	
$\Delta_2(K_1) = 20 - 40 = -20$	
Logo C-Teste de 2 falhou	
C-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2, 3, 4\}$
$\delta_3(K_1) = 100 - 90 = 10$	
$\Delta_3(K_1) = 10 - 10 = 0$	
Logo C-Teste de 3 funcionou	
C-Teste para i = 4	$K_0 = \{3\}\ K_1 = \{1\}\ K_2 = \{2, 4\}$
$\delta_4(K_1) = 100 - 70 = 30$	
$\Delta_4(K_1) = 20 - 30 = -10$	
Logo C-Teste de 4 falhou	

Tabela 3.7: Aplicação do C-Teste

Com a realização dos C-Testes fechou-se a facilidade 3, ficando com $K_0 = \{3\}$, $K_1 = \{1\}$ e $K_2 = \{2, 4\}$. Como, com a aplicação do C-Teste, teve-se sucesso para $i = 3$, então passa-se a realizar o O-Teste novamente $\forall i \in K_2$. Os resultados da nova aplicação dos testes estão na tabela 3.8.

Com a realização dos O-Testes abriu-se a facilidade 2, ficando com $K_0 = \{3\}$, $K_1 = \{1, 2\}$ e $K_2 = \{4\}$. Como isso pode-se aplicar novamente o C-Teste $\forall i \in K_2$. Os resultados da nova aplicação dos testes são mostrados na tabela 3.9. Após a realização do C-Teste $\forall i \in K_2$ tem-se $K_0 = \{3, 4\}$, $K_1 = \{1, 2\}$ e $K_2 = \emptyset$, indicando o fechamento do gap, atingindo assim a solução ótima do problema.

•

O-Teste para i = 2	$K_0 = \{3\}\ K_1 = \{1\}\ K_2 = \{2,4\}$
$\delta_2(K_1 \cup K_2 - 2) = 70 - 50 = 20$	
$\Delta_2(K_1 \cup K_2 - 2) = 20 - 20 = 0$	
Logo O-Teste de 2 funcionou	
O-Teste para i = 4	$K_0 = \{3\}\ K_1 = \{1,2\}\ K_2 = \{4\}$
$\delta_4(K_1 \cup K_2 - 4) = 60 - 50 = 10$	
$\Delta_4(K_1 \cup K_2 - 4) = 20 - 10 = 10$	
Logo O-Teste de 4 falhou	

Tabela 3.8: Aplicação do O-Teste

C-Teste para i = 4	$K_0 = \{3\}\ K_1 = \{1,2\}\ K_2 = \{4\}$
$\delta_4(K_1) = 60 - 50 = 10$	
$\Delta_4(K_1) = 20 - 10 = 10$	
Logo C-Teste de 4 funcionou	

Tabela 3.9: Aplicação do C-Teste

3.4 Algoritmos que verificam o fechamento do gap

Os algoritmos que apresentaremos procuram através dos testes de redução encontrar a solução ótima de um PLC. Como dito anteriormente, utilizando-se o O-Teste e o C-Teste, somente pode-se encontrar a solução ótima para o problema se dado o impasse na aplicação de um tipo de teste, o outro teste obtiver sucesso, isto é, conseguir definir o *status* de pelo menos uma facilidade adicional. Caso desta forma consiga-se fechar o gap tem-se uma solução ótima.

Os algoritmos que apresentaremos muitas vezes não conseguem encontrar a solução ótima, pois como visto na seção 2.1 na maioria dos problemas os testes

de redução não conseguem fechar o gap. Contudo pode-se aplicar os algoritmos com o objetivo de reduzir a dimensão do problema.

3.4.1 Algoritmo 1

O algoritmo 1 é um algoritmo geral para resolver o problema através dos testes, ou seja, caso exista um teste que tenha sucesso para alguma facilidade o algoritmo o realizará. O algoritmo usa além dos dados do problema e dos conjuntos definidos, uma variável boleana, a qual é chamada de **Parada**. Esta variável representará a possibilidade ou não de sucesso dos testes. Ela assumirá o valor verdadeiro quando os testes não puderem ser realizados com sucesso para nenhuma facilidade pertence a K_2. Parte-se do princípio que para aplicação do algoritmo 1 o problema seja viável, ou seja, $\sum_{i \in I} a_i \geq \sum_{j \in J} b_j$. O algoritmo trabalha da seguinte maneira: inicializa os conjuntos $K_0 = K_1 = \emptyset$ e $K_2 = I$ e começa a realização dos testes. Primeiro realiza o O-Teste $\forall i \in K_2$, passando para a realização dos C-Testes. Para que se possa realizar os C-Testes é necessário que K_1 seja viável. Sempre quando um teste obtiver sucesso para algum $i \in K_2$ o outro deve ser testado, conforme foi mostrado na seção 3.3. Este controle é realizado no algoritmo através da variável **Parada**, que neste caso recebe o valor Falso. Caso o O-Teste ou o C-Teste não tenham sucesso para nenhum $i \in K_2$ o algoritmo termina, pois a variável **Parada** assume o valor verdadeiro. Este processo de realização dos O-Testes e depois C-Testes repete-se até que eles consigam fechar o gap, ou até quando $\forall i \in K_2$ nenhum teste obtiver sucesso.

inicio Algoritmo 1

$K_2 = I$

$K_0 = K_1 = \emptyset$

Parada = Falso;

enquanto $K_2 \neq 0$ e Parada == Falso **fazer { laço 1 }**

Parada = Verdadeiro

calcular $w(K_2 \cup K_1)$

para cada $i \in K_2$ **fazer { laço 2 }**

calcular $w(K_2 \cup K_1 - \{i\})$

$\delta_i = w(K_2 \cup K_1 - \{i\}) - w(K_2 \cup K_1)$

$\Delta_i = f_i - \delta_i$

se $\Delta_i \leq 0$ **então**

$K_1 = K_1 \cup \{i\}$

$K_2 = K_2 - \{i\}$

Parada = Falso

se K_1 *viável* e Parada == Falso **então**

Parada = Verdadeiro

calcular $w(K_1)$

para cada $i \in K_2$ **fazer {laço 3}**

calcular $w(K_1 \cup \{i\})$

$\delta_i = w(K_1) - w(K_1 \cup \{i\})$

$\Delta_i = f_i - \delta_i$

se $\Delta_i \geq 0$ **então**

$K_0 = K_0 \cup \{i\}$

$K_2 = K_2 - \{i\}$

Parada = Falso

fim

O laço 3 representa a realização do C-Teste $\forall i \in K_2$, O laço 2 representa a realização do O-Teste $\forall i \in K_2$ e o laço 1 representa a execução do algoritmo como um todo, que terminará quando $K_2 = \emptyset$ ou quando for verificado que os testes não conseguem fechar o gap. Seja n o número de elementos em I e $O(w)$ a complexidade do algoritmo para resolver o Problema de Transporte, como a cada iteração do laço 1 pelo menos uma facilidade é retirada de K_2 teremos, no máximo, n iterações do laço 1. Por outro lado, dentro do laço 1, os laços 2 e 3 implicam, no pior caso, em resolver $2n$ problemas de transporte. Logo a complexidade do algoritmo 1 é $O(n^2O(w))$.

Exemplo 3.3 *Exemplo de aplicação do algoritmo 1.*

Para mostrar como o algoritmo 1 trabalha e como os testes se relacionam, será utilizado o mesmo exemplo da seção 3.3, para o qual consegue-se através dos testes fechar o gap. Os dados do problema estão na tabela 3.4. A tabela 3.5 apresenta somente as soluções dos Problemas de Transporte necessários para a realização dos testes. Os conjuntos são inicializados da seguinte forma: $K_0 = K_1 = \emptyset$ e $K_2 = I = \{1, 2, 3, 4\}$. A variável **Parada** é sempre inicializada com Verdadeiro antes da aplicação do O-Teste e do C-Teste. A tabela 3.6 mostra a realização do O-Teste $\forall i \in K_2$, que representa a execução do laço 2 no algoritmo 1.

Com a realização dos O-Testes obteve-se sucesso para $i = 1$, indicando que **Parada** assumiu o valor Falso e os conjuntos ficaram $K_0 = \emptyset$, $K_1 = \{1\}$ e $K_2 = \{F2, F3, F4\}$. Como K_1 é viável e **Parada** é Falso executa-se o laço 3, ou

seja, realiza-se C-Teste $\forall i \in K_2$. Os resultados dos C-Testes são mostrados na tabela 3.7.

Com a realização dos C-Testes obteve-se sucesso para $i = 3$, indicando que **Parada** assumiu o valor Falso e os conjuntos ficaram $K_0 = \{3\}$, $K_1 = \{1\}$ e $K_2 = \{2, 4\}$. Como **Parada** é Falso e $K_2 \neq \emptyset$ executa-se o laço 2 novamente, ou seja, realiza-se O-Teste $\forall i \in K_2$. Os resultados dos O-Testes são mostrados na tabela 3.8.

Com a realização dos O-Testes obteve-se sucesso para $i = 2$, indicando que **Parada** assumiu o valor Falso e os conjuntos ficaram $K_0 = \{3\}$, $K_1 = \{1, 2\}$ e $K_2 = \{4\}$. Como K_1 é viável e **Parada** é Falso executa-se o laço 3, ou seja, realiza-se C-Teste para $\forall i \in K_2$. Os resultados dos C-Testes são mostrados na tabela 3.9.

Com a realização dos C-Testes obteve-se sucesso para $i = 4$, indicando que **Parada** assumiu o valor Falso e os conjuntos ficaram $K_0 = \{3, 4\}$, $K_1 = \{1, 2\}$ e $K_2 = \emptyset$. **Parada** é Falso, mas $K_2 = \emptyset$, logo pára-se a execução do laço 3, indicando com isso o final do algoritmo. Como $K_2 = \emptyset$ o gap fechou, obtendo-se com isso a solução ótima para o problema. •

3.4.2 Algoritmo 2

O algoritmo 2 difere do algoritmo 1 por não considerar a influência de um teste no outro. Isso torna o algoritmo mais específico, ou seja, restringirá ainda

mais a quantidade de problemas para os quais os testes conseguem gerar a solução ótima. Contudo o algoritmo é mais eficiente. O algoritmo 2 tenta encontrar um conjunto $\overline{K} \subseteq I$ tal que:

1. $\Delta_i(I - i) \leq 0 \quad \forall i \in \overline{K}$
2. $\Delta_i(\overline{K}) \geq 0 \quad \forall i \in I - \overline{K}$

Neste caso pode-se fazer $K_1 = \overline{K}$, $K_0 = I - \overline{K}$ levando $K_2 = \emptyset$. O algoritmo realiza o O-Teste $\forall i \in I$ colocando i em $\overline{K}$ caso $\Delta_i(I - i) \leq 0$, Se $\overline{K}$ for viável realiza-se o C-teste $\forall i \in I - \overline{K}$. Caso para algum $i \in I - \overline{K}$ $\Delta_i(\overline{K}) \leq 0$ o algoritmo pára, não tendo sido possível encontrar a solução ótima.

inicio Algoritmo 2

$\overline{K} = \emptyset$

calcular $w(I)$

para cada $i \in I$ **fazer { laço 1 }**

calcular $w(I - \{i\})$

$\delta_i = w(I - \{i\}) - w(I)$

$\Delta_i = f_i - \delta_i$

se $\Delta_i \leq 0$ **então**

$\overline{K} = \overline{K} \cup \{i\}$

se $\sum_{i \in \overline{K}} a_i \geq \sum_{j \in J} b_j$ **então**

calcular $w(\overline{K})$

para cada $i \in I - \overline{K}$ **fazer { laço 2 }**

calcular $w(\overline{K} \cup \{i\})$

$\delta_i = w(\overline{K}) - w(\overline{K} \cup i)$

$\Delta_i = f_i - \delta_i$

se $\Delta_i \leq 0$ **então**

o algoritmo não consegue a otimalidade pois não consegue-se fechar o gap

terminar

senao

o algoritmo não consegue a otimalidade pois $\overline{K}$ *é inviável*

terminar

o algoritmo encontrou a otimalidade

fim

O laço 1 realiza o O-Teste e o laço 2 o C-Teste. O laço 1 é determinante para a complexidade, pois $I > I - \overline{K}$. Sendo n o número de elementos em I e $O(w)$ a complexidade do algoritmo para resolver o Problema de Transporte, então a complexidade do algoritmo 2 é $O(nw(K))$.

Exemplo 3.4 *Exemplo de aplicação do algoritmo 2.*

Para mostrar como o algoritmo 2 trabalha será apresentado um exemplo onde o gap é fechado. Os dados do problema são fornecidos na tabela 3.10. Como no exemplo 3.2 F_i $i = 1, ..., 4$ representa as facilidades, C_j $j = 1, ..., 6$ os clientes, f_i $i = 1, ..., 4$ representa o custo fixo de instalação de cada facilidade, a_i $i = 1, ..., 4$

a capacidade de cada facilidade e b_j $j = 1, ..., 6$ a demanda de cada cliente.

	C_1	C_2	C_3	C_4	C_5	C_6	f_i	a_i
F_1	4	4	2.5	2.5	2	5	10	31
F_2	2	2	1.25	5	4	2.5	10	22
F_3	4.2	4.2	2.375	5.25	4	2.75	10	20
F_4	1	1	3.75	7.5	6	7.5	20	20
b_j	5	5	8	4	5	4		

Tabela 3.10: Dados do problema (F_1, F_2 - ótimo)

A tabela 3.11 apresenta somente as soluções dos Problemas de Transporte necessários para a realização dos testes. No inicio do algoritmo faz-se $\overline{K} = \emptyset$. A tabela 3.12 mostra a realização do O-Teste $\forall i \in I$, que representa a execução do laço 1 no algoritmo 2.

Problema de Transporte	**Solução**
w(F1,F2,F3,F4)	50
w(F1,F2,F3)	60
w(F2,F3,F4)	70
w(F1,F3,F4)	60
w(F1,F2,F4)	50
w(F1, F4)	70
w(F1, F2)	60
w (F1,F3)	90
w (F1)	100

Tabela 3.11: Resolução de alguns $w(K)$

Com a realização dos O-Testes obteve-se $\overline{K} = \{1, 2\}$. Como $\sum_{i \in \overline{K}} a_i \geq \sum_{j \in J} b_j$ realiza-se o C-Teste $\forall i \in I - \overline{K}$, a tabela 3.13 mostra os resultados da aplicação dos C-Testes.

O-Teste para i = 1	$\overline{K} = \emptyset$
$\delta_1(K_1 \cup K_2 - 1) = 70 - 50 = 20$	
$\Delta_1(K_1 \cup K_2 - 1) = 10 - 20 = -10$	
Logo O-Teste de 1 funcionou	
O-Teste para i = 2	$\overline{K} = \{1\}$
$\delta_2(K_1 \cup K_2 - 2) = 60 - 50 = 10$	
$\Delta_2(K_1 \cup K_2 - 2) = 10 - 10 = 0$	
Logo O-Teste de 2 funcionou	
O-Teste para i = 3	$\overline{K} = \{1, 2\}$
$\delta_3(K_1 \cup K_2 - 3) = 50 - 50 = 0$	
$\Delta_3(K_1 \cup K_2 - 3) = 10 - 0 = 10$	
Logo O-Teste de 3 falhou	
O-Teste para i = 4	$\overline{K} = \{1, 2\}$
$\delta_4(K_1 \cup K_2 - 4) = 60 - 50 = 10$	
$\Delta_4(K_1 \cup K_2 - 4) = 20 - 10 = 10$	
Logo O-Teste de 4 falhou	

Tabela 3.12: Aplicação do O-Teste

C-Teste para i = 3	$\overline{K} = \{1, 2\}$
$\delta_3(K_1) = 60 - 60 = 0$	
$\Delta_3(K_1) = 10 - 0 = 10$	
Logo C-Teste de 3 funcionou	
C-Teste para i = 4	$\overline{K} = \{1, 2\}$
$\delta_4(K_1) = 60 - 50 = 10$	
$\Delta_4(K_1) = 20 - 10 = 10$	
Logo C-Teste de 4 funcionou	

Tabela 3.13: Aplicação do C-Teste

Como $\forall i \in I - \overline{K}$ o C-Teste obteve sucesso, então o algoritmo conseguiu fechar o gap, ou seja, atingiu a solução ótima. Logo tem-se que $K_1 = \overline{K} = \{1, 2\}$, $K_0 = I - \overline{K} = \{3, 4\}$ e $K_2 = \emptyset$. •

3.4.3 Conclusão

Como conclusão de 3.4 podemos dizer que o algoritmo 1 apesar de verificar o caso $K_2 = \emptyset$ não o permite fazer *a priori*, ou seja, é preciso aplicar o algoritmo para saber se $K_2 = \emptyset$. O algoritmo 2 exige condições muito fortes para que ele consiga fazer $K_2 = \emptyset$, e da mesma forma que o 1 é necessário aplicar o algoritmo para saber se $K_2 = \emptyset$. Ambos não levam em conta os dados diretamente. Prosseguiremos procurando encontrar condições mais fracas para se identificar o fechamento do gap.

3.5 Casos especiais

Para encontrar classes de problemas onde a aplicação do O-Teste e do C-Teste façam $K_2 = \emptyset$, ou seja, a aplicação do algoritmo 1 da seção 3.4.1 feche o gap, precisa-se encontrar classes que não sofram dos problemas citados na seção 3.1, isto é, que não tenham *regiões de indecisão*. Os dados do problema representam um "emaranhado" de inter-relações, tendo no entanto o maior peso os c_{ij} e os f_i que aparecem diretamente na fórmula Δ_i. Por esta razão será considerado primeiramente o relacionamento entre os c_{ij} e os f_i na seção 3.5.1. A seguir, consideremos o relacionamento dos c_{ij} entre si na seção 3.5.2.

3.5.1 Classes de problemas considerando o relacionamento dos c_{ij} e b_j com os f_i

Consideraremos os casos mais significativos de relacionamentos dos c_{ij} e b_j com os f_i, com o objetivo de encontrar classes de problemas práticos em que o gap feche. Para que os resultados fiquem bem claros e para facilitar a apresentação, serão relaxadas as restrições de capacidade das facilidades, ou seja, os exemplos apresentados serão de problemas não capacitados.

A primeira relação a ser considerada é a $\{min\ f_i > max\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$. Mostraremos através de contra exemplo que esta relação não garante o fechamento do gap. É importante obsevar que esta relação favorece o sucesso do C-Teste. Quanto maior for a diferença entre o menor f_i e o maior c_{ij}, a favor do f_i, maior a probalidade de C-Teste obter sucesso. Isso porque o C-Teste tem sucesso quando $\Delta_i(K_1) \geq 0$ e $\Delta_i(K_1) = f_i - \delta_i(K_1)$, então quanto maiores forem os f_i em relação aos c_{ij} maiores as suas chances de superar os $\delta_i(K_1)\ \forall i \in K_2$.

Exemplo 3.5 *Contra exemplo para a relação* $\{min\ f_i > max\ c_{ij}b_j$
$,\ \forall i \in I\ e\ \forall j \in J\}$

Os dados relativos a $c_{ij}b_j$ e f_i são apresentados na tabela 3.14, onde $F_i\ \ i = 1,..,4$ representa as facilidades, os $C_j\ \ j = 1,...,6$ os clientes e $f_i\ \ i = 1,...,4$ representa os custos fixos. Por exemplo a linha 2 coluna 2 cujo valor é 20 representa o custo da facilidade F_1 atender toda a demanda de C_1. O problema será resolvido aplicando o algoritmo 1.

	C_1	C_2	C_3	C_4	C_5	C_6	f_i
F_1	20	20	18	10	10	20	31
F_2	10	10	10	30	30	10	31
F_3	11	11	30	30	30	5	31
F_4	5	5	30	30	30	11	31

Tabela 3.14: Dados do problema (F_1, F_4 - ótimo)

A tabela 3.15 apresenta somente as soluções dos Problemas de Transporte necessários para a realização dos testes. A tabela 3.16 mostra a realização do O-Teste $\forall i \in K_2$.

Problema de Transporte	**Solução**
w(F1,F2,F3,F4)	45
w(F1,F2,F3)	55
w(F2,F3,F4)	85
w(F1,F3,F4)	53
w(F1,F2,F4)	50
w(F1, F4)	59
w(F1, F2)	60
w (F1,F3)	65
w (F1)	98

Tabela 3.15: Resolução de alguns $w(K)$

Com a realização dos O-Testes obteve-se sucesso para $i = 1$, ficando com $K_0 = \emptyset$, $K_1 = \{1\}$ e $K_2 = \{2, 3, 4\}$. Como K_1 é viável e **Parada** é Falso realiza-se o C-Teste $\forall i \in K_2$. Os resultados dos C-Testes são mostrados na tabela 3.17.

Na realização dos C-Testes nenhum $i \in K_2$ obteve sucesso, como **Parada** é verdadeiro o algoritmo termina sua execução sem a obtenção da solução ótima. Tem-se então que a relação $\{min\ f_i > max\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$ não garante

O-Teste para i = 1	$K_0 = K_1 = \emptyset\ K_2 = I = \{1,2,3,4\}$
$\Delta_1 = -9$. Logo O-Teste de 1 funcionou	
O-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_2 = 23$. Logo O-Teste de 2 falhou	
O-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_3 = 26$. Logo O-Teste de 3 falhou	
O-Teste para i = 4	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_4 = 21$. Logo O-Teste de 4 falhou	

Tabela 3.16: Aplicação do O-Teste

C-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_2 = -7$. Logo C-Teste de 2 falhou	
C-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_3 = -2$. Logo C-Teste de 3 falhou	
C-Teste para i = 4	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_4 = -8$. Logo C-Teste de 4 falhou	

Tabela 3.17: Aplicação do C-Teste

o fechamento do gap. •

Seja R a relação $\{min\ f_i > max\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$. De R pode-se retirar que as seguintes relações também não garantem o fechamento do gap.

1. $R_1 \Rightarrow \{max\ f_i > max\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$
2. $R_2 \Rightarrow \{min\ f_i > min\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$
3. $R_3 \Rightarrow \{max\ f_i > min\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$
4. $R_4 \Rightarrow \{min\ f_i > max\ c_{ij},\ \forall i \in I\ e\ \forall j \in J\}$
5. $R_5 \Rightarrow \{min\ f_i > max\ b_j,\ \forall i \in I\ e\ \forall j \in J\}$

Para verificar 1, 2 e 3 basta ver que R_1, R_2, R_3 se deixam reduzir a R. Logo o exemplo 3.5 é contra exemplo de fechamento de gap para R_1, R_2 e R_3. Outro resultado que pode ser obtido através do exemplo 3.5 é que a relação R_4 não garante que $K_2 = \emptyset$, para isso basta colocar a demanda como sendo unitária. O mesmo resultado pode ser obtido da relação R_5 considerando $c_{ij} = 1$, $\forall i \in I$ e $\forall j \in J$.

Pode-se extrair facilmente que a relação $S = \{min\ f_i > \lambda max\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$, onde λ é uma constante qualquer maior que zero, não garante o fechamento do gap. Para verificar esta relação, utilizando os dados do exemplo 3.5, basta dividir os $c_{ij}b_j$, $\forall i \in I$ e $\forall j \in J$ por λ, e encontrar um λ em que o gap não feche. Para $\lambda = 1$ é o próprio exemplo 3.5, para $\lambda = 5$ nenhum O-Teste terá sucesso, obviamente nenhum C-Teste terá também. A relação $T = \{\lambda\ min\ f_i >$

$max\ c_{ij}b_j,\ \forall i \in I\ e\ \forall j \in J\}$, onde λ é uma constante qualquer maior que zero, não garante o fechamento do gap também. Para verificar esta relação, utilizando os dados do exemplo 3.5, basta dividir os f_i, $\forall i \in I$ por λ, e encontrar um λ em que o gap não feche. Para $\lambda = 1$ é o próprio exemplo 3.5, para $\lambda = 3$ somente o O-Teste para $i = 1$ terá sucesso e nenhum C-Teste terá sucesso $\forall i \in K_2$. Da mesma forma que obteve-se as relações R_1, R_2, R_3, R_4 e R_5 de R pode-se obter as relações S_1, S_2, S_3, S_4 e S_5 de S e as as relações T_1, T_2, T_3, T_4 e T_5 de T.

Agora mostraremos através de um contra exemplo que a relação $\{min\ c_{ij}b_j > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$ não garante o fechamento do gap. É importante observar que esta relação favorece o sucesso do O-Teste. Quanto maior for a diferença entre o menor c_{ij} e o maior f_i, a favor do c_{ij}, maior a probalidade de O-Teste obter sucesso. Isso porque o O-Teste tem sucesso quando $\Delta_i(K_1 \cup K_2 - i) \leq 0$ e $\Delta_i(K_1 \cup K_2 - i) = f_i - \delta_i(K_1 \cup K_2 - i)$, então quanto maiores forem os c_{ij} em relação aos f_i, maiores as chances de $\delta_i(K_1 \cup K_2 - i)$ superar os f_i $\forall i \in K_2$.

Exemplo 3.6 *Contra exemplo para a relação* $\{min\ c_{ij}b_j > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$

Os dados relativos a $c_{ij}b_j$ e f_i são apresentados na tabela 3.18, onde $F_i\ \ i = 1,..,4$ representa as facilidades, os $C_j\ \ j = 1,...,6$ os clientes e $f_i\ \ i = 1,...,4$ representa os custos fixos. Por exemplo a linha 2 coluna 2 cujo valor é 20 representa o custo da facilidade F_1 atender toda a demanda de C_1. O problema será resolvido aplicando o algoritmo 1.

	C_1	C_2	C_3	C_4	C_5	C_6	f_i
F_1	20	20	18	10	10	20	6
F_2	8	8	10	20	20	9	6
F_3	30	30	30	30	30	7	6
F_4	7	7	11	30	30	30	6

Tabela 3.18: Dados do problema (F_1, F_2 - ótimo)

A tabela 3.19 apresenta somente as soluções dos Problemas de Transporte necessários para a realização dos testes. A tabela 3.20 mostra a realização do O-Teste para $\forall i \in K_2$.

Problema de Transporte	**Solução**
w(F1,F2,F3,F4)	51
w(F1,F2,F3)	53
w(F2,F3,F4)	71
w(F1,F3,F4)	52
w(F1,F2,F4)	53
w(F1, F4)	65
w(F1, F2)	55
w (F1,F3)	87
w (F1)	98

Tabela 3.19: Resolução de alguns $w(K)$

Com a realização dos O-Testes obteve-se sucesso para $i = 1$, ficando com $K_0 = \emptyset$, $K_1 = \{1\}$ e $K_2 = \{2, 3, 4\}$. Como K_1 é viável e **Parada** é Falso realiza-se C-Teste $\forall i \in K_2$. Os resultados dos C-Testes são mostrados na tabela 3.21.

Na realização dos C-Testes nenhum $i \in K_2$ obteve sucesso, como **Parada** é verdadeiro o algoritmo termina sua execução sem a obtenção da solução ótima. Tem-se então que a relação $\{min\ c_{ij}b_j > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$ não garante

O-Teste para i = 1	$K_0 = K_1 = \emptyset\ K_2 = I = \{1,2,3,4\}$
$\Delta_1 = -14$. Logo O-Teste de 1 funcionou	
O-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_2 = 5$. Logo O-Teste de 2 falhou	
O-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_3 = 4$. Logo O-Teste de 3 falhou	
O-Teste para i = 4	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_4 = 4$. Logo O-Teste de 4 falhou	

Tabela 3.20: Aplicação do O-Teste

C-Teste para i = 2	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_2 = -37$. Logo C-Teste de 2 falhou	
C-Teste para i = 3	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_3 = -5$. Logo C-Teste de 3 falhou	
C-Teste para i = 4	$K_0 = \emptyset\ K_1 = \{1\}\ K_2 = \{2,3,4\}$
$\Delta_4 = -27$. Logo C-Teste de 4 falhou	

Tabela 3.21: Aplicação do C-Teste

o fechamento do gap. •

Seja R' a relação $\{min\ c_{ij}b_j > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$. De R' pode-se retirar que as seguintes relações também não garantem o fechamento do gap.

1. $R'_1 \Rightarrow \{max\ c_{ij}b_j > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$
2. $R'_2 \Rightarrow \{min\ c_{ij}b_j > min\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$
3. $R'_3 \Rightarrow \{max\ c_{ij}b_j > min\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$
4. $R'_4 \Rightarrow \{min\ c_{ij} > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$
5. $R'_5 \Rightarrow \{min\ b_j > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$

Para verificar 1, 2 e 3 basta ver que R'_1, R'_2 e R'_3 se deixam reduzir a R'. Logo o exemplo 3.6 é contra exemplo de fechamento de gap para R'_1, R'_2 e R'_3. Outro resultado que pode ser obtido através do exemplo 3.6 é que a relação R'_4 não garante que $K_2 = \emptyset$, para isso basta colocar a demanda como sendo unitária. O mesmo resultado pode ser obtido da relação R'_5 considerando $c_{ij} = 1$, $\forall i \in I$ e $\forall j \in J$.

Pode-se extrair facilmente que a relação $S' = \{\lambda min\ c_{ij}b_j > max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$, onde λ é uma constante qualquer maior que zero, não garante o fechamento do gap. Para verificar esta relação, utilizando os dados de exemplo 3.6, basta dividir os $c_{ij}b_j$, $\forall i \in I$ e $\forall j \in J$ por λ, e encontrar um λ em que o gap não feche. Para $\lambda = 1$ é o próprio exemplo 3.6, para $\lambda = 8$ nenhum O-Teste terá sucesso, obviamente nenhum C-Teste terá também. A relação $T' = \{min\ c_{ij}b_j >$

$\lambda max\ f_i,\ \forall i \in I\ e\ \forall j \in J\}$, onde λ é uma constante qualquer maior que zero, não garante o fechamento do gap também. Para verificar esta relação, utilizando os dados do exemplo 3.6, basta dividir os f_i, $\forall i \in I$ por λ, e encontrar um λ em que o gap não feche. Para $\lambda = 1$ é o próprio exemplo 3.6, para $\lambda = 2$ somente o O-Teste para $i = 1$ terá sucesso, e nenhum C-Teste terá sucesso $\forall i \in K_2$. Da mesma forma que obteve-se as relações R'_1, R'_2, R'_3, R'_4 e R'_5 de R' pode-se obter as relações S'_1, S'_2, S'_3, S'_4 e S'_5 de S' e as as relações T'_1, T'_2, T'_3, T'_4 e T'_5 de T'.

Com o que foi apresentado tem-se que os relacionamentos dos c_{ij} e b_j com os f_i mais significativos não garantem o fechamento do gap. Isto era de se esperar, pois para alocar facilidades utilizando os O-Testes o grande problema é que uma indefinição grande demais torna os custos de transporte insensíveis à retirada de uma facilidade. No caso extremo esta sensibilidade será zero e poderia ser gerada por uma distribuição espacial(por exemplo junto de cada local candidato teria um outro que pode-se alocar tão perto quanto se queira). Assim esta situação independeria dos c_{ij} e dos b_j. O mesmo se pode dizer para os C-Testes.

3.5.2 Classes de problemas considerando o relacionamento entre os cij

Em função do que foi visto nas seções 3.1 e 3.2 parece que o único meio de encontrar classes de problemas relevantes em que o gap feche através dos testes de redução, é encontrar classes de problemas que sejam espacialmente bem distribuidas em função dos c_{ij}, tentando eliminar as *regiões de indecisão.*

Apresentaremos duas classes de problemas espacialmente bem distribuidas. As outras classes podem ser obtidas com uma combinação das classes que serão apresentadas. Além das restrições sobre os c_{ij} impostas para obtenção de uma situação espacialmente bem definida serão colocadas outras restrições sobre os dados do problema,

A primeira classe de problemas, a classe $C1$, é mostrada na figura 3.3, onde os circulos hachurados representam os clientes e os brancos as facilidades. Esta distribuição espacial é obtida em função dos c_{ij}. Além das restrições sobre os c_{ij}, visiveis na figura, é imposto que todos os f_i e a_i sejam iguais.

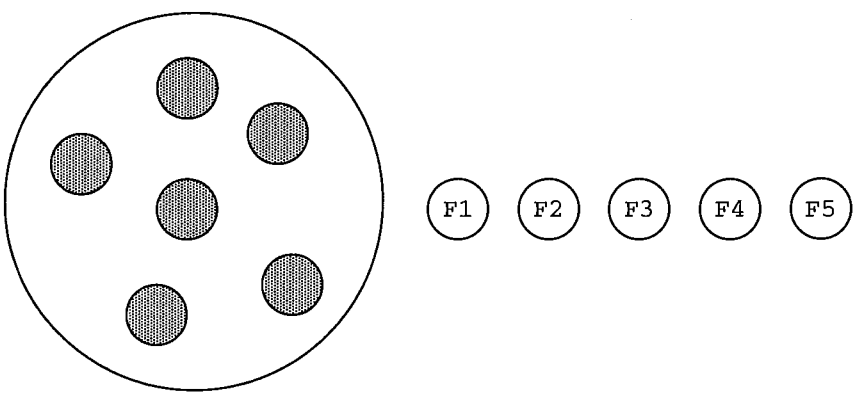


Figura 3.3: Classe C_1

Então as condições impostas a classe $C1$ são: com relação aos f_i todos devem ter o mesmo valor, aos a_i também todos devem ter o mesmo valor, e com relação aos c_{ij} tem-se $c_{1j} < c_{2j} < ... < c_{nj}\ \forall j \in J$.

A solução da classe $C1$ é obvia, e para resolver o problema não há necessidade da aplicação dos testes de redução. A solução consiste em: ordenar as facilidades pelos seus custos variáveis, já que esta é uma condição imposta pela classe $C1$,

e abrir as p primeiras facilidades que façam o problema viável. Apesar disto, a aplicação dos testes de redução não garante a obtenção de uma solução ótima (isto é, o fechamento do gap) para este caso. É o que será visto no exemplo 3.7.

Exemplo 3.7 *Exemplo da classe* C_1.

A figura 3.4 mostra um exemplo da classe de problemas $C1$, os dados são apresentados juntamente com a figura para facilitar a visualização, onde F_i $i = 1, ..., 5$ representa as facilidades, C_j $j = 1, ..., 6$ os clientes, f_i $i = 1, ..., 5$ os custos fixos de instalação das facilidades, a_i $i = 1, ...5$ as capacidades, b_j $= 1, ..., 6$ a demanda de cada cliente e c_{ij} $i = 1, ..., 5$ e $j = 1, ..., 6$ o custo de transporte de i para j de uma unidade. Para facilitar a apresentação dos dados colocou-se que o custo variável de uma facilidade para todos os clientes são iguais.

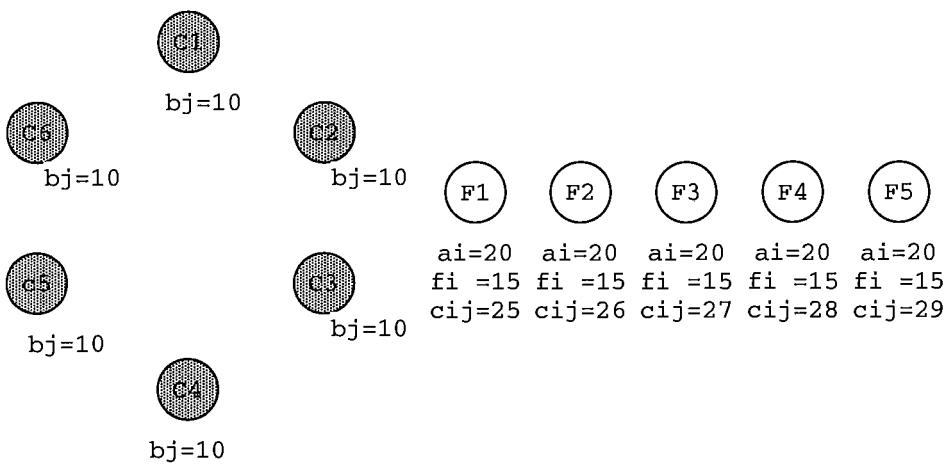


Figura 3.4: Dados do problema (F_1, F_2, F_3 - ótimo)

Será aplicado o algoritmo 1 para resolver o problema. A tabela 3.22 apresenta somente as soluções dos Problemas de Transporte necessários para a realização

dos testes. A tabela 3.23 mostra a realização do O-Teste $\forall i \in K_2$.

Problema de Transporte	**Solução**
w(F1,F2,F3,F4,F5)	156
w(F2,F3,F4,F5)	162
w(F1,F3,F4,F5)	160
w(F1,F2,F4,F5)	158
w(F1,F2,F3,F5)	156
w(F1,F2,F3,F4)	156

Tabela 3.22: Resolução de alguns $w(K)$

O-Teste para i = 1	$K_0 = K_1 = \emptyset\ K_2 = I = \{1,2,3,4,5\}$
$\Delta_1 = 9$. Logo O-Teste de 1 falhou	
O-Teste para i = 2	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5\}$
$\Delta_2 = 11$. Logo O-Teste de 2 falhou	
O-Teste para i = 3	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5\}$
$\Delta_3 = 13$. Logo O-Teste de 3 falhou	
O-Teste para i = 4	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5\}$
$\Delta_4 = 15$. Logo O-Teste de 4 falhou	
O-Teste para i = 5	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5\}$
$\Delta_5 = 15$. Logo O-Teste de 5 falhou	

Tabela 3.23: Aplicação do O-Teste

Os testes realizados na tabela 3.23 mostram que nenhum dos O-Testes obteve sucesso, indicando assim que não pode-se fazer nenhum C-Teste, pois K_1 não é viável e **Parada** é verdadeiro, conseguentemente o algoritmo 1 termina. Logo os testes de redução não garantem o fechamento do gap para os problemas da classe C1. •

A segunda classe de problemas, a classe $C2$, espacialmente bem distribuida é apresentada graficamente na figura 3.5. Determina-se para $C2$ que cada facilidade

que tem um conjunto de clientes próximos tenha capacidade de atende-los, e que todas as facilidades tenham custo fixo iguais. Será verificado através de um contra exemplo que não é possível garantir o fechamento do gap aplicando-se o O-Teste e o C-Teste na classe $C2$.

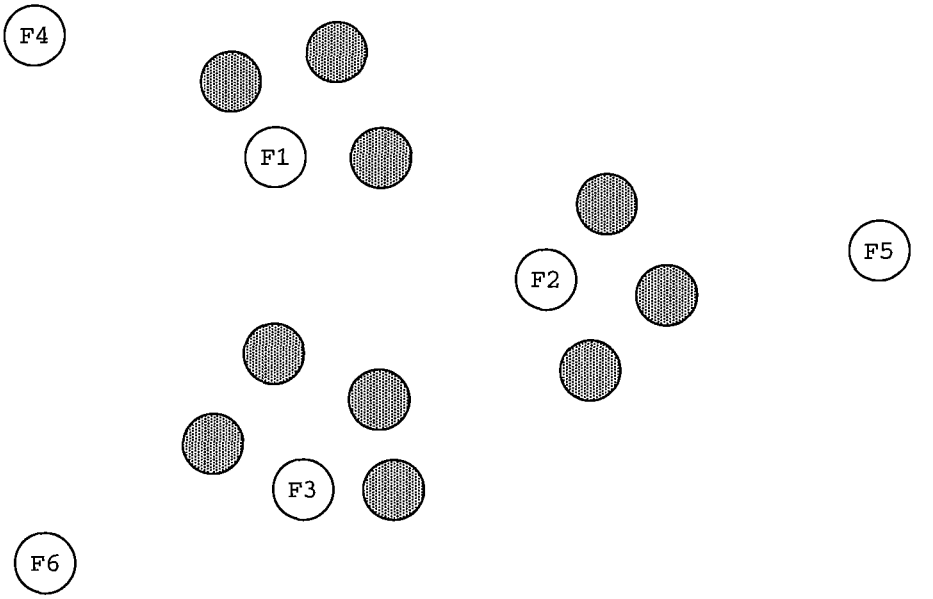


Figura 3.5: Classe C_2

Exemplo 3.8 *Exemplo da classe C_2.*

A figura 3.6 mostra um exemplo da classe de problemas $C2$. O problema não tem restrições de capacidade, por isso ao relacionar o custo de transporte de uma facilidade para um cliente coloca-se o custo total (custo correspondente ao atendimento de toda a demanda) ao invés do custo unitário.

Será aplicado o algoritmo 1 para resolver o problema. A tabela 3.24 apresenta somente as soluções dos Problemas de Transporte necessários para a realização

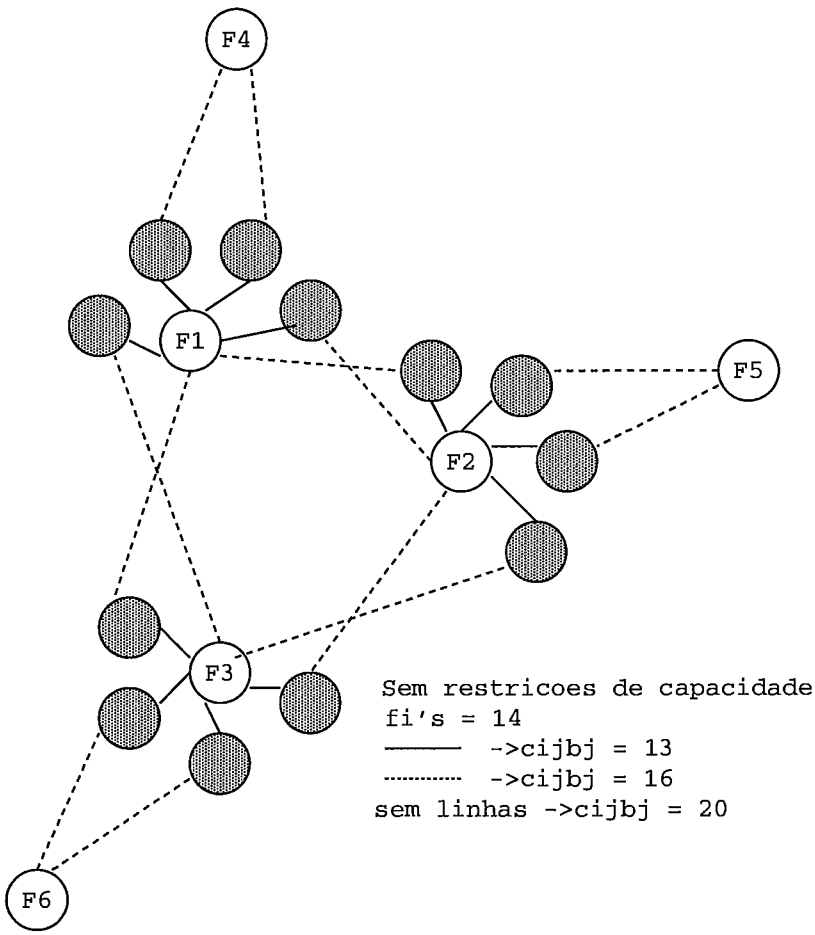


Figura 3.6: Dados do problema (F_1, F_2, F_3 - ótimo)

dos testes. A tabela 3.25 mostra a realização do O-Teste $\forall i \in K_2$.

Problema de Transporte	**Solução**
w(F1,F2,F3,F4,F5,F6)	156
w(F2,F3,F4,F5,F6)	168
w(F1,F3,F4,F5,F6)	168
w(F1,F2,F4,F5,F6)	168
w(F1,F2,F3,F5,F6)	156
w(F1,F2,F3,F4,F6)	156
w(F1,F2,F3,F4,F5)	156

Tabela 3.24: Resolução de alguns $w(K)$

O-Teste para i = 1	$K_0 = K_1 = \emptyset\ K_2 = I = \{1,2,3,4,5,6\}$
$\Delta_1 = 2$. Logo O-Teste de 1 falhou	
O-Teste para i = 2	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5,6\}$
$\Delta_2 = 2$. Logo O-Teste de 2 falhou	
O-Teste para i = 3	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5,6\}$
$\Delta_3 = 2$. Logo O-Teste de 3 falhou	
O-Teste para i = 4	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5,6\}$
$\Delta_4 = 14$. Logo O-Teste de 4 falhou	
O-Teste para i = 5	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5,6\}$
$\Delta_5 = 14$. Logo O-Teste de 5 falhou	
O-Teste para i = 6	$K_0 = K_1 = \emptyset\ K_2 = \{1,2,3,4,5,6\}$
$\Delta_6 = 14$. Logo O-Teste de 6 falhou	

Tabela 3.25: Aplicação do O-Teste

Os testes realizados na tabela 3.25 mostram que nenhum dos O-Testes obteve sucesso, indicando assim que não pode-se fazer nenhum C-Teste, pois K_1 não é viável e **Parada** é verdadeiro, conseguentemente o algoritmo 1 termina. Logo não pode-se garantir que os testes de redução conseguem fechar o gap para os problemas da classe $C2$. •

Como visto, as duas classes $C1$ e $C2$ são espacialmente definidas de modo a induzir a uma solução ótima, e no entanto nem esta distristribuição espacial garantiu o fechamento do gap, com a aplicação dos testes. Para que $K_2 = \emptyset$ as distribuições teriam que ser bem mais expressivas, o que caracteriza uma sub classe pequena das classes que já são bem restritas, e estas sub classes seriam resolvidas de forma mais eficiente sem a necessidade da aplicação dos testes de redução.

Nas seções 3.5.1 e 3.5.2 examinamos uma série de casos especiais (classes de problemas) que, em principio, devem levar a um desempenho favorável(bom) dos O e C-Testes. Atráves de contra exemplos foi no entanto possível verificar que para nenhuma dessas classes pode-se garantir o fechamento do gap, isto é, a obtenção de uma solução ótima através da aplicação dos testes. O conhecimento adquirido através deste estudo pode, no entanto, ser de grande valor, para o desenvolvimento de um algoritmo *Branch and Bound*, a ser visto no próximo capítulo. Lá, conforme veremos, decisões "temporarias" (provisórias) de abrir e fechar facilidades (ramificação da árvore de busca), potencializam os testes de redução. E o conhecimento adquirido nestas seções pode ser importante para definir critérios para estas ramificações.

Capítulo 4

Um algoritmo Branch and Bound

O objetivo deste capítulo é o desenvolvimento de um algoritmo exato para a resolução do PLC. A ideia é fazer um *Branch and Bound* especializado para o problema, que forneça uma solução próxima da ótima nas primeiras buscas em profundidade e consiga medir a qualidade da melhor solução encontrada conforme o método for realizando a exploração na árvore de busca. A especialização está na tentativa de fazer uma poda eficiente, para isso, procura-se fazer um processamento rigoroso em cada nó da árvore de busca. Na apresentação do método, será dada mais ênfase no seu funcionamento.

Os testes de redução abrem/fecham facilidades *a priori*, contudo para o bom desempenho necessitam da abertura/fechamento das facilidades *a priori*, pois para K_0 pequeno tem-se baixa sensibilidade no custo de transporte de $(K_1 \cup K_2)$ para i, ocasionado a falha do O-Teste, e para K_1 pequeno tem-se grande sensibilidade no custo de transporte de K_1 para i, ocasionando a falha do C-Teste. Isso pode ocassionar um ciclo vicioso, o qual pode ser quebrado utilizando

um método *Branch and Bound*, conforme ilustra a figura 4.1. A partir disso desenvolvemos um método *Branch and Bound* para encontrar a solução ótima de um PLC, que utiliza os testes de redução para reduzir a dimensão do problema, relaxação lagrangeana para obtenção do limite inferior e heurísticas como regras de ramificação.

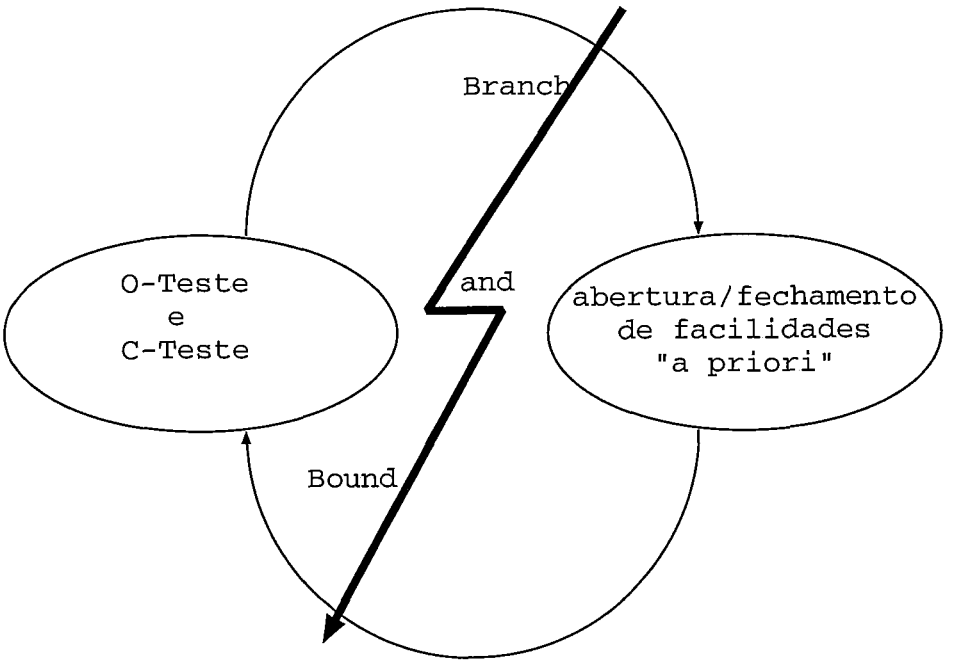


Figura 4.1: Quebra do possível ciclo vicioso

4.1 Definições

Definição 4.1 *Nó:*

representa uma alocação de elementos aos conjuntos K_0, K_1 e K_2, definidos na seção 2.1. Por abuso de linguagem nos referenciamos a um nó como um problema ou sub-problema do problema original, visto que, na tentativa de fixar valores

para os elementos do conjunto K_2 frequentemente resolvemos um PLC associado a estas facilidades.

Definição 4.2 *Árvore de Busca:*

Árvore de busca é um conjunto de nós que represeta todas as combinações possíveis de elementos nos conjuntos K_0, K_1 e K_2. Então seja n o tamanho de I a árvore de busca terá $2^{n+1} - 1$ nós.

Definição 4.3 *Folha:*

Uma folha é um nó cujo o conjunto $K_2 = \emptyset$.

Definição 4.4 *Solução Viável:*

Uma solução viável para o PLC é uma folha que contém K_1 viável. Com isso uma solução viável fornece $K_0 = I - K_1$, e o valor da solução é dada por $w(K_1) + \sum_{i \in K_1} f_i$.

Definição 4.5 *Ramificação (branching):*

Ramificação na árvore é dada da seguinte maneira: Seja v um nó não folha da árvore de busca, K_0, K_1 e K_2 os conjuntos de facilidades associados a v, e i uma facilidade pertencente a K_2, então pode-se ter duas ramificação conforme

ilustrado na figura 4.2. Uma a esquerda que faz $K_0 = K_0 \cup \{i\}$ e $K_2 = K_2 - \{i\}$ e outra a direita que faz $K_1 = K_1 \cup \{i\}$ e $K_2 = K_2 - \{i\}$, gerando respectivamente os nós v' e v''. Também chama-se estas ramificações de sub-árvore esquerda e sub-árvore direita de v, onde v' e v'' são respectivamente as raizes das sub-árvores. O que ocorre na ramificação é a divisão do problema em dois, onde em um deles considera-se uma determinada facilidade como sendo fechada e no outro como sendo aberta. A solução ótima do problema associado ao nó v é a melhor das soluções ótimas dos dois sub-problemas gerados na ramificação.

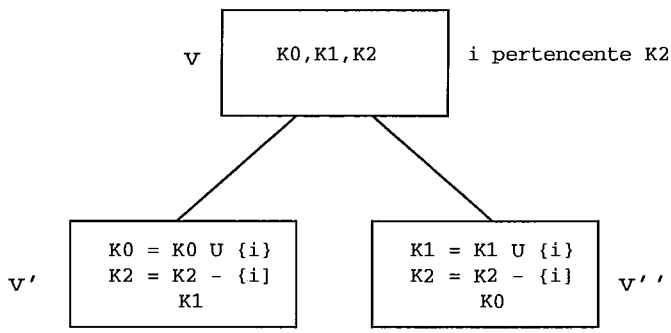


Figura 4.2: Ramificação do nó v em v' e v''

Definição 4.6 *Pai e Filho:*

Um nó v é pai de um nó u se u é uma ramificação a esquerda ou direita de v. Se v é pai de u então u é filho de v.

Definição 4.7 *Raiz:*

Um nó é raiz quando ele não possui pai.

Definição 4.8 *Subir na Árvore:*

Subir na árvore é o ato de estar em um determinado nó v e ir para o pai de v. Não é possível subir se o nó for raiz.

Definição 4.9 *Descer na Árvore:*

Descer na árvore é o ato de estar em um determinado nó v e ir para um de seus filhos. Não é possível descer caso v seja folha.

Definição 4.10 *Nó Descendente:*

Um nó u é descendente do nó v se subindo na árvore a partir de u, encontra-se v.

Definição 4.11 *Soluções Descendentes:*

Seja v um nó da árvore de busca, uma solução descendente de v é uma solução viável associada a um nó descendente de v.

Definição 4.12 *Limite Superior de um PLC:*

O limite superior de um PLC é uma solução viável do PLC. O melhor limite superior encontrado durante a exploração da árvore de busca é denotado por LS_{min}.

Definição 4.13 *Limite Inferior de um PLC:*

O limite inferior de um PLC é um valor garantidamente menor ou igual ao valor da função objetivo associado à solução ótima. O melhor limite inferior encontrado durante a exploração da árvore de busca é denotado por LI_{max}.

Definição 4.14 *Limite Inferior de um Nó:*

O limite inferior de um nó v é um valor garantidamente menor ou igual ao valor objetivo associado a qualquer solução descendente de v. O limite inferior de v é denotado por LI_v. $LI_{v'}$ denota o limite inferior para a ramificação a esquerda de v, e $LI_{v''}$ denota o limite inferior para a ramificação a direita de v.

4.2 Poda

A exploração completa da árvore de busca é inviavel do ponto de vista computacional para problemas de grandes porte. Supondo que o conjunto I tenha tamanho 50, isso fornece $2^{51} - 1$ nós na árvore de busca. Caso o tempo computacional para exploração de um nó seja de 0.00001 segundo, demoraria-se aproximadamente 7,2 séculos para que se consiga explorar toda a árvore. Com isso deve-se encontrar uma maneira que tente explorar um número pequeno de nós fornecendo a solução ótima para o problema. A maneira para não se explorar todas as combinações possíveis é fazendo podas, indicando que parte da árvore

de busca não precisa ser explorada. O método desenvolvido fornece 2 maneiras de podas.

1. Limite Inferior: Seja v o nó a ser explorado, se $LI_v \geq LS_{max}$ então pode-se efetuar uma poda na árvore, pois não existem soluções descendentes de v melhores que a melhor solução encontrada até então. Isto indica que não é necessário explorar as soluções descendentes de v.

2. Testes de Redução: é a principal contribuição dos testes de redução ao *Branch and Bound* na tentativa de enumerar pouco (explorar um número pequeno de nós). Consiste da aplicação do O-Teste e C-Teste vistos na seção 2.1. Os testes alocam valores em K_0 e K_1, indicando que não há necessidade de considerar o valor alternativo. Com isto poupa-se uma ramificação. Isto é, se o O-Teste decide que $i \in K_1$ então evidentemente não é preciso considerar o ramo correspontente a $i \in K_0$.

A figura 4.3 mostra como funcionam as podas. A tesoura 1 representa uma poda gerada pelo sucesso do C-Teste, a tesoura 2 representa uma poda gerada pelo sucesso do O-Teste e a tesoura 3 representa uma poda gerado pelo uso do limite inferior. As linhas pontilhadas representam sub-árvores que devem ser exploradas. Note que a poda realizada pelos testes é feita somente em uma sub-árvore, enquanto que a poda realizada pelo limite inferior é feita nas duas sub-árvores.

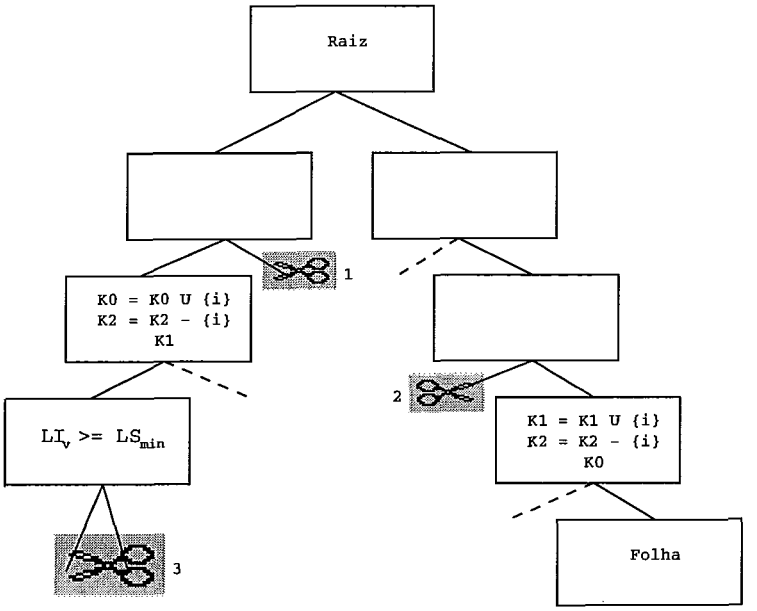


Figura 4.3: Funcionamento das podas

4.3 *Backtracking*, Condições de Parada e Busca na Árvore

O *Backtracking* em um nó v indica que não existe mais necessidade de continuar explorando as soluções descendentes de v, fornecendo a informação que a solução do sub-problema que estava sendo resolvido já foi encontrada, ou sabe-se que ele fornecerá uma solução pior ou igual a uma já encontrada. Na exploração da árvore de busca pode ocorrer *Backtracking* em um nó v em 3 situações:

1. v é uma folha, ou seja, quando o conjunto $K_2 = \emptyset$, indicando que não exitem mais soluções descendentes de v para serem exploradas.

2. $LI_v \geq LS_{min}$, ou seja, não é mais necessário explorar as soluções descendentes de v, porque elas fornecem um valor maior ou igual do que o melhor

valor encontrado.

3. todos os nós descendentes de v já foram explorados, ou seja, quando todas as soluções descendentes de v já foram encontradas de forma explícita (folhas) ou implícita (podas).

Backtracking é o ato de subir na árvore e encontrar uma sub-árvore que ainda não foi explorada, ou seja, encontrar um sub-problema que ainda não foi resolvido. A condição de parada do método é quando ocorrer *backtracking* no nó raiz, ou seja, todas as soluções descendentes do problema original foram encontradas de forma explícita ou implícita.

O método utiliza busca em profundidade para obtenção da solução ótima. Busca em profundidade é o ato de descer de um nó v até um nó que forneça uma condição de *backtracking*, ou seja, quando ocorrer uma ramificação, opta-se por um lado e segue-se até que seja realizado *backtracking*, para depois seguir pelo outro lado. Realiza-se busca em profundidade porque no método uma solução viável somente é encontrada em um nó folha. Ora, são as soluções viáveis que fornecem limites superiores. Para que se possa realizar podas através do limite inferior são necessários limites superiores cada vez melhores, ou seja, é necessário explorar o máximo número de folhas possível. No método desenvolvido o *backtracking* sobe na árvore até encontrar a primeira sub-árvore que ainda não foi explorada, isto é, a última ramifição ocorrida que não foi completamente explorada. Encontrada esta sub-árvore realiza-se a busca em profundidade novamente.

Para representar a busca pela solução ótima na árvore será utilizado uma estrutura de lista, a qual representará uma sequência de exploração na árvore. A manipulação dos nós será na ordem LIFO (*last in first out*), ou seja, o último a entrar será o primeiro a sair. Isso caracteriza uma pilha, a qual será denominada de π. A pilha conterá os sub-problemas que precisam ser resolvidos para que se possa encontrar a solução ótima. Com isso, cada nó de π será um sub-problema do PLC original e terá os conjuntos K_0, K_1 e K_2 e o valor LI_v associados a ele, ou seja, será a raiz de uma sub-árvore que deve ser explorada. Então quando houver uma ramificação deve-se inserir em π, v' e v'' na ordem determinada pelas regras de *branching*, as quais serão vistas na seção 4.4. A condição de parada do algoritmo é quando $\pi = \emptyset$. A pilha π é inicializada com um nó v contendo $K_0 = K_1 = \emptyset$ e $K_2 = I$ e $LI_v = -\infty$.

4.4 Regras de *Branching*

Ocorre um *branching* em um nó v quando $K_2 \neq \emptyset$ e nenhuma poda puder ser realizada. Isso quer dizer que foi detectada pelo menos uma *região de indecisão* e é necessário decidir temporariamente o valor de alguma facilidade para tentar eliminar a indecisão de uma região. A questão aqui está em qual facilidade $i \in K_2$ escolher para alocar nos conjuntos K_0 e K_1 e qual ramificação deve ser explorada primeiro, ou seja, qual nó, v' ou v'', deve ser inserido por último em π. Uma boa escolha da facilidade a ser fixada e da ramificação que será explorada primeiro pode aumentar as chances de sucesso dos testes além de propiciar a obtenção de

novos/melhores limites superiores. Ambos os casos implicam em uma melhora de eficiência das podas, pois o sucesso dos testes é equivalente, como visto, a uma poda e novos limites superiores aumentam as chances de podas através do limite inferior.

Antes de apresentar as regras de *branching* do método, será mostrado a O-Heurística e a C-Heurística apresentadas em Jacobsen [20].

- O-Heurística: Se $\Delta_i(K_1) < 0$ então $y_i = 1$
- C-Heurística: Se $\Delta_i(K_1 \cup K_2 - i) > 0$ então $y_i = 0$

As heurísticas são utilizadas para as facilidades $i \in K_2$ que não mais satisfazem as condições do O- e C-Teste. Neste caso tem-se $\Delta_i(K_1) < 0 < \Delta_i(K_1 \cup K_2 - i)$ indicando que as regras permitem tanto $y_i = 1$ como $y_i = 0$. Para escapar deste dilema Jacobsen [20] sugere tomar $min\ \Delta_i(K_1)$ ou $max\ \Delta_i(K_1 \cup K_2 - i)$.

No método desenvolvido quando K_1 é viável criou-se uma heurística, para obtenção da facilidade a ser fixada, que leva em conta tanto a O-Heurística como a C-Heurística, ou seja, procura-se fazer um balanço entre as duas heurísticas. Além disso leva-se em conta a capacidade de i, obtendo-se assim uma medida por unidade de capacidade. A facilidade a ser fixada é a facilidade cujo o índice fornece o valor da expressão $max_{i \in K_2}\ abs(\frac{\Delta_i(K_1 \cup K_2 - i) + \Delta_i(K_1)}{a_i})$. Se $\Delta_i(K_1 \cup K_2 - i) + \Delta_i(K_1) > 0$ então explora-se primeiro v' $(K_0 = K_0 \cup \{i\})$, ou seja, primeiro insere-se v'' e depois v' em π. Caso contrário $\Delta_i(K_1 \cup K_2 - i) + \Delta_i(K_1) \leq 0$ explora-se primeiro v''. Esta heurística será referenciada por **O/C-Heurística**.

A **O/C-Heurística** é um melhoramento da O- e C-Heurística e foi desenvolvida para suprir a falha dessas. Na O-Heurística tomando-se o $min\ \Delta_i(K_1)$ não pode-se garantir que $|\Delta_i(K_1)| < |\Delta_i(K_1 \cup K_2 - i)|$. Neste caso parece razoável que a facilidade i seja fechada em vez de ser aberta como a O-Heurística propõe. Seja $min\ \Delta_i(K_1) = -5$ então, conforme mostra a figura 4.4, a facilidade $i = 1$ está muito mais próxima de ser fechada do que aberta. Se no entanto fosse aplicada a O-Heurística a facilidade 1 seria aberta. Na C-Heurística pode ocorre uma falha parecida. Quando se toma o $max\ \Delta_i(K_1 \cup K_2 - i)$ pode ser que $|\Delta_i(K_1 \cup K_2 - i)| < |\Delta_i(K_1)|$, indicando que é melhor a facilidade i ser aberta do que fechada conforme estabelece a C-Heurística. Seja $max\ \Delta_i(K_1 \cup K_2 - i) = 5$ então, conforme mostra a figura 4.5, a facilidade $i = 1$ está muito mais próxima de ser aberta do que fechada. Se fosse aplicada a C-Heurística a facilidade 1, no entanto, seria fechada. Usando a **O/C-Heurística** procura-se a facilidade i com maior diferença em valor absoluto entre $|\Delta_i(K_1 \cup K_2 - i)|$ e $|\Delta_i(K_1)|$, pois é esta a facilidade que tem mais claramente definida a sua tendência a ser aberta/fechada. Caso para esta facilidade $|\Delta_i(K_1 \cup K_2 - i)| > |\Delta_i(K_1)|$ então a facilidade i está mais próxima da região em que as facilidades são fechadas, logo a prioridade é considerar o fechamento de i. Caso, ao contrário, $|\Delta_i(K_1 \cup K_2 - i)| < |\Delta_i(K_1)|$ então a prioridade é para abertura de i. Cabe lembrar que estamos em uma situação em que os O- e C-teste não se aplicam mais, isto é, $\Delta_i(K_1) < 0 < \Delta_i(K_1 \cup K_2 - i)$, ou seja, não mais estamos na região de abertura/fechamento, e portanto o que fazemos é testar a maior proximidade de uma destas regiões. Por exemplo na figura 4.6 a facilidade 1 fornece o valor de $max\ \Delta_i(K_1 \cup K_2 - i)$ e a facilidade 2

o *min* $\Delta_i(K_1)$. Contudo é a facilidade 3 a mais propícia a ser fechada, pois é a facilidade 3 que tem mais claramente definido o balanço entre as tendências a ser fechada e aberta. No caso, o balanço tende para o fechamento da facilidade 3.

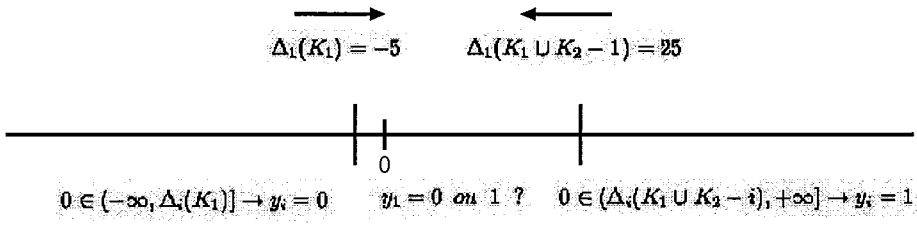


Figura 4.4: Funcionamento não adquado da O-Heurísticas

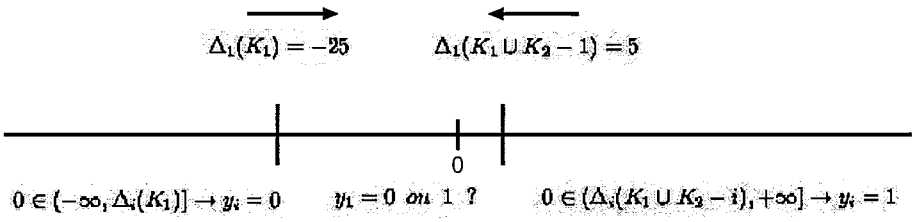


Figura 4.5: Funcionamento não adquado da C-Heurísticas

Pode ser fornecida uma interpretação para **O/C-Heurística** que justifica a sua aplicação de forma mais precisa. $\delta_i(K_1 \cup K_2 - i)$ representa o mínimo ganho(sensibilidade) dos custos de transporte e $\delta_i(K_1)$ o máximo ganho(sensibilidade) se a facilidade i for aberta. Quando o custo fixo for muito pequeno, ou seja, menor que o mínimo ganho dos custos de transporte, a facilidade será aberta (O-Teste). Neste caso o custo fixo fica à esquerda de $\delta_i(K_1 \cup K_2 - i)$, conforme mostra a figura 4.7. Quando o custo fixo for maior que o máximo ganho dos custos de transporte a facilidade será fechada (C-Teste). Neste caso os custo fixo fica à direita de $\delta_i(K_1)$, conforme mostra a figura 4.7. Quando o custo fixo ficar entre $\delta_i(K_1 \cup K_2 - i)$ e $\delta_i(K_1)$ nada pode-se garantir. É neste caso que se aplica a **O/C-Heurística**. Caso f_i esteja mais próximo de $\delta_i(K_1 \cup K_2 - i)$ do

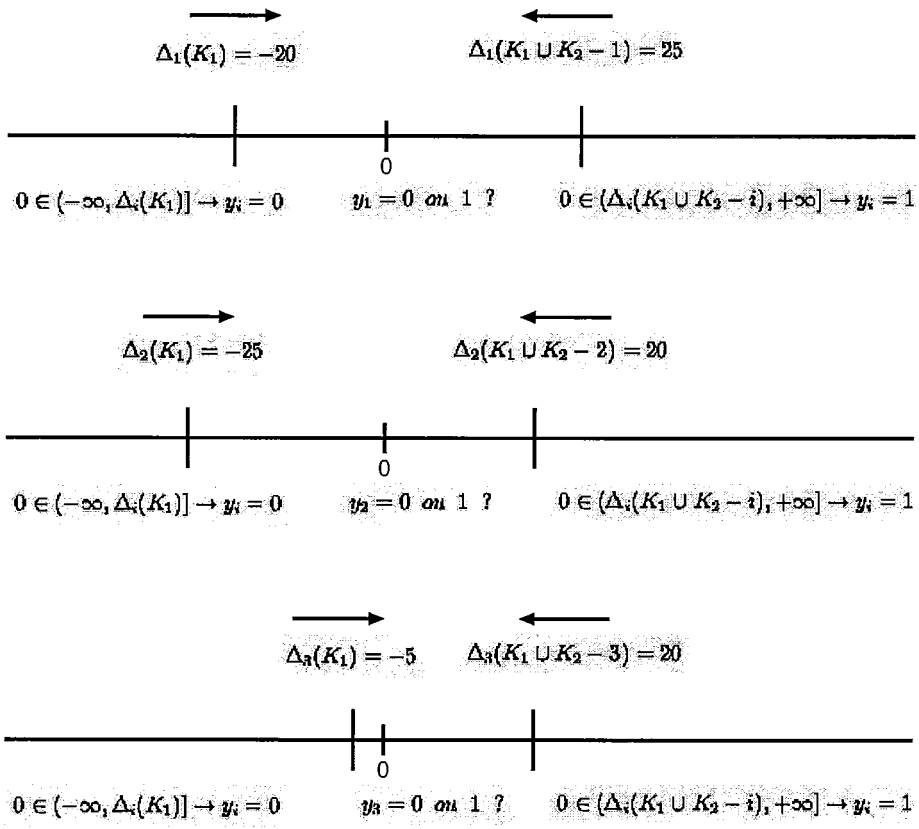


Figura 4.6: Funcionamento da O/C-Heurísticas

que $\delta_i(K_1)$ abre-se a facilidade i. Isso porque o custo fixo se aproxima do mínimo ganho. Provavelmente o ganho real abrindo-se i irá superar o custo fixo. Quando f_i está mais próximo de $\delta_i(K_1)$ do que $\delta_i(K_1 \cup K_2 - i)$ a facilidade i é fechada. Isso porque que o custo fixo aproxima-se do máximo ganho dos custos de transporte, ou seja, da pouca chance da facilidade i contribuir para a redução global dos custos. Abrindo-se i provavelmente o ganho real nos custos de transporte irá ser superado pelo custo fixo.

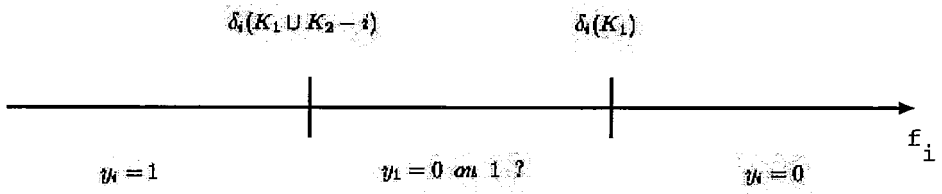


Figura 4.7: Inderpretação da O/C-Heurísticas

Quando K_1 é inviável é impossível calcular $\Delta_i(K_1)$ e se tem somente os dados da C-Heurística. Neste caso desenvolveu-se duas heurísticas baseadas nesta. As medidas também são obtidas por unidade de capacidade. A primeira heurística consiste em fixar a facilidade $i \in K_2$ cujo o índice fornece o valor da expressão $max_{i \in K_2} (\frac{\Delta_i(K_1 \cup K_2 - i)}{a_i})$. A expressão fornece entre as facilidades indefinidas aquela que deve ser fechada, indicando que devemos explorar primeiro v' ($K_0 = K_0 \cup \{i\}$), ou seja, primeiro insere-se v'' e depois v' em π. Esta heurística será referenciada por **CMax-Heurística**. A segunda heurística consiste em fixar a facilidade $i \in K_2$ cujo o índice fornece o valor da expressão $min_{i \in K_2} (\frac{\Delta_i(K_1 \cup K_2 - i)}{a_i})$. A expressão fornece entre as facilidades indefinidas aquela que deve ser aberta, indicando que devemos explorar primeiro v'' ($K_1 = K_1 \cup \{i\}$). Esta heurística será referenciada por **CMin-Heurística**.

CMax-Heurística e **CMin-Heurística** fornecem respectivamente uma maneira de fechar e abrir uma determinada facilidade quando K_1 for inviável. A figura 4.8 mostra que a facilidade 1 está mais próxima da região de abertura de facilidades do que a 2. CMin-Heurística procura a facilidade mais próxima da região de abertura, enquanto que CMax-Heurística a que está mais distante da região de abertura. Estas heurísticas podem ser muito falhas, isso porque como a região de fechamento não está definida a noção de perto ou longe perde um pouco o sentido. Por exemplo, uma facilidade que está longe da região de abertura poderia estar mais longe ainda da "região de fechamento" (quando esta vier a ser definida). Tal facilidade deveria portanto ser aberta e não fechada. Por outro lado uma facilidade pode parecer estar perto da região de abertura e no entanto estar longe se tomarmos como referencial a sua proximidade da região de fechamento.

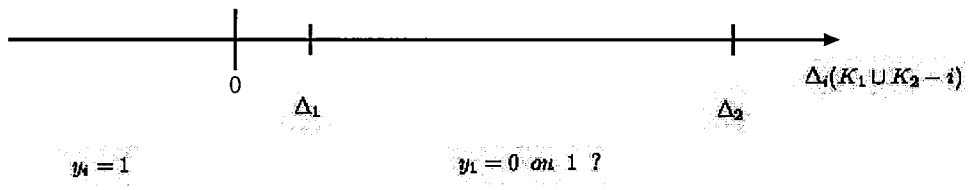


Figura 4.8: CMax- e CMin-Heurística

Seque-se outro tipo de interpretação. Tem-se que $\delta_i(K_1 \cup K_2 - i)$ representa o mínimo ganho quando a facilidade i é aberta. Quando o custo fixo de i é menor que o mínimo ganho a facilidade i será aberta (O-Teste). Por outro lado, quando o custo fixo for maior que o máximo ganho $\delta_i(K_1)$ a facilidade será fechada (C-Teste). Quando, no entanto, o custo fixo for maior que o mínimo ganho mas menor que o máximo ganho nada pode-se garantir sobre abertura/fechamento de

i. Neste caso aplicam-se as heurísticas. Quando o custo fixo de i está próximo do mínimo ganho abre-se i (CMax-Heurística). Nesta condição está implícita a suposição que o máximo ganho está distante do custo fixo. Com isso, provavelmente o ganho real abrindo-se i irá superar o custo fixo. Quando o custo fixo de i está distante do mínimo ganho fecha-se i (CMin-Heurística). Nesta condição está implícita a suposição que o máximo ganho está próximo do custo fixo. Assim abrindo-se i provavelmente o ganho real nos custos de transporte irá ser superado pelo custo fixo. A figura 4.9 mostra através das linhas pontilhadas as chances da facilidade i ser aberta. Quanto mais próximo o custo fixo estiver do mínimo ganho maiores as chances de i ser aberta. Contudo, como já dito anteriormente, pode ser que alguma coisa que pareça perto na verdade esteja longe se for considerado o máximo ganho, que aqui apenas estamos supondo.

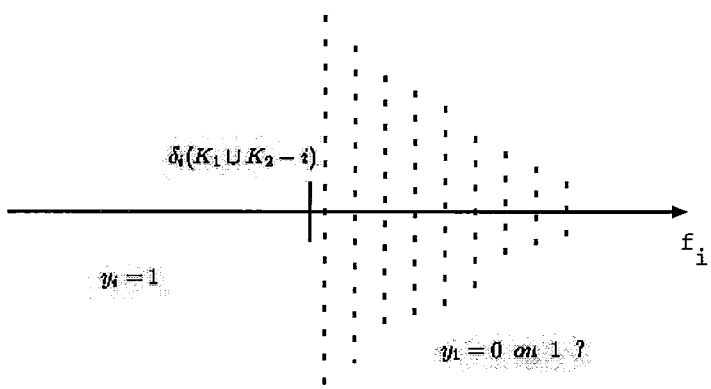


Figura 4.9: Chances de abertura de i

Além de um parâmetro de distância, outra coisa que deve ser observada para aplicação de uma ou outra heurística é o sucesso dos testes de redução. A **CMax-Heurística** por aumentar K_0, diminuir $K_1 \cup K_2$ e conseguentemente aumenta $\delta_i(K_1 \cup K_2 - i)$ diminuindo $\Delta_i(K_1 \cup K_2 - i)$ favorece o sucesso do O-Teste,

enquanto que **CMin-Heurística** por aumentar K_1 e consequentemente diminuir $\delta_i(K_1)$ aumentando $\Delta_i(K_1)$ favorece o sucesso do C-Teste(ver seção 3.3). Caso o PLC seja mais propício ao sucesso do C-Teste provavelmente utilizando a **CMin-Heurística** serão realizadas mais podas na árvore do que utilizando a **CMax-Heurística**. Os resultados computacionais sobre os testes de Kühn and Hamburger [22], Beasley [8] e de Cornuejols, Sridharan R. and Thizy [14] mostraram que **CMax-Heurística** fornece geralmente limites superiores melhores nas primeiras buscas em profundidade. Este resultado é apenas um parâmetro que não é por si só determinante. Por exemplo nos problemas de Cornuejols, Sridharan R. and Thizy [14] utilizou-se **CMin-Heurística**, a qual demonstrou ser bastante superior a **CMax-Heurística** na obtenção da solução ótima.

4.5 Limite Inferior

No método desenvolvido utiliza-se relaxação lagrangeana para obtenção do limite inferior, segundo Beasley [9]. A diferença é que não consideramos a restrição que fixa um número de facilidades a serem abertas. A formulação do PLC é um pouco diferente da trabalhada até agora. Considera-se c_{ij}, $\forall i \in I$ e $\forall j \in J$, como sendo o custo da facilidade i atender toda a demanda do cliente j. Com isso x_{ij} passa ser a fração de demanda do cliente j que será atendida pela facilidade i (assim $0 \leq x_{ij} \leq 1$). As outras variáveis tem o mesmo significado da formulação dada no capítulo 1. O modelo matemático é o seguinte:

$$minimize \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i \tag{4.1}$$

sujeito a :

$$\sum_{j \in J} b_j x_{ij} \leq a_i y_i, \quad \forall i \in I \tag{4.2}$$

$$\sum_{i \in I} x_{ij} = 1, \quad \forall j \in J \tag{4.3}$$

$$0 \leq x_{ij} \leq 1, \quad \forall i \in I, \forall j \in J \tag{4.4}$$

$$y_i \in \{0,1\}, \quad \forall i \in I \tag{4.5}$$

A equação 4.2 assegura que nenhum cliente seja atendido por uma facilidade fechada e que o total de demanda atendida pela facilidade não ultrapasse a sua capacidade. A equação 4.3 assegura que a demanda de cada cliente seja satisfeita. A equação 4.4 proporciona os limites de alocação para as variáveis (x_{ij}), enquanto que a 4.5 é a restrição de integralidade, onde $y_i = 1$ se a facilidade i for aberta e $y_i = 0$ caso contrário.

Para obtenção do limite inferior faz-se relaxação lagrangeana nas equações 4.2 e 4.3. Com os multiplicadores t_i $(i = 1, ..., n)$ para 4.2 e os multiplicadores s_j $(j = 1, ..., m)$ para 4.3 o PLC com relaxação lagrangeana (PLCRL) fica:

$$min \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i + \sum_{i \in J} s_j [1 - \sum_{i \in I} x_{ij}] + \sum_{i \in I} t_i [-y_i + \sum_{j \in J} (b_j / a_i) x_{ij}]$$

$$sujeito\ a: \ (4.4)(4.5)$$

$$t_i \geq 0 \tag{4.6}$$

Definindo

$$C_{ij} = c_{ij} - s_j + t_i(b_j/a_i)$$

o problema torna-se:

$$min \sum_{i \in I} \sum_{j \in J} C_{ij} x_{ij} + \sum_{i \in I} (f_i - t_i) y_i + \sum_{i \in J} s_j \tag{4.7}$$

$$sujeito\ a:\ (4.4)(4.5)(4.6)$$

O problema é facilmente resolvido porque se $y_i = 1$ (isto é a facilidade i é aberta) a contribuição para a função objetiva lagrangeana 4.7 é dado por:

$$p_i = f_i - t_i + \sum_{j \in J} \min(0, C_{ij})$$

O PLC com relaxação lagrangeana reduz-se para:

$$min \sum_{i \in I} p_i y_i + \sum_{i \in J} s_j$$

$$sujeito\ a: y_i \in \{0, 1\}, \quad \forall i \in I$$

Para resolver este problema basta fazer $y_i = 1$ quando $p_i < 0$, e $y_i = 0$ caso contrário. Denota-se a solução por $\hat{y}_i$. Os valores correspondentes para as variáveis (x_{ij}) são dados por $\hat{x}_{ij} = 1$ se $\hat{y}_i = 1$ e $C_{ij} \leq 0$ e $\hat{x}_{ij} = 0$ caso contrário.

Como o limite inferior será utilizado no *Branch and Bound* juntamente com os conjuntos K_0, K_1 e K_2 o cálculo do limite inferior de um determindo nó é obtido da seguinte forma:

- $\forall i \in K_0$ fazer $\hat{y}_i = 0$ e consequentemente $\hat{x}_{ij} = 0$, $\forall j \in J$.

- $\forall i \in K_1$ fazer $\hat{y}_i = 1$ e consequentemente $\hat{x}_{ij} = 1$ se $C_{ij} \leq 0$ e $\hat{x}_{ij} = 0$ caso contrário, $\forall j \in J$

- $\forall i \in K_2$ resolver o PLCRL substituindo o conjunto I por K_2, a formulação é a seguinte::

$$min \sum_{i \in K_2} p_i y_i + \sum_{i \in J} s_j$$
$$sujeito\ a : y_i \in \{0,1\}, \quad \forall i \in K_2$$

O valor do limite inferior é dado por:

$$LI = \sum_{i \in i} p_i \hat{y}_i + \sum_{j \in J} s_j$$

O cálculo do PLCRL fornece LI, $\hat{x}_{ij}$, $\forall i \in I$ e $\forall j \in J$ e $\hat{y}_i$, $\forall i \in I$.

4.5.1 Atualização do Limite Inferior

No método *Branch and Bound* os testes de redução e a própria ramificação fixam valores para as facilidades. Quando um valor for fixado a uma facilidade pode-se atualizar o limite inferior, o qual representará o limite inferior condicionado ao valor fixado à facilidade. Como no método sempre fixa-se um valor a uma facilidade que pertence a K_2 a atualização é realizada da seguinte maneira:

- quando a facilidade $i \in K_2$ for incluida em K_1 tem-se:

se $p_i > 0$ **então**

$\hat{y}_i = 1$

$LI = LI + p_i$

se $C_{ij} < 0$ **então** $(\forall j \in J)$

$\hat{x}_{ij} = 1$

senão

$\hat{x}_{ij} = 0$

senão

permancem os valores de $\hat{y}_i$ e $\hat{x}_{ij}$ calculados

Quando coloca-se i em K_1 deve-se verificar se no cálculo do limite inferior a facilidade foi considerada aberta ou fechada. Caso $p_i < 0$ a facilidade i foi considerada aberta. Neste caso não existe nada que possa ser realizado para melhorar o limite inferior considerando a fixação de i em K_1. Contudo, caso $p_i > 0$, a facilidade i foi considerada fechada. Assim sendo, deve-se atualizar o limite inferior abrindo-se i.

- quando a facilidade $i \in K_2$ for incluida em K_0 tem-se:

  se $p_i < 0$ **então**

  $\hat{y}_i = 0$

  $LI = LI - p_i$

  $\hat{x}_{ij} = 0, \forall j \in J$

  senão

  permancem os valores de $\hat{y}_i$ e $\hat{x}_{ij}$ calculados

  Da mesma forma que é verificado se no cálculo do limite inferior a facilidade foi considerada aberta ou fechada quando coloca-se i em K_1, deve-se verificar quando coloca-se i em K_0. Caso $p_i > 0$ a facilidade i foi considerada fechada. Neste caso não existe nada que possa ser realizado para melhorar o limite inferior considerando a fixação de i em K_0. Contudo, caso $p_i < 0$ a facilidade i foi considerada aberta. Assim sendo, deve-se atualizar o limite inferior fechando-se i.

4.5.2 Calculo do Subgradiente

A otimização com subgradiente é usada para tentar maximizar o limite inferior obtido pelo PLCRL. O procedimento adotado é o seguinte:

1. Calcular os subgradientes G_j e H_i usando:

   $G_j = 1 - \sum_{i \in I} \hat{x}_{ij}, \forall j \in J$

   $H_i = -\hat{y}_i + \sum_{j \in J} (b_j / a_i) \hat{x}_{ij}, \forall i \in I$

2. Ajustar H_i usando:

   $H_i = 0$ se $t_i = 0$ e $H_i < 0, \forall i \in I$

3. Definir o tamanho do passo T usando:

   $T = \frac{f(LS-LI)}{\sum_{j\in J}(G_j)^2+\sum_{i\in I}(H_i)^2}$

4. Atualizar os multiplicadores de lagrange usando:

   $s_j = s_j + TG_j, \quad \forall j \in J$

   $t_i = max(0, t_i + TH_i), \quad \forall i \in I$

4.5.3 Algoritmo Geral para o Cálculo do Limite Inferior

O algoritmo usa dois parâmetros que são fornecidos conforme as necessidades requeridas pelo método. Um parâmetro é o escalar $f \leq 2$ que serve para ajustar o passo do subgradiente, o outro é o número máximo permitido de iterações de subgradiente sem aumentar o limite inferior para um dado valor de f, denotado por n. Deve-se utilizar o seguinte procedimento para calcular o primeiro limite inferior.

1. inicializar os multiplicadores usando:

   $t_i = 0 \quad \forall i \in I$ e $s_j = \min(c_{ij} | c_{ij} > 0, \forall i \in I$ e $\forall j \in J)$

2. resolver o PLCRL com os multiplicadores definidos em 1, sendo a solução dada por LI, $\hat{x}_{ij}$ e $\hat{y}_i$.

Quando o limite inferior já foi calculado pelo menos uma vez os valores LI, $\hat{x}_{ij}$, $\hat{y}_i$ e os multiplicadores relativos a esses valores já existem, então passa-se a aplicar o procedimento de otimização por subgradiente na tentativa de melhorar o limite inferior encontrado. O melhor limite inferior encontrado é representado por $\overline{LI}$, o cálculo do subgradiente é realizado sempre com o último LI encontrado, ou seja, o valor do último PLCRL calculado e com o conjunto corrente de multiplicadores (os últimos multiplicadores calculados). O algoritmo é dado a seguir, onde N representa o número de iterações de subgradiente que não aumentaram $\overline{LI}$.

inicio

$N = 0$

enquanto $f \geq 0.0005$ **fazer**

Calcular o Subgradiente

Resolver PLCRL

se $\overline{LI} < LI$

$\overline{LI} = LI$

$N = 0$

senão

$N = N + 1$

se $N = n$ **entao**

$f = f/2$

$N = 0$

fim

Após a realização da primeira busca em profundidade calcula-se o primeiro limite inferior considerando os conjuntos K_0, K_1 e K_2 obtidos até a primeira ramificação e os seguintes parâmetros: $n = 30$, $f = 2$ e $LS = LS_{min} + (LS_{min} * 0.25)$. LS utilizado para o cálculo do subgradiente é obtido através da fórmula $LS = LS_{min} + (LS_{min} * 0.25)$, onde LS_{min} é o valor obtido na primeira busca em profundidade. A fórmula utilizada é resultado da experimentações feitas. Beasley [9] usa $LS = LS_{min} + (LS_{min} * 0.05)$. A diferença deve ser provavelmente ao fato de nossos limites superiores serem melhores, isto é, menores. Este método fornece o primeiro limite inferior. Os demais limites inferiores, obtidos no correr do método *Branch and Bound*, podem ser obtidos de duas maneiras distintas. A primeira, denominada de A-LI, consiste apenas na atualização de v conforme seção 4.5.1. A segunda maneira, denominada de C-LI, além da atualização, calcula o limite inferior com a geração de novos multiplicadores de lagrange, ou seja, com novas operações de sub-gradiente. Neste caso utiliza-se os seguintes parâmetros: $n = 10$, $f = 0.125$ e LS com um valor 5% maior que LS_{min}. Estes parâmetros são os mesmos utilizado por Beasley [8].

4.5.4 Redução do Problema Utilizando LI

Seja v o nó da árvore que se está explorando. Ao invés de calular LI_v e fazer a poda correspondente, uma ideia consite em tomar $i \in K_2$, alternadamente alocar i a K_0 e K_1 gerando v_i' e v_i'' e calcular $LI_{v_i'}$ e $LI_{v_i''}$ (utilizando A-LI) comparando estes valores com LS_{min}. Se $LI_{v_i'} \geq LS_{min}$ não precisamos considerar a alternativa da facilidade i ser fechada, pois no máximo poderia gerar um solução igual a uma

já encontrada. Logo aloca-se i em K_1. Se $LI_{v_i''} \geq LS_{min}$ significa que a facilidade i não precisa ser aberta, pois no máximo poderia gerar um solução igual a uma já encontrada. Logo aloca-se i em K_0. Este procedimento feito $\forall i \in K_2$ implica, a semelhança dos O e C-Testes, em uma serie de podas, que podem reduzir em muito o processo enumerativo. Ressalta-se que esta redução é feita através dos limites inferiores calculados para ramificações possíveis do nó. O procedimento para redução do problema ultizando LI é realizado da seguinte forma:

inicio

para cada $i \in K_2$ **fazer**

calcular $LI_{v_i'}$

se $LI_{v_i'} \geq LS_{min}$ **então**

$K_2 = K_2 - \{i\}$

$K_1 = K_1 \cup \{i\}$

senão

calcular $LI_{v_i''}$

se $LI_{v_i''} \geq LS_{min}$ **então**

$K_2 = K_2 - \{i\}$

$K_0 = K_0 \cup \{i\}$

fim

4.6 Algoritmo

Apresentaremos uma visão geral do algoritmo. O primeiro passo é a inicialização, onde é gerado o primeiro nó que deve ser explorado (raiz), o qual representa o PLC a ser resolvido. O laço mais externo do passo 1 ao 10, representa o processo enumerativo. Este laço somente termina quando não existir mais nenhum sub-problema para ser resolvido, ou seja, quando todas as soluções descendentes da raiz tiverem sido exploradas implicitamente ou explicitamente. Esta condição acontece quando $\pi = \emptyset$, indicando a parada do algoritmo.

O laço representado pelos passos 3 e 4 consiste da realização do O-Teste $\forall i \in K_2$. O passo 3 representa a execução propriamente dita do O-Teste, enquando que o passo 4 verifica se o passo 3 não levou o nó gerado a uma condição de *backtracking*. O laço representado pelos passos 5 e 6 é igual ao laço representado pelos passos 3 e 4, exceto por se tratar do C-Teste. O laço composto pelos passos de 3 a 6 representa a alternância dos O e C-Testes. Este laço é quebrado somente quando nenhum teste tiver sucesso $\forall i \in K_2$.

Quando os testes não tiverem sucesso passa-se para o passo 7 ou 8, dependendo da viabilidade de K_1. O passo 7 representa a ramificação utilizando a O/C-Heurística, a qual depende de K_1 ser viável para que se possa utiliza-la. O passo 8 representa a ramificação utilizando a CMax ou CMin-Heurística, que é utilizada quando K_1 for inviável. O passo 9 atualiza o limite inferior do problema caso seja possível, calcula o limite inferior dos nós gerados na ramificação e coloca estes em

π caso não ocorra *backtracking*. O passo 10 é executado quando o método chega a uma folha, nele atualiza-se o limite superior do problema caso seja possível.

Para não sobrecarregar a apresentação e porque não afeta a essência do algoritmo não colocamos a redução do problema utilizando LI, como foi mostrado em 4.5.4.

No passo 9 fizemos a atualização do limite inferior LI_{max}. Esta atualização é feita sempre que a lista de nós π estiver vazia, significando que um ramo da árvore foi explorada completamente. Neste caso o limite inferior do problema é o limite inferior do ramo ainda não explorado.

Passo 0: Inicialização

Gerar a raiz e colocar em π

Passo 1: Pegar um nó de π

Se $\pi \neq \emptyset$ remover último nó e fazer igual a NÓ

Senão parar e imprimir LS_{min} e SOL_{min}

Passo 2: Identificando uma folha

Se $K_2 = \emptyset$ ir para **Passo 10**

Passo 3: Poda na ramificação esquerda (O-Teste)

TESTE $= 0$

Usar O-Teste para $i \in K_2$

Se não obtiver sucesso na aplicação do O-Teste ir para **Passo 5**

Colocar a solução referente ao primeiro sucesso da aplicação do O-Teste em NÓ e atualizar K_1 e K_2

Passo 4: Poda depois do O-Teste usando limite inferior

Se $K_2 = \emptyset$ ir para **Passo 10**

Calcular $LI_{NÓ}$

Se $LI_{NÓ} < LS_{min}$ ir para **Passo 3**

Senão ir para **Passo 1**

Passo 5: Poda na ramificação direita (C-Teste)

Usar C-Teste para $i \in K_2$

Se não obtiver sucesso na aplicação do C-Teste e TESTE=0 ir para **Passo 7**

Se não obtiver sucesso na aplicação do C-Teste e TESTE=1 ir para **Passo 3**

Colocar a solução referente ao primeiro sucesso da aplicação do C-Teste em NÓ, atualizar K_0 e K_2 e fazer TESTE $= 1$

Passo 6: Poda depois do C-Teste usando limite inferior

Se $K_2 = \emptyset$ ir para **Passo 10**

Calcular $LI_{NÓ}$

Se $LI_{NÓ} < LS_{min}$ ir para **Passo 5**

Senão ir para **Passo 1**

Passo 7: Ramificação / K_1 viável (O/C-Heurística)

Se K_1 inviável ir para **Passo 8**

Usar a O/C-Heurística em NÓ ramificando em NÓ' e NÓ"

Dependendo da saida da heurística faz-se NODELAST = NÓ' ou NÓ"

$\overline{NODELAST}$ = {NÓ', NÓ"} - NODELAST

ir para **Passo 9**

Passo 8: Ramificação / K_1 inviável (CMax e CMin-Heurística)

Usar a CMax ou a CMin heurística em NÓ ramificando em NÓ' e NÓ"

Dependendo da saida da heurística faz-se NODELAST = NÓ' ou NÓ"

$\overline{NODELAST}$ = {NÓ', NÓ"} - NODELAST

Passo 9: Poda depois da ramificação usando limite inferior

Calcular $LI_{NODELAST}$ e $LI_{\overline{NODELAST}}$

Se $\pi = \emptyset$ fazer $LI_{max} = min(LI_{NODELAST}, LI_{\overline{NODELAST}})$

Se $LI_{\overline{NODELAST}} < LS_{min}$ colocar $\overline{NODELAST}$ no final de π

Se $LI_{NODELAST} < LS_{min}$ colocar $NODELAST$ no final de π

ir para **Passo 1**

Passo 10: Atualizar o Limite Superior

Fazer LS igual ao valor da função objetivo do NÓ

Se $LS < LS_{min}$ fazer $LS_{min} = LS$ e $SOL_{min} = NÓ$

ir para **Passo 1**

Embora não esteja explícito no algoritmo descrito acima, o limite inferior só é realmente calculado quando o método atinge a primeira folha. Isso acontece porque para calcular o limite inferior precisa-se de um limite superior. O cálculo do limite inferior é realizado nos passos 4, 6 e 9. Nos passos 4 e 6 o cálculo é feito somente pela maneira A-LI, pois o custo computacional para realizar o cálculo através de sub-gradientes a cada sucesso dos testes é grande. No passo 9 o cálculo pode ser realizado utilizando A-LI ou C-LI.

4.7 Resultados Computacionais

Utilizou-se a modelagem Orientada a Objetos e a linguagem C++ para implementação do algoritmo apresentado. Os testes foram rodados em uma estação

Sun Ultra-5 sobre os problemas de Kühn & Hamburger [22], Beasley [8] e Cornuejols, Sridharan and Thizy [14]. O algoritmo de transporte foi obtido conforme dito na seção 2.3.

A tabela 4.1 mostra os resultados obtidos na primeira busca em profundidade, ou seja, os resultados da primeira solução viável encontrada. Mostra-se o tempo em segundos obtido para encontrar a primeira solução e a qualidade desta com relação ao primeiro limite inferior encontrado pelo método (LI_{max}) e com relação a solução ótima do problema (Z^*). A comparação é realizada com o algoritmo heurístico desenvolvido por Beasley [9], onde é fornecido o tempo em segundos e a qualidade da solução obtida (Z_{UB}) pelo seu método com relação a solução ótima (Z^*). Beasley utilizou um Cray X-MP/28 para obtenção de seus resultados.

As tabelas 4.2 e 4.3 mostram os resultados obtidos pelo método *Branch and Bound* proposto e os resultados obtidos por Beasley [8] sobre os problemas de Kühn & Hamburger, sendo que a tabela 4.3 é continuação da tabela 4.2. As medidas retiradas do método proposto são o número de ramificações realizadas e o tempo em segundos para o método encontrar a solução ótima, enquanto que no método de Beasley obtem-se o número de nós da árvore e o tempo em segundos obtido em um Cray-1S.

A tabela 4.4 mostra os resultados obtidos pelo método *Branch and Bound* desenvolvido sobre alguns problemas de Kühn & Hamburger, e compara-se com os resultados obtidos por Van Roy [25]. As medidas relativas ao método proposto são o número de ramificações realizadas e o tempo em segundos para se encontrar

Problema	nxm	Algoritmo Proposto[1]			Beasley[2]	
		$100 * \frac{Z^h - LI}{LI}$	$100 * \frac{Z^h - Z^*}{Z^*}$	sec.	$100 * \frac{Z_{UB} - Z^*}{Z^*}$	sec.
41	16x50	0.01	0	0.04	0	0.97
42		0.06	0	0.04	0	0.57
43		0.08	0	0.05	0	0.58
44		0.20	0	0.04	0	0.59
51		0.28	0.21	0.04	0	0.80
61		0.01	0	0.02	0	0.46
62		0	0	0.02	0	0.44
63		0.30	0.16	0.03	0	0.71
64		0.73	0.72	0.05	0	0.36
81	25x50	0.47	0.29	0.09	0	2.27
82		0.07	0	0.10	0	1.74
83		0.08	0	0.11	0	1.90
84		0.57	0.38	0.12	0	2.86
91		0	0	0.05	0	1.23
92		0.08	0	0.06	0	1.11
93		0.33	0.13	0.08	0	1.17
94		0.41	0	0.13	0.06	1.25
111	50x50	0.01	0	0.68	0	2.74
112		0.14	0	0.75	0	3.11
113		0.36	0.08	0.78	0	2.71
114		0.81	0.55	0.84	0.44	4.22
121		0	0	0.29	0	2.37
122		0.17	0.07	0.24	0	1.72
123		0.52	0.40	0.24	0	1.47
124		0.43	0	0.82	0.17	1.91
média		0.24	0.12	0.23	0.03	1.57

Tabela 4.1: Primeira solução (folha) - Problemas de Kühn & Hamburger (1. Sun Ultra-5 2. Cray X-MP/28)

Problema	nXm	Algoritmo Proposto[1]		Beasley[2]	
		branch.	sec.	nós	sec.
41	16x50	-	0.24	-	1.2
42		-	0.24	-	1.0
43		-	0.26	13	1.9
44		-	0.31	-	0.9
51		4	0.33	-	1.2
61		1	0.22	-	0.3
62		-	0.20	-	0.5
63		5	0.26	7	1.1
64		5	0.27	-	0.3
71		-	0.18	-	0.2
72		-	0.19	-	0.2
73		-	0.20	-	0.2
74		-	0.19	-	0.1
81	25x50	13	0.56	33	3.8
82		1	0.45	5	1.8
83		3	0.55	9	1.8
84		32	1.01	41	4.0

Tabela 4.2: *Branch and Bound* - Problemas de Kühn & Hamburger (1. Sun Ultra-5 2. Cray-1S)

Problema	nXm	Algoritmo Proposto[1]		Beasley[2]	
		branch.	sec.	nós	sec.
91		-	0.37	-	0.8
92		2	0.43	-	0.9
93		28	0.67	23	2.4
94		11	0.57	57	3.1
101		-	0.29	-	0.4
102		-	0.33	-	0.4
103		3	0.34	-	0.5
104		-	0.32	-	0.2
111	50x50	2	1.35	3	2.4
112		8	1.59	37	4.3
113		99	3.56	129	8.7
114		284	6.39	151	8.8
121		-	0.78	-	1.5
122		9	1.78	11	2.3
123		80	1.41	49	3.8
124		109	2.59	175	7.5
131		-	0.98	-	0.8
132		6	0.84	-	0.6
133		-	0.76	-	0.6
134		7	0.63	-	0.7
Média		19.24	0.86	20.1	1.9

Tabela 4.3: *Branch and Bound* - Problemas de Kühn & Hamburger continução

(1. Sun Ultra-5 2. Cray-1S)

a solução ótima, enquanto que no método do Van Roy obtem-se o número de nós da árvore e o tempo em segundos obtido em um MV8000.

A tabela 4.5 mostra os resultados obtidos pelo método *Branch and Bound* desenvolvido e pelo método desenvolvido por Beasley [8] sobre os problemas maiores gerados aleatoreamente por Beasley [8]. As medidas retiradas do método proposto são o número de ramificações realizadas e o tempo em segundos para se encontrar a solução ótima, enquanto que no método de Beasley obtem-se o número de nós da árvore e o tempo em segundos obtido em um Cray-1S.

A tabela 4.6 mostra os resultados obtidos pelo método *Branch and Bound* proposto para os problemas gerados por Cornuejols, Sridharan and Thizy [14], comparando-se com o tempo de execução dos metodos utilizados por este último para encontrar Z_c e Z^I, que são respectivamente os limites inferiores obtidos por relaxação lagrangeana da restrição de capacidade do PLC e por relaxação completa da restrição de integralidade do PLC. Thizy utiliza o método de Van Roy [25], comparado anteriormente com os problemas de Kühn & Hamburger, para encontrar Z_c mencionando que para problemas difíceis o método frequentemente reduz-se para um método ineficiente de geração de colunas. Thizy procurou gerar uma bateria de testes com dificuldades razoáveis para serem resolvidos. Mostra-se na tabela 4.6 a média de ramificações realizadas e do tempo em segundos para o método encontrar a solução ótima de 30 problemas nas dimensões 8x25, 16x25, 25x25, 16x50, 33x50 e 50x50, e a média do tempo em segundos para encontrar Z_c e Z^I dos problemas. Thizy utilizou um IBM 3081 para obter os resultados de

Problema	nxm	Algoritmo Proposto[1]		Roy[2]	
		branch.	sec.	nós	sec.
41	16x50	-	0.24	-	0.24
42		-	0.24	-	0.25
43		-	0.26	3	1.50
44		-	0.31	-	0.97
61		1	0.22	-	0.38
62		-	0.20	-	0.39
63		5	0.26	6	1.92
64		5	0.27	15	2.71
71		-	0.18	-	0.25
72		-	0.19	2	1.33
73		-	0.20	2	1.43
74		-	0.19	2	1.37
81	25x50	13	0.56	-	0.33
82		1	0.45	12	2.26
83		3	0.55	26	3.53
84		32	1.01	118	6.37
91		-	0.37	-	0.31
92		2	0.43	-	1.66
93		28	0.67	10	3.09
94		11	0.57	41	4.93
101		-	0.29	38	2.46
102		-	0.33	5	5.47
103		3	0.34	34	7.55
104		-	0.32	132	26.44
média		3.87	0.36	18.58	3.21

Tabela 4.4: *Branch and Bound* - Problemas de Kühn & Hamburger (1. Sun Ultra-5 2. MV 8000)

Problema	nXm	Algoritmo Proposto[1]		Beasley[2]	
		branch.	sec.	nós	sec.
A1	100x1000	1744	21735.2	43	87.4
A2	100x1000	1786	24422.3	39	79.9
A3	100x1000	1886	24055.6	27	59.3
A4	100x1000	1886	23698.9	-	28.1
B1	100x1000	155	1887.4	7	74.2
B2	100x1000	150	1883.1	309	321.6
B3	100x1000	136	1672.2	257	244.9
B4	100x1000	153	1923.9	47	89.5
C1	100x1000	21	339.3	59	150.6
C2	100x1000	19	307.9	237	294.7
C3	100x1000	22	266.2	45	108.7
C4	100x1000	27	342.4	7	87.7
média		665.4	8544.5	89.7	135.5

Tabela 4.5: *Branch and Bound* - Problemas de Beasley (1. Sun Ultra-5 2. Cray-1S)

Z_c e um DEC 20 para obter os resultados de Z^I.

Para os problemas de Kühn & Hamburger o cálculo do limite inferior no passo 9 foi realizado utilizando A-LI. Para os problemas de Beasley e Thizy o cálculo do limite inferior no passo 9 foi realizado utilizando C-LI. A razão deste procedimento é que para os problemas menores não há necessidade de um procedimento de poda tão acurado. Já para os problemas grandes, o maior gasto de tempo no cálculo de limites é compensado pelo maior número de podas realizado.

No passo 8 do algoritmo utilizou-se CMax-Heurística para os problemas de Kühn & Hamburger e Beasley e CMin-Heurística para os problemas de Thizy. Esta decisão foi tomada por experimentos práticos onde levou-se em conta o número de ramificações para se chegar à primeira solução viável (folha) e a quali-

nxm	Algoritmo Proposto (1)		$Z_C^{(2)}$	$Z^{I\,(3)}$
	branch.	sec	sec	sec
8x25	9.72	0.25	0.9	2.1
16x25	43.92	1.76	14	5.7
25x25	108.76	6.42	140	11
16x50	45.8	4.47	34	8.7
33x50	264.96	42.44	1900	28
50x50	877.72	212.88	6800	62

Tabela 4.6: *Branch and Bound* - Média de 6x25 Problemas de Thizy (1 - Sun Ultra-5 2. DEC 20)

dade desta solução. Esta estratégia não foi melhor para todos os problemas, mas ela foi melhor levando-se em conta o conjunto total de problemas.

Os resultados obtidos mostraram que o método desenvolvido é bastante eficiente para problemas de pequeno e médio porte, inclusive para os problemas difíceis gerados por Thizy. Para os problemas grandes (veja tabela 4.5) os resultados não foram tão expressivos. Ha no entanto que se levar em conta que o método desenvolvido por Beasley é especialmente eficiente para problemas de grande porte que requerem um número pequeno de facilidades abertas na solução ótima. Conforme Beasley [8], seu método é capaz de resolver problemas de grande porte desde que o número de facilidades abertas seja pequeno. Seu algoritmo de transporte só leva em conta facilidades abertas, implicando na resolução de problemas de transporte pequenos. Para os problemas da tabela 4.5 seu método resolveu problemas de transporte com no máximo 11 centros de oferta. Já no método proposto o fato de termos poucas ou muitas facilidades abertas na solução ótima não é relevante, já que grande parte dos problemas de transporte é

resolvido para $K_1 \cup K_2$. Resulta dai uma maior robustes do método proposto. Por exemplo, para a resolução de qualquer problema da tabela 4.5, para obtenção da primeira solução viável o método proposto resolve no mínimo 100 problemas de transporte com 99 centros de oferta e 100000 centros de demanda. A diferença dos dois métodos com relação ao número de facilidades abertas pode ser vista para os problemas de Kühn & Hamburger (tabelas 4.2 e 4.3), onde um número razoável de facilidades são abertas na solução ótima o desempenho do algoritmo de Beasley não é tão favorecido. Há que também levar em conta que a implementação de Beasley utiliza processamento paralelo além dos resultados terem sido obtidos em máquina de maior desempenho. Portanto pouco pode-se afirmar sobre a eficiência comparativa dos dois métodos quando o problema requer um número médio ou grande de facilidades abertas na solução ótima. Vale observar que a performance do método proposto não é muito afetada pelo número de facilidades abertas na solução ótima o que caracteriza a maior robustez do método. Até pelo contrário, possivelmente o método proposto é mais eficiente para problemas que requeram um número grande de facilidades abertas na solução ótima. Isso porque o O-Teste pode fazer K_1 viável mais rapidamente, proporcionando o sucesso do C-Teste e o uso da **O/C-Heuristíca**, a qual toma decisões melhores do que as outras heuristícas aplicadas para K_1 inviável.

Capítulo 5

Conclusões

Apesar de que algumas vezes os testes de redução conseguem determinar a solução ótima de um PLC, conforme visto no exemplo 3.3, a principal aplicabilidade deles é com relação à redução do problema original. Com isso ulitiza-se os testes tanto em métodos heurísticos (Veja Bornstein and Azlan usaram em [10]), como em métodos exatos. Este é o caso do método apresentado no capítulo 4.

As aproximações desenvolvidas, na seção 2.2 para os testes, mostraram que a aproximação para o O-Teste é mais rápida que outras aproximações e tem uma boa qualidade. Já a aproximação para o C-Teste mesmo sendo mais rápida não justifica a sua aplicação devido a problemas com a qualidade. A eficiência do método de transporte utilizado praticamente elimina a necessidade da aplicação das aproximações, conforme visto na seção 2.3. A única questão que fica aberta é a aplicação da aproximação desenvolvida na seção 2.2.1, para o O-Teste. Esta aproximação demonstrou ser muito mais rápida para problemas grandes.

Na maioria dos PLC os testes de redução não conseguem fechar o gap de indefinição. O principal problema da falta de sucesso do O-Teste é quando $K_1 \cup K_2$ é grande demais, o que gera uma insensibilidade do custo de transporte ao retirar-se uma facilidade, ocasionando assim a falha do teste. O principal problema da falta de sucesso do C-Teste é quando K_1 é muito pequeno, o que torna muito sensivel o custo de transporte quando é inserido uma facilidade, ocasionando assim a falha do teste. Estes problemas podem acontecer em várias regiões de um PLC, as chamadas regiões de indecisão. O sucesso de um teste influência positivamente no sucesso do outro, pois o O-Teste aumenta K_1 e o C-Teste diminui $K_1 \cup K_2$.

Existe uma dificuldade muito grande para se encontrar classes de problemas para as quais se consegue fechar o gap de indefinição. As dificuldades são provenientes da grande quantidade de inter-relações dos dados do problema. Várias classes de problemas foram exploradas, sendo algumas delas bem específicas e de fácil solução. Contudo, foram apresentados contra exemplos onde o gap não fecha.

O C-Teste tende a ser mais favorecido para obtenção do sucesso quando os f_i forem maiores que os c_{ij} e quando as diferenças entre os c_{ij} forem pequenas. Por exemplo, pegando um problema e aumentando os valores dos f_i e/ou diminuindo a diferença entre os c_{ij} será aumentada a pontencialidade do C-Teste. O oposto ocorre para o O-Teste, ou seja, quanto maiores forem os c_{ij} em relação aos f_i e quanto maiores forem as diferenças entre os c_{ij}, maiores as chances do O-Teste obter sucesso.

Os testes de redução abrem/fecham facilidades *a priori*. Contudo para o bom desempenho destes eles necessitam da abertura/fechamento das facilidade *a priori*. Este ciclo vicioso é quebrado pelo *Branch and Bound*. O método *Branch and Bound* desenvolvido utiliza-se dos testes de redução e da relaxação lagrangeana para fazer as podas na árvore. Quando os testes falham o processo de ramificação (*branching*) abre e fecha as facilidades até que os testes voltem a funcionar. Utilizou-se busca em profundidade e regras de *branching* baseadas em heurísticas decorrentes dos testes de redução. As regras fornecem uma boa estratégia de descida na árvore, proporcionando uma solução viável próxima da ótima no ínicio da exploração. O limite inferior do problema é calculado utilizando relaxação lagrangeana, e este juntamente com a melhor solução viável encontrada pelo método fornecem uma medida da qualidade da solução, proporcionando assim ao usuário a possibilidade de parar a execução do método antes mesmo de ser encontrado a solução ótima.

As regras de *branching* utilizadas são baseadas na O- e C-Heurística desenvolvidas por Jacobsen [20]. A O/C-Heurística procura sanar as falhas das regras de Jacobsen fazendo um balanceamento entre as duas heurísticas para a tomada de uma decisão mais precisa quando K_1 for viável. A O/C-Heurística leva em conta a mínima e máxima sensibilidade dos custos de transporte, quando uma facilidade é aberta, e as comparam com o custo fixo. Para o caso de K_1 ser inviável foi desenvolvida a CMax- e CMin-Heurística que implica no fechamento/abertura de uma determinada facilidade. Resta ainda saber em quais situações uma ou a outra ou ambas devem ser usadas. De um lado a CMax-Heurística, proporcionando

o fechamento de facilidades, portanto, a diminuição de $K_1 \cup K_2$ e a consequente diminuição de $\Delta_i(K_1 \cup K_2 - i)$, aumenta o potencial de sucesso do O-Teste. Por outro lado, a CMin-Heurística, abrindo facilidades, portanto, aumentando K_1, aumenta não só as chances de se utilizar a O/C-Heurística que certamente representa um heurística mais elaborada, como aumenta as chances de uso e de sucesso do C-Teste. Cabe no futuro determinar qual a melhor política.

Os resultados computacionais demonstraram que o algoritmo conseguiu encontrar rapidamente uma solução próxima da ótima na primeira busca em profundidade, não ficando muito longe, em termos de qualidade, da solução obtida pelos melhores métodos heurísticos encontrados na literatura. Os resultados para os problemas de pequeno e médio porte, fáceis e difíceis, foram excelentes. Para os problemas de grande porte os resultados não foram tão expressivos. Excetuando-se o método desenvolvido por Beasley [8] não se têm parâmetros de comparação com nenhum outro método. Mesmo para o método de Beasley a única coisa que pode-se afirmar é que seu método é mais eficiente para problemas que têm poucas facilidades abertas na solução ótima. Para outros tipos de problemas de grande porte nada pode-se afirmar sobre a eficiência comparativa dos dois métodos, a não ser a robustez do método proposto, pois ele não é sensível ao número de facilidades abertas na solução ótima.

Muita coisa ainda pode ser desenvolvida no método *Branch and Bound*:

1. aplicação das aproximações no lugar dos testes exatos para verificar o comportamento do método. Deve ser dada maior ênfase para a aproximação do

O-Teste desenvolvida na seção 2.2.1, pois parece ser a única que pode ter alguma vantagem sobre os testes exatos. Pode ser realizada também uma alternância entre testes exatos e aproximados.

2. encontrar medidas para que se possa automatizar o uso da CMax e CMin-heurísticas, ou seja, dar possibilidade ao método para que ele possa mudar de estratégia durante o processo enumerativo. Atualmente o processo é fixado *a priori* para um determinado problema. As medidas provavelmente devem ser obtidas com os resultados dos próprios testes.

3. desenvolver uma aproximação para w(K1) quando $K1$ for inviável, com isso, pode-se aplicar a O/C-Heurística em vez de CMax e CMin-Heurística.

4. fazer um estudo sobre a atualização do limite inferior ao realizar uma ramificação, verificando quando deve ser utilizada C-LI e A-LI, e se possível automatizar este procedimento.

5. para problemas de grande porte utilizar técnicas de decomposição ou de agrupamento, dependendo da característica do problema em questão.

6. fazer uma combinção entre busca em profundidade e em largura, ou seja, continuar percorrendo a árvore em profundidade, mas não necessariamente continuar a manipulação da lista na ordem LIFO. O objetivo poderia ser tentar percorrer a sub-árvore que por algum critério estabelecido tenha mais chances de conter a solução ótima. Este critério pode ser obtido, por exemplo, utilizando os valores das heurísticas. Temos o sentimento que na

O/C-Heurística quando menor for o valor calculado, maior é a chance de ter sido feita uma escolha errada.

7. nas fórmulas das heurísticas, ao fazer a divisão pela capacidade, consideramos a capacidade total da facilidade i. Talvez seja melhor utilizar somente a capacidade ociosa do Problema de Transporte em que i participa.

8. fazer um estudo sobre a possibilidade de se caracterizar as regiões de indecisão. Caso isso seja possível, provavelmente um boa estratégia de ramificação é tomar a facilidade que faça parte do maior número de regiões de indecisão para fazer a ramificação.

9. fazer um estudo mais detalhado para verificar quando um dado problema é mais propício para o sucesso do O-Teste ou do C-Teste. Isso trará informações sobre qual ramificação deve ser explorada primeiro, proporcionando um número de podas maior.

10. aplicar de forma mais eficiente a redução utilizando LI. A maneira para fazer isso é aplicar a redução somente quando o intervalo entre LI_{max} e LS_{min} for relativamente pequeno.

Referências Bibliográficas

[1] J. H. Ahrens and G. Finke. Primal transportation and transshipment algorithms. *Zeitshrift for Operations Research*, 24:1–32, 1980.

[2] C. H. Aikens. Facility location models for distribution planning. *European Journal of Operational Research*, 22:263–279, 1985.

[3] U. Akinc and B. M. Khumawala. An efficient branch and bound algorithm for the capacitated wharehouse location problem. *Management Science*, 23, $N°$ 6:585–594, 1977.

[4] H. A. B. Azlán. *Aplicação do algoritmo simulated annealing ao problema de localização capacitado, usando testes de redução.* PhD thesis, Engenharia de Sistemas e Computação - COPPE/UFRJ, Rio de Janeiro, Brasil, 1995.

[5] B. M. Baker. Linear relaxations of the capacitated warehouse location problem. *European Journal of Operational Research Society*, 33:475–479, 1982.

[6] B. M. Baker. An partial dual algorithm for the capacitated warehouse location problem. *European Journal of Operational Research*, 23:48–56, 1986.

[7] E. Bartezzaghi, A. Colorni, and P. C. Palermo. A tree search algorithm for plant location problems. *European Journal of Operational Research*, 7:371–379, 1981.

[8] J. E. Beasley. An algorithim for solving large capacited warehouse location problems. *European Journal of Operational Research*, 33:314–325, 1988.

[9] J. E. Beasley. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65:383–399, 1993.

[10] C. T. Bornstein and H. B. Azlán. The use of reduction tests and simulated annealing for the capacitated plant location problem. *Location Science*, 6:67–81, 1998.

[11] C. T. Bornstein and G. R. Mateus. Dominance criteria for the capacitated warehouse location problem. *Operational Research Society*, 42:145–149, 1991.

[12] J. Branman. A note on pivoting in transportation codes. *European Journal of Operational Research*, 2:377–378, 1978.

[13] N. Christofides and J. E. Beasley. Extensions to a lagrangean relaxation approach for the capacitated warehouse location problem. *European Journal of Operational Research*, 12:19–28, 1983.

[14] G. Cornuejols, R. Sridharan, and J. M. Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. *European Journal of Operational Research*, 50:280–297, 1991.

[15] W. Domschke and A. Drexl. Add-heuristics - starting procedures for capacitated plant location models. *European Journal of Operational Research*, 21:47–53, 1985.

[16] W. Domschke and A. Drexl. *Location and Layout Planning: An International Bibliography*, volume 238 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 1985.

[17] R. L. Francis, L. F. McGinnis, and J. A. White. Locational analysis. *European Journal of Operational Research*, 12:220–252, 1983.

[18] M. Guignard-Spielberg and S. Kim. A strong lagrangean relaxation for capacitated plant location problem. Technical Report 56, Department of Statistics, the Wharton School, University of Pennsylvania, PA, 1983.

[19] S. K. Jacobsen. On the use of tree-indexing methods in transportation algorithms. *European Journal of Operational Research*, 2:54–65, 1978.

[20] S. K. Jacobsen. Heuristics for the capacitated plant location model. *European Journal of Operational Research*, 12:253–261, 1983.

[21] P. C. Jones, T. J. Lower, and G. Muller et al. Specially structured uncapacitated facility location problems. *Operations Research*, 43(4):661–669, July-August 1995.

[22] A. A. Kühn and M. J. Hamburger. A heuristic program for locating warehouses. *Mgmt Sci*, 9:643–666, 1963.

[23] G. R. Mateus. *Algoritmo Exato e Heurísticas para o Problema de Localização.* PhD thesis, Engenharia de Sistemas e Computação - COPPE/UFRJ, Rio de Janeiro, Brasil, 1986.

[24] M. B. Campêlo Neto. Testes de redução e heurística add/drop para o problema de localização capaditado. Master's thesis, Engenharia de Sistemas e Computação - COPPE/UFRJ, Rio de Janeiro, Brasil, 1993.

[25] T. J. Van Roy. A cross decomposition algorithm for capacitated facility location. *Operations Research*, 34:145–163, 1986.

[26] R. Sridharan. A survey of the capacitated plant location problem. Technical report, Graduate School of Industrial Administration, Carnegier-Mello University, Pittsburgh PA 15213, USA, 1984.

[27] L.A. Wolsey. Fundamental properties of certain discrete location problem. *Location Analysis of Public Facilities*, North-Holland:331–355, 1983.