



UNIVERSIDADE ESTADUAL PAULISTA

Programa de Pós-Graduação em Engenharia Elétrica

**Análise de Algoritmos tipo Branch and
Bound para Problemas de Designação
Generalizada**

Maira Pauline de Lima Alencar

Ilha Solteira - SP

UNIVERSIDADE ESTADUAL PAULISTA - UNESP
CAMPUS DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**Análise de Algoritmos tipo Branch and Bound
para Problemas de Designação Generalizada**

Maira Pauline de Lima Alencar

Prof. Dr. Francisco Villarreal Alvarado

Orientador

1210001418



Proc. 053/2003-NPD 160

UNESP - "CAMPUS DE ILHA SOLTEIRA"	
SERVIÇO TÉCN. DE BIBLIOTECA E DOCUMENTAÇÃO	
DATA DE CHEGADA	DATA DE TOMBO
02.09.03	30.09.03
REGISTRADO POR	TOMBO
Ailza	Te. 1418
AQUISIÇÃO	CLASSIFICAÇÃO
ilustrad anda R\$ 10,00	A368a

Dissertação submetida à Faculdade de Engenharia de Ilha Solteira - UNESP- como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

Ilha Solteira, julho de 2003.

UNIVERSIDADE ESTADUAL PAULISTA - UNESP
CAMPUS DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Análise de Algoritmos tipo Branch and Bound
para Problemas de Designação Generalizada

Maira Pauline de Lima Alencar

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação/Serviço Técnico
de Biblioteca e Documentação da FEIS/UNESP

A368a Alencar, Maira Pauline de Lima
Análise de algoritmos tipo Branch and Bound para problemas de designação
generalizada / Maira Pauline de Lima Alencar. -- Ilha Solteira : [s.n.], 2003
ix, 80 p.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia
de Ilha Solteira, 2003.

Orientador: Francisco Villarreal Alvarado
Bibliografia: p. 44-45

1. Algoritmos. 2. Programação dinâmica. 3. Programação inteira. 4. Otimização.

30400007?

Análise de Algoritmos Tipo Branch and Bound para Problemas de Designação Generalizada

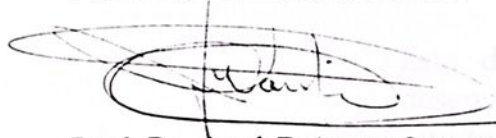
Maira Pauline de Lima Alencar

DISSERTAÇÃO SUBMETIDA À FACULDADE DE ENGENHARIA - CAMPUS DE ILHA SOLTEIRA – UNESP – COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA ELÉTRICA

COMISSÃO EXAMINADORA:


Prof. Dr. Francisco Villarreal Alvarado – orientador


Prof. Dr. Edivaldo Romanini


Prof. Dr. José Roberto Sanches Mantovani

Ilha Solteira-SP, julho de 2003.



Esta dissertação de Mestrado foi desenvolvida com apoio (bolsa de estudo e reserva técnica) da Fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP, processo nº 00/04292-6.

Agradecimento Especial

Ao meu Orientador, Prof. Dr. Francisco Villarreal Alvarado e ao Prof. Dr. Rubén Augusto Romero Lázaro, que com dedicação, paciência e compreensão me auxiliaram neste trabalho. A vocês meu carinho e admiração.

Ao meu marido, minha filha, sogra
e aos meus pais, por todo apoio e
carinho.

Dedico.



Agradecimento Especial

Ao meu Orientador, Prof. Dr. Francisco Villarreal Alvarado e ao Prof. Dr. Rubén Augusto Romero Lázaro, que com dedicação, paciência e compreensão me auxiliaram neste trabalho. A vocês meu carinho e admiração.

Agradeço aos meus pais, Marilén e Mauricio, pelo incentivo e apoio em todos os momentos da minha vida e pela confiança que em mim depositaram.

Ao meu irmão, pelo companheirismo, incentivo e por compartilhar comigo os momentos bons e ruins da minha vida.

A minha sogra, que com suas palavras de conforto e de estímulo me ajudou a superar vários obstáculos.

Aos meus amigos do Departamento que de certa forma contribuíram para a realização deste trabalho, Celso, Manoel, Fernando, Marlene, Cláudio e Elzeir.

Ao Prof. Dr. Rubén Augusto pela amizade, confiança e por contribuir para minha formação profissional e realização deste trabalho.

Aos funcionários da seção de pós-graduação da Unesp, pela atenção.

Aos funcionários do Departamento, Decio, Carlos Alex, Cláudio, Sueli e Irineide, pela amizade e auxílio.

Aos Professores do curso de Pós-Graduação em Engenharia Elétrica da UNESP, pela competência e apoio.



Agradecimentos

A concretização deste trabalho tornou-se possível mediante colaboração de muitas pessoas, isto implica apoio técnico, financeiro, ou até mesmo emocional. Portanto, gostaria de agradecer a todos que participaram de alguma forma na conclusão deste trabalho.

Agradeço aos meus pais, Marilza e Maurício, pelo incentivo e apoio em todos os momentos da minha vida e pela confiança que em mim depositaram.

Ao meu marido, pelo companheirismo, compreensão e por compartilhar dos melhores momentos da minha vida com serenidade e amor.

A minha sogra, que com suas palavras de conforto e de otimismo me ajudaram a superar vários obstáculos.

Aos meus amigos do Departamento que de certa forma contribuíram para a realização deste trabalho, Celso, Mara, Fernando, Marilene, Cilim e Elizete.

Ao Prof. Dr. Rubén Augusto pela amizade, confiança e por contribuir para minha formação profissional e realização deste trabalho.

Aos funcionários da seção de pós-graduação da Unesp, pela atenção.

Aos funcionários do Departamento, Deoclécio, Carlos Alex, Cristina, Sueli e Luzinete, pela amizade e auxílio.

Aos Professores do curso de Pós-Graduação em Engenharia Elétrica da UNESP, pela competência e apoio.

À Fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP,
pelo auxílio financeiro dado a esta pesquisa.

Àqueles que não foram citados, mas de alguma forma, contribuíram
para a realização deste trabalho.

E finalmente, agradeço a Deus, por mais esta conquista, por ter me
dado força e ter colocado as pessoas certas em meu caminho, possibilitando
este momento em que posso agradecer a todos por mais uma etapa vencida.



Resumo

Este trabalho faz uma análise de Algoritmos tipo Branch and Bound para o Problema de Designação Generalizada, analisando principalmente o algoritmo de Ross-Soland. Este faz parte da família de algoritmos Branch and Bound para resolver problemas do tipo linear inteiro 0-1. Entretanto a diferença fundamental desse algoritmo é que em cada nó da árvore de Branch and Bound não é resolvido um problema de programação linear como nos algoritmos Branch and Bound convencionais. Assim, em cada nó é encontrada uma limitante inferior do problema relaxado usando uma heurística e resolvendo problemas da Mochila. Neste trabalho é realizado uma análise crítica desse tipo de algoritmo, novas implementações em vários passos desse algoritmo e um teste com um sistema de médio porte.



Abstract

This work affectuates a analysis of the Branch and Bound algorithm for the generalized assignment problem, analysing the Ross-Soland algorithm mainly. This pertains to the Branch and Bound family to solve problems like integer linear programming 0-1. However, the fundamental difference of this work is that in each node of the Branch and Bound tree, it is not solve a linear programming problem as in the conventional Branch and Bound algorithms. Thus, in each node it is a lower limitant of the relaxed problem using a heuristic and solving knapsack problems. Therefore, it is effectuated a critical analysis of this algorithm, new implementation in several steps of this algorithm and a test with a medium size system.

3.3 - Análise Crítica do Algoritmo de Ross-Soland	20
3.3.1 - Algoritmo Fundamental	26
3.3.2 - O Algoritmo de Ross-Soland	32
3.4 - Testes de Eficiência do Algoritmo	24
3.4.1 - Programação Linear	24
Capítulo 4 - Testes e Resultados	
4.1 - Algoritmo Implementado	37
4.2 - Testes do Algoritmo	40
Considerações Finais	42
Referências Bibliográficas	44



Índice

Capítulo 1

1.1- Introdução	1
-----------------	---

Capítulo 2 –Problema de designação generalizada

2.1 - Introdução	5
2.2 - Formulação Matemática	6
2.3 - Técnicas de Solução Propostas	6

Capítulo 3 – Técnica de solução usando Relaxação de Restrições

3.1- Introdução	12
3.2 - Algoritmo Branch and Bound	13
3.3 - Análise Crítica do Algoritmo de Ross-Soland	16
3.3.1 - Estratégia Fundamental	16
3.3.2 - O algoritmo de Ross-Soland	22
3.4- Resolução do Problema da Mochila	24
3.4.1 - Programação Dinâmica	24

Capítulo 4 – Testes e Resultados

4.1- Algoritmo Implementado	37
4.2 - Teste com o Algoritmo	40

Considerações Finais	42
----------------------	----

Referências Bibliográficas	44
----------------------------	----



Apêndices

Apêndice A – Exemplo Ilustrativo do Algoritmo de Ross-Soland

A.1 - Exemplo	46
---------------	----

Apêndice B – Programação Dinâmica para um problema da Mochila Binário (Exemplo Ilustrativo)

B.1 - Exemplo	59
---------------	----

Apêndice C – Algoritmo Branch and Bound Especializado para o Problema da Mochila Binário

C.1 – Introdução	63
C.2 – Estratégia para resolver de forma eficiente o PL no problema da Mochila Binário	65
C.3 – Estratégia de separação	66
C.4 – Estratégia de escolha do Subproblema da Lista	66
C.5 – Estratégia de sondagem	67
C.6 – Outras características Importantes	67
C.7 – Sondagem usando Enumeração Explícita das variáveis livres	67
C.8 – Exemplos	68

Apêndice D – Análise Crítica do Algoritmo de Trick

D.1 – Introdução	71
D.2 – Estratégia Fundamental	72
D.3 – O Algoritmo de Trick	74
D.4 – Exemplo Ilustrativo	74



Capítulo 1

1.1 - Introdução

O problema de designação generalizada (GAP) consiste em encontrar o custo mínimo de designação de n tarefas para m agentes, de tal forma que cada tarefa seja designada somente por um único agente, sujeito às restrições de capacidade dos agentes [1].

Muitos problemas da vida real podem ser representados e solucionados através de subproblemas, os quais são do tipo de problemas de designação generalizada (GAP), desta forma este trabalho tem como objetivo analisar as técnicas de resolução existentes na literatura para este problema, com a finalidade de obter uma base teórica e conceitual sólida sobre estas técnicas e posteriormente formular um algoritmo especializado, eficaz para a solução do problema de designação generalizada.

Como exemplos de problemas que apresentam subproblemas do tipo GAP, pode-se citar: a localização de plantas com carga fixa, programação de



projetos de redes, projetos de redes de comunicação, problemas de designação de tarefas em redes de computadores [2]. Desse modo seja existem uma série de problemas dentro da engenharia elétrica que possuem subproblemas do tipo GAP, e apesar de ser um problema genérico, o GAP tem despertado muito interesse em pesquisadores de engenharia elétrica.

O problema GAP é um problema de otimização combinatorial do tipo NP-hard e muitos pesquisadores trabalham no desenvolvimento de algoritmos eficientes para esse tipo de problema, que em grande parte são apresentados como diferentes versões do algoritmo Branch and Bound.

Diferentes estratégias têm sido desenvolvidas para o algoritmo Branch and Bound em aplicações específicas, com a finalidade de solucionar problemas tais como planejamento da expansão de subestações e alimentadores em sistemas de distribuição radial e no problema de planejamento da expansão a longo prazo de sistemas de transmissão de energia elétrica.

O algoritmo Branch and Bound é um algoritmo enumerativo, cuja maneira de resolução se baseia na construção de uma árvore onde os nós representam os subproblemas e os ramos as novas restrições que devem ser consideradas. O problema começa a ser resolvido excluindo a integralidade das restrições do problema original, este problema relaxado é dividido sucessivamente em subproblemas, até que esteja completamente resolvido. É de grande importância para o algoritmo que seja encontrada uma solução incumbente, pois uma boa incumbente diminui o número de subproblemas que precisam ser analisados, devido ao fato que durante o processo de solução é gerada uma lista contendo as melhores soluções que serão comparadas com a incumbente inicial, sendo assim eliminadas da lista aquelas soluções de qualidade inferior a incumbente. O algoritmo mais eficiente é aquele que resolve o problema gerando e resolvendo a menor quantidade de subproblemas.



Na literatura especializada, os algoritmos Branch and Bound mais difundidos apresentam a desvantagem de resolver um problema de programação linear (PL) usando o método Simplex. Portanto, uma das técnicas analisadas nesta pesquisa é a técnica proposta por Ross-Soland, que possibilita uma rápida solução para o problema GAP, baseada na técnica de relaxação de uma parcela das restrições, utilizando o algoritmo Branch and Bound combinado com uma série de problemas da Mochila para determinar os limites inferiores, com isso fica diminuído o número de nós da árvore de Branch and Bound. Também foi desenvolvido um breve estudo sobre o algoritmo Branch and Bound para resolver o problema da Mochila, pois na literatura este algoritmo especializado é o mais adequado para a resolução de tais problemas. Entretanto, quando o número de variáveis é pequeno e o termo independente não for grande, um algoritmo de Programação Dinâmica seria o ideal, pois, estas características acontecem no problema da mochila gerado em cada nó da árvore de Branch and Bound do algoritmo analisado. Portanto neste trabalho, estes problemas foram resolvidos usando um algoritmo de programação dinâmica.

Existem outras técnicas de solução para o problema GAP, além da Relaxação baseada na Eliminação de Restrições, destacam-se: Relaxação do PL, Relaxação Lagrangeana, Relaxação Substituta e Decomposição Lagrangeana.

Alguns algoritmos desenvolvidos para a solução do problema GAP, como visto anteriormente, utilizam-se de técnicas de Branch and Bound e relaxações sobre algumas restrições do problema. Algumas propostas desenvolvem ainda restrições válidas que permitem encontrar limites mais eficientes para o problema relaxado. Uma das principais diferenças desses algoritmos está relacionada com a forma que são calculados os limitantes inferior e superior do algoritmo Branch and Bound.

Também foi analisado o algoritmo de Trick (Apêndice C), baseado na técnica de relaxação de uma parcela das restrições.



O algoritmo apresentado neste trabalho corresponde à família de técnicas de otimização clássica, isto é, um algoritmo Branch and Bound que utiliza a relaxação das restrições de desigualdade no problema GAP.

Finalmente após análise destes algoritmos, foi elaborado e implementado um algoritmo do tipo Branch and Bound, que integra de forma adequada as principais características dos algoritmos analisados para o problema GAP usando a experiência acumulada no estudo dos algoritmos antes mencionados.

Este trabalho está organizado da seguinte forma, no capítulo 2 segue uma breve introdução sobre o Problema de Designação Generalizada e algumas técnicas para resolvê-lo. Em seguida, no capítulo 3 é apresentado uma introdução sobre técnicas de solução usando relaxação das restrições, destacando a Análise do Algoritmo de Ross-Soland e Programação Dinâmica para a resolução dos Problemas da Mochila. Testes e resultados são encontrados no capítulo 4, assim como exemplos explicativos no seus respectivos apêndices.

2.2 - Introdução

O problema de designação generalizada (GAP) consiste em encontrar o custo mínimo de designação de n tarefas para m agentes, de tal forma que cada tarefa seja designada para um único agente e sujeito a restrições de capacidade de cada agente.

Existem várias técnicas para resolver o problema GAP, dentre elas: Relaxação do PL, Relaxação baseada na Eliminação de Restrições, Relaxação Lagrangeana, Relaxação Substituta e Decomposição Lagrangeana. Este trabalho irá analisar as técnicas: Relaxação do PL e a Relaxação baseada na eliminação de restrições, pois os algoritmos que são aqui analisados estão relacionados com estas técnicas.



A formulação do problema de designação generalizada é dada por

$$\text{Minimizar } Z = \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} \quad (1)$$

$$\text{sujeito a: } \sum_{j=1}^m x_{ij} \leq C_i, \quad i = 1, 2, \dots, n \quad (2)$$

Capítulo 2

Problema de designação generalizada

2.1 - Introdução

O problema de designação generalizada (GAP) consiste em encontrar o custo mínimo de designação de n tarefas para m agentes, de tal forma que cada tarefa seja designada para um único agente e sujeita a restrições de capacidade deste agente.

Existem várias técnicas para resolver o problema GAP, dentre elas: Relaxação do PL, Relaxação baseada na Eliminação de Restrições, Relaxação Lagrangeana, Relaxação Substituta e Decomposição Lagrangeana. Este trabalho dá ênfase as técnicas: Relaxação do PL e a Relaxação baseada na eliminação de restrições, pois os algoritmos que são aqui analisados estão relacionados com estas técnicas.



2.2 - Formulação Matemática

A formulação do problema de designação generalizada é dada por:

$$\text{Minimizar } z = \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} , \quad (1)$$

$$\text{sujeito à: } \sum_{j=1}^n a_{ij} x_{ij} \leq b_i , \quad \text{para todo } i \in I , \quad (2)$$

$$\sum_{i=1}^m x_{ij} = 1 \quad \text{para todo } j \in J , \quad (3)$$

$$x_{ij} \in \{0,1\}, \quad i \in I, \quad j \in J. \quad (4)$$

Nesta formulação $I \equiv \{1,2,\dots,m\}$ é o conjunto dos índices dos agentes, e $J \equiv \{1,2,\dots,n\}$ é o conjunto dos índices das tarefas, c_{ij} é o custo se o agente i é designado para a tarefa j , a_{ij} é o recurso requerido pelo agente i para realizar a tarefa j e $b_i > 0$ é a quantia de recurso disponível para o agente i .

A variável de decisão x_{ij} é igual a 1 se o agente i é designado para a tarefa j , caso contrário x_{ij} é igual à zero.

2.3 - Técnicas de Solução Propostas

Nesta seção são apresentadas algumas técnicas existentes na literatura [2] para o problema GAP.



2.3.1. Relaxação do PL

Primeiramente, o problema GAP será relaxado, através da substituição da restrição (4) por $x_{ij} \geq 0$. A solução do problema relaxado, geralmente contém algumas tarefas, as quais $x_{ij} \neq 0$ para algum i , ou seja, a tarefa j é dividida entre os vários agentes. Quando o número de agentes é menor se comparado com o número de tarefas, uma solução heurística pode ser um bom ponto de partida para resolver o problema. Esta fornece um limite superior, enquanto na relaxação do PL fornece o limite inferior. O limite no PL tende a aumentar quando o número de tarefas é maior se comparado ao número de agentes e quando a capacidade das restrições é bastante grande.

A desvantagem da formulação da Relaxação do PL no problema GAP é que ele é degenerativo. O tempo computacional de solução aumenta rapidamente com o crescimento da dimensão do problema.

2.3.2. Relaxação baseada na eliminação de restrições

O problema GAP é relaxado, primeiramente eliminando-se a restrição (2):

$$(RS) \text{ in } \left\{ \sum_i \sum_j c_{ij} x_{ij} \text{ s.a. } (3), (4) \right\}. \quad (5)$$

Em seguida, a solução para o problema é obtida designando cada tarefa para o agente de menor custo, em seguida a factibilidade é testada através da restrição (2). As restrições que forem violadas são transformadas em problemas da Mochila, a formulação destes problemas é dada por:



$$(PK_i) \min z_i = \sum_{j \in J_i} p_j y_{ij} \quad (6)$$

$$\text{s.a.} \quad \sum_{j \in J_i} a_{ij} y_{ij} \geq d_i, \quad (7)$$

$$y_{ij} = 0 \text{ ou } 1 \quad (8)$$

onde $d_i = \sum_j a_{ij} x_{ij}^* - b_i$, com x_{ij}^* representando a solução ótima do PL relaxado (5), p_j é a diferença entre o menor e o segundo menor custo (c_{ij}) e $J_i = \{j / x_{ij}^* = 1\}$.

A variável de separação x_{ij} é selecionada entre aquelas com $y_{ij} = 0$ da solução do (PK_i), deste modo:

$$t_{i^* j^*} = \max \left\{ p_j / (a_{ij} / (b_i - \sum_{j \in F_i} a_{ij} x_{ij})) \right\}. \quad (9)$$

Martello e Toth [2] usam um processo similar. Eles reformularam o problema como um problema de maximização, pois ambas as versões (maximização e minimização) são equivalentes. As restrições (3) são eliminadas para determinar o limite inferior.

2.3.3 - Relaxação Lagrangeana

Há duas relaxações Lagrangeanas para o problema GAP. A primeira é obtida dualizando as restrições do tipo (3). Isto reduz o problema em problemas da Mochila 0-1. A segunda relaxação é obtida pela dualização das restrições (2).

A primeira relaxação é explorada em Fisher, Jaikumar e Van Wassenhove [2]. As restrições (3) são relaxadas e o conjunto de

multiplicadores $(\mu_j, j = 1, \dots, n)$ são iguais ao segundo maior c_{ij} (problema de maximização).

$$(FJV) \max \left\{ \sum_i \sum_j (c_{ij} - \mu_{ij}) x_{ij} \mid (2), (4) \right\}. \quad (10)$$

A segunda relaxação Lagrangeana na qual os subproblemas são problemas GUBs [2] é equivalente a relaxação do PL. As duas fases heurísticas baseadas nesta segunda relaxação é apresentada por Klastorin [2]. Na primeira fase a aproximação modificada do subgradiente é utilizada na busca da solução ótima dual. Na segunda fase a busca pela árvore é executada na vizinhança da solução inicial a fim de encontrar uma melhor solução primal. Esta segunda fase não parece prover melhorias significativas à solução da primeira fase, além disto exige muito esforço computacional [2].

2.3.4 - Relaxação Substituta

Será brevemente comentada a relaxação substituta das restrições tipo (2) na formulação original, segundo Jörnsten e Näsberg [2]. Esta substituição fornece uma escolha múltipla do problema da Mochila,

$$(SR) \min \left\{ \sum_i \sum_j c_{ij} x_{ij} \mid \sum_i \sum_j \mu_i a_{ij} x_{ij} \leq \sum_i \mu_i b_i, (3), (4) \right\}. \quad (11)$$

Os limites obtidos por esta relaxação são tão bons quanto aqueles obtidos pela relaxação do PL do problema original (propriedade de integralidade) e aqueles que resultaram da relaxação Lagrangeana de restrições (2). Na prática o limite é ligeiramente inferior ao limite obtido pela relaxação Lagrangeana das restrições (3).

O segundo processo de Jörnsten e Värbrand [2] gera limites pela solução da relaxação substituta das restrições do Problema da Mochila e desigualdades válidas deste mesmo problema. A formulação é:

$$\max S(u, \delta) = \min \sum_i \sum_j c_{ij} x_{ij} \quad (12)$$

$$s.a. \sum_i u_i \sum_j x_{ij} a_{ij} + \sum_k \delta_k \sum_i \sum_j b_{kij} x_{ij} \leq \sum_i \mu_i b_i + \sum_k \delta_k d_k, \quad (13)$$

2.3.5 - Decomposição Lagrangeana

Outra aproximação é dada por Jörnsten e Näsberg (1986), que reformularam o problema como:

$$\min \alpha \sum_i \sum_j c_{ij} x_{ij} + \beta \sum_i \sum_j c_{ij} y_{ij} \quad (14)$$

$$s.a. \sum_j a_{ij} x_{ij} \leq b_i, \quad i \in I, \quad (15)$$

$$\sum_i y_{ij} = 1, \quad j \in J \quad (16)$$

$$x_{ij} = y_{ij}, \quad i \in I, \quad j \in J, \quad (17)$$

$$x_{ij}, y_{ij} \in \{0,1\}, \quad i \in I, \quad j \in J. \quad (18)$$

Usando a relaxação Lagrangeana na restrição (15) e observando que $\alpha + \beta = 1$, o problema é decomposto em problemas binários da Mochila para as variáveis x e problemas de semi designação ou GUBS nas variáveis y . O

método de subgradiente é utilizado para calcular os multiplicadores. Os dois problemas citados são mostrados à seguir:

$$PX(\mu) \min \sum_i \sum_j (\alpha c_{ij} - \mu_{ij}) x_{ij} \quad (19)$$

$$s.a. \sum_j a_{ij} x_{ij} \leq b_i, \quad i \in I, \quad (20)$$

$$x_{ij} \in \{0,1\}, \quad i \in I, \quad j \in J. \quad (21)$$

$$PX(\mu) \min \sum_i \sum_j (\beta c_{ij} - \mu_{ij}) y_{ij} \quad (22)$$

$$s.a. \sum_i y_{ij} = 1, \quad j \in J, \quad (23)$$

$$y_{ij} \in \{0,1\}, \quad i \in I, \quad j \in J. \quad (24)$$

O limite proposto é comparado com a relaxação Lagrangeana tradicional e conclui-se que é melhor que aquele obtido pela relaxação das restrições (3) ou (2) na formulação do problema original (GAP).



Capítulo 3

Técnicas de Solução Usando Relaxação das Restrições

3.1 - Introdução

A técnica de solução usando Relaxação de Restrições aqui analisada é o Algoritmo de Ross-Soland que utiliza eliminação de restrições tipo (2), ou seja, as restrições de desigualdade. Também é analisado o algoritmo de Trick (Apêndice C), o qual não foi implementado. Neste capítulo também é feita uma introdução sobre o algoritmo de Branch and Bound convencional para melhor compreensão desta técnica e também para o reconhecimento das diferenças e vantagens entre este e Ross-Soland.



3.2 - Algoritmo Branch and Bound

O algoritmo Branch and Bound é uma técnica enumerativa, cuja estrutura de solução está baseada na construção de uma árvore em que os nós representam os problemas candidatos e os ramos representam novas restrições que devem ser adicionadas aos problemas. Essa árvore permite que todas as soluções inteiras do problema original sejam analisadas implícita ou explicitamente. Assim, se o algoritmo converge então é possível encontrar a solução ótima global do problema linear inteiro binário. A estrutura geral do algoritmo apresenta três elementos fundamentais: separação, relaxação e sondagem.

Na **separação**, o problema original é dividido em subproblemas de forma que toda solução viável do problema original pertença somente a um dos subproblemas e uma solução viável de um dos subproblemas também seja viável para o problema original. Os subproblemas são chamados de descendentes e podem gerar novos subproblemas. A filosofia da separação (branching) consiste em usar a estratégia de dividir para conquistar. Assim, se não for possível resolver um problema, este é separado em dois novos problemas, gerando uma lista de problemas candidatos. A seguir, deve ser selecionado um dos problemas gerados, tentando encontrar uma solução viável, se a solução não for possível, dois novos problemas são gerados, em caso contrário foi encontrada uma nova solução para este problema. Nesse caso, é verificado se a nova solução é melhor que a incumbente (melhor solução viável encontrada), se for, é feita a troca de incumbente e continua o processo analisando o próximo problema da lista. A forma usual de separação de um problema inteiro binário é através de restrições contraditórias, isto é, encolhendo uma variável x_{ij} e adicionando a restrição $x_{ij} = 1$ a um dos problemas descendentes e $x_{ij} = 0$ ao outro.

A **relaxação** consiste em eliminar temporariamente algumas restrições do problema para facilitar a resolução do problema modificado. A condição



que deve ser satisfeita é que o conjunto de soluções viáveis do problema original deve também ser viável para o problema relaxado. Assim, por exemplo, as restrições de integralidade do problema são relaxadas transformando um problema linear inteiro em um problema de programação linear (PL). As restrições do tipo (2) ou (3) do problema GAP também podem ser eliminadas.

A **sondagem** está relacionada com a eliminação dos problemas gerados na estratégia Branch and Bound. Assim, geralmente é necessário avaliar os problemas gerados na lista para identificar os promissores e eliminar aqueles que podem ser descartados. Portanto, um problema pode ser descartado, sem direito a gerar novos descendentes, se satisfizer um dos critérios de sondagem:

Critério 1: Se o problema candidato relaxado é infactível. Significa que esse problema não tem solução inteira binária factível.

Critério 2: Se a solução do problema candidato relaxado é não inteira mas de pior qualidade que a melhor solução inteira conhecida, chamada de incumbente, significa que uma solução inteira do problema descendente deste que está sendo analisado deverá ser de pior qualidade que a incumbente corrente.

Critério 3: Se a solução do problema candidato relaxado é inteira. Nesse caso, a solução encontrada é a melhor solução inteira que existe na região factível do problema candidato analisado e, portanto, pode ser sondado. Assim, deve ser analisada a possibilidade de trocar de incumbente.

No algoritmo Branch and Bound existem quatro estratégias que definem as características e a qualidade desse tipo de algoritmo:

- (1) A forma de encontrar uma incumbente inicial;
- (2) A estratégia usada para selecionar o próximo subproblema da lista a ser analisado;
- (3) A estratégia usada para selecionar a variável que deve ser usada para separar um subproblema em dois subproblemas sucessores ;



(4) A forma de encontrar uma limitante inferior de cada subproblema avaliado.

A maioria dos algoritmos Branch and Bound usam um algoritmo heurístico rápido para encontrar uma incumbente inicial. Uma incumbente inicial de excelente qualidade faz com que o teste de sondagem se torne mais eficiente. No caso básico, sem incumbente, é assumido simplesmente uma incumbente de valor muito elevado.

Existem também várias estratégias para selecionar o próximo subproblema da lista que deve ser analisado. O caso mais simples consiste em usar a regra LIFO, em que o último subproblema que foi armazenado na lista é o primeiro a ser selecionado. Essa estratégia prioriza a busca em profundidade permitindo encontrar rapidamente soluções inteiras não necessariamente de boa qualidade e também produzindo uma lista de subproblemas armazenados de tamanho reduzido.

Os algoritmos Branch and Bound mais difundidos na literatura especializada para resolver problemas de programação linear inteira mista são aqueles que resolvem um PL usando o método Simplex para encontrar a limitante inferior de cada subproblema analisado. Esta estratégia apresenta a vantagem adicional de conhecer os valores não inteiros das variáveis inteiras na solução ótima de cada subproblema que é um problema relaxado do problema original. Entretanto, apresenta a desvantagem de resolver um problema de PL. Na literatura especializada também existem outras estratégias como o Algoritmo de Ross-Soland. Levando em consideração as características mencionadas anteriormente para um algoritmo Branch and Bound, é considerado que o algoritmo de Ross-Soland possui as seguintes características:

(1) Não existe incumbente inicial (a incumbente inicial é um número muito grande);

(2) Usa a regra LIFO para selecionar o próximo subproblema que deve ser avaliado, sendo que o subproblema com $x_j = 1$ é analisado primeiramente, por isso é armazenado em último lugar;



(3) Usa uma estratégia específica para identificar a variável que deve ser usada para gerar os dois subproblemas sucessores sendo que o subproblema com $x_j = 0$ é armazenado primeiro na lista de subproblemas;

(4) usa uma estratégia específica para encontrar uma limitante inferior de cada subproblema que está sendo avaliado, isto é, não usa o método Simplex para resolver um PL para encontrar a limitante inferior de cada subproblema selecionado para a avaliação.

3.3 - Análise Crítica do Algoritmo de Ross-Soland

O algoritmo de Ross-Soland encontra a limitante inferior de cada nó através da resolução de Problemas da Mochila binários em contraposição ao algoritmo Branch and Bound convencional, que resolve PLs (programas lineares) usando o Método Simplex. Ou seja, o algoritmo de Ross-Soland aproveita a estrutura do problema e procura de forma direta soluções inteiras factíveis que permitam sondar o nó da árvore de Branch and Bound.

O algoritmo de Ross-Soland é eficiente porque os Problemas da Mochila binários gerados são resolvidos com um mínimo de esforço, e pela estratégia LIFO escolhida para resolver os subproblemas que correspondem à árvore de Branch and Bound.

3.3.1 - Estratégia Fundamental

Dado o problema GAP(P):

$$\text{Minimizar } z = \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij}, \quad (1)$$



$$(P) \text{ sujeito à: } \sum_{j=1}^n a_{ij} x_{ij} \leq b_i, \text{ para todo } i \in I, \quad (2)$$

$$\sum_{i=1}^m x_{ij} = 1 \text{ para todo } j \in J, \quad (3)$$

$$x_{ij} \in \{0,1\}, \quad i \in I, \quad j \in J. \quad (4)$$

Primeiramente, será feita uma relaxação sobre o problema (P), eliminando a restrição (2), como pode ser observado a seguir:

$$\text{Minimizar } z = \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij}, \quad (1)$$

$$(PR) \text{ sujeito à } \sum_{i=1}^m x_{ij} = 1 \text{ para todo } j \in J, \quad (3)$$

$$x_{ij} \in \{0,1\}.$$

Uma solução do PR é encontrada designando cada tarefa ao agente que realiza a tarefa pelo menor custo c_{ij} , ou seja, considerando os conjuntos $I \equiv \{1,2,\dots,m\}$ dos índices dos agentes, e $J \equiv \{1,2,\dots,n\}$ dos índices das tarefas. Uma solução ótima do PR é encontrada identificando para cada tarefa j o agente $p \in I$ que deve realizá-la com menor custo. Portanto foi obtido o seguinte:

$$c_{pj} = \min_{i \in I} \{c_{ij}\} \Rightarrow x_{pj} = 1 \text{ e } x_{ij} = 0 \quad \forall i \neq p$$

Essa solução ótima do PR produz o seguinte valor da função objetivo:

$$z_{LI} = \sum_{j \in J} c_{pj}.$$

A solução do PR, será utilizada em cada restrição (2) em (P) com a finalidade de testar a factibilidade. As restrições que forem violadas serão

então reformuladas transformando-se em Problemas da Mochila e assim melhorando o limite inferior. Esta reformulação é dada a seguir:

$$\text{Para: } J_i \equiv \{x_{ij} = 1, j \in J\} \text{ e } I' \equiv \left\{ i : \sum_{j \in J_i} a_{ij} x_{ij} > b_i, i \in I \right\}.$$

(PK_i):

$$\text{Minimizar } z_i = \sum_{j \in J_i} p_j y_{ij}, \quad (6)$$

$$\text{sujeito a: } \sum_{j \in J_i} a_{ij} y_{ij} \geq d_i, \text{ para todo } i \in I, \quad (7)$$

$$y_{ij} = 0 \text{ ou } 1, \quad (8)$$

em que:

$$d_i = \sum_{j \in J_i} a_{ij} x_{ij} - b_i,$$

$$p_j = c_{sj} - c_{ij} = \min_{K \in I - \{i\}} \{c_{kj} - c_{ij}\}.$$

Cada problema (PK_i) é um problema binário da Mochila, sua solução identifica a tarefa que deve ser redesignada do agente i para outro agente para transformar a restrição i em factível, isto é, para que seja satisfeita a capacidade de trabalho do agente i. Assim, s identifica o agente para o qual pode ser transferida a tarefa j, o parâmetro d_i representa o valor da capacidade violada, isto é, indica o valor em que foi excedido a capacidade de trabalho do agente. O parâmetro p_j representa a penalidade mínima na função objetivo que deve acontecer se a tarefa j é redesignada do agente i para o agente s. A solução ótima de (PK_i) identifica a tarefa j que deve ser redesignada e essa informação aparece em y_{ij}^* produzindo o menor

incremento de z_i^* . Assim o limite inferior Z pode ser aumentado pela soma de valores das funções objetivos obtidas pela solução dos (PK_i), deste modo o limite inferior será:

$$LIM = Z + \sum_{i \in I'} z_i^*.$$

Desse modo, uma LI do subproblema avaliado pode ser melhorado incrementando os valores ótimos da função objetivo dos problemas da mochila binários.

A solução ótima y_{ij}^* de cada problema da mochila é utilizada para realizar a transferência de tarefas, e portanto, atualizar a proposta de solução x_{ij} . Assim, uma variável $y_{ij}^* = 1$ deve produzir a seguinte atualização:

$$y_{ij}^* = 1 \Rightarrow x_{ij} = 0 \text{ e } x_{sj} = 1.$$

A factibilidade é checada com esta nova solução, se não existir restrição de capacidade dos agentes violada, uma solução factível foi encontrada, caso contrário, existe disponível apenas uma LI do subproblema, com algumas restrições de capacidade de agentes violados, deste modo, o problema deverá ser dividido em dois subproblemas. A variável de separação escolhida x_{i^*,j^*} , será uma entre aquelas com $y_{ij}^* = 0$ obtidas na solução ótima do problema (PK_i), a escolha será feita através da fórmula:

$$t_{i^*,j^*} = \max \left\{ p_j / (a_{ij} / (b_i - \sum_{j \in F_i} a_{ij} x_{ij})) \right\}, \quad (9)$$

onde F_i denota o conjunto de tarefas designadas para o agente i . O problema candidato com a variável fixada em 1 será o próximo a ser resolvido.

As soluções obtidas através dos problemas da Mochila (PK_i) contribuem de duas maneiras para sondar o problema candidato:

- É possível melhorar a limitante inferior da função objetivo;

- A redesignação de tarefas pode levar a soluções factíveis para (P). Este caso pode acontecer com frequência se os recursos disponíveis não são muito restritivos.

Justificativa para o uso do Limitante Inferior Modificado(LIM)

A forma de calcular a (LIM) para o problema (P) apresentada anteriormente, na verdade é um caso especial de relaxação Lagrangeana muito usada para encontrar (LI) de problemas inteiros. Portanto, fazendo uma análise da proposta em termos de relaxação lagrangeana.

Transformando o problema anterior usando multiplicadores de lagrange λ_j para as restrições tipo (3) de (P), e assim o problema (P) modificado assume a seguinte forma:

$$\begin{aligned} \min \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{j \in J} \lambda_j (1 - \sum_{i \in I} x_{ij}) \\ \text{s.a.} \quad & \sum_{j \in J} a_{ij} x_{ij} \leq b_i, \quad \forall \quad i \in I \quad (PR_\lambda) \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

Através de manipulações, a função objetivo pode ser escrita como:

$$\sum_{j \in J} \lambda_j - \max \left[\sum_{j \in J} \sum_{i \in I} (\lambda_j - c_{ij}) x_{ij} \right] \quad (\beta)$$

Assim o (PR_λ) pode ser separado em vários problemas (PK_i) 0-1 para cada $i \in I$. Logicamente, o valor do limitante inferior obtido por (PR_λ) depende dos valores λ_j escolhidos. A melhor escolha para esses λ_j são os multiplicadores duais ótimos das restrições (3) quando se resolve o problema (P) após relaxar a integralidade dos x_{ij} . Entretanto, calcular esses valores

ótimos dos λ_j implica um esforço de processamento elevado porque um PL deve ser resolvido.

Outra estratégia para encontrar valores adequados para os λ_j consiste em encontrar os multiplicadores duais ótimos do seguinte PL canalizado:

$$\left\{ \begin{array}{l} \min \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\ \text{s.a. } \sum_{i \in I} x_{ij} = 1, \quad \forall j \in J \Rightarrow \lambda_j \\ 0 \leq x_{ij} \leq 1. \end{array} \right. \quad (PR_l)$$

É importante observar que cada multiplicador dual ótimo λ_j de (PR_l) deve ter um valor dentro do intervalo, $c_{1j} \leq \lambda_j \leq c_{2j}$ em que c_{1j} e c_{2j} são os dois valores menores de c_{ij} , $\forall i \in I$, sendo c_{1j} o menor entre eles. Assim, é possível calcular os valores ótimos λ_j de (PR_l) facilmente.

O resultado mais importante é que a limitante inferior (LI) de (P) é idêntico ao limitante inferior fornecido por (PR_λ) quando cada λ_j é igual a c_{2j} o segundo menor valor de c_{ij} para todo $i \in I$. Esta afirmativa é fácil de ser verificada, basta substituir $(1 - x_{ij})$ por y_{ij} em cada (PK_i) , lembrando que cada $p_j = c_{2j} - c_{1j}$ para todo $j \in J$ e observar que cada $x_{ij} = 0$ no problema relaxado (PR) também podem ser iguais a zero no (PR_λ) .

Portanto, resolvendo o (PR) e depois os (PK_i) , $\forall i \in I'$ é encontrada uma limitante inferior igual ao fornecido por (PR_λ) , que claramente, é uma limitante inferior válida para o problema (P).

É importante notar que a medida em que o algoritmo Branch and Bound evolui, particularmente após a resolução do primeiro problema, soluções mais refinadas para o problema (PR) podem ser encontradas. Assim, é possível identificar os dois menores custos dos agentes para uma

tarefa e verificar que algumas variáveis devem assumir valores iguais a zero porque $a_{ij} > b_i - \sum_{j \in F_i} a_{ij} x_{ij}$ em que F_i é o conjunto de tarefas designadas ao agente i na solução corrente. Esta estratégia pode aumentar a limitante inferior de (PR) assim como as penalidades p_j na função objetivo dos (PK_i).

3.3.2 - O Algoritmo de Ross-Soland

Passo 1:

Inicialização.

Escolher a incumbente inicial $z \rightarrow \infty$ e inicializar a lista dos subproblemas candidatos com o problema original (P) relaxando as restrições (2) para obter o subproblema corrente chamado (PR).

Passo 2:

Teste de convergência: Se a lista dos candidatos é vazia então o processo terminou e a solução incumbente atual é a solução ótima do problema (P); caso contrário prossiga.

Passo 3:

Dentre os subproblemas candidatos ainda não sondados, escolher qual será o próximo a ser examinado, usando LIFO e escolhendo primeiro aquele subproblema com o valor de $x_{ij}=1$.

Resolver o subproblema relaxado corrente (PR)_c encontrando uma limitante inferior para o (PR)_c, da seguinte forma:

(a) Encontrando uma limitante inferior inicial:

Encontrar uma LI para o (PR)_c usando uma estratégia heurística, considerando que a tarefa j seja realizada pelo agente i de menor custo, cujo valor corresponde a $x_{ij}=1$ e os demais iguais a zero.

(b) Verificar se a LI do passo anterior é factível. Se for factível, ir ao passo 4. Caso contrário ir ao passo (c).

(c) Construção e solução dos Problemas da Mochila:

Para cada restrição tipo (2) violada, construir e resolver um Problema da Mochila.

(d) Atualizar a limitante inferior e a nova solução:

$$LIM = Z + \sum_{i \in I'} z_i^*.$$

Caso a nova solução seja factível ir ao passo 4, caso contrário ir ao passo 5.

Passo 4:

Testes de Sondagem:

O subproblema corrente é factível, portanto:

Se a nova solução factível encontrada é menor que a incumbente $\Rightarrow$ atualizar a incumbente. Nesse caso, tentar sondar outros subproblemas usando a nova incumbente. Ir ao passo 2.

Passo 5:

Separação do subproblema:

Escolher a variável de separação para gerar dois novos subproblemas. Existe uma proposta para escolher a variável para a separação, colocando o subproblema com $x_{ij}=1$ em último lugar na lista de candidatos. Ir ao passo 2.

Observação:

No apêndice A o exemplo A1 ilustra a técnica de Ross-Soland.



3.4 – Resolução do Problema da Mochila

Para resolver problemas da mochila binários sem características específicas, a literatura especializada considera que um algoritmo Branch and Bound especializado é o mais adequado. Entretanto, quando o número de variáveis é pequeno e o termo independente não for tão grande, um algoritmo de programação dinâmica pode ser mais rápido. Essas características acontecem no problema da mochila que aparece em cada nó da árvore de Branch and Bound do algoritmo analisado. Portanto, neste trabalho, os problemas da mochila que estão envolvidos foram resolvidos usando um algoritmo de programação dinâmica.

3.4.1 – Programação Dinâmica

Processo de Decisão Sequencial

Existem aplicações em que é preciso tomar uma sequência de decisões, uma após a outra. Neste tipo de caso o sistema assume um estado particular no início de cada período. Deste modo, em cada estado, uma decisão é tomada. Assim acontecem dois fatores, ambos dependentes do estado atual do sistema:

- (a) um custo imediato é assumido;
- (b) a decisão leva o sistema para o outro estado no próximo período.

O mesmo processo é repetido por n períodos. Portanto uma sequência de decisões são tomadas em instantes discretos de tempo.

A idéia é otimizar um objetivo que leva o sistema ao estado final desejado. O aspecto importante deste tipo de problema é que as decisões



sequenciais não podem ser tratadas de forma isolada, pela necessidade de um balanço das decisões implementadas.

Portanto é obtido um problema multi estágio envolvendo n estágios finitos. O sistema pode estar em vários estados possíveis. No decorrer do tempo, o estado do sistema muda dependendo da sequência de decisões tomadas e do estado inicial. Como os estados mudam, a técnica de otimização deste tipo de sistema é chamada de Programação Dinâmica (PD).

A seleção que determina a ação tomada em cada ponto de decisão é chamada de "Policy" (política). O objetivo da PD é determinar uma política ótima que minimize os custos totais em todos os estágios. A PD resolve este problema de forma recursiva para os n estágios. Assim, em cada ponto de decisão deve minimizar a soma dos custos correntes e do melhor custo futuro.

Um exemplo deste tipo de problema é o "shortest route problem". A informação necessária para usar PD em um determinado problema é o conjunto de todos os estados possíveis do sistema (finito). O conjunto de todas as decisões que podem ser tomadas em cada estado e o custo da decisão local assim como o próximo estado que o sistema deve atingir após cada decisão. Com este tipo de informação é conhecida a dinâmica do sistema. Como a transição de estados acontece em pontos discretos então tal sistema é chamado de sistema dinâmico discreto no tempo e também é assumido que a função objetivo é aditiva no tempo. Este tipo de problema pode ser resolvido usando técnicas de PD que permitem aumentar a política ótima (decisão ótima em cada estado do sistema).

Exemplo: Problema da mochila com coeficientes não negativos:

Nesta versão, para uma mochila de volume w_0 , pretende-se maximizar o custo dos produtos que são colocados na mochila.

Este problema pode ser formulado como um processo de decisão sequencial considerando que em cada passo é colocado um produto na



mochila. Neste contexto, o estado do sistema em qualquer passo (ponto de decisão) ou estágio está dado pelo volume disponível da mochila. Deste modo, existem $w_0 + 1$ estados possíveis do sistema. Em qualquer estágio, um objeto (produto) é candidato a ser colocado na mochila se o volume desse objeto é menor que o volume disponível.

A decisão neste estágio deve ser colocar um produto na mochila. Existe uma compensação (lucro) que deve ser incorporada na função objetivo pela decisão tomada de colocar um produto na mochila. A decisão de colocar um produto na mochila reduz o volume da mochila e o novo estado é determinado pelo volume ainda disponível na mochila.

Como exemplo considera-se que em um determinado estado do problema da mochila existe um volume disponível de 20 unidades e $n=6$ produtos fora da mochila, com os seguintes valores de volume e lucro:

N ₀	Volume	lucro
1	14	700
2	8	900
3	5	500
4	4	600
5	22	2700
6	25	3500

Os produtos 5 e 6 são descartados porque têm volume maior que o volume disponível na mochila e, portanto, os produtos 1, 2, 3 e 4 estão disponíveis para serem colocados na mochila. Assim, existem 4 decisões no estado corrente, isto é, colocar um dos produtos de 1 a 4. O resultado dessas decisões levam aos seguintes estados:



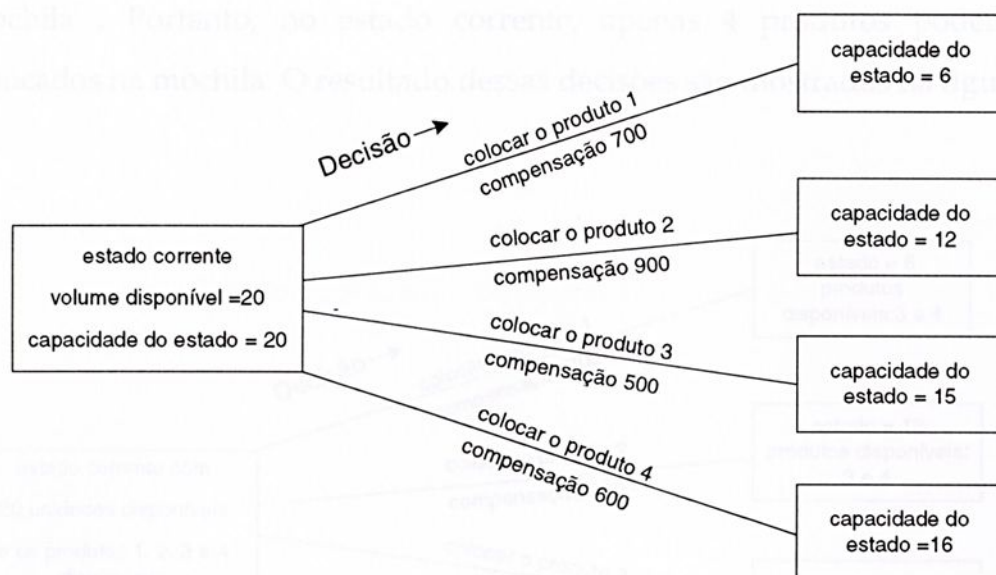


Figura 1: Representação de decisões.

Exemplo: Resolvendo o problema da mochila 0-1 usando Programação Dinâmica:

Neste caso existe apenas uma unidade de cada produto. Assim, em qualquer estágio, um produto é considerado candidato a ser colocado na mochila se:

- Seu volume é menor ou igual ao volume ainda disponível na mochila no estágio corrente;
- O produto que ainda não foi colocado na mochila.

Portanto, o estado do sistema pode ser representado pelo volume ainda disponível na mochila e pelo subconjunto de produtos que ainda não foram colocados na mochila e que têm volume menor ou igual ao volume ainda disponível na mochila. A decisão nesse estado é colocar um produto na mochila e passar para o novo estado.

Como exemplos é considerado que no estado corrente o volume ainda disponível da mochila é 20 unidades e existem ainda 6 produtos fora da

mochila . Portanto, no estado corrente, apenas 4 produtos podem se colocados na mochila. O resultado dessas decisões são mostrados na figura 2.

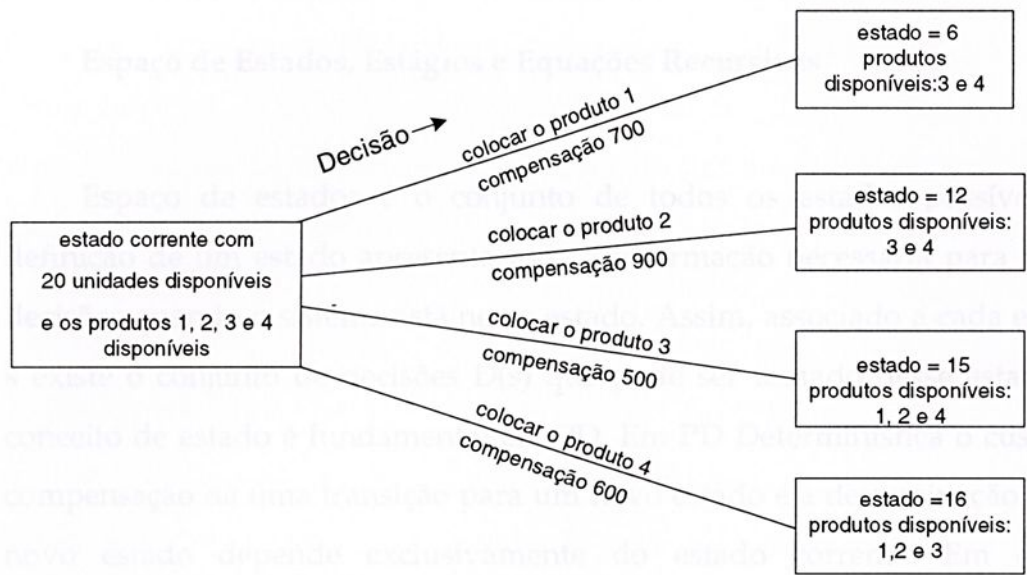


Figura 2: Representação de decisões

É notado que no problema da mochila padrão (existem vários produtos de cada tipo) com volume w_0 e com formato de decisão sequencial, é preciso de $(w_0 + 1)$ estados. Por outro lado, para o problema da mochila 0-1 é necessário $n(w_0 + 1)$ estados para o problema com n produtos porque deve-se manter o subconjunto de produtos ainda não colocados na mochila na definição do estado desse tipo de problema da mochila.

Na resolução de um problema usando PD deve ser definido os estados de forma adequada levando em conta a estrutura do problema. A definição de estados deve conter sempre informação suficiente para realizar todas as decisões possíveis em qualquer estado e sem ambiguidade na tomada de decisões até terminar o processo.

Assim, o resultado de uma decisão define a compensação imediata obtida e a transição para o novo estado conhecido. Este ramo da PD em que

não existe incerteza e existe informação suficiente sobre o efeito de cada decisão em cada estado do sistema é chamado de Programação Dinâmica Determinística.

Espaço de Estados, Estágios e Equações Recursivas

Espaço de estados é o conjunto de todos os estados possíveis. A definição de um estado apresenta toda a informação necessária para tomar decisões quando o sistema está nesse estado. Assim, associado a cada estado s existe o conjunto de decisões $D(s)$ que pode ser tomado nesse estado. O conceito de estado é fundamental em PD. Em PD Determinística o custo de compensação de uma transição para um novo estado e a determinação desse novo estado depende exclusivamente do estado corrente. Em outras palavras, não depende da trajetória percorrida para chegar ao estado corrente. Esta propriedade é chamada propriedade Markoviana [5].

Como os estados estão nos pontos em que são tomadas as decisões, o processo de decisão sequencial implica a transição de um estado para o estado seguinte. A sequência de estados visitados pelo sistema até terminar o processo é uma trajetória chamada de realização e depende do estado inicial e da estratégia (política) adotada. Esta trajetória é mostrada na figura 3 onde os nós representam os estados, os arcos as decisões e a função objetivo a forma das compensações em todos os arcos da trajetória.

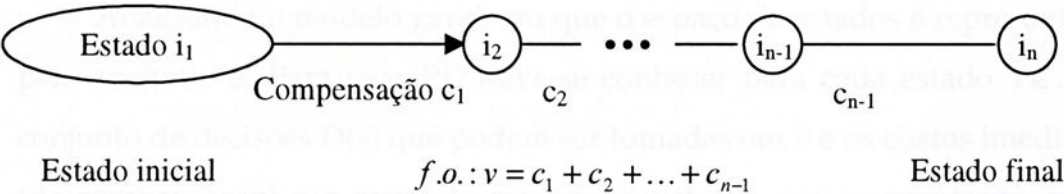


Figura 3: Trajetória de Estado.

Para resolver um problema usando PD deve ser identificado o espaço de estados e essa tarefa exige muito engenho.

Cada modelo de PD possui estados que podem ser utilizados para a resolução deste modelo, mas em alguns destes modelos existem também estágios. Os estágios são modelos de decisão sequencial, os quais são formados por estados que estão ordenados de alguma forma. Nesses modelos, as transições sempre acontecem de um estado de algum estágio para um estado do estágio seguinte. Assim, o processo é iniciado em um determinado estado do estágio 1 e faz a transição para os estágios 2, 3, etc. Esses modelos são usados quando as decisões são tomadas de forma periódica, isto é, uma decisão por período e no início do período.

Se um processo de decisão sequencial tem uma organização natural em estágios então o espaço de estados S pode ser particionado na forma $S_1 \cup S_2 \cup \dots \cup S_n$ e o sistema sempre se desloca de S_1 a S_2 , de S_2 para S_3 , e assim sucessivamente até S_{n-1} a S_n . Todos os estados em S_n (estágio n) são estados finais, isto é, o processo termina quando é atingido um estado em S_n .

Considerando um processo sequencial que tem final definido, ou seja, aquele em que a forma das decisões começa e termina, isto é, modelos com horizonte de planejamento finito. Se a modelagem é realizada em estágios com N estágios então começa no estágio 1 e termina no estágio N após $(N-1)$ transições. Em caso contrário, o modelo termina quando é encontrado o estado final desejado ou após esgotar um número determinado de transições.

Analisando o modelo geral, em que o espaço de estados é representado pelo conjunto S . Para usar PD deve-se conhecer para cada estado $s \in S$, o conjunto de decisões $D(s)$ que podem ser tomadas em S e os custos imediatos (de compensação) e o seguinte estado do sistema que é encontrado como consequência de selecionar cada uma das decisões. A solução deve especificar as decisões selecionadas de $D(s)$ para cada $s \in S$ e essas decisões



são chamadas de estratégia. A estratégia define completamente a sequência de decisões que são tomadas após cada transição no processo de solução.

Função de Valor Ótimo

Para cada estado $s \in S$, seja $f(s)$ igual ao custo mínimo (ou máximo) total para adotar uma estratégia ótima iniciando do estado inicial S . A função $f(s)$ definida sobre o espaço de estados S é chamada de função de valor ótimo (FVO) (Optimum Value Function). Supor que o problema define o estado terminal desejado, isto é, o processo termina se atinge um dos estados do conjunto terminal. Assim, quando o estado terminal for atingido não podem ser tomadas novas decisões e o custo futuro (de compensação) é igual a zero. Portanto, se o sistema é iniciado em um desses estados finais então

$$f(s) = 0, \quad (25)$$

se s for o estado final. A relação anterior é chamada de condição de fronteira (boundary conditions) que satisfaz uma FVO.

Princípio de Otimalidade

Uma técnica de PD é consequência de um princípio muito simples chamado Princípio de Otimalidade (Principle of Optimality). Será apresentado 3 versões equivalentes deste princípio:

Princípio de Otimalidade –versão 1

Uma estratégia ótima tem a propriedade de que se S é um estado encontrado inicialmente através de uma estratégia ótima, então S_0 será a parcela desta realização de s até terminar o processo, ou seja, representa uma realização ótima se o processo é iniciado com o sistema em s .



Princípio de Otimalidade – versão 2

Dado o estado corrente em um determinado momento do processo, as decisões ótimas de cada estado encontradas no futuro não dependem dos estados passados ou das decisões realizadas nos estados passados.

Princípio de Otimalidade – versão 3

Uma estratégia ótima tem a propriedade de que se os estados iniciais e finais são ótimos, então as outras decisões intermediárias devem constituir uma estratégia ótima como consequência das transições a partir da primeira transição.

Em outras palavras, conhecido o estado corrente de uma realização ótima em algum instante do processo, uma estratégia ótima para o restante do processo é independente da estratégia adotada no passado. Assim, o conhecimento do estado corrente do sistema elimina a necessidade do conhecimento do comportamento prévio para determinar a sequência ótima do futuro. Esta propriedade é uma consequência da propriedade Markoviana. Deve-se assumir também o princípio de aditividade da função objetivo.

A Equação Funcional para a FVO

Considerando um problema de minimização. Seja s_0 o estado corrente do sistema em um determinado ponto do processo. Supor que existem k decisões possíveis nesse estado, e, é escolhida uma dessas decisões. Supor que o custo da escolha é igual a c_t e o sistema passa para o estado s_t se t é escolhido no estado corrente para $t = 1, \dots, k$. Como definido anteriormente tem-se o seguinte:

$f(s_t)$ = custo total mínimo por adotar uma estratégia ótima iniciando o processo a partir de s_t , para $t = 0, 1, \dots, k$.

Para $t = 1, \dots, k$, se for escolhida a decisão t e seguir uma estratégia ótima no futuro, então o custo total nesse momento é $c_t + f(s_t)$. O motivo é simples: c_t é o custo pela decisão escolhida e essa decisão leva o sistema ao estado s_t . Como $f(s_t)$ é o custo para o processo iniciado no estado s_t e seguindo uma estratégia ótima no futuro então pela hipótese de aditividade o custo total até terminar o processo é a soma desses custos, isto é, $c_t + f(s_t)$.

Portanto, uma decisão ótima no estado corrente s_0 consiste em escolher um t entre 1 e k que minimize $c_t + f(s_t)$. Assim é apresentada a seguinte equação:

$$f(s_0) = \min\{c_t + f(s_t) : t = 1, \dots, k\} \quad (26)$$

A relação anterior é chamada de equação de otimalidade.

Se os valores de $f(s_1), \dots, f(s_k)$ são conhecidos então (26) pode ser usado para determinar o valor de $f(s_0)$ e determinar a decisão ótima em s_0 . Essa proposta é chamada fixação recursiva (recursive fixing) porque fixa os valores de $f(s_0)$ a partir dos valores conhecidos de $f(s_1), \dots, f(s_k)$.

De (25) tem-se que $f(s) = 0$ para cada estado final s . Iniciando por um valor conhecido de $f(s)$ em estados finais (obtidos das condições de fronteira) é calculado os valores das FVO em todos os estados usando (26) retrocedendo um movimento em cada passo. Este método de encontrar os valores de FVO em todos os estados é chamado de técnica recursiva ("recursive technique") porque começa o processo nos estados terminais e realiza um movimento de retrocesso em cada passo.

Se os estados são agrupados em estágios em um problema então a condição de fronteira estabelece que $f(s) = 0$ para todos os estados s no estágio final N . Em tais problemas formulados em estágios, a técnica recursiva é iniciada no estágio N e em cada passo é realizado um retrocesso de um estágio para outro, encontrando em cada passo o valor de FVO e a decisão ótima para cada estado desse estágio.

Para resolver um problema usando PD deve-se encontrar as equações funcionais para esse problema. Esse processo é crítico porque deve ser verificada a possibilidade de resolver um problema usando PD.

O produto final da aplicação de PD é uma lista de valores de FVO e a decisão ótima em cada estado possível do sistema.

Programação Dinâmica para o Problema da Mochila 0-1

Neste caso existe apenas um produto de cada tipo. Existe apenas um tipo (cópia) de cada produto e $w_0, w_1, \dots, w_k$ são positivos.

Usando PD então o problema consiste em um processo de colocar na mochila um produto em cada passo. Entretanto, como existe apenas uma cópia de cada produto, a definição do estado em qualquer passo do algoritmo deve ter disponível a informação do volume ainda disponível da mochila e dos produtos disponíveis para alocação nesse passo. Assim, o processo é representado através de n estágios. Para $k = 1, \dots, n$, os estados no estágio k são identificados por (k, w) onde $0 \leq w \leq w_0$ representa o volume ainda disponível da mochila no estágio k . Para qualquer estado (k, w) no estágio k existem pelo menos duas decisões que podem ser realizadas: (i) decidir não colocar o produto k na mochila (somente possível se $w_k > w$) com uma recompensa de zero e passagem para o estado $(k+1, w)$ no estágio $(k+1)$; ou (ii) alocar o produto na mochila (possível somente se $w \geq w_k$) com uma recompensa de V_k e passagem para o estado $(k+1, w - w_k)$ no estágio $k+1$. Isto produz uma estrutura de estágio artificial com n estágios, sendo que no estágio k foi decidida a inclusão ou exclusão do produto k para $k = 1, \dots, n$. Assim, o destino de cada produto é considerado em um único estágio. As decisões disponíveis no estado e a transição do estado resultante são mostrados na figura 4.



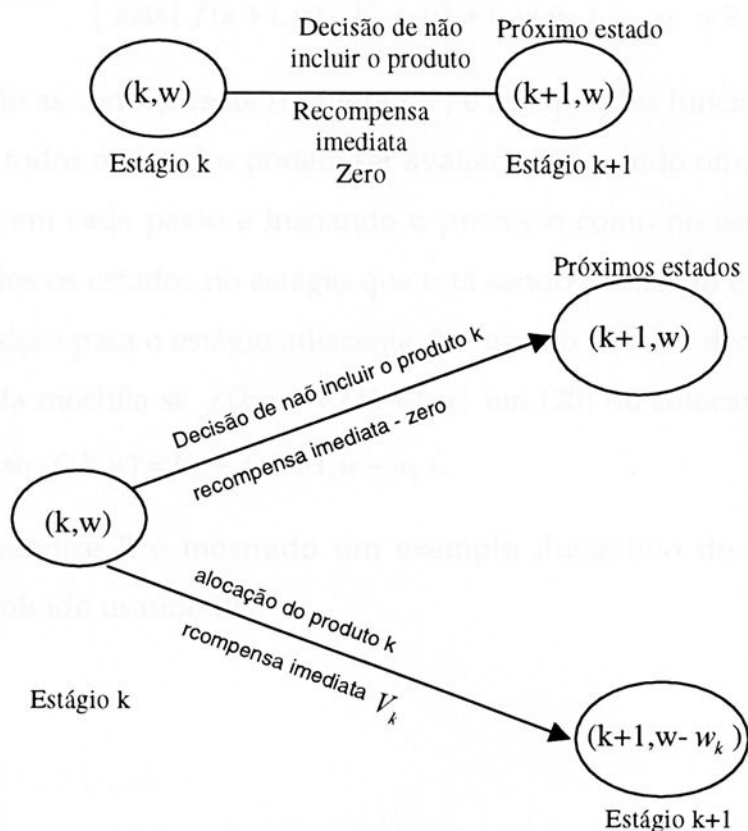


Figura 4: Representação de Estados.

A FVO é definida no estado (k, w) da seguinte forma:

$f(k, w)$ = máximo valor possível que pode ser colocado na mochila com volume ainda disponível de w e escolha de produtos restritos ao conjunto $\{k, k+1, \dots, n\}$.

Desde que somente uma cópia de cada produto esteja disponível é obtido o seguinte:

$$f(n, w) = \begin{cases} 0 & \text{se } w < w_n \\ V_n & \text{se } w > w_n \end{cases} \quad (27)$$

A relação (27) representa a condição de fronteira que a FVO, $f(k, w)$ deve satisfazer no problema da mochila 0-1. Das decisões disponíveis no estado (k, w) , as equações que devem ser satisfeitas pelo FVO são as seguintes:

$$f(k, w) = \begin{cases} f(k+1, w) & \text{se } w < w_n \\ \max\{f(k+1, w), V_k + f(k+1, w-w_k)\} & \text{se } w \geq w_n \end{cases} \quad (28)$$

Usando as condições de fronteira (27) e as equações funcionais em (28), as FVO em todos os estados podem ser avaliadas e fazendo uma transição de um estágio em cada passo e iniciando o processo como no estágio (n-1). A FVO de todos os estados no estágio que está sendo analisado é calculada e é feita a transição para o estágio adjacente. No estado (k,w), a decisão é excluir o objeto k da mochila se $f(k, w) = f(k+1, w)$ em (28) ou colocar o produto k na mochila se $f(k, w) = V_k + f(k+1, w-w_k)$.

No apêndice B é mostrado um exemplo ilustrativo do problema da mochila resolvido usando PD.

Testes e Resultados

4.1 - Algoritmo Implementado

O algoritmo implementado tem como finalidade resolver o problema de Decisão de Capacidade (DCA) de maneira mais eficiente que os algoritmos analisados nesta pesquisa.

O algoritmo proposto foi estruturado a partir dos algoritmos de Branch and Bound e Programação Dinâmica.

Os Problemas de Mochila que são gerados pelo algoritmo de Branch and Bound devem de ser resolvidos pelo Algoritmo de Branch and Bound para problemas binários, e passaram a ser resolvidos por Programação Dinâmica (Proposta de melhoria) que possui melhor desempenho.

Capítulo 4

Testes e Resultados

4.1 – Algoritmo Implementado

O algoritmo Implementado tem como finalidade resolver o problema de Designação Generalizada (GAP) de maneira mais eficiente que os algoritmos analisados nesta pesquisa.

O algoritmo proposto foi estruturado a partir dos algoritmos de Ross-Soland e Programação Dinâmica.

Os Problemas da Mochila que são gerados pelo algoritmo de Ross-Soland deixam de ser resolvidos pelo Algoritmo de Branch and Bound para problemas binários, e passam a ser resolvidos por Programação Dinâmica (Proposta do trabalho), que possui melhor desempenho.



Assim foi implementado o seguinte algoritmo:

Passo 1. Inicialização: Fazer $i = 0$, e a incumbente inicial $z_{inicial} \rightarrow \infty$ e inicializar a lista dos subproblemas candidatos com o problema original.

Passo 2. Teste de convergência: Se a lista dos problemas candidatos está vazia, então termina o processo com a incumbente corrente sendo a solução ótima do problema. Em caso contrário, ir ao passo 3.

Passo 3. Seleção e resolução do subproblema candidato: Escolher um subproblema usando a regra de LIFO. Resolver o subproblema usando os seguintes passos:

(a) Obter o subproblema reduzido após especificar os valores das variáveis x_{ij} já fixadas em 0 ou 1 pelo processo Branch and Bound.

(b) No problema reduzido, identificar as variáveis que podem assumir os valores $x_{ij} = 0$ usando a relação $a_{ij} > b_{ij} - \sum_{j \in I_i} a_{ij} x_{ij}$ e montar o novo problema reduzido.

(c) Se no novo problema reduzido alguma tarefa não pôde ser alocada para nenhum agente então o problema é INFACTÍVEL, então ir ao passo 4. Nesse caso todas as variáveis de uma coluna estão fixadas em $x_{ij} = 0$.

(d) Encontrar uma solução do subproblema reduzido PR usando heurística (o agente i realiza a tarefa j de menor custo, assim a sua correspondente x_{ij} é igual a 1). Neste caso pode acontecer duas possibilidades:

i. Existe uma solução inteira FACTÍVEL para o problema (1): Neste caso, foi encontrada uma candidata a incumbente. Ir ao passo 4.



ii. Não existe uma solução inteira FACTÍVEL para o problema (1): Neste caso, gerar um problema da mochila para cada restrição violada tipo (2).

(e) Montagem e resolução dos problemas da mochila:

i. Montar os problema da mochila. Nessa montagem levar em conta que, caso não exista uma variável x_{sj} então, escolher um $p_j = M$ grande. Este caso pode acontecer quando no subproblema que está sendo montado não existe um outro agente para transferir uma tarefa realizada por um agente com capacidade ultrapassada.

ii. Resolver os problemas da mochila (usando Programação Dinâmica). Se na resolução do problema da mochila i , $z_i \geq M$ significa que não é possível eliminar a infactibilidade do agente i e portanto, o subproblema deve ser INFACTÍVEL e deve ser sondado. Nesse caso ir ao passo 4. Em caso contrário ir ao passo seguinte.

iii. Encontrar a nova limitante inferior do subproblema e realizar a transferência de tarefas de acordo com a solução dos problemas da mochila. Se a nova proposta de solução for FACTÍVEL para o problema (1), isto é, nenhum agente tem sua capacidade violada então foi encontrada uma solução inteira factível. Nesse caso, ir ao passo 4 para sondar o subproblema e analisar a possibilidade de trocar de incumbente. Em caso contrário ir ao passo 5 para separar o subproblema.

Passo 4. Testes de sondagem: O subproblema avaliado pode ser sondado se satisfizer uma das seguintes condições:

(a) Se for verificado que a proposta de solução é infactível.



(b) Se a limitante inferior do subproblema avaliado é de pior qualidade que a incumbente corrente.

(c) Se a limitante inferior do problema avaliado for inteira e factível. Nesse caso, avaliar a possibilidade de trocar de incumbente. Caso exista troca de incumbente, aplicar o teste 4(b) para todos os subproblemas candidatos armazenados na lista e ainda não sondados.

Se o subproblema avaliado foi sondado retornar ao passo 2. Em caso contrário ir ao passo 5.

Passo 5. Separação: Separar o subproblema avaliado em dois subproblemas sucessores e escolher a variável de separação usando (9).

4.2 - Teste com o Algoritmo

Para mostrar o desempenho do algoritmo apresenta-se um exemplo típico de um problema com 5 agentes e 20 tarefas. Os dados do problema são os seguintes:

Coeficientes de custo c_{ij}

36	46	34	25	29	17	15	29	38	50	39	50	37	39	25	13	11	11	32	50
12	33	26	10	32	49	46	44	41	49	41	45	43	44	24	14	17	21	32	39
34	23	36	34	21	39	48	37	34	10	44	39	41	13	45	11	35	43	35	46
10	42	25	24	16	31	11	25	21	43	41	27	38	31	36	35	14	17	18	36
39	19	24	16	50	20	39	11	26	40	18	44	37	26	13	25	16	12	39	19

Recursos utilizados pelos agentes (restrições)

15 8 11 16 24 8 19 9 15 7 6 13 10 12 25 16 21 18 21 21
8 10 7 23 20 9 7 11 21 9 16 20 9 25 10 14 20 23 23 25
22 9 13 12 22 7 10 8 15 12 15 15 12 16 7 15 17 20 16 12
6 5 24 24 24 23 8 8 6 18 20 7 8 6 7 5 21 25 5 18
23 19 18 17 25 14 15 12 11 5 25 14 8 8 16 17 11 12 8 14

Recursos dos agentes (função objetivo)

50 50 50 50 50

O algoritmo analisado resolve o problema após gerar uma árvore de Branch and Bound de 1018 nós encontrando a seguinte solução:

Função objetivo: 374

Variáveis com valor 1:

$$x_{1.6} = x_{1.11} = x_{1.16} = x_{1.18} = x_{2.1} = x_{2.3} = x_{2.4} = x_{2.15} = x_{3.2} = x_{3.10} = x_{3.14} = x_{4.5} = \\ x_{4.7} = x_{4.9} = x_{4.12} = x_{4.19} = x_{5.8} = x_{5.13} = x_{5.17} = x_{5.20} = 1$$

Considerações Finais

Nesta pesquisa foram analisados vários algoritmos especializados em problemas de Programação Linear Inteira Binária, dentre os quais o algoritmo de Trick (Apêndice D) que não foi implementado computacionalmente por apresentar problemas na fase de redesignação de tarefas quando a capacidade dos agentes é muito restrita.

Portanto, a pesquisa deu ênfase aos algoritmos de Branch and Bound, Ross-Soland e Programação Dinâmica.

O algoritmo de Ross-Soland apresentou melhor desempenho, em relação ao algoritmo de Branch and Bound convencional, pois encontra a limitante inferior de cada nó através da resolução dos Problemas da Mochila binários em contraposição ao algoritmo convencional de Branch and Bound, que resolve um problema de PL em cada nó da árvore de Branch and Bound usando o método Simplex. Ou seja, o algoritmo de Ross-Soland aproveita a estrutura do problema e procura de forma direta soluções inteiras factíveis que permitam sondar o nó da árvore de Branch and Bound.



Em problemas de programação inteira binária, os únicos algoritmos com capacidade de encontrar as soluções ótimas de um problema, são os algoritmos tipo Branch and Bound. Mas estes algoritmos não tem capacidade de resolver problemas de grande porte porque o tempo de processamento cresce exponencialmente com o tamanho do problema. Portanto, os problemas da mochila gerados pelo algoritmo de Ross-Soland foram resolvidos por programação dinâmica, os quais apresentaram excelente desempenho. O tamanho dos problemas da mochila gerados nos testes foi muito pequeno, tornando o algoritmo de programação dinâmica muito rápido. A maioria dos problemas da mochila tiveram um número de variáveis muito reduzido, entre 4 e 6 variáveis. O problema da mochila de maior dimensão gerado teve 10 variáveis. No futuro pretende-se testar sistemas de grande porte e implementar melhorias no algoritmo especialmente na identificação de uma incumbente inicial de qualidade e na escolha da variável para a separação dos subproblemas.



[5] K. G. MURTY: "Operations Research: Linear and Nonlinear Optimization Models", Prentice-Hall, New Jersey, USA, 1993.

[6] K. G. MURTY: "Linear Programming", John Wiley and Sons, 1983.

[7] R. HILLIER, G. J. LIEBERMAN: "Introduction to Operations Research", McGraw Hill Publishing Company, 1990.

[8] AMIR, I. S. & GOLANIE, R. S.: "Integer Programming", John Wiley and Sons, New York, 1988.

Referências Bibliográficas

[1] G.T. ROSS-SOLAND: A branch and bound algorithm for the generalized assignment problem, Mathematical programming 8, pp 91-103, 1975.

[2] G.G. CATTRYSSE-L.N. VAN WASSENHOUSE: A survey of algorithms for the generalized assignment problem, European Journal of Operational Research 60, pp 260-272, 1992.

[3] M.A. TRICK: A linear relaxation heuristic for the generalized assignment problem, Naval Research Logistic, Vol. 39, pp 137-151, 1992.

[4] E. BALAS, C.H. MARTIN: "Pivot and Complement – A Heuristic for 0-1 Programming", Management Science, Vol. 26, No. 1, pp. 86-96, January 1980.



[5] K. G. MURTY: "Operatins Research: Deterministic Optimizatim Models",
Prentice Hall, New Jersey, USA, 1995.

[6] K. G. MURTY: "Linear Programming", John Wiley and Sons, 1983.

[7] F. HILLIER, G. J. LIEBERMAN: "Introduction to Operations Research",
Mc Graw Hill Publishing Company, 1990.

[8] NEMHAUSER G. L., GARTINNEE R. S. : "Integer Programming", John
Wiley and Sons, New York, 1972.

Exemplo Ilustrativo do Algoritmo de Ross-Soland

A.1- Exemplo

O exemplo a seguir é um problema de três agentes e cinco tarefas. Os dados são mostrados na tabela 1. O asterisco (*) indica que não é possível a associação de um agente-tarefa. Os valores de b_i nas restrições são 28 para cada agente.



Tabela 1

	Tarefas						Tarefas				
	1	2	3	4	5		1	2	3	4	5
Agente 1	11	18	*	20	14	Agente 2	12	19	*	11	18
Agente 3	2	40	*	20	40	Agente 3	2	4	*	11	15
	3	*	15	16	10		3	*	10	14	22

Coeficientes da função objetivo

Coeficientes de designações múltiplas

Apêndice A

Exemplo Ilustrativo do Algoritmo de Ross-Soland

A.1- Exemplo

O exemplo a seguir é um problema de três agentes e cinco tarefas, os dados serão mostrados na tabela 1. O asterisco (*) indica que não é possível a associação do par agente-tarefa. Os valores de b_i nas restrições são 28 para todo $i \in I$.

Tabela 1

Tarefas							Tarefas						
	1	2	3	4	5		1	2	3	4	5		
	1	14	38	*	26	14	1	12	19	*	11	18	
Agentes	2	49	*	20	46	49	Agentes	2	6	*	11	15	18
	3	*	45	16	10	*	3	*	10	14	22	*	
Coeficientes da função objetivo							coeficientes de designações múltiplas						

Formulação do problema:

a) Problema P.

$$\min z = 14x_{11} + 38x_{12} + 26x_{14} + 14x_{15} + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34}$$

$$s.a \quad 12x_{11} + 19x_{12} + 11x_{14} + 18x_{15} \leq 28$$

$$6x_{21} + 11x_{23} + 15x_{24} + 18x_{25} \leq 28$$

$$10x_{32} + 14x_{33} + 22x_{34} \leq 28$$

$$x_{11} + x_{21} + x_{31} = 1$$

$$x_{12} + x_{22} + x_{32} = 1$$

$$x_{13} + x_{23} + x_{33} = 1$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{15} + x_{25} + x_{35} = 1$$

$$x_{ij} = 0 \text{ ou } 1.$$

b) Problema relaxado (PR):

$$\min z = 14x_{11} + 38x_{12} + 26x_{14} + 14x_{15} + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34}$$

$$s.a \quad x_{11} + x_{21} + x_{31} = 1$$

$$x_{12} + x_{22} + x_{32} = 1$$

$$x_{13} + x_{23} + x_{33} = 1$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{15} + x_{25} + x_{35} = 1$$

$$x_{ij} = 0 \text{ ou } 1.$$

Resolvendo o PR do Nó 0:

Tarefa 1. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 1, isto implica que $x_{11} = 1$, $x_{21} = 0$ e x_{31} não é possível.

Tarefa 2. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 1, isto implica que $x_{12} = 1$, $x_{32} = 0$ e x_{22} não é possível.

Tarefa 3. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 3, isto implica que $x_{33} = 1$, $x_{23} = 0$ e x_{13} não é possível.

Tarefa 4. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 3, isto implica que $x_{34} = 1$ e os outros x_{24} e x_{14} são iguais à zero.

Tarefa 5. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 1, isto implica que $x_{15} = 1$, $x_{25} = 0$ e x_{35} não é possível.

Portanto a solução do problema(PR) será:

$$Z = 14 * 1 + 38 * 1 + 26 * 0 + 14 * 1 + 49 * 0 + 20 * 0 + 46 * 0 + 49 * 0 + 45 * 0 + 16 * 1 + 10 * 1$$
$$Z = 92$$

A solução do (PR) não é factível para (P), pois:

$$12 * 1 + 19 * 1 + 11 * 0 + 18 * 1 \leq 28 \Rightarrow 49 \leq 28 \quad \text{restrição violada}$$

$$6 * 0 + 11 * 0 + 15 * 0 + 18 * 0 \leq 28 \Rightarrow 0 \leq 28 \quad \text{satisfaz}$$

$$10 * 0 + 14 * 1 + 22 * 1 \leq 28 \Rightarrow 36 \leq 28 \quad \text{restrição violada}$$

As restrições violadas são reformuladas a fim de satisfazer o problema (P), ou seja, serão transformadas em Problemas da Mochila:

$$J_i \equiv \{x_{ij} = 1, j \in J\} \Rightarrow \begin{cases} J_1 \equiv \{x_{11} = x_{12} = x_{15} = 1\} = \{1, 2, 5\} \\ J_2 \equiv \{\emptyset\} \\ J_3 \equiv \{x_{33} = x_{34} = 1\} = \{3, 4\} \end{cases}$$

e

$$I' \equiv \left\{ i : \sum_{j \in J_i} a_{ij} x_{ij} > b_i, i \in I \right\} \equiv \{1, 3\}$$

$$d_i = \sum_{j \in J_i} a_{ij} x_{ij} - b_i = \begin{cases} d_1 = a_{11}x_{11} + a_{12}x_{12} + a_{15}x_{15} - b_1 = 12 + 19 + 18 - 28 = 21, J_1 = \{1, 2, 5\} \\ d_2 = \phi \\ d_3 = a_{33}x_{33} + a_{34}x_{34} - b_3 = 14 + 22 - 28 = 8, J_3 = \{3, 4\} \end{cases}$$

$$p_j = \min_{K \in I - \{i_j\}} \{c_{kj} - c_{i_j j}\} = \begin{cases} p_1 = \min\{(49 - 14); (\infty - 14)\} = 35 \\ p_2 = \min\{(\infty - 38); (45 - 38)\} = 7 \\ p_3 = \min\{(\infty - 16); (20 - 16)\} = 4 \\ p_4 = \min\{(26 - 10); (46 - 10)\} = \min\{16, 26\} = 16 \\ p_5 = \min\{(\infty - 14); (49 - 14)\} = 35 \end{cases}$$

Deste modo,

$$z_i = \sum_{j \in J_i} p_j y_{ij} = \begin{cases} z_1 = p_1 y_{11} + p_2 y_{12} + p_5 y_{15} = 35 y_{11} + 7 y_{12} + 35 y_{15} \\ z_2 = \phi \\ z_3 = p_3 y_{33} + p_4 y_{34} = 4 y_{33} + 16 y_{34} \end{cases}$$

Com esses dados um novo problema pode ser construído para a restrição violada 1 e 3:

Restrição 1:

$$\min z = 35 y_{11} + 7 y_{12} + 35 y_{15}$$

$$s.a. \quad 12 y_{11} + 19 y_{12} + 18 y_{15} \geq 21$$

$$y_{11}, y_{12}, y_{15} \in \{0, 1\}$$

Restrição 3:

$$\min z = 4y_{33} + 16y_{34}$$

$$s.a. \quad 14y_{33} + 22y_{34} \geq 8$$

$$y_{33}, y_{34} \in \{0,1\}$$

Estes problemas são resolvidos através de Branch and Bound para problemas binários e as soluções encontradas são $z_1 = 42$, com $y_{11} = 0, y_{12} = 1, y_{15} = 1$ e $z_3 = 4$, com $y_{34} = 0, y_{33} = 1$ respectivamente. Os valores das funções objetivos são adicionados ao limite inferior de 92, encontrando o limite revisado de 138, como mostrado a seguir:

$$LIM = Z + \sum_{i \in I'} z_i^* = 92 + 42 + 4 = 138$$

Interpretando os resultados:

Para $i = 1$

$y_{12} = 1 \Rightarrow$ Tarefa 2 é repassada do agente 1 para o agente 3 (agente que realiza a tarefa 2 com o segundo menor custo).

$y_{15} = 1 \Rightarrow$ Tarefa 5 é repassada do agente 1 para o agente 2 (agente que realiza a tarefa 5 com o segundo menor custo).

$$\Rightarrow x_{12} = 0 \Rightarrow x_{32} = 1$$

$$\Rightarrow x_{15} = 0 \Rightarrow x_{25} = 1$$

Para $i = 3$

$y_{33} = 1 \Rightarrow$ Tarefa 3 é repassada do agente 3 para o agente 2 (agente que realiza a tarefa 2 com o segundo menor custo).

$$\Rightarrow x_{33} = 0 \Rightarrow x_{23} = 1$$

Nova solução: $x_{11} = x_{32} = x_{23} = x_{34} = x_{25} = 1$; observar se esta nova solução é factível para P,



$12 \leq 28$ *satisfaz*

$29 \leq 28$ *restrição violada*

$32 \leq 28$ *restrição violada*

ou seja, a proposta não é factível para P, pois as restrições 2 e 3 estão violadas.

O próximo passo é separar o problema, deste modo, a variável de separação baseada na resposta do (PK_i) será escolhida. Como $y_{11} = 0$ e $y_{34} = 0$, deve-se calcular t_{11} e t_{34} , onde F_i é o conjunto de tarefas designadas pelo agente i,

$$t_{i^*j^*} = \max \left\{ \frac{p_j}{\frac{a_{ij}}{(b_i - \sum_{j \in F_i} a_{ij} x_{ij})}} \right\},$$

$$t_{11} \Rightarrow F_1 = \{1\}$$

$$t_{1^*1^*} = \frac{p_1}{\frac{a_{11}}{b_1 - r_{11}}} = \frac{35}{\frac{12}{28-12}} = \frac{35}{\frac{12}{16}} = \frac{140}{3}$$

$$t_{34} \Rightarrow F_3 = \{2, 4\}$$

$$t_{3^*4^*} = \frac{p_4}{\frac{a_{34}}{b_3 - a_{32} - a_{34}}} = \frac{16}{\frac{22}{28-10-22}} = \frac{16}{\frac{22}{-4}} = \frac{-32}{11}$$

$$t_{i^*j^*} = \max\{t_{11}, t_{34}\} = \max\left\{\frac{140}{3}; -\frac{32}{11}\right\} \Rightarrow t_{11} \text{ é o vencedor.}$$

A variável escolhida é x_{11} , onde o nó 1 é calculado com $x_{11} = 1$ e o nó 2, com $x_{11} = 0$. Primeiramente o nó 1 é analisado, deste modo o próximo problema a ser resolvido é:

Nó (1)

$$\min z = 14 + 38x_{12} + 26x_{14} + 14x_{15} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34}$$

$$s.a \quad 19x_{12} + 11x_{14} + 18x_{15} \leq 16$$

$$11x_{23} + 15x_{24} + 18x_{25} \leq 28$$

$$10x_{32} + 14x_{33} + 22x_{34} \leq 28$$

$$x_{12} + x_{22} + x_{32} = 1$$

$$x_{13} + x_{23} + x_{33} = 1$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{15} + x_{25} + x_{35} = 1$$

Critério estabelecido: Na identificação do agente de primeiro menor custo para uma dada tarefa, algumas variáveis são consideradas zero devido a condição $a_{ij} > b_i - \sum_{j \in F_i} a_{ij}x_{ij}$, onde F_i é o conjunto que armazena as tarefas que são designadas pelo agente i no problema atual ou seja:

Como a variável fixada até o momento é x_{11} , então $F_1 = \{1\}$, assim

Para $i = 1$, tem-se

$$b_i - \sum_{j \in F_i} a_{ij}x_{ij} = 28 - 12 = 16$$

$$(a_{12}) \rightarrow 19 > 16 \rightarrow x_{12} = 0$$

$$(a_{14}) \rightarrow 11 > 16$$

$$(a_{15}) \rightarrow 18 > 16 \rightarrow x_{15} = 0$$

Este processo se repete para todos os valores de i. Após serem definidas as variáveis iguais à zero, verifica-se entre as variáveis não nulas quais representam o menor custo de designação entre o agente i e a tarefa j.

Deste modo a solução do problema relaxado(PR) é:

Tarefa 1: realizada pelo agente 1 $\Rightarrow x_{11} = 1$

Tarefa 2: realizada pelo agente 3 $\Rightarrow x_{32} = 1$

Tarefa 3: realizada pelo agente 3 $\Rightarrow x_{33} = 1$

Tarefa 4: realizada pelo agente 3 $\Rightarrow x_{34} = 1$

Tarefa 5: realizada pelo agente 2 $\Rightarrow x_{25} = 1$

Portanto o valor da função objetivo de (PR), com $x_{11}, x_{32}, x_{33}, x_{34}, x_{25} = 1$, e os outros $x_{ij} = 0$ é $Z = 134$. Testar se esta solução é factível para (P):

$$19 \cdot 0 + 11 \cdot 0 + 18 \cdot 0 \leq 16 \Rightarrow 0 \leq 16 \quad \text{satisfaz}$$

$$11 \cdot 0 + 15 \cdot 0 + 18 \cdot 1 \leq 28 \Rightarrow 18 \leq 28 \quad \text{satisfaz}$$

$$10 \cdot 1 + 14 \cdot 1 + 22 \cdot 1 \leq 28 \Rightarrow 46 \leq 28 \quad \text{restrição violada}$$

A solução do (PR) não é factível para o (P), portanto a restrição 3 será transformada em um problema da Mochila, e a seguir será resolvida:

$$\text{Restrição 3: } 10x_{32} + 14x_{33} + 22x_{34} \leq 28$$

$$d_i = \sum_{j \in J_i} a_{ij}x_{ij} - b_i = \{d_3 = a_{32}x_{32} + a_{33}x_{33} + a_{34}x_{34} - b_3 = 10 + 14 + 22 - 28 = 18,$$

$$d_3 = 18$$

$$p_j = \min_{K \in I - \{i\}} \{c_{kj} - c_{ij}\} = \begin{cases} p_2 = \min\{\infty - 38\} = \infty \\ p_3 = \min\{20 - 16\} = 4 \\ p_4 = \min\{26 - 10, 46 - 10\} = \min\{16, 26\} = 16 \end{cases}$$

$$\min z_i = \sum_{j \in J_i} p_j y_{ij} = \{z_3 = p_2 y_{32} + p_3 y_{33} + p_4 y_{34}$$

$$\text{s.a.} \quad a_{32}y_{32} + a_{33}y_{33} + a_{34}y_{34} \geq 18$$

ou seja:

$$\min z_3 = \infty y_{32} + 4y_{33} + 16y_{34}$$

$$\text{s.a.} \quad 10y_{32} + 14y_{33} + 22y_{34} \geq 18$$

$$y_{3j} = 0 \quad \text{ou} \quad 1 \quad \text{para todo } j$$

A solução obtida para este problema é a seguinte: $y_{32} = 0, y_{34} = 1, y_{33} = 0$, com $Z = 16$, assim o limite pode ser aumentado para 150.

$$LIM = Z + \sum_{i \in I'} z_i^* = 134 + 16 = 150$$

Interpretando os resultados:

$y_{34} = 1 \Rightarrow$ Tarefa 4 é repassada do agente 3 para o agente 1.

$$x_{34} = 0 \Rightarrow x_{14} = 1$$

Nova solução: $x_{11} = x_{32} = x_{33} = x_{14} = x_{25} = 1$

Testando a factibilidade do problema:

$$\min z \ 14 + 26 * 1 + 49 * 1 + 45 * 1 + 16 * 1 = 150$$

$$19 * 0 + 11 * 1 + 18 * 0 \leq 16 \Rightarrow 11 \leq 16 \quad \text{satisfaz}$$

$$11 * 0 + 15 * 0 + 18 * 1 \leq 28 \Rightarrow 18 \leq 28 \quad \text{satisfaz}$$

$$10 * 1 + 14 * 1 + 22 * 0 \leq 28 \Rightarrow 24 \leq 28 \quad \text{satisfaz}$$

$$0 + * + 1 = 1 \quad \text{satisfaz}$$

$$* + 0 + 1 = 1 \quad \text{satisfaz}$$

$$1 + 0 + 0 = 1 \quad \text{satisfaz}$$

$$0 + 1 + * = 1 \quad \text{satisfaz}$$

O nó 1 está sondado, agora o próximo problema a ser resolvido será aquele com $x_{11} = 0$.

Nó 2

$$\min z \ 38x_{12} + 26x_{14} + 14x_{15} + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34}$$

$$s.a \quad 19x_{12} + 11x_{14} + 18x_{15} \leq 28$$

$$6x_{21} + 11x_{23} + 15x_{24} + 18x_{25} \leq 28$$

$$10x_{32} + 14x_{33} + 22x_{34} \leq 28$$

$$x_{21} + x_{31} = 1$$

$$x_{12} + x_{22} + x_{32} = 1$$

$$x_{13} + x_{23} + x_{33} = 1$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{15} + x_{25} + x_{35} = 1$$

Solução para o PL relaxado(PR):

Tarefa 1. Como $x_{11} = 0$, o agente que realizará esta tarefa será aquele de menor custo entre os agentes 2 e 3, isto implica que o agente escolhido é 2, pois x_{31} não é possível, assim $x_{21} = 1$.

Tarefa 2. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 1, isto implica que $x_{12} = 1$, $x_{32} = 0$ e x_{22} não é possível.

Tarefa 3. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 3, isto implica que $x_{33} = 1$, $x_{23} = 0$ e x_{13} não é possível.

Tarefa 4. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 3, isto implica que $x_{34} = 1$ e os outros x_{24} e x_{14} são iguais à zero.

Tarefa 5. O agente que realizará esta tarefa será aquele de menor custo entre os agentes 1,2 e 3, ou seja, o agente escolhido é 1, isto implica que $x_{15} = 1$, $x_{25} = 0$ e x_{35} não é possível.

Portanto a solução do problema é:

$$Z = 14 * 0 + 38 * 1 + 26 * 0 + 14 * 1 + 49 * 1 + 20 * 0 + 46 * 0 + 49 * 0 + 45 * 0 + 16 * 1 + 10 * 1$$

$$Z = 127$$

Novamente a solução não é factível para (P), as restrições 1 e 3 são violadas, assim os Problemas da Mochila são obtidos analogamente aos nós anteriores:

$$\min z_1 = 7y_{12} + 35y_{15}$$

$$s.a. \quad 19y_{12} + 18y_{15} \geq 9$$

$$y_{1j} = 0 \quad ou \quad 1$$

$$\min z_3 = 16y_{34} + 4y_{33}$$

$$s.a. \quad 22y_{34} + 14y_{33} \geq 8$$

$$y_{3j} = 0 \quad ou \quad 1$$

As soluções ótimas $y_{12} = 1$, $y_{15} = 0$ e $y_{33} = 1$, $y_{34} = 0$ aumentam o limite de 127 para 138:



$$LIM = Z + \sum_{i \in I'} z_i^* = 127 + 7 + 4 = 138$$

Interpretando os resultados:

$y_{12} = 1 \Rightarrow$ Tarefa 2 é repassada do agente 1 para o agente 3

$y_{33} = 1 \Rightarrow$ Tarefa 3 é repassada do agente 3 para o agente 2

$$x_{12} = 0 \Rightarrow x_{32} = 1$$

$$x_{33} = 0 \Rightarrow x_{23} = 1$$

Nova solução: $x_{21} = x_{32} = x_{23} = x_{34} = x_{15} = 1$. Substituindo estes resultados nas restrições observa-se que o problema é infactível, ocasionando uma divisão deste nó, assim o próximo passo será escolher a variável de separação:

Como $y_{15} = 0$ e $y_{34} = 0 \Rightarrow$ a escolha é realizada entre x_{15} e x_{34} para separar o problema.

$$t_{15} \Rightarrow F_1 = \{5\}$$

$$t_{15*} = \frac{p_1}{\frac{a_{15}}{b_1 - a_{15}}} = \frac{35}{\frac{18}{28 - 18}} = \frac{35}{\frac{18}{10}} = \frac{140}{3} = 19,444$$

$$t_{34} \Rightarrow F_3 = \{2, 4\}$$

$$t_{34*} = \frac{p_4}{\frac{a_{34}}{b_3 - a_{32} - a_{34}}} = \frac{16}{\frac{22}{28 - 10 - 22}} = \frac{16}{\frac{22}{-4}} = \frac{-32}{11} = -2,9090$$

t_{15} é o vencedor $\Rightarrow x_{15}$ será a próxima variável de separação.

O nó 3 será calculado com $x_{15} = 1$ e o nó 4 com $x_{15} = 0$. O problema com $x_{15} = 1$ é o primeiro a ser analisado.



Nó 3

$$\min z \ 38x_{12} + 26x_{14} + 14 + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34}$$

$$s.a \quad 19x_{12} + 11x_{14} \leq 10$$

$$6x_{21} + 11x_{23} + 15x_{24} + 18x_{25} \leq 28$$

$$10x_{32} + 14x_{33} + 22x_{34} \leq 28$$

$$x_{21} + x_{31} = 1$$

$$x_{12} + x_{22} + x_{32} = 1$$

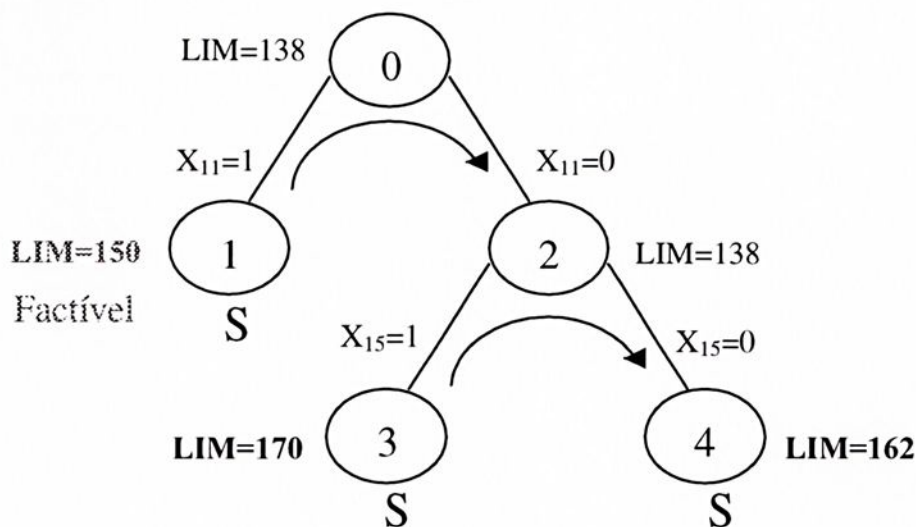
$$x_{13} + x_{23} + x_{33} = 1$$

$$x_{14} + x_{24} + x_{34} = 1$$

Solução para o PR: $x_{15} = x_{32} = x_{33} = x_{34} = x_{21} = 1$, com $z = 134$, mas esta solução não é factível para P, pois a 3ª restrição é violada:

O próximo passo será transformar esta restrição em um Problema da Mochila, a factibilidade será testada, como o valor de Z será maior do que o encontrado no nó 1, impossibilitando assim o encontro de uma melhor solução no decorrer deste ramo, o nó 3 é sondado. Posteriormente o nó 4 será analisado analogamente aos anteriores, encontrando $Z = 162$, que também é maior que a solução do nó 1, portanto a solução do nosso problema será $Z = 150$.

Visualização da árvore do problema final:



Solução ótima é $Z = 150$, com $x_{11} = x_{32} = x_{33} = x_{14} = x_{25} = 1$.

Observações: O exemplo foi também resolvido através do Algoritmo Branch and Bound convencional e sua solução é: $Z=150$, com $x_{11} = x_{32} = x_{33} = x_{14} = x_{25} = 1$, e a árvore resultante possui 19 nós.

Conclusões: Para este exemplo o algoritmo de Ross-Soland requer menos esforço que o algoritmo Branch and Bound convencional, isto pode ser observado ao se comparar o número de nós de suas respectivas árvores, ou seja, 5 nós contra 19 do algoritmo Branch and Bound convencional.

Apêndice B

Programação Dinâmica para um Problema da Mochila Binário (Exemplo Ilustrativo)

B.1 - Exemplo

O exemplo a seguir é um problema com $n=5$ e $w_0=12$. Outras informações são dadas na tabela a seguir:

Produto	Volumen v_i	Lucro l_i
1	3	12
2	4	12
3	3	15
4	7	42
5	8	18

Apêndice B

Programação Dinâmica para um Problema da Mochila Binário (Exemplo Ilustrativo)

B.1 - Exemplo

O exemplo a seguir é um problema com $n=5$ e $w_0=12$. Outras informações são dadas na tabela à seguir:

Produto	Volume w_i	Lucro v_i
1	3	12
2	4	12
3	3	15
4	7	42
5	9	18



Estágio 5: Condição de fronteira:

$$n = 5 \Rightarrow w_n = 9$$

$$f(5,0) = f(5,1) = f(5,2) = f(5,3) = f(5,4) = f(5,5) = f(5,6) = f(5,7) = f(5,8) = 0$$

$$f(5,9) = f(5,10) = f(5,11) = f(5,12) = v_s = 18$$

colocar o produto 5 e terminar.

Estágio 4:

$$w_n = 7$$

$$f(4,0) = f(4,1) = f(4,2) = f(4,3) = f(4,4) = f(4,5) = f(4,6) = f(5, w)$$

Excluir o produto 4 e continuar com no estado $(5, w)$.

$$f(4,7) = \max\{f(5,7); 42 + f(5,0)\} = \max\{0, 42\} = 42$$

Colocar o produto 4 e continuar como no estado $(5,0)$

$$f(4,8) = \max\{f(5,8); 42 + f(5,1)\} = \max\{0, 42\} = 42$$

Colocar o produto 4 e continuar como no estado $(5,1)$

$$f(4,9) = \max\{f(5,9); 42 + f(5,2)\} = \max\{18, 42\} = 42$$

Colocar o produto 4 e continuar como no estado $(5,2)$

$$f(4,10) = \max\{f(5,10); 42 + f(5,3)\} = \max\{18, 42\} = 42$$

Colocar o produto 4 e continuar como no estado $(5,3)$

$$f(4,11) = \max\{f(5,11); 42 + f(5,4)\} = \max\{18, 42\} = 42$$

Colocar o produto 4 e continuar como no estado $(5,4)$



$$f(4,12) = \max\{f(5,12); 42 + f(5,5)\} = \max\{18, 42\} = 42$$

Colocar o produto 4 e continuar como no estado (5,5)

Continuando este processo, os valores da FVO são obtidos e as decisões ótimas nos estados dos estágios 3 e 2.

w	Estágio 3		Estágio 2	
	FVO $f(3, w)$	Decisão Ótima	FVO $f(2, w)$	Decisão Ótima
0, 1, 2	$f(4, w)$	Excluir prod. 3 Continuar com (4, w)	$f(3, w)$	Excluir prod. 2 Continuar com (3, w)
3	15	Colocar prod. 3 Continuar com (4,0)	15	Excluir prod. 2 Continuar com (3,3)
4	15	Colocar prod. 3 Continuar com (4,1)	15	Excluir prod. 2 Continuar com (3,4)
5	15	Colocar prod. 3 Continuar com (4,2)	15	Excluir prod. 2 Continuar com (3,5)
6	15	Colocar prod. 3 Continuar com (4,3)	15	Excluir prod. 2 Continuar com (3,6)
7	42	Excluir prod. 3 Continuar com (4,7)	42	Excluir prod. 2 Continuar com (3,7)
8	42	Excluir prod. 3 Continuar com (4,8)	42	Excluir prod. 2 Continuar com (3,8)
9	42	Excluir prod. 3 Continuar com (4,9)	42	Excluir prod. 2 Continuar com (3,9)
10	57	Colocar prod. 3 Continuar com (4,7)	57	Excluir prod. 2 Continuar com (3,10)
11	57	Colocar prod. 3 Continuar com (4,8)	57	Excluir prod. 2 Continuar com (3,11)
12	57	Colocar prod. 3 Continuar com (4,9)	57	Excluir prod. 2 Continuar com (3,12)

E finalmente, obtém-se:

$$f(4,12) = \max\{0 + f(2,12); 12 + f(2,9)\} = \max\{0 + 57, 12 + 42\} = 57, \quad \text{com}$$

decisão ótima no estado (1, 2) excluindo o produto 1, e continuando como no estado (2,12). Deste modo, seguindo as decisões nos vários estágios, observa-se que a estratégia ótima no problema original para maximizar o valor



colocado na mochila é colocar os produtos 3 e 4 dentro dela, conduzindo ao valor máximo de 57.

Apêndice C

Algoritmo Branch and Bound Especializado para o Problema da Mochila Binária

C.1 - Introdução

Neste apêndice é apresentado um algoritmo Branch and Bound especializado para resolver o problema da mochila binária. A característica fundamental do algoritmo apresentado é que este não resolve problemas de programação linear em cada nó da árvore de Branch and Bound. Portanto, este algoritmo é muito rápido e eficiente, pois utiliza algoritmos de programação linear que resolvem o problema de forma eficiente.



processamento em algoritmos Branch and Bound tradicionais. Para maiores detalhes ver [70].

Consideremos o problema da mochila binária da seguinte forma:

$$\text{Min } z(x) = - \sum_{i=1}^n v_i x_i \quad (1)$$

$$\text{sujeito a } \sum_{i=1}^n w_i x_i \leq b \quad (2)$$

$$0 \leq x_i \leq 1 \quad (3)$$

Apêndice C

Uma decisão inicial trivial consiste em utilizar aqueles produtos cujo volume é maior que o volume da mochila, isto é, $v_i > b$. Assim, deve-se identificar todos esses produtos e fixar as variáveis correspondentes em $x_i = 0$ e assim apenas o problema restante.

Algoritmo Branch and Bound Especializado para o Problema da Mochila Binária

Este tipo de problema pode ser resolvido de forma eficiente sem a necessidade de usar o método simplex para resolver problemas de PL.

C.1 - Introdução

Neste apêndice é apresentado um algoritmo Branch and Bound especializado para resolver o problema da mochila binária. A característica fundamental do algoritmo apresentado é que este não resolve problemas de programação linear em cada nó da árvore de Branch and Bound. Portanto, este algoritmo é muito rápido já que dispensa o uso de algoritmos de programação linear que usam o método simplex que aumentam o tempo de



processamento em algoritmos Branch and Bound tradicionais. Para maiores detalhes ver [10].

Consideremos o problema da mochila binária da seguinte forma:

$$\text{Min } z(x) = - \sum_{j=1}^n c_j x_j, \tag{1}$$

$$\text{sujeito à: } \sum_{j=1}^n a_j x_j \leq b \tag{2}$$

$$0 \leq x_j \leq 1 \tag{4}$$

$$x_j \text{ inteiro}$$

Uma decisão inicial trivial consiste em identificar aqueles produtos j cujo volume é maior que o volume da mochila, isto é, $a_j > b$. Assim, deve-se identificar todos esses produtos e fixar as variáveis correspondentes em $x_j = 0$ e resolver apenas o problema restante.

Quando relaxamos a integralidade no problema da mochila, o problema relaxado resultante é um problema de programação linear com uma única restrição. Esse tipo de problema pode ser resolvido de forma eficiente sem a necessidade de usar o método simplex para resolver problemas de PL.

Na análise do algoritmo usamos o seguinte exemplo:

Produto	Volume a_j	Custo c_j
1	3	12
2	4	12
3	3	15
4	7	42
5	9	18

Volume da mochila: $b = 12$.

C.2 - Estratégia para resolver de forma eficiente o PL no Problema da Mochila Binária

O PL resultante após relaxar a integralidade no problema da mochila pode ser resolvido de forma eficiente sem necessidade de usar o método simplex. Para encontrar a solução ótima desse tipo de problema inicialmente, deve-se fixar todas as variáveis x_j correspondentes as variáveis j tal que $a_j > b$ em $x_j = 0$ e remover essas variáveis de análise posterior desse PL. Depois, deve-se encontrar a densidade das variáveis restantes, isto é, deve-se calcular a relação $d_j = c_j / a_j$ para cada variável j e ordenar as variáveis em ordem decrescente de densidade.

O PL pode ser resolvido facilmente fixando as primeiras variáveis $x_j = 1$ e descontando o volume correspondente ao volume da mochila. O processo termina quando não é possível colocar um produto na mochila, devido a insuficiência do volume remanescente. Essa variável assume um valor fracional determinado pelo volume disponível na mochila e o volume desse produto. As outras variáveis assumem valores iguais a $x_j = 0$.

Usando a estratégia mencionada anteriormente resolvemos o PL correspondente ao exemplo enunciado anteriormente. Ordenando os produtos pela densidade obtemos o seguinte ordenamento dos produtos:

Produto	Volume a_j	Custo c_j	Densidade	Volume acumulado
4	7	42	6	7
3	3	15	5	10
1	3	12	4	16
2	4	12	3	20
5	9	18	2	29

Volume da mochila: $b = 12$.



Assim é possível colocar apenas os produtos 4 e 3 integralmente e o produto 1 parcialmente. Portanto a solução ótima do PL assume a seguinte forma:

$$x_1 = 2/3$$

$$x_3 = x_4 = 1 \Rightarrow z(x) = -65$$

$$x_2 = x_5 = 0$$

Portanto, qualquer PL resultante da relaxação do problema da mochila binária se resolve de forma trivial e sem a necessidade de usar o método simplex.

C.3 – Estratégia de separação

Se a solução de um PL correspondente a um nó da árvore de branch and bound não é inteira então este problema deve ser separado em dois subproblemas. A variável usada para separação é justamente aquela única variável que assume um valor fracionário na solução do PL. Assim, não existe necessidade de usar estratégias sofisticadas de escolha de variáveis j para separação de subproblemas porque existe apenas uma variável para separação. Portanto, o problema é separado em dois subproblemas sucessores usando a variável x_j que tem valor fracionário. Um subproblema incorpora a restrição $x_j = 0$ e o outro subproblema a restrição $x_j = 1$.

C.4 – Estratégia de Escolha do Subproblema da Lista

Quando existe necessidade de escolher o próximo problema da lista que deve ser escolhida para análise na estratégia branch and bound pode ser usada alguma estratégia existente para algoritmos branch and bound. A proposta mais simples consiste em escolher a regra LIFO e armazenar



primeiro na lista o subproblema com a restrição $x_j = 0$. Assim, é analisado primeiro o subproblema com a restrição $x_j = 1$.

C.5 - Estratégia de Sondagem

Neste caso devem ser usados os três testes de sondagem típicos de um algoritmo branch and bound para problemas de programação linear inteira mista.

C.6 - Outras Características Importantes

Na resolução de cada subproblema, deve-se levar em conta que algumas variáveis podem assumir valores iguais a zero porque o volume do produto representado por essa variável é maior que o volume remanescente da mochila. Esta característica está cada vez mais presente quando algumas variáveis são fixadas no valor $x_j = 1$ na estratégia branch and bound diminuindo o volume disponível da mochila.

C.7 - Sondagem usando Enumeração Explícita das variáveis livres

Supor que em determinado estágio do processo branch and bound, pretende-se resolver um problema de PL correspondente a um nó da árvore de branch and bound. Supor que neste estágio já foram fixados os valores de uma parcela das variáveis na seguinte forma:

$$x_{j1} = x_{j2} = \dots = x_{jr} = 0$$

$$x_{p1} = x_{p2} = \dots = x_{pu} = 1$$

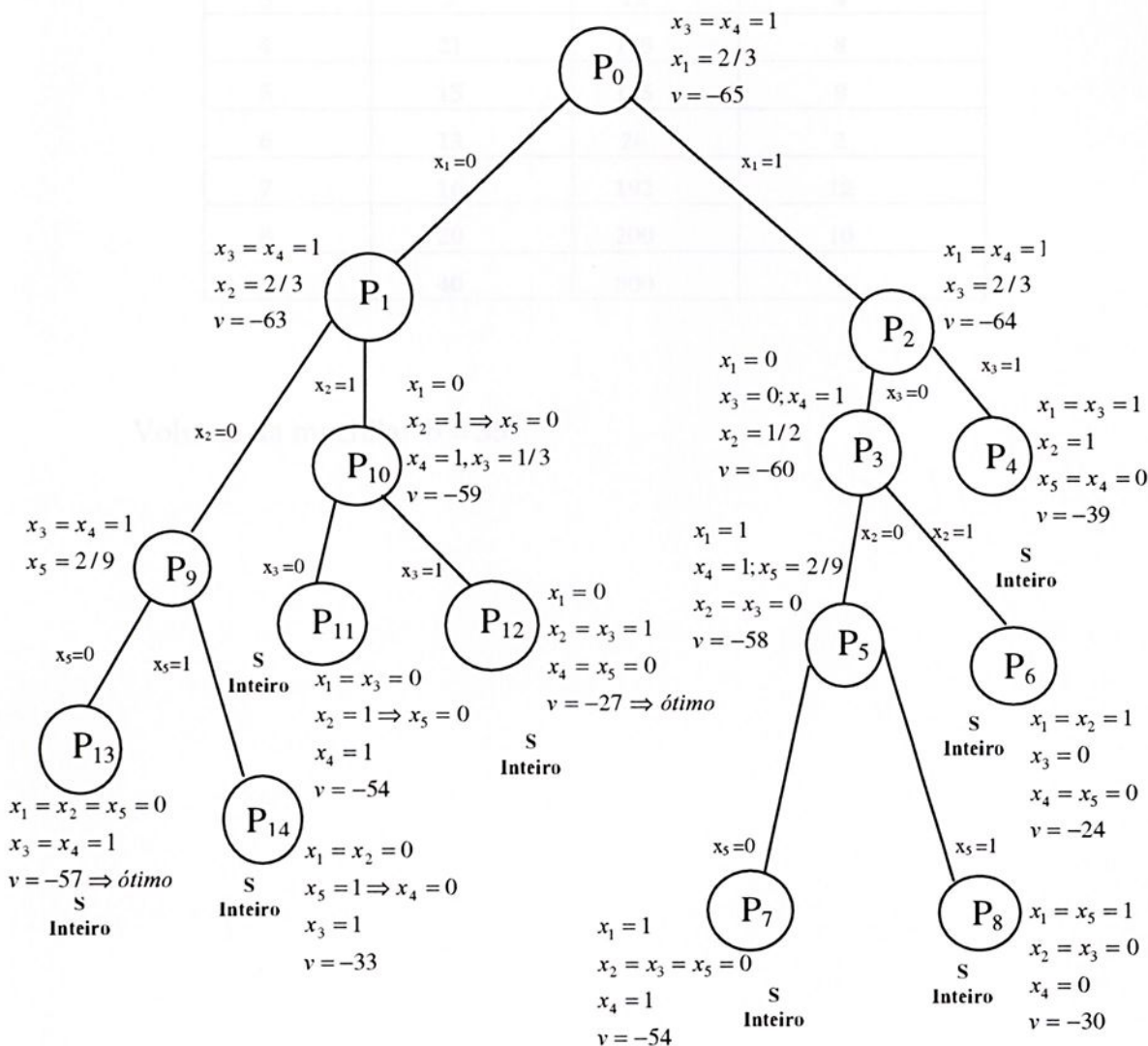
Assim, o problema com n variáveis tem $(r+u)$ variáveis fixadas. Neste estágio, deve-se resolver um PL reduzido com apenas $s = n - (r + u)$



variáveis. Essas variáveis são chamadas variáveis livres. Se o número de variáveis livres é pequeno então pode ser mais eficiente resolver e sondar esse subproblema por enumeração explícita. Na enumeração explícita, deve-se avaliar duas propostas de solução identificando a solução ótima rapidamente se S for pequena [5].

C.8 - Exemplos

C.8.1. Resolvemos o exemplo mencionado anteriormente usando o algoritmo branch and bound especializado para o problema da mochila binário.

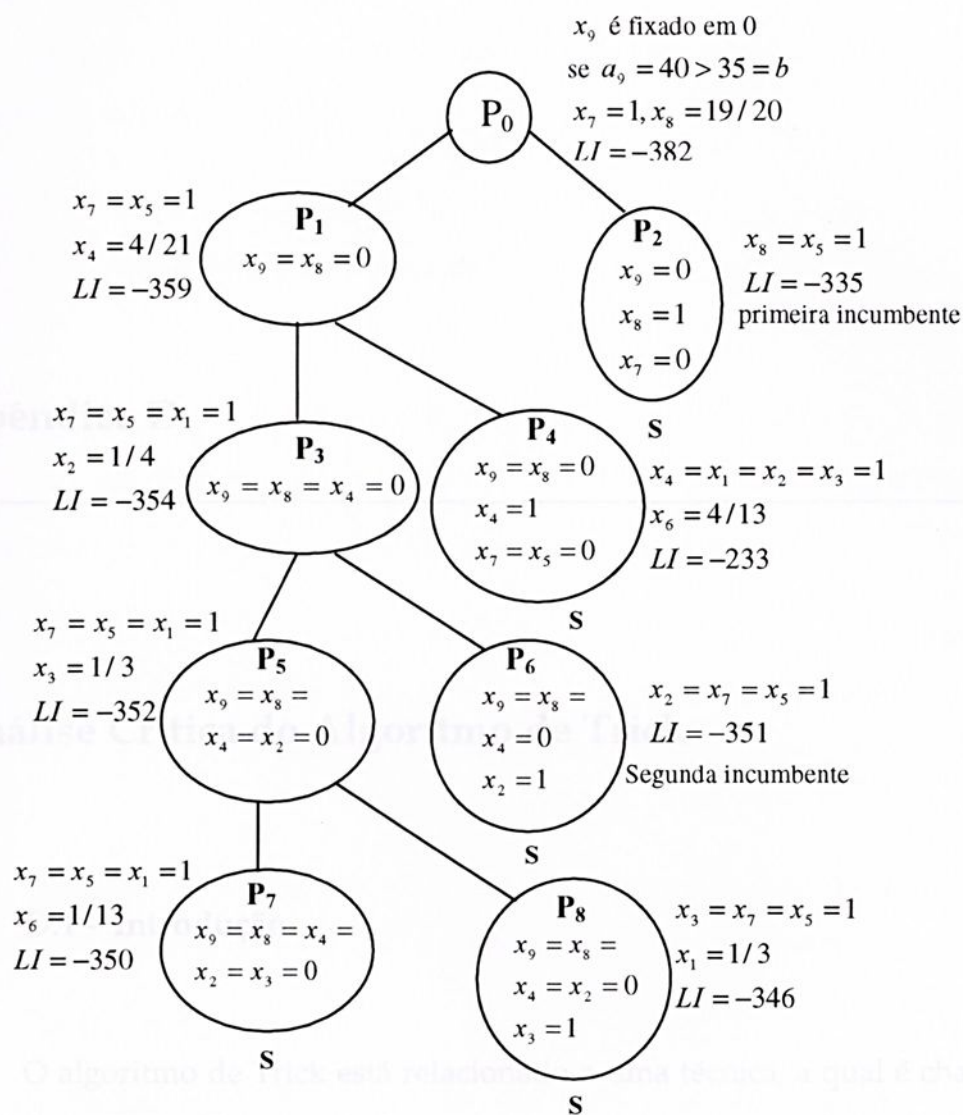


$$\text{Solução: } \begin{cases} x_3 = x_4 = 1 \\ x_1 = x_2 = x_5 = 0 \\ v = -57 \end{cases}$$

C.8.2. Considere os dados da tabela abaixo:

Produto	Volume a_j	custo c_j	densidade
1	3	21	7
2	4	24	6
3	3	12	4
4	21	168	8
5	15	135	9
6	13	26	2
7	16	192	12
8	20	200	10
9	40	800	

Volume da mochila: $b = 35$.



Portanto, a solução é $z = -351$, isto implica que a melhor escolha é colocar os produtos 2, 5 e 7 na mochila.

Apêndice D

Análise Crítica do Algoritmo de Trick

D.1 – Introdução

O algoritmo de Trick está relacionado a uma técnica, a qual é chamada de Relaxação do PL, esta técnica consiste em relaxar o problema GAP. Esta relaxação é obtida através da substituição da restrição tipo (4) por $x_{ij} \geq 0$.

O algoritmo de Trick é um algoritmo heurístico que usa relaxação linear. Existe na literatura outro algoritmo baseado na relaxação de x_{ij} (relaxação do PL). Este algoritmo é apresentado por Bender-Van Nunen, e apresenta problemas na fase de redesignação de tarefas quando a capacidade dos agentes é muito restrita, segundo uma análise de Cattrysse [2].



Quando aparece esse tipo de problema não é possível encontrar uma solução factível. O algoritmo de Trick apresenta este mesmo problema, portanto a medida que aumenta a capacidade dos agentes o algoritmo tende a fornecer soluções factíveis.

D.2 - Estratégia Fundamental

Considerando o problema de designação de n tarefas para m agentes, de forma que cada agente i tenha uma capacidade b_i , cada tarefa j usa uma capacidade a_{ij} a um custo c_{ij} se for designada ao agente i .

A relaxação do problema GAP é obtida pela substituição da integralidade da restrição tipo (3) pela restrição de não negatividade. O algoritmo é adequado especialmente para resolver problemas em que $n \gg m$ [3].

Teorema 1: O número de tarefas compartilhadas na solução do problema GAP relaxado (LGAP) é no máximo igual ao número de agentes.

Prova: ver [3].

Comentário: Após a solução do LGAP, podem ser retiradas da formulação matemática todas as variáveis x_{ij} relacionados com as variáveis alocadas a um único agente. Nesse caso resta ainda um problema GAP reduzido, unicamente com as variáveis relacionadas com as tarefas que apresentam solução compartilhada por mais de um agente na solução do LGAP.

Teorema 2: Se existe algumas tarefas compartilhadas por mais de um agente na solução do LGAP, então fixando todas as variáveis com valor 1,



resultará em pelo menos uma variável inútil com respeito a nova capacidade do agente.

Prova: ver [3].

Comentário: Este Teorema mostra que no problema reduzido existe pelo menos uma variável inútil na formulação matemática, isto é, existe um x_{ij} com $a_{ij} > b_i$ que inviabiliza a possibilidade de que essa variável assuma um valor igual a 1. Se todas as variáveis x_{ij} inúteis são eliminadas do problema reduzido então a solução do LGAP do novo problema reduzido apresenta pelo menos uma tarefa integralmente designada a um único agente.

Portanto, a estrutura básica da resolução do problema é a seguinte:

Partindo de um problema GAP, realiza-se a relaxação através da substituição da restrição tipo (4) pela restrição $x_{ij} \geq 0$. Resolve-se o problema LGAP através do Minos, sua solução fornece as informações necessárias para a construção do novo problema, ou seja, as variáveis x_{ij} que possuem valor 1 serão fixadas no problema, atualizando assim a capacidade dos agentes, desta forma o problema conterá apenas as variáveis compartilhadas, formando o problema LGAP reduzido.

Partindo do problema LGAP reduzido, será feito um reconhecimento das variáveis inúteis, as quais serão eliminadas deste problema, ou seja, as variáveis x_{ij} que possuem $a_{ij} > b_i$. Assim resultando no problema LGAP reduzido modificado, o qual poderá ser solucionado através do Minos, e sua solução será utilizada para construir um novo problema, ou seja, o processo continua sucessivamente até que não haja mais variáveis a serem fixadas.



D.3 - O Algoritmo de Trick

Entrada: um problema GAP.

Saída: uma solução heurística.

Passo 1: Remover todas as variáveis desnecessárias; se não existir variáveis, então pare.

Passo 2: Resolver a relaxação linear do problema GAP.

Passo 3: Fixar todas as variáveis que possuem valor 1, eliminando as tarefas correspondentes e atualizando a capacidade dos agentes. Vá para o passo 1. Montar o novo GAP, usando somente as variáveis compartilhadas.

D.4 - Exemplo Ilustrativo

Será utilizado o exemplo resolvido anteriormente pelo algoritmo de Ross-Soland (Exemplo A.1).



Exemplo:

Problema GAP.

$$\begin{aligned} \min \quad & 14x_{11} + 38x_{12} + 26x_{14} + 14x_{15} + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34} \\ \text{s.a} \quad & 12x_{11} + 19x_{12} + 11x_{14} + 18x_{15} \leq 28 \\ & 6x_{21} + 11x_{23} + 15x_{24} + 18x_{25} \leq 28 \\ & 10x_{32} + 14x_{33} + 22x_{34} \leq 28 \\ & x_{11} + x_{21} + x_{31} = 1 \\ & x_{12} + x_{22} + x_{32} = 1 \\ & x_{13} + x_{23} + x_{33} = 1 \\ & x_{14} + x_{24} + x_{34} = 1 \\ & x_{15} + x_{25} = 1 \\ & x_{ij} = 0 \quad \text{ou} \quad 1. \end{aligned}$$

Problema GAP relaxado(LGAP).

$$\begin{aligned} \min \quad & 14x_{11} + 38x_{12} + 26x_{14} + 14x_{15} + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34} \\ \text{s.a} \quad & 12x_{11} + 19x_{12} + 11x_{14} + 18x_{15} \leq 28 \\ & 6x_{21} + 11x_{23} + 15x_{24} + 18x_{25} \leq 28 \\ & 10x_{32} + 14x_{33} + 22x_{34} \leq 28 \\ & x_{11} + x_{21} + x_{31} = 1 \\ & x_{12} + x_{22} + x_{32} = 1 \\ & x_{13} + x_{23} + x_{33} = 1 \\ & x_{14} + x_{24} + x_{34} = 1 \\ & x_{15} + x_{25} = 1 \\ & x_{ij} \geq 0. \end{aligned}$$

Passo 1: Nenhuma variável foi eliminada no primeiro LGAP, ou seja na formulação matemática não existe variáveis x_{ij} inúteis com $a_{ij} > b_i$.



Passo 2: Resolvendo o problema LGAP usando o Minos:

Solução obtida:

$$x_{11} = x_{23} = x_{32} = 1; x_{12} = x_{13} = x_{21} = x_{22} = x_{31} = x_{33} = 0$$

$x_{15} = 0,8889; x_{25} = 0,1111; x_{24} = 0,18182; x_{34} = 0,81818$ (Tarefas 4 e 5 são compartilhadas).

Passo 3: Fixando o valor de algumas variáveis e montando o novo GAP (GAP reduzido), ou seja as variáveis que serão fixadas são aquelas que foram integralmente alocadas por um único agente:

$$x_{11} = x_{23} = x_{32} = 1$$

Problema GAP reduzido:

$$\min \quad 26x_{14} + 14x_{15} + 46x_{24} + 49x_{25} + 10x_{34}$$

$$s.a \quad 11x_{14} + 18x_{15} \leq 16$$

$$15x_{24} + 18x_{25} \leq 17$$

$$22x_{34} \leq 18$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{15} + x_{25} = 1$$

$$x_{ij} \geq 0.$$

Passo1: Eliminando variáveis inúteis, resultando no LGAP modificado.



Problema GAP reduzido Modificado:

$$\min \quad 26x_{14} + 46x_{24}$$

$$s.a \quad 11x_{14} \leq 16$$

$$15x_{24} \leq 17$$

$$x_{14} + x_{24} = 1$$

$$0 = 1$$

$$x_{ij} \geq 0.$$

Passo 2: A solução é infactível.

Passo 3: Não existe nenhuma variável a ser fixada.

Passo 4: Stop.

(B) Repetindo o exemplo aumentando a capacidade dos agentes:

Problema GAP.

$$\min \quad 14x_{11} + 38x_{12} + 26x_{14} + 14x_{15} + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34}$$

$$s.a \quad 12x_{11} + 19x_{12} + 11x_{14} + 18x_{15} \leq 30$$

$$6x_{21} + 11x_{23} + 15x_{24} + 18x_{25} \leq 30$$

$$10x_{32} + 14x_{33} + 22x_{34} \leq 30$$

$$x_{11} + x_{21} + x_{31} = 1$$

$$x_{12} + x_{22} + x_{32} = 1$$

$$x_{13} + x_{23} + x_{33} = 1$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{15} + x_{25} = 1$$

$$x_{ij} = 0 \quad \text{ou} \quad 1.$$

Problema GAP relaxado(LGAP).

$$\min \quad 14x_{11} + 38x_{12} + 26x_{14} + 14x_{15} + 49x_{21} + 20x_{23} + 46x_{24} + 49x_{25} + 45x_{32} + 16x_{33} + 10x_{34}$$

$$s.a \quad 12x_{11} + 19x_{12} + 11x_{14} + 18x_{15} \leq 30$$

$$6x_{21} + 11x_{23} + 15x_{24} + 18x_{25} \leq 30$$

$$10x_{32} + 14x_{33} + 22x_{34} \leq 30$$

$$x_{11} + x_{21} + x_{31} = 1$$

$$x_{12} + x_{22} + x_{32} = 1$$

$$x_{13} + x_{23} + x_{33} = 1$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{15} + x_{25} = 1$$

$$x_{ij} \geq 0.$$

Passo 1: Nenhuma variável foi eliminada no primeiro LGAP, ou seja, na formulação matemática não existe variáveis x_{ij} inúteis com $a_{ij} > b_i$.

Passo 2: Resolvendo o problema LGAP usando o Minos:

Solução obtida:

$$x_{11} = x_{15} = x_{23} = x_{32} = 1; \quad x_{12} = x_{13} = x_{21} = x_{22} = x_{25} = x_{31} = x_{33} = 0$$

$$x_{24} = 0,09091; \quad x_{34} = 909090 \text{ (Tarefa 4 é compartilhada)}.$$

Passo 3: Fixando o valor de algumas variáveis e montando o novo GAP(GAP reduzido), ou seja, as variáveis que serão fixadas são aquelas que foram integralmente alocadas por um único agente:

$$x_{11} = x_{15} = x_{23} = x_{32} = 1$$

Problema GAP reduzido:

$$\min \quad 26x_{14} + 46x_{24} + 10x_{34}$$

$$s.a \quad 11x_{14} \leq 0$$

$$15x_{24} \leq 19$$

$$22x_{34} \leq 20$$

$$x_{14} + x_{24} + x_{34} = 1$$

$$x_{ij} \geq 0.$$

Passo1: Eliminando variáveis inúteis, resultando no LGAP modificado.

Problema GAP reduzido Modificado:

$$\min \quad 46x_{24}$$

$$s.a \quad 15x_{24} \leq 19$$

$$x_{24} = 1$$

$$x_{ij} \geq 0$$

Passo 2: A solução é factível, pode-se observar que todas as tarefas foram alocadas por um único agente.

Passo 3: Não existe nenhuma variável a ser fixada.

Passo 1: Stop.

Solução Final: $x_{11} = x_{15} = x_{23} = x_{24} = x_{32} = 1$, com $z = 139$.

Conclusões:

O exemplo foi primeiramente apresentado com a capacidade restrita dos agentes em (A), e a solução encontrada foi infactível. A medida em que se aumentou a capacidade dos agentes, ou seja no exemplo (B) foi possível observar que a solução obtida foi factível, mostrando assim que o algoritmo de Trick fornece soluções factíveis a medida que se aumenta a capacidade dos agentes.





UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Câmpus de Ilha Solteira
Programa de Pós-Graduação em Engenharia Elétrica
Av. Brasil Centro, 56
15385-000 Ilha Solteira - SP
www.dee.feis.unesp.br

