UNESP
Faculdade de Engenharia de Guaratinguetá

Guaratinguetá
1997

UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE GUARATINGUETÁ
FACULDADE DE ENGENHARIA
DEPARTAMENTO DE MATEMÁTICA

CARO LEITOR NÃO
RISQUE O LIVRO

EM BUSCA DE UM AMBIENTE INTELIGENTE DE MODELAGEM EM REDES

Edson Luiz França Senne

FACULDADE DE ENGENHARIA
DE GUARATINGUETA
BIBLIOTECA

Trabalho apresentado, como parte dos requisitos exigidos, para o concurso de Livre-Docência na disciplina Computação e Cálculo Numérico, do Departamento de Matemática da Faculdade de Engenharia do Campus de Guaratinguetá da Universidade Estadual Paulista.

345

345

Junho de 1997

T.007.52(043)
S478e

EDSON LUIZ FRANÇA SENNE

ENGENHEIRO MECÂNICO, Faculdade de Engenharia do Campus de Guaratinguetá - UNESP, 1974.
MESTRE EM CIÊNCIAS, Instituto Tecnológico de Aeronáutica - ITA, 1980.
DOUTOR EM COMPUTAÇÃO APLICADA, Instituto Nacional de Pesquisas Espaciais - INPE, 1987.
PÓS-DOUTORADO EM PROGRAMAÇÃO MATEMÁTICA, Brunel University, Inglaterra, 1994.

EM BUSCA DE UM AMBIENTE INTELIGENTE DE MODELAGEM EM REDES

Trabalho apresentado, como parte dos requisitos exigidos, para o concurso de Livre-Docência na disciplina Computação e Cálculo Numérico, do Departamento de Matemática da Faculdade de Engenharia do Campus de Guaratinguetá da Universidade Estadual Paulista.

À

Maria Lígia, Luís Francisco e Thaís Helena,

com muito amor.

AGRADECIMENTOS

Este trabalho contou com colaboração de várias pessoas. A todas elas desejo expressar meus agradecimentos, e o faço pela ordem dos capítulos.

A *Fernando Augusto da Silva Marins*, do Departamento de Produção da Unesp-Campus de Guaratinguetá, sou grato pela cessão de sua implementação do algoritmo Primal Simplex para redes lineares por partes, descrito no Capítulo 2. Ao *Marins* agradeço também pela leitura crítica deste capítulo e por suas sugestões. Agradeço também ao meu aluno *Luís Gustavo de Carvalho Bechuate* pela implementação de interface gráfica para a edição de redes e visualização de soluções.

A *Cormac Lucas*, da Brunel University, Inglaterra, agradeço a ajuda no projeto da interface gráfica discutida no Capítulo 3. Sou grato também a *Flávio Roberto Dias Velasco* pela cessão de sua implementação da linguagem OPS5, sem o que não teria sido possível a implementação do sistema especialista em formulação de modelos apresentado neste capítulo. Agradeço muito aos meus alunos *André Ricardo Marchezan* e *Evando Celso Nakagawa*, pela implementação de parte deste projeto de interface.

Sou muito grato a *Luís Antonio Nogueira Lorena*, do Instituto Nacional de Pesquisas Espaciais, por toda sua colaboração e pela formulação dos problemas de localização descritos no Capítulo 4. Ao *Lorena*, agradeço também suas críticas e sugestões sobre o texto inicial deste capítulo.

Parte deste trabalho foi realizado durante estágio de pós-doutorado. Agradeço, portanto, à *Fapesp* (Fundação de Amparo à Pesquisa do Estado de São Paulo) por esta oportunidade.

Agradeço também ao *CNPq* (Conselho Nacional de Desenvolvimento Científico e Tecnológico) que tem me auxiliado, através de bolsa de pesquisa, nestes últimos dois anos.

unesp

RESUMO

A importância da modelagem como ferramenta de apoio à tomada de decisão tem dirigido muitos trabalhos de pesquisa para o desenvolvimento de formas de representação e técnicas de solução de modelos. Nos últimos anos, com o avanço dos computadores - tanto em *hardware*, com máquinas mais poderosas, como em *software*, com algoritmos mais inteligentes - o número de aplicações bem-sucedidas de modelos de otimização tem aumentado. Consequentemente, tem crescido o interesse na tarefa de construção de modelos e no desenvolvimento de ferramentas para simplificar esta tarefa.

Este trabalho reúne contribuições do autor para o desenvolvimento de um ambiente de modelagem em redes. As contribuições discutidas neste trabalho são as seguintes: (i) o desenvolvimento de um Sistema de Processamento de Modelos e sua integração a um Sistema de Visualização e Análise de Soluções e a um Sistema de Gerenciamento de Bancos de Dados. Discute-se a construção de uma linguagem algébrica de modelagem em redes lineares por partes e sua utilização na formulação e na solução de modelos; (ii) uma proposta de Sistema de Interface com Usuário e seu relacionamento a um Sistema Especialista em Formulação de Modelos. Discute-se o desenvolvimento de uma interface gráfica para a representação conceitual de um problema de Programação Linear e a utilização de um sistema especialista para obter a formulação algébrica do problema; (iii) a construção de uma Base de Algoritmos de Solução. Discute-se o desenvolvimento e a implementação de algoritmos para a solução de modelos de fluxo em redes lineares por partes e de algoritmos heurísticos eficientes para a solução de problemas clássicos de localização, tais como: p-medianas, localização simples e localização capacitada.

unesp

ABSTRACT

The importance of optimization models as a decision making tool has directed much research work towards developing representation schemes and solution techniques for these problems. In recent years, with the advances in computer technology, not only in hardware, with more powerful machines, but also in software, with more intelligent algorithms, the ability to solve optimization models has improved. As a consequence, there has been an increase of interest in the modeling process itself and in computer systems which can support the modeling task.

This work discusses some contributions which have been developed by the author in order to build a network modeling system. These contributions refer to: (i) the development of a Model Processor System and its integration to a Solution Analyzer System and a Data Base Management System. The development of an algebraic piecewise-linear network modeling language is discussed; (ii) a proposal of an User Interface System and its relationship with a Model Formulation Expert System. The work discusses the components of a graphical interface as the basis for a system to assist users in the process of formulating Linear Programming problems; (iii) the development of a Base of Solution Algorithms. The work presents the algorithm used by the algebraic language in order to solve piecewise-linear network flow models and discusses a new heuristic approach for facility location problems. The use of this new approach to solve p-median problems, uncapacitated facility location problems and capacitated facility location problems is shown.

unesp

SUMÁRIO

1 - INTRODUÇÃO

1.1 - Preliminares 1
1.2 - Ambientes de Modelagem 2
1.3 - Objetivos do Trabalho 7
1.4 - Sumário do Trabalho 8

2 - LINGUAGEM ALGÉBRICA DE MODELAGEM EM REDES

2.1 - Introdução 10
2.2 - Linguagem de Modelagem em Redes 12
2.3 - Programação em Redes Lineares Por Partes 16
2.4 - A Estrutura da Linguagem NPL2 25
2.4.1 - Conjuntos de Índices e Parâmetros 26
2.4.2 - Conjuntos de Índice, Parâmetros e Relações 29
2.4.3 - Declaração de Estrutura de Rede Linear Por Partes ... 37
2.5 - Exemplos de Modelos NPL2 43
2.5.1 - Modelo Generalizado de Distribuição 44
2.5.2 - Modelo de Alocação 46

3 - UMA PROPOSTA DE INTERFACE INTELIGENTE DE MODELAGEM

3.1 - Introdução 48
3.2 - O Modelo de Dados 53
3.3 - O Modelo de PL 54
3.4 - Algoritmo de Interpretação de Restrições 60
3.5 - O Modelo Conceitual 73
3.6 - Modelagem Baseada em Conhecimentos 83

4 - ALGORITMOS HEURÍSTICOS PARA PROBLEMAS DE LOCALIZAÇÃO

4.1 - Introdução 92
4.2 - As Relaxações Lagrangeana e *Surrogate* 93
4.3 - Problema das P-Medianas 99
4.4 - Problema de Localização Simples 112
4.5 - Problema de Localização Capacitado 121

5 - CONCLUSÃO

5.1 - O Trabalho Atual 136
5.2 - O Trabalho Futuro 138

REFERÊNCIAS BIBLIOGRÁFICAS 140

APÊNDICES

A - GRAMÁTICA DA LINGUAGEM NPL2

B - ALGORITMO DE INTERPRETAÇÃO DE RESTRIÇÕES

C - SISTEMA ESPECIALISTA EM FORMULAÇÃO DE MODELOS

D - MoNet - AMBIENTE DE MODELAGEM EM REDES

ÍNDICE DE FIGURAS

1.1 - Arquitetura de Ambiente de Modelagem em Redes 8
2.1 - Modelo de Fluxo Máximo 13
2.2 - Problema de Fluxo Máximo 15
2.3 - Função Convexa Linear por Partes 17
2.4 - Etapas na Solução de um Problema NPL2 25
2.5 - Declaração de Conjuntos de Índice e Parâmetros 26
2.6 - Declaração Alternativa de Conjuntos de Índice e Parâmetros 27
2.7 - Tabela Termelet: Exemplo de Relação 30
2.8 - Tabela Resultante de Operação de Seleção 31
2.9 - Tabela Resultante de Operação de Projeção 32
2.10 - Tabela Rotas 33
2.11 - Tabela Resultante de Operação de Junção 33
2.12 - Tabela Combust 36
2.13 - Funções Lineares por Partes 38
2.14 - Limite Inferior como Ponto de Quebra 39
2.15 - Modelo de Fluxo de Custo Mínimo 41
3.1 - Gerações de Ferramentas de Modelagem 51
3.2 - Paradigma de Formulação de Modelos 52
3.3 - Classes no Modelo de Dados 53
3.4 - Objetos e Tabela de Dados no Modelo de Dados 54
3.5 - Representação Gráfica da Função-Objetivo 55
3.6 - Representação Gráfica de Restrição 55
3.7 - Variáveis de Decisão e Objetos Primitivos 56
3.8 - Variáveis de Decisão e Instâncias de Índices 56
3.9 - Representação Gráfica de Modelo de PL 58
3.10 - Parte de Modelo de Dados 58
3.11 - Modelo de Dados para o Problema das Estações de Potência 67
3.12 - Modelo Conceitual: Problema de Distribuição com Transbordo 74
3.13 - Esquema Conceitual: Problema de Distribuição com Transbordo 75
3.14 - Modelo Conceitual: Problema de Distribuição de Múlti-Produtos .. 78
3.15 - Modelo Conceitual: Problema de Produção Multi-Períodos 82
3.16 - Estrutura Básica de Sistema Especialista 83
4.1 - Limitantes na Função LS 103
4.2 - Comportamento Computacional de Heurísticas 111
5.1 - Estágio Atual de Desenvolvimento do Sistema MoNet 136
5.2 - Estágio Atual de Desenvolvimento da Interface com Usuário 137

unesp

D.1 - Interface NPL2 ..... D2
D.2 - Fase de Compilação de Programa NPL2 ..... D3
D.3 - Interface IGPL com Ferramentas de Edição D4
D.4 - Representação Gráfica de Modelo Conceitual D5
D.5 - Compilação de Modelo Conceitual D5
D.6 - Tela Principal da Interface VIEW D6
D.7 - Visualização de Rede D7
D.8 - Quadro de Diálogo de Características de Nós D9
D.9 - Quadro de Diálogo de Características de Arcos D9

ÍNDICE DE TABELAS

4.1 - Problema de P-Medianas: Resultados computacionais da heurística lagrangeana 107

4.2 - Problema de P-Medianas: Resultados computacionais da heurística LS 108

4.3 - Problema das P-Medianas: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação lagrangeana 109

4.4 - Problema das P-Medianas: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação LS 110

4.5 - Problema das P-Medianas: Razões de tempo para as heurísticas LS e lagrangeana 112

4.6 - Problema de Localização Simples: Resultados computacionais da heurística lagrangeana 118

4.7 - Problema de Localização Simples: Resultados computacionais da heurística LS 118

4.8 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação lagrangeana para problemas pequenos 119

4.9 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação LS para problemas pequenos 119

4.10 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação lagrangeana para problemas grandes 119

4.11 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação LS para problemas grandes . 120

4.12 - Problema de Localização Simples: Razões de tempo para as heurísticas LS e lagrangeana na solução de pequenas instâncias 120

4.13 - Problema de Localização Simples: Razões de tempo para as heurísticas LS e lagrangeana na solução de grandes instâncias . 120

4.14 - Problema de Localização Capacitado: Resultados computacionais da heurística lagrangeana na solução de pequenas instâncias ... 127

4.15 - Problema de Localização Capacitado: Resultados computacionais da heurística LS na solução de pequenas instâncias 128

4.16 - Problema de Localização Capacitado: Resultados computacionais da heurística lagrangeana na solução de grandes instâncias 129

4.17 - Problema de Localização Capacitado: Resultados computacionais da heurística LS na solução de grandes instâncias 129

unesp

4.18 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de pequenas instâncias obtidas pela relaxação lagrangeana 130
4.19 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de pequenas instâncias obtidas pela relaxação LS . 131
4.20 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de grandes instâncias obtidas pela relaxação lagrangeana 132
4.21 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de grandes instâncias obtidas pela relaxação LS .. 132
4.22 - Problema de Localização Capacitado: Razões de tempo para as heurísticas LS e lagrangeana na solução de pequenas instâncias 133
4.23 - Problema de Localização Capacitado: Razões de tempo para as heurísticas LS e lagrangeana na solução de grandes instâncias . 134

Capítulo 1

Introdução

1.1 - Preliminares

O processo de tomada de decisão é, em geral, uma tarefa difícil. Muitas técnicas já foram desenvolvidas para auxiliar neste processo como, por exemplo, a Análise Estatística, a Simulação de Sistemas e a Teoria de Sistemas Especialistas. Dentre tais técnicas, uma das mais empregadas tem sido a Programação Matemática e, principalmente a partir do desenvolvimento do método Simplex (Dantzig, 1963), em especial, a Programação Linear (PL).

A essência de tais técnicas reside na construção de modelos capazes de descrever situações complexas. No caso da Programação Linear procura-se modelar o problema de tomada de decisão como a otimização (minimização ou maximização) de uma função linear, denominada função-objetivo, sujeita a um conjunto de restrições (igualdades ou desigualdades) lineares. Muitos problemas práticos como, por exemplo, problemas de transporte e distribuição, problemas de localização, problemas de *scheduling*, planejamento e controle da produção, planejamento financeiro, dentre muitos outros têm sido formulados e resolvidos como modelos de otimização (Williams, 1993).

Nos últimos anos, uma classe especial de problemas tem recebido muita atenção: problemas que podem ser tratados por meio de modelos em redes. As aplicações de modelos em redes são inúmeras. Uma revisão geral de tais aplicações pode ser vista nos trabalhos de Glover *et al.* (1992); Murty (1992); Ahuja *et al.* (1993); Marins e Perin (1996). Uma das descobertas recentes da área foi a de que muitos modelos matemáticos, incluindo diversos modelos de Programação Linear e Programação Inteira, podem ser tratados como modelos de fluxo em redes (Baston *et al.*, 1991). A grande vantagem dos modelos em rede deve-se ao fato de ser possível explorar a estrutura especial destes modelos para obter algoritmos eficientes de solução. Assim, não é surpresa que muitas das aplicações mais bem-sucedidas - tanto no setor público como no setor privado - pertencem a esta classe de problemas.



A importância da modelagem como ferramenta de apoio à tomada de decisão tem dirigido muitos trabalhos de pesquisa para o desenvolvimento de formas de representação e técnicas de solução de modelos. Desde a metade dos anos 1980 estes trabalhos procuram reunir contribuições das áreas de Ciência da Computação (como Tecnologia de Bancos de Dados, Linguagens de Programação, Computação Gráfica e Inteligência Artificial) e Pesquisa Operacional (como Otimização, Simulação e Análise de Decisão), e têm levado ao conceito de Sistemas de Apoio à Decisão (Mitra, 1988).

Nos últimos anos, com o avanço dos computadores (tanto em *hardware*, com máquinas mais poderosas, como em *software*, com algoritmos mais inteligentes), o custo de obtenção de soluções tem diminuído. Diversos problemas práticos, considerados de difícil solução a alguns anos atrás podem, atualmente, ser resolvidos em poucos minutos em microcomputadores (ver, por exemplo, Suhl, 1994). Com estes avanços, tem aumentado o número de aplicações bem-sucedidas de modelos de otimização. Entretanto, diversos autores, dentre eles Murphy *et al.* (1992a), Darby-Dowman (1993), Forster and Mevert (1994), têm argumentado que a despeito de todo este progresso, muitas pessoas envolvidas em processos de tomada de decisão ainda hesitam em empregar a Programação Matemática na solução de seus problemas devido à complexidade do processo de modelagem.

Com isto a preocupação tem-se voltado para a tarefa de construção de modelos e muitas ferramentas têm sido desenvolvidas para simplificar esta tarefa. Os trabalhos mais recentes procuram enfatizar a necessidade do desenvolvimento de Ambientes de Modelagem para auxiliar na formulação, documentação, gerenciamento e interpretação de modelos (ver Bharadwaj *et al.*, 1992; Chung and O'Keefe, 1992, para uma revisão). Tais ambientes constituem, atualmente, uma das principais ferramentas para a construção de Sistemas de Apoio à Decisão (Bhargava and Krishnan, 1993; Chang *et al.*, 1993).

1.2 - Ambientes de Modelagem

As características desejáveis de um Ambiente de Modelagem já foram detalhadamente especificadas por Geoffrion (1987; 1989; 1994) em sua discussão sobre modelagem estruturada, e compreendem basicamente:

- uma estrutura suficientemente geral para as diversas fases do processo de modelagem;

unesp

- o emprego de três representações básicas para um modelo: uma representação conveniente para comunicação com não-especialistas, uma representação matemática conveniente para uso por especialistas e uma representação executável por computador;
- uma fundamentação conceitual rigorosa para as três representações básicas do modelo;
- a independência entre a representação do modelo e a de sua solução, com o emprego de interface que facilite a construção de bibliotecas de modelos ou de sub-modelos;
- a independência entre a representação do modelo e a representação dos dados que definem instâncias específicas do modelo;
- a facilidade de integração com bancos de dados;
- a implementação de interface visual e interativa com o usuário.

Segundo Murphy *et al.* (1992a), um Ambiente de Modelagem deve incluir vários módulos, tais como: Sistema de Processamento de Modelos, Sistema de Interface com Usuário, Sistema de Gerenciamento de Bancos de Dados, Sistema Gerenciador de Modelos, Sistema de Análise de Soluções, além de uma Base de Algoritmos de Solução.

Diversos autores vêm se preocupando com os componentes e os relacionamentos de componentes de um Ambiente de Modelagem. Drud (1988), por exemplo, concentra-se na interface entre o Sistema de Processamento de Modelos e o algoritmo de solução (ou otimizador). Este autor discute princípios gerais a serem considerados nas representações do modelo para que exista independência entre o modelo e o algoritmo de solução.

O relacionamento entre o Sistema de Processamento de Modelos (SPM) e o Sistema de Gerenciamento de Bancos de Dados (SGBD) está também bastante discutido na literatura (Dolk, 1988b; Bhargava *et al.*, 1992; Kristjansson *et al.*, 1993). O interesse neste relacionamento deve-se à importância de manter separados o modelo (genérico), dos dados necessários à uma instância do modelo. Isto, basicamente, por duas razões:

- a separação facilita a abstração, no sentido de que cada uma das partes pode ser manipulada e compreendida separadamente;
- a separação facilita a manutenção e verificação de modelos, uma vez que alterações nos dados, que ocorrem, em geral, muito mais frequentemente, não afetam o modelo.

unesp

Além disto, embora, a rigor, não seja necessário um SGBD para implementar tal separação, é uma prática comum em empresas e outras organizações a utilização de bancos de dados para armazenamento e recuperação de informações.

Em alguns sistemas de modelagem, as características de bancos de dados estão embutidas no próprio SPM. Choobineh (1991a), por exemplo, mostra como a linguagem SQL (*Structured Query Language*) pode ser estendida de modo a incorporar novas construções para a definição da função-objetivo e das restrições de um modelo, constituindo assim a linguagem SQLMP (*SQL for Mathematical Programming*). O trabalho de Moody (1994) vai em outra direção: não desenvolver um sistema de bancos de dados próprio, mas prover a conexão entre o SPM e um SGBD conhecido. Tal idéia parece ser interessante em aplicações práticas, onde, em geral, os dados necessários ao modelo estão armazenados em bancos de dados já existentes.

Dolk (1988a) discute as funções de um SGM de modo a apoiar todo o ciclo de vida da modelagem. Segundo este autor, o SGM deve prover para os modelos as mesmas facilidades proporcionadas aos dados por um Sistema Gerenciador de Bancos de Dados como, por exemplo, acesso compartilhado, independência, segurança e privacidade. Além desta função de controle, este autor aponta ainda para um SGM as funções de:

- descrição do modelo, a partir de um formalismo capaz de representar diversas classes de modelos e que proporcione visões do modelo em diferentes níveis de abstração, e
- manipulação do modelo, com operações como avaliação, verificação de viabilidade, análise de sensibilidade, além de operações que assegurem a integridade do modelo e de seus dados.

Outros autores discutem também a arquitetura de um SGM. Lenard (1988; 1993), por exemplo, propõe uma arquitetura baseada na modelagem estruturada, no paradigma de programação orientada a objetos e em bancos de dados relacionais. Na proposta deste autor, os modelos são vistos como uma coleção de objetos (podendo, portanto, ser organizados em classes e subclasses) e a manipulação de modelos é vista como trocas de mensagens entre objetos. O trabalho propõe ainda que estes objetos sejam implementados como tabelas, de forma que os métodos para tratamento das mensagens possam estar escritos na linguagem de manipulação de dados de um sistema de banco de dados relacional.

unesp

Trabalhos mais recentes sobre SGM discutem ainda questões relacionadas à reutilização e integração de modelos, ou seja, o desenvolvimento de modelos de grande porte a partir de modelos (ou sub-modelos) já existentes. Muhanna (1994), por exemplo, enfatiza que o objetivo de um SGM é aumentar a produtividade das atividades de modelagem e tomada de decisão e para isto é necessário ter em mente que modelos (a exemplo dos dados) são recursos organizacionais e que a tarefa de modelagem é um conjunto continuado e cumulativo de atividades, as quais devem ser gerenciadas, integradas e coordenadas. Assim, este autor entende que um SGM deve ser construído com base nos seguintes princípios:

- armazenar coleções de modelos e informações relacionadas com modelos e otimizadores em uma Base de Conhecimento de Modelos, para controlar o uso e a reutilização de modelos;
- manter a independência modelo-dados, para permitir que valores, tamanho, estrutura de armazenamento e método de acesso dos dados possam ser alterados sem alteração na estrutura do modelo;
- manter a independência modelo-otimizador, para permitir que o modelo seja resolvido por diferentes algoritmos de solução;
- dividir a especificação da estrutura do modelo em duas partes: especificação externa (interface) e especificação interna (estrutura e comportamento), para facilitar o compartilhamento e a reutilização de modelos.

Alguns trabalhos procuram enfatizar que a integração de modelos pode considerar, até mesmo, modelos construídos a partir de diferentes paradigmas. Dolk (1993), por exemplo, em sua discussão sobre Ambientes de Modelagem Integrados, ilustra esta possibilidade com a integração de um Modelo de Transporte (modelo de otimização) com um Modelo de Demanda (modelo econométrico) e um Modelo de Fornecimento (modelo de simulação). Outra possibilidade, também levantada por Dolk (1993) é a integração de modelos a processadores de texto, planilhas, bancos de dados e pacotes gráficos. Kottemann and Dolk (1992), apresentam uma revisão das capacidades de integração de modelos existentes em algumas linguagens de modelagem e discutem as abordagens teóricas atualmente empregadas para a integração de modelos. Estes autores argumentam que a integração de modelos envolve não apenas a junção de representações de componentes de modelos mas também o acoplamento de processos que manipulam estes componentes e apontam os fatores que limitam a capacidade de integração de modelos tanto nas linguagens de modelagem como nas abordagens teóricas. Estes mesmos autores, em outro trabalho, apresentam os requisitos (e uma seleção de trabalhos relevantes acerca de cada um destes

unesp

requisitos) necessários para um Ambiente de Modelagem ser capaz de integrar a definição e a manipulação de modelos (Dolk and Kottemann, 1993).

De uma maneira geral, a pesquisa sobre Sistemas de Gerenciamento de Modelos é influenciada pelas três maneiras com que um modelo tem sido tratado (Chang *et al.*, 1993). Um enfoque é considerar modelos como procedimentos ou programas que devem ser executados para resolver um determinado problema. Em outro enfoque, os modelos são tratados como dados a serem analisados, como nos trabalhos de Dolk (1988a) e Lenard (1988; 1993). Este enfoque facilita a síntese de modelos complexos a partir de modelos já existentes (como nos trabalhos discutidos acima, relativos às questões de reutilização e integração de modelos). Um terceiro enfoque é tratar a própria modelagem como um problema a ser resolvido. Um SGM desenvolvido sob este enfoque deve ser capaz de converter uma descrição do problema em equações matemáticas. Os trabalhos sobre sistemas capazes de formular modelos de Programação Matemática adotam este último enfoque.

Existem duas abordagens predominantes para o desenvolvimento de um SGM capaz de dar suporte à formulação de modelos. Uma abordagem baseia-se apenas no conhecimento estrutural e sintático dos modelos matemáticos (como, por exemplo, em Stohr, 1988). A segunda abordagem combina o conhecimento sintático dos modelos com o conhecimento semântico do domínio de aplicação (como em Binbasioglu and Jarke, 1986, por exemplo). O trabalho de Santhanam and Schniedejans (1993) descreve o desenvolvimento de um SGM que emprega esta segunda abordagem, combinando conhecimento (sintático) de programação por metas (*goal programming*), com conhecimento (semântico) de seleção de projetos de sistemas de informação.

A literatura sobre Sistemas de Gerenciamento de Modelos apresenta também vários trabalhos de revisão. Blanning (1993) examina a literatura sobre SGM a partir de dois ângulos: o uso de conceitos relacionais e o uso de técnicas de Inteligência Artificial no gerenciamento de modelos. Uma revisão complementar sobre SGM é apresentada por Chang *et al.* (1993), e aborda questões relacionadas com o emprego de técnicas de manipulação de conhecimento para a análise de dados em Sistemas de Apoio à Decisão e para o desenvolvimento de SGMs.

Um Sistema de Análise de Soluções é descrito por Greenberg (1983; 1993; 1994). Este sistema (ANALYZE) aceita a solução obtida por um otimizador, realiza análises destes resultados e produz relatórios para o usuário. A integração dos sistemas CAMPS (Lucas, 1986; Lucas and Mitra, 1988) e ANALYZE é discutida por Greenberg *et*

unesp

al. (1987) como um relacionamento entre Sistema de Processamento de Modelos e Sistema de Análise de Soluções. Neste caso, procura-se utilizar as capacidades de formulação de modelos do sistema CAMPS, acrescidas de algumas informações textuais acerca das entidades básicas que compõem o modelo, para gerar o arquivo de entrada para o sistema ANALYZE que cuida, então, da análise do modelo e de suas soluções, sendo capaz, também, de explicar em linguagem pseudo-natural o significado de componentes do modelo.

O emprego de técnicas de Inteligência Artificial, principalmente Sistemas Especialistas, para auxiliar as funções de um Ambiente de Modelagem tem recebido também grande atenção nos últimos anos (ver, por exemplo, Angehrn and Lüthi, 1990; Connell and Powell, 1990; Doukidis and Paul, 1992; Murphy *et al.*, 1992b; Potter *et al.*, 1992; Raghunathan, 1992; El-Najdawi and Stylianou, 1993; McBride and O'Leary, 1993; Mookerjee and Chaturvedi, 1993; Santhanam and Schniederjans, 1993; Sen *et al.*, 1992-93; Sen *et al.*, 1993; Mentzas, 1994).

1.3 - Objetivos do Trabalho

Este trabalho reúne contribuições do autor para o desenvolvimento de um ambiente de modelagem em redes. As contribuições discutidas neste trabalho referem-se a:

- o desenvolvimento de um Sistema de Processamento de Modelos e sua integração a um Sistema de Visualização e Análise de Soluções e a um Sistema de Gerenciamento de Bancos de Dados. Discute-se a construção de uma linguagem algébrica de modelagem em redes lineares por partes e sua utilização na formulação e na solução de modelos.

- uma proposta de Sistema de Interface com Usuário e seu relacionamento a um Sistema Especialista em Formulação de Modelos. Discute-se o desenvolvimento de uma interface gráfica para a representação conceitual de um problema de Programação Linear e a utilização de um sistema especialista para obter a formulação algébrica do problema.

- a construção de uma Base de Algoritmos de Solução. Discute-se o desenvolvimento e a implementação de algoritmos para a solução de modelos de fluxo em redes lineares por partes e de algoritmos heurísticos eficientes para a solução de problemas

unesp

clássicos de localização, tais como: p-medianas, localização simples e localização capacitada.

Pretende-se que tais contribuições levem a um ambiente de modelagem constituído como ilustra a Figura 1.1.

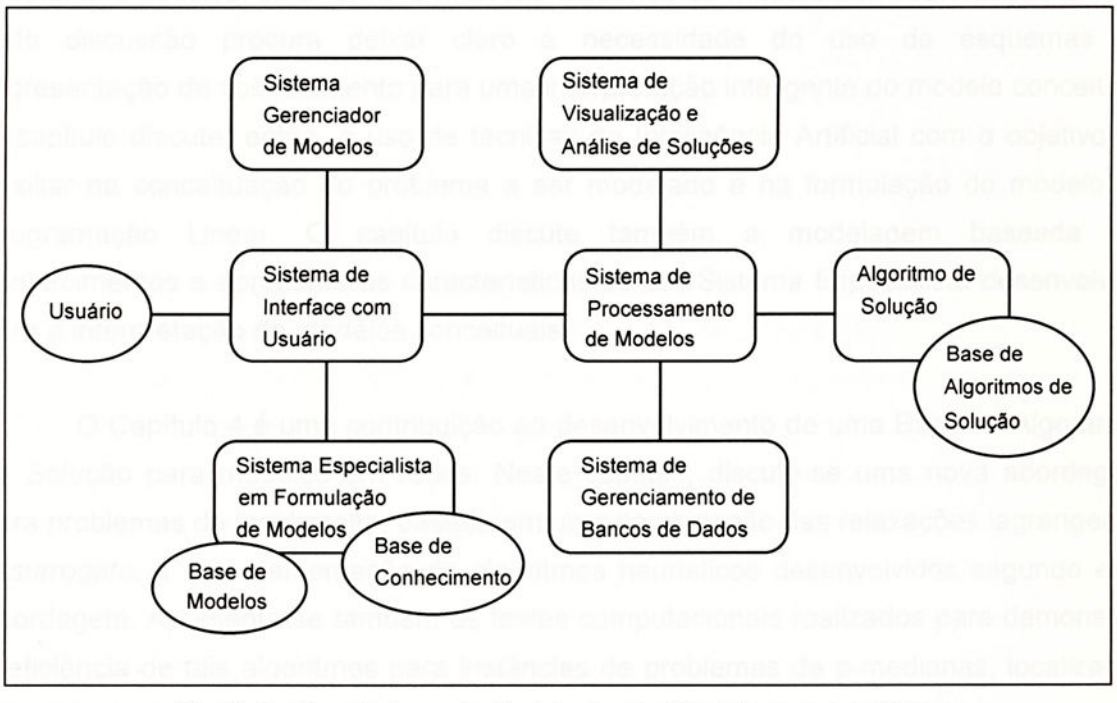


Fig. 1.1 - Arquitetura de Ambiente de Modelagem em Redes

1.4 - Sumário do Trabalho

Este primeiro capítulo procurou situar o trabalho no contexto de ambientes de modelagem e relacionar as contribuições do autor para o desenvolvimento de um Ambiente de Modelagem em Redes. Tais contribuições são consideradas em detalhes nos capítulos seguintes.

O Capítulo 2 apresenta uma revisão bibliográfica das principais ferramentas já desenvolvidas para auxiliar na construção, documentação e manutenção de modelos. Comenta-se sobre características importantes de algumas linguagens algébricas existentes e, em especial, das linguagens AMPL e MPL. Discute-se, então, a linguagem NPL2, desenvolvida especialmente para a modelagem em redes.

No Capítulo 3 discute-se uma proposta de interface inteligente para formulação de modelos de Programação Linear. Esta interface considera a tarefa de modelagem

unesp

dividida em três estágios, nos quais são especificados um modelo de dados (como são organizados os dados necessários ao modelo), um modelo conceitual (como o problema a ser modelado é visto em termos de entidades e seus relacionamentos) e um modelo de Programação Linear. Apresenta-se uma representação gráfica para estes modelos e discute-se, para alguns exemplos clássicos, como a partir da conceitualização do problema é possível obter uma formulação algébrica do modelo de Programação Linear. Esta discussão procura deixar claro a necessidade do uso de esquemas de representação de conhecimento para uma interpretação inteligente do modelo conceitual. O capítulo discute, então, o uso de técnicas de Inteligência Artificial com o objetivo de auxiliar na conceituação do problema a ser modelado e na formulação do modelo de Programação Linear. O capítulo discute também a modelagem baseada em conhecimentos e apresenta as características de um Sistema Especialista desenvolvido para a interpretação de modelos conceituais.

O Capítulo 4 é uma contribuição ao desenvolvimento de uma Base de Algoritmos de Solução para modelos em redes. Neste capítulo, discute-se uma nova abordagem para problemas de localização, baseda em uma combinação das relaxações lagrangeana e *surrogate*, e a implementação de algoritmos heurísticos desenvolvidos segundo esta abordagem. Apresenta-se também os testes computacionais realizados para demonstrar a eficiência de tais algoritmos para instâncias de problemas de p-medianas, localização simples e localização capacitada, conhecidos na literatura.

Apresenta-se no Capítulo 5 uma discussão acerca das contribuições visando a construção de um ambiente de modelagem em redes, tratadas neste trabalho, comenta-se sobre os desenvolvimentos atuais e aponta-se a direção de futuros trabalhos.

O trabalho inclui ainda alguns apêndices. No Apêndice A apresenta-se a descrição da linguagem NPL2 discutida no Capítulo 2. Os apêndices B e C contêm implementações de algoritmos discutidos no Capítulo 3. No Apêndice B, a listagem de um programa Lisp que implementa o algoritmo de interpretação de restrições e, no Apêndice C, a listagem de um programa OPS5 que implementa o sistema especialista em formulação de modelos. No Apêndice D apresenta-se o estágio atual de desenvolvimento do ambiente MoNet, que incorpora os assuntos tratados neste trabalho.

unesp

Capítulo 2

Linguagem Algébrica de Modelagem em Redes

2.1 - Introdução

Para que um modelo de Programação Linear possa ser resolvido por computador é necessário representá-lo em uma forma aceitável pelo programa otimizador. O padrão de representação melhor estabelecido e mais largamente utilizado como entrada para otimizadores é o formato MPS (IBM, 1976). Esta forma, entretanto, não é conveniente para tratamento por especialistas. Assim, para facilitar a construção, a documentação e a manutenção de modelos, muitas ferramentas têm sido desenvolvidas para traduzir uma representação conveniente para especialistas em uma representação conveniente a um programa otimizador.

Murphy *et al.* (1992a) relacionam diversos esquemas de representação de modelos e discutem a influência do esquema de representação na efetividade do processo de modelagem. Para isto, comparam os esquemas em termos de seis dimensões: *generalidade* (aplicabilidade do esquema na representação de diferentes tipos de modelos), *declaratividade* (de que forma o esquema permite a definição do modelo desvinculada da forma pela qual será resolvido), *complexidade* (da representação de modelos utilizando o esquema), *concretude* (nível de abstração dos objetos do modelo no esquema), *modularidade* (capacidade de representação de modelos em classes e subclasses no esquema) e *independência* (do modelo representado no esquema, em relação aos dados e ao programa otimizador).

Os autores concluem que, embora alguns esquemas são melhores do que outros em alguns sentidos, nenhum deles domina os demais em todas as dimensões. Assim, as linguagens algébricas são apontadas como bons esquemas de representação em relação à generalidade, declaratividade, complexidade e independência, mas falham em relação à concretude e modularidade, dimensões nas quais a modelagem estruturada, os grafos atividade-restrição, a representação por redes e as linguagens de ícones, são melhores.

unesp

Das várias formas de representação de modelos já propostas, as linguagens algébricas constituem um dos principais esquemas (Greenberg and Murphy, 1992). O número de linguagens algébricas de modelagem existentes é uma prova disto. Basicamente, as linguagens algébricas caracterizam-se por:

- uso de conjuntos para representação de domínios;
- capacidade de indexação;
- expressões aritméticas simples e diretas;
- descrição simbólica de conjuntos de dados;
- capacidade de instanciação de dados;
- habilidade em impor restrições sobre parâmetros e variáveis.

Em consequência da grande diversidade de linguagens algébricas, vários são os trabalhos que comparam as características destas linguagens. Os trabalhos mais recentes comparam as capacidades de linguagens que já alcançaram o *status* de produto comercial. O trabalho de Steiger and Sharda (1993), por exemplo, faz referência e contrasta as características das linguagens AMPL, GAMS, LINGO, LPL, MPL, PC-PROG e XPRESS-LP. Estes autores observam pontos que se relacionam com a efetividade da linguagem, como a possibilidade de utilização de conjuntos de índices, expressões aritméticas livres de formato, conjuntos e funções. Em relação à eficiência, as linguagens são comparadas quanto a capacidade de expressar o modelo de forma compacta, a capacidade de análise e recuperação de erros, a facilidade em ser aprendida e quanto à qualidade da interface que apresentam. O trabalho indica ainda como importante ponto de comparação, a existência de técnicas inteligentes incorporadas à linguagem, como a redução automática do tamanho do modelo (como é discutida por Andrei and Barbulescu, 1993 e Babayev and Mardanov, 1994), a eliminação de índices e a identificação de restrições inviáveis.

Outro trabalho (Kuip, 1993) compara as linguagens LANDO, UIMP, LPM, CAMPS, LMC, MGG, LAMP, ULP, GAMS, LPL, AMPL e MPL em função da capacidade de indexação, da separação entre modelo e dados, e em termos da linguagem e seu ambiente. Este trabalho apresenta também as contribuições recentes às linguagens de modelagem, algumas já incorporadas nas linguagens estudadas. Com relação à melhoria dos conceitos existentes nas linguagens, o trabalho aponta como contribuições mais recentes:

- *conjuntos hierárquicos* - possibilidade de se ter conjuntos de índices como elementos de conjuntos de índices;

unesp

- *conjuntos compostos* - possibilidade de dar nome a subconjuntos de um produto cartesiano de dois conjuntos;
- *abstração de modelo* - possibilidade de enxergar o modelo como um objeto e assim permitir que modelos possuam características como tamanho e classe, além de poderem ser combinados para formar novos modelos;
- *asserções lógicas* - possibilidade de estabelecer assertivas explícitas sobre os dados do modelo;
- *análise dimensional* - possibilidade de verificar o uso consistente de unidades em equações;
- *estruturas especiais* - possibilidade de especificar estruturas especiais como modelos em redes, modelos lineares por partes, dentre outras;
- *especificação interativa* - possibilidade de uso de interfaces, menus, e outras formas de comunicação com o usuário;
- *programação inteira* - possibilidade de especificar modelos de programação inteira.

Algumas destas contribuições recentes às linguagens algébricas são discutidas por Geoffrion (1992a; 1992b; 1992c), Bisschop and Kuip (1993a; 1993b), Bisschop and Fourer (1994), Maturana (1994), Mitra *et al.* (1994) e Hadjiconstaninou and Mitra (1994).

Neste capítulo discute-se o desenvolvimento da linguagem algébrica NPL2 (*Network Piecewise-Linear Programming Language*), desenvolvida para a representação de modelos em redes lineares por partes. Esta linguagem baseia-se nas linguagens AMPL (Fourer *et al.*, 1990; Fourer *et al.*, 1993) e MPL (Kristjansson, 1993). Da linguagem AMPL adota-se as construções sintáticas e as extensões propostas por Fourer and Gay (1993) referentes às estruturas de rede e de modelos lineares por partes. Da linguagem MPL adota-se a representação e o tratamento de dados segundo o modelo relacional, como proposto por Kristjansson *et al.* (1993).

2.2 - Linguagem de Modelagem em Redes

A formulação algébrica e a solução de problemas de Programação Linear podem fazer uso de certas estruturas especiais. É o caso, por exemplo, de modelos de fluxo em redes. Neste caso, dispondo-se de uma linguagem capaz de expressar estruturas de rede, um modelo de Programação Linear pode ser formulado de forma mais simples e natural e, mais importante, pode ser submetido a algoritmos que, fazendo uso da estrutura especial do problema, são capazes de obter soluções eficientemente.

unesp

A formulação de um modelo NPL2 é simples porque, nesta linguagem, a estrutura de rede é estabelecida diretamente através da descrição de seus nós e arcos. A solução de um problema NPL2 é rápida pois emprega-se para isto um algoritmo especialmente desenvolvido para redes lineares por partes (Marins, 1987), com eficiência comprovada por estudo comparativo com outros algoritmos para redes (Marins *et al.*, 1994; Senne *et al.*, 1994; Marins *et al.*, 1997).

É importante ressaltar a distinção que se faz, neste capítulo, entre um modelo (ou programa) e um problema (ou instância de modelo) NPL2. Entende-se por modelo NPL2, a formulação em notação algébrica, obedecendo a sintaxe da linguagem NPL2, de uma estrutura de rede (por meio da descrição de seus componentes) e de um modelo de otimização (por meio da descrição de conjuntos de índices, parâmetros, variáveis e objetivo). Já um problema NPL2 é visto como uma particularização do modelo a um determinado conjunto de dados, ou seja, a uma particularização de seus conjuntos e parâmetros. O exemplo a seguir (adaptado de Fourer and Gay, 1993) ilustra as características essenciais de um modelo e de um problema NPL2 (Figura 2.1).

```
model maxflow;
set N;
par
   srce where (srce in N);
   sink where (sink in N) and (sink <> srce);
set A := (N diff {sink}) cross (N diff {srce});
par
   lb[<i,j> in A] where (lb[i,j] >= 0);
   ub[<i,j> in A] where (ub[i,j] >= lb[i,j]);
node Bal[i in N];
arc Flow[<i,j> in A]
   from Bal[i] to Bal[j]
   bounds < lb[i,j] : ub[i,j] >;
Flow_In
   to Bal[srce]
   bound >= 0;
Flow_Out
   from Bal[sink];
maximize Total_Flow.
```

Fig. 2.1 - Modelo de Fluxo Máximo

Neste exemplo tem-se um modelo de fluxo máximo. O objetivo é maximizar o fluxo que entra em `srce` (o nó-origem), ou equivalentemente, o fluxo que sai de `sink` (o nó-destino), sujeito às restrições de conservação de fluxo nos nós e às limitações de fluxo nos arcos. A formulação começa com o nome do modelo:

```
model maxflow;
```

e com a especificação de um conjunto (`N`) e de dois parâmetros (`srce` e `sink`) correspondentes a elementos deste conjunto:

```
set N;
par
   srce where (srce in N);
   sink where (sink in N) and (sink <> srce);
```

Observe que, da forma como declarado, `srce` e `sink` podem ser quaisquer dois elementos distintos do conjunto N. A seguir a formulação estabelece um conjunto A e os parâmetros `lb` e `ub`, definidos para elementos do conjunto A:

```
set A := (N diff {sink}) cross (N diff {srce});
par
   lb[<i,j> in A] where (lb[i,j] >= 0);
   ub[<i,j> in A] where (ub[i,j] >= lb[i,j]);
```

O modelo estabelece a seguir, o conjunto de nós (`Bal`), cujos índices pertencem ao conjunto N, e o conjunto de arcos (`Flow`), cujos índices estão em A:

```
node Bal[i in N];
arc Flow[<i,j> in A]
   from Bal[i] to Bal[j]
   bounds < lb[i,j] : ub[i,j] >;
```

Deve-se observar a distinção entre conjuntos de componentes de rede (como `Bal` e `Flow`) e conjuntos de índices de componentes (como `N` e `A`).

Os parâmetros `lb` e `ub` definidos acima correspondem aos limites (inferior e superior, respectivamente) de capacidade de fluxo nos arcos do conjunto `Flow`. O modelo estabelece ainda dois conjuntos de arcos especiais:

```
Flow_In
   to Bal[srce]
   bound >= 0;
Flow_Out
```

```
from Bal[sink];
```

correspondentes ao arco cujo destino é o nó de índice `srce`, e ao arco cuja origem é o nó indexado por `sink`. Por fim, tem-se a declaração do objetivo do modelo:

```
maximize Total_Flow.
```

A figura a seguir ilustra (utilizando, em parte, a notação adotada por Glover *et al.*, 1990; Glover *et al.*, 1992) o problema NPL2 correspondente, para uma instância do modelo associada aos seguintes dados:

```
N = { 1, 2, 3, 4, 5 }
srce = 1
sink = 5
```

i	j	lb[i,j]
1	2	0
1	3	0
2	3	10
2	4	0
3	4	20
3	5	0
4	5	0

i	j	ub[i,j]
1	2	300
1	3	300
2	3	50
2	4	50
3	4	400
3	5	250
4	5	300

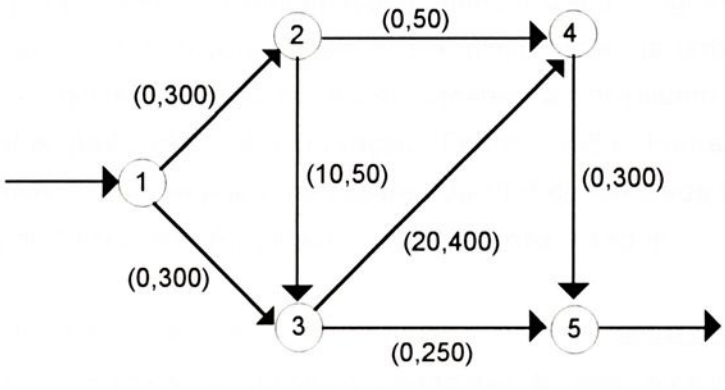


Fig. 2.2 - Problema de Fluxo Máximo

onde o conjunto `Flow_In` corresponde ao arco que entra no vértice 1, o conjunto `Flow_Out` corresponde ao arco que sai do vértice 5 e os demais arcos correspondem ao conjunto `Flow`.

Observe que a instância de dados acima estabelece os valores de lb e ub apenas para alguns pares (i,j) do conjunto A, correspondentes aos arcos da rede da

Figura 2.2. Para as demais combinações de índices (i,j) não presentes nas tabelas de dados, considera-se que lb[i,j] = 0 e ub[i,j] = 0, ou seja, a inexistência do arco entre os nós indexados por i e j. Observe também que o modelo não estabelece condições explícitas de balanço de fluxo para os nós pois, neste caso, o fluxo que entra iguala o fluxo que sai de cada nó.

Este exemplo ilustra apenas algumas características da linguagem NPL2. Na Seção 2.4 discute-se, em linhas gerais, as características essenciais da linguagem. Mais detalhes podem ser vistos no Apêndice A. Antes, apresenta-se, sucintamente, a programação em redes lineares por partes e o algoritmo de solução (devido a Marins, 1987) utilizado pela linguagem.

2.3 - Programação em Redes Lineares Por Partes

O modelo de fluxo em redes é um dos mais conhecidos e mais importantes na Programação Linear. Este modelo baseia-se numa rede cujos nós são conectados por arcos orientados (como a da Figura 2.2) nos quais imagina-se fluir alguma entidade, de forma que as variáveis de decisão do modelo correspondem aos níveis destes fluxos. O objetivo é minimizar ou maximizar alguma função linear dos fluxos, sujeito a restrições de limites nos fluxos e de balanceamento de fluxo nos nós.

Outra área importante da Programação Matemática é a Programação Linear por Partes (PLP). Modelos PLP preocupam-se com a minimização de uma função-objetivo linear por partes convexa, sujeita a restrições lineares, e constituem uma formulação natural para muitos problemas de otimização (Fourer, 1985; Fourer, 1988; Fourer, 1992). Neste trabalho, considera-se uma subárea da PLP denominada Programação em Redes Lineares por Partes (PRLP), caracterizada da forma a seguir.

Uma função de várias variáveis $C(x_1, x_2, \cdots, x_n)$ é <u>separável</u> se puder ser representada como uma soma de funções dependentes de uma única variável, ou seja,

$$C(x_1, x_2, \cdots, x_n) = C_1(x_1) + C_2(x_2) + \cdots + C_n(x_n)$$

Uma função de uma variável $C_j(x_j)$ é <u>convexa</u> e <u>linear por partes</u> se for linear sobre cada um de uma série de intervalos consecutivos $[d_j^k, d_j^{k+1}]$ para os quais existem coeficientes de custo c_j^k associados, onde os coeficientes c_j^k e os limitantes (ou *breakpoints*) d_j^k formam seqüências monotonicamente crescentes para cada variável x_j, ou seja:

$$c_j^1 < c_j^2 < \cdots < c_j^k < c_j^{k+1} < \cdots < c_j^{m_j}$$

e

$$d_j^1 < d_j^2 < \cdots < d_j^k < d_j^{k+1} < \cdots < d_j^{m_j}$$

A figura a seguir ilustra uma função convexa linear por partes típica. Detalhes sobre propriedades fundamentais de funções convexas, separáveis e lineares por partes, podem ser encontrados em Fourer (1985).

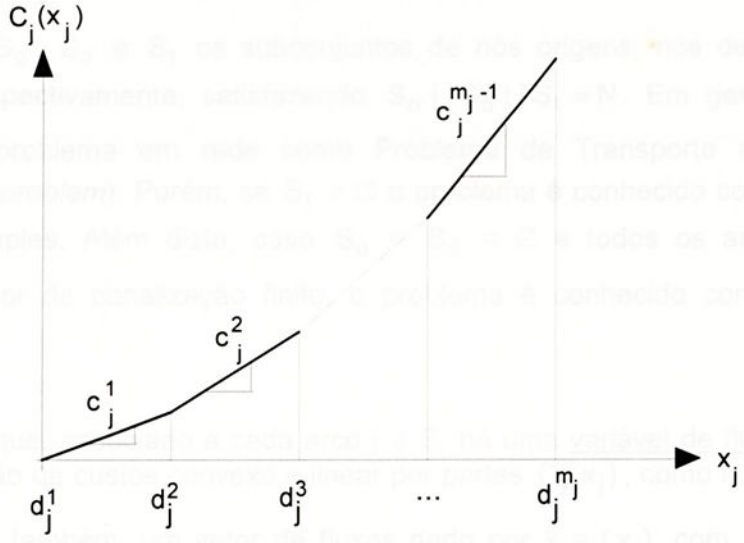


Fig. 2.3 - Função Convexa Linear por Partes

Neste trabalho, os modelos em redes lineares por partes são tratados como uma generalização da teoria existente para o problema de fluxo com custo mínimo da Programação em Redes Lineares (ver Ahuja *et al.*, 1993).

Seja G = (N, E) um grafo, onde N é um conjunto com | N | = n nós (ou vértices) e E é um conjunto de | E | = m arcos, tal que cada arco $a_k \in E$ é identificado por um par de vértices (v_i, v_j), $v_i, v_j \in N$. Um caminho entre dois vértices v_o e v_d ($v_o, v_d \in N$, $v_o \neq v_d$) em um grafo G é uma sequência alternada de vértices e arcos $(v_1, a_1, \cdots, v_k, a_k, v_{k+1})$, com $k > 0$, $v_1 = v_o$ e $v_{k+1} = v_d$, tal que cada arco a_j (j = 1, ..., k) da sequência é identificado pelo par de nós (v_j, v_{j+1}). Um grafo G é conectado se existe, pelo menos, um caminho entre qualquer par de vértices de G. Se existe uma função Ψ que mapeia cada arco $j \in E$ a um par ordenado (t_j, h_j), com $t_j, h_j \in N$, $t_j \neq h_j$, sendo denominados, respectivamente, nó cauda e nó cabeça do arco j, então diz-se que G é um grafo orientado. Define-se uma rede como um grafo conectado orientado.

Seja $A = \left[a_{ij}\right]_{n \times m}$ a matriz de incidência nó-arco (de ordem n x m) de uma rede G = (N, E), tal que, para todos os arcos $j \in E$, têm-se: $a_{t_j j} = -1$, $a_{h_j j} = +1$, e os demais elementos da coluna j da matriz A nulos.

Considere que o valor da demanda existente no nó $i \in N$ seja dado por b_i. Neste caso, o nó i é classificado como sendo um nó origem, um nó destino, ou um nó de transbordo se $b_i < 0$, $b_i > 0$ ou $b_i = 0$, respectivamente. Suponha, ainda, que a soma das demandas para todos os nós $i \in N$ seja igual a zero.

Sejam S_o, S_d e S_t os subconjuntos de nós origens, nós destinos e nós de transbordo, respectivamente, satisfazendo $S_o \cup S_d \cup S_t = N$. Em geral, denomina-se este tipo de problema em rede como Problema de Transporte com Transbordo (*transshipment problem*). Porém, se $S_t = \varnothing$ o problema é conhecido como Problema de Transporte Simples. Além disto, caso $S_o = S_d = \varnothing$ e todos os arcos tenham um limitante superior de canalização finito, o problema é conhecido como Problema de Circulação.

Admita que, associado a cada arco $j \in E$, há uma variável de fluxo não-negativa x_j, e uma função de custos convexa e linear por partes $C_j(x_j)$, como ilustrado na Figura 2.3. Considere, também, um vetor de fluxos dado por $x = (x_j)$, com $x_j \in [d_j^{f_j}, d_j^{f_j+1}]$. Define-se, neste caso, f_j como sendo o intervalo corrente do arco j. Pode-se mostrar que é possível calcular o valor da função de custos associada, $C_j(x_j)$, através da expressão:

$$C_j(x_j) = \sum_{k=1}^{f_j-1} \left[c_j^k \left(d_j^{k+1} - d_j^k \right) \right] + c_j^{f_j} (x_j - d_j^{f_j})$$

Um problema primal (P) da Programação em Redes Lineares por Partes pode ser formulado como segue:

$$\min C(x)$$

(P) sujeito a:

$$Ax = b,\ x \geq 0$$

onde $C(x) = \sum_{j \in E} C_j(x_j)$, e o vetor de fluxos x, o vetor de demandas b, e a matriz de incidência A são como definidos anteriormente.

Este problema pode ser transformado num problema linear equivalente, de maior porte. Basta criar, para cada arco $j \in E$, identificado pelo par de nós distintos (t_j, h_j), no multigrafo resultante da transformação, tantos arcos paralelos entre os nós t_j e h_j quantos sejam em $C_j(x_j)$ o número de segmentos lineares, com limitantes inferior e superior de fluxo adequados. Como os coeficientes de custo c_j^k são admitidos formar uma sequência monotonicamente crescente, o problema linear equivalente, resultante desta transformação, torna-se um problema de fluxo com custo linear padrão, podendo ser resolvido por métodos convencionais. Contudo, Marins (1987) mostra que a eficiência de tal procedimento pode ser melhorada pela resolução direta do problema de PRLP.

Como apontado e discutido por Fourer (1985), de um modo geral, intercambiando-se os papéis dos *breakpoints* d_j^k, e dos correspondentes coeficientes de custos c_j^k, na função $C_j(x_j)$, pode-se obter outra função convexa e linear por partes, $D_j(z_j)$ que é a função dual (ou conjugada) de $C_j(x_j)$. Esta função dual $D_j(z_j)$ terá inclinação d_j^{k+1} no intervalo $[c_j^k, c_j^{k+1}]$ e pode ser calculada pela expressão abaixo:

$$D_j(z_j) = \sum_{k=1}^{f_j - 1} \left[d_j^{k+1} \left(c_j^{k+1} - c_j^k \right) \right] + d_j^{f_j+1} (z_j - c_j^{f_j})$$

Assim, pode-se formular o problema dual (D) de um programa em redes lineares por partes como sendo:

$$\max \; b' y - D(z)$$

(D) sujeito a:

$$A' y - z \leq 0, \text{ com } z_j \geq c_j^1 \text{ para } j \in E$$

onde $D(z) = \sum_{j \in E} D_j(z_j)$. Neste trabalho, os vetores de variáveis duais y e z, da formulação acima, são denominados como vetor de preços duais nos nós e vetor de custos correntes nos arcos, respectivamente. Fourer (1985) demonstra que, devido à relação dual que existe entre as funções $C_j(x_j)$ e $D_j(z_j)$, se, no problema primal, o fluxo no arco j é tal que $x_j \in [d_j^{f_j}, d_j^{f_j+1}]$, então, no problema dual, tem-se que $z_j \in [c_j^{f_j}, c_j^{f_j+1}]$.

Apresenta-se, a seguir, algumas definições importantes para a PRLP, as quais são extensões de definições válidas para a Programação em Redes Lineares.

Uma base de trabalho para um problema PRLP é definida por uma árvore geradora (*spanning tree*), T, com (n-1) arcos básicos, um nó especificado como sendo a raiz de T, e um vetor de intervalos correntes $f = (f_j)$.

Para cada arco básico $j \in T$, define-se R(T, j) como sendo o subconjunto de nós na sub-árvore que contém a raiz de T, obtida quando o arco j é removido. Deste modo, um arco básico $(t_j, h_j) \in T$ pode ser classificado como:

- arco direcionado a partir da raiz (ou arco descendente) se $h_j \in R(T, j)$;
- arco direcionado para a raiz (ou arco ascendente) se $t_j \in R(T, j)$.

Define-se, ainda, para todo arco básico $j \in T$ dois subconjuntos de arcos não-básicos incidentes aos nós de R(T, j), dados por:

$Tail(T, j) = \{ k \notin T : t_k \in R(T, j) \text{ e } h_k \notin R(T, j) \}$;
$Head(T, j) = \{k \notin T : t_k \notin R(T, j) \text{ e } h_k \in R(T, j) \}$.

Finalmente, define-se para cada arco $j \in E$ um valor p_j tal que:

$p_j = 0$, se $j \notin T$;
$p_j = +1$, se $t_j \in R(T, j)$;
$p_j = -1$, se $h_j \in R(T, j)$.

Seja P_i o único caminho existente em T, da raiz até o nó i. Assim, dada uma base e um vetor de custos correntes f para o problema (P) acima, a única solução básica { x, y, z } associada pode ser calculada por:

$y_i = 0$, se i é a raiz de T,

$y_i = \sum_{j \in T} [c_j^{f_j} \mid j \text{ é arco ascendente em } P_i] - \sum_{j \in T} [c_j^{f_j} \mid j \text{ é arco descendente em } P_i]$, se $i \neq$ raiz de T,

$z_j = c_j^{f_j}$, se $j \in T$,

$z_j = \max \{ c_j^1 , y_{h_j} - y_{t_j} \}$, se $j \notin T$,

$x_j = d_j^{f_j}$, se $j \notin T$,

$$x_j = p_j \left(\sum_{i \notin R(T,j)} b_i + \sum_{k \in Tail(T,j)} x_k - \sum_{k \in Head(T,j)} x_k \right), \text{ se } j \in T$$

onde T é a árvore geradora correspondente à base. Esta solução básica é primal viável se $x_j \in [\, d_j^{f_j}, d_j^{f_j+1}]$, para todo j.

Deve-se observar que a solução básica assim determinada satisfaz as condições de folgas complementares válidas para a PRLP (ver Marins, 1987), as quais, supondo para todo arco $j \in E$ coeficientes de custo dados por $c_j^{f_j-1} = -\infty$, caso não estejam definidos, podem ser expressas por:

$$d_j^{f_j} < x_j < d_j^{f_j+1} \quad \Rightarrow \quad y_{h_j} - y_{t_j} = c_j^{f_j}$$

$$c_j^{f_j-1} < y_{h_j} - y_{t_j} < c_j^{f_j} \quad \Rightarrow \quad x_j = d_j^{f_j}$$

$$z_j = \max \left\{ c_j^1,\ y_{h_j} - y_{t_j} \right\}$$

Note-se, ainda, que se $x_j \in [\, d_j^{f_j}, d_j^{f_j+1}]$ para $j \in T$, o vetor de fluxos nos arcos $x = (x_j)$ é viável para o problema (P), e os vetores de variáveis duais $y = (y_j)$, e $z = (z_j)$ são sempre viáveis para o problema (D) associado. Com o auxílio do vetor de preços duais y, pode-se definir, para todo arco $j \in E$, dois tipos de coeficientes de custo reduzido, que avaliam a influência, no valor da função objetivo de (P), de modificações no atual valor da variável de fluxo x_j:

- c_j^-, denominado <u>coeficiente de custo reduzido à esquerda</u>, relacionado com reduções no atual valor da variável de fluxo x_j;
- c_j^+, denominado <u>coeficiente de custo reduzido à direita</u>, relacionado com aumentos no atual valor da variável de fluxo x_j.

Para uma dada solução básica { x, y, z }, estes coeficientes de custo reduzido podem ser calculados por meio de:

$c_j^- = c_j^+ = 0$, para $j \in T$;

$c_j^- = c_j^{f_j-1} - y_{h_j} + y_{t_j}$ e $c_j^+ = c_j^{f_j} - y_{h_j} + y_{t_j}$, para $j \notin T$.

Analogamente ao que ocorre nos programas em redes lineares canalizadas, o valor da função objetivo C(x) decrescerá, como é de interesse, nos seguintes casos:

$c_j^+ < 0$ e o valor do fluxo x_j no arco j for aumentado;

$c_j^- > 0$ e o valor do fluxo x_j no arco j for diminuído.

unesp

Pode ser demonstrado (ver Fourer, 1985) que uma dada solução básica viável { x, y, z }, associada a uma árvore geradora T e um vetor de custos correntes f, é solução ótima se para todo arco não-básico $s \notin T$, tem-se $c_s^- \leq 0$ ou $c_s^+ \geq 0$. Deve-se observar, que não é possível ocorrer simultaneamente as condições $c_s^- \leq 0$ e $c_s^+ \geq 0$ pois, sempre tem-se que $c_s^{f_s - 1} < c_s^{f_s}$, devido a hipótese de convexidade admitida para a função objetivo separável e linear por partes C(x).

Além dos conceitos apresentados acima, faz-se necessário para a apresentação dos passos do algoritmo Primal Simplex para PRLP, o conceito de ciclo de arcos na rede. Da teoria de Programação em Redes Lineares sabe-se que, o acréscimo de um arco não-básico s à árvore geradora T, associada a uma base, gera um ciclo $Q_s = Q_s^+ \cup Q_s^-$ em $T \cup \{s\}$, definido por $Q_s^+ = \{ j \in Q_s \mid$ o arco j e o arco s têm a mesma orientação em $Q_s \}$ e $Q_s^- = Q_s \setminus Q_s^+$.

Adota-se, a seguir, a seguinte notação: $\hat{c}_s = c_s^+$ se $c_s^+ < 0$; $\hat{c}_s = c_s^-$ se $c_s^- > 0$, e $\hat{c}_s = 0$, caso contrário. Deve-se observar que valores de $\hat{c}_s$ não nulos estão associados a fluxos em arcos não-básicos candidatos a melhorar a solução corrente. Os passos básicos do algoritmo Primal Simplex para a PRLP podem ser estabelecidos como se segue.

Algoritmo Primal Simplex para a PRLP

Seja { x, y, z } uma solução básica viável associada à uma árvore geradora T e um vetor de intervalos correntes f.

1. [Teste de Otimalidade]
   Selecione um arco não-básico $s \notin T$, tal que $\hat{c}_s \neq 0$, para entrar na base.

   Se não existe tal arco então pare: Solução ótima.
   Caso contrário faça $s_0 \leftarrow s$ e obtenha o ciclo $Q = Q^+ \cup Q^-$ formado em $T \cup \{s_0\}$.
2. [Exploração Continuada]
   se $\hat{c}_s < 0$

   então faça:

   $\delta_0 \leftarrow d_s^{f_s+1} - d_s^{f_s}$

   $\delta_1 \leftarrow \min \left\{ d_j^{f_j+1} - x_j : j \in Q^+ \setminus \{s_0\} \right\}$

   $\delta_2 \leftarrow \min \left\{ x_j - d_j^{f_j} : j \in Q^- \setminus \{s_0\} \right\}$

senão, faça:

$$\delta_0 \leftarrow d_s^{f_s} - d_s^{f_s - 1}$$

$$\delta_1 \leftarrow \min\left\{x_j - d_j^{f_j} : j \in Q^+ \setminus \{s_0\}\right\}$$

$$\delta_2 \leftarrow \min\left\{d_j^{f_j+1} - x_j : j \in Q^- \setminus \{s_0\}\right\}$$

Obtenha $\delta \leftarrow \min\left\{\delta_0, \delta_1, \delta_2\right\}$.

Selecione um arco $r \in Q$, que minimiza δ, para deixar a base.

Se não existe tal arco então pare: Solução ilimitada.

Se $\hat{c}_s < 0$ então faça $f_s \leftarrow f_s - 1$.

Atualize o vetor de fluxos x (os arcos no ciclo Q), e atualize o vetor de preços duais y.

Se $\hat{c}_s \neq 0$ então faça $s \leftarrow r$ e volte ao passo 2.

Se $r \neq s_0$ então atualize a árvore geradora T para $T \cup \{r\} \setminus \{s_0\}$.

Volte ao passo 1.

Para este algoritmo, pode-se evitar o uso da fase 1 do método das duas fases (Hillier and Lieberman, 1988) para a obtenção de uma solução básica viável inicial. Para isto, basta considerar uma árvore geradora qualquer, aplicar as expressões para o cálculo dos valores das variáveis primal e dual, vistas anteriormente, acrescentar intervalos adicionais para cada arco básico que esteja com fluxo negativo, com coeficiente de custo $c = -\infty$ e limitante de intervalo $d = -\infty$, e aplicar o próprio método imediatamente.

A versão implementada do algoritmo utiliza a exploração continuada de variável básica e não-básica, sendo que, no contexto da PRLP, define-se como operação de pivoteamento, cada atualização do vetor de fluxos x, e define-se como iteração, cada vez que um arco não-básico é escolhido para entrar na base.

O algoritmo utiliza o conceito de bases fortemente viáveis para a PRLP, conforme definido abaixo, que é uma extensão do mesmo conceito introduzido por Cunningham (1976; 1979) e Barr *et al.* (1977) para redes lineares. Pode-se interpretar tal tipo de solução como a possibilidade de enviar uma quantidade positiva de fluxo da raiz a qualquer nó da árvore geradora, sem violação de qualquer limitante de fluxo nos arcos. Neste caso o algoritmo é denominado algoritmo simplex fortemente viável para a PRLP (Marins and Perin, 1988).

Um arco básico j é denominado degenerado se o valor da variável de fluxo x_j correspondente iguala o valor de um dos *breakpoints*, isto é, $x_j = d_j^k$ para algum

intervalo k. Dado um vetor de fluxos básico x, e uma árvore geradora associada T, x é denominado fortemente viável se, para todo arco básico degenerado $j \in T$ tem-se:

- se $x_j = d_j^{f_j+1}$ então o arco j é direcionado para a raiz em T;
- se $x_j = d_j^{f_j}$ então o arco j é direcionado a partir da raiz em T.

Um vetor de fluxos básico fortemente viável inicial x, e um vetor de custos correntes f associado, podem ser obtidos a partir de uma árvore geradora viável qualquer aumentando, ou decrescendo, do valor 1, os atuais valores de intervalo corrente f_j de cada arco básico degenerado j que não satisfaça a propriedade de viabilidade forte formulada acima. Deve-se observar que este artifício altera os vetores duais y e z.

O algoritmo implementado tem a propriedade de evitar os fenômenos indesejáveis de ciclagem (*cycling*) e empacamento (*stalling*), e converge sempre. A ciclagem pode ser definida como sendo uma seqüência cíclica de pivoteamentos degenerados, e o empacamento pode ser definido como sendo uma seqüência finita de pivoteamentos degenerados com comprimento exponencial com relação a dimensão do problema. Um tratamento completo destas situações de ciclagem e empacamento para redes lineares pode ser encontrada em Cunningham (1976; 1979) e Maros (1993).

A ciclagem é evitada mantendo na aplicação do algoritmo, soluções básicas fortemente viáveis, através da escolha adequada da variável que deve ser substituída na base, como se segue. Seja s o arco considerado para entrar na base, e seja w o último nó comum (mais perto da raiz) aos dois caminhos, na árvore geradora T, da raiz aos nós cauda e cabeça do arco s. Considere o ciclo Q formado em $T \cup \{s\}$. Percorra este ciclo Q, partindo do nó w com a mesma orientação de s se $\hat{c}_s < 0$, ou com a orientação contrária de s se $\hat{c}_s > 0$. Escolha como r, o arco que deve deixar a base, o primeiro arco do ciclo Q que satisfaça o teste da razão.

Já o empacamento é evitado utilizando-se a viabilidade forte em conjunto com uma regra adequada de escolha da variável considerada para entrar na base. O algoritmo implementado adota a regra LRC (*Least Recently Considered*), que pode ser descrita da forma a seguir. Seja 1, 2,..., m uma ordenação arbitrária, pré-fixada, dos arcos da rede. Suponha j como o arco que entrou na base na última iteração. Na próxima iteração, selecione como o arco s a ser considerado para entrar na base, o primeiro arco candidato ($\hat{c}_s \neq 0$) da seqüência de arcos j+1, j+2,..., m, 1,..., j.

unesp

Mais detalhes sobre este algoritmo Primal Simplex (além de outros algoritmos para PRLP), incluindo provas de convergência, podem ser encontrados em Marins (1996).

2.4 - A Estrutura da Linguagem NPL2

NPL2 é uma linguagem de computador especialmente implementada para o desenvolvimento de modelos em redes lineares por partes, que emprega uma notação algébrica, usualmente, familiar a especialistas em construção de modelos de Programação Matemática. Um programa NPL2 corresponde à formulação de um modelo em rede com coeficientes de custo estabelecidos por funções lineares por partes, como discutido na seção anterior.

A utilização da linguagem NPL2 para a solução de um problema de fluxo em redes passa pelas seguintes etapas (Figura 2.4):

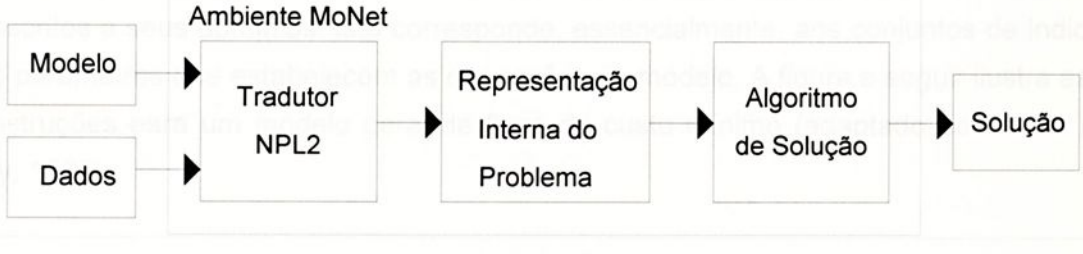


Fig. 2.4 - Etapas na Solução de um Problema NPL2

a) escrever um modelo em rede obedecendo a estrutura da linguagem NPL2;
b) fornecer os dados necessários ao modelo;
c) traduzir o problema em uma representação interna conveniente ao algoritmo de solução;
d) submeter o problema ao algoritmo de solução;
e) visualizar e analisar a solução obtida.

As etapas (c), (d) e a visualização incluída na etapa (e) são realizadas automaticamente (ver descrição do ambiente MoNet, no Apêndice D), cabendo ao usuário da linguagem estabelecer o problema e analisar a solução obtida.

Nesta seção, discute-se, em linhas gerais, como escrever um modelo NPL2. A seção discute também como podem ser representados os dados necessários ao modelo. Para isto, serão utilizados alguns extratos da gramática da linguagem para exemplificar e

explicar conceitos importantes da programação NPL2. A gramática completa da linguagem está contida no Apêndice A.

Deve-se iniciar a descrição de um modelo NPL2 com a palavra-reservada `model` seguida de um identificador que corresponde ao nome do modelo. A seguir devem ser definidos os conjuntos de índice e os parâmetros necessários ao modelo. A definição dos conjuntos de índice deve seguir a palavra-reservada `set`, e a dos parâmetros, a palavra-reservada `par`. Após a declaração de conjuntos e parâmetros, o modelo NPL2 deve especificar os nós (após a palavra-reservada `node`) e os arcos (após a palavra-reservada `arc`) que compõem a rede. Por fim, deve-se especificar o objetivo do modelo. Nas subseções a seguir discute-se cada uma destas construções que compõem um modelo NPL2.

2.4.1 - Conjuntos de Índices e Parâmetros

O primeiro passo na construção de um modelo NPL2 é a declaração de subscritos e seus domínios. Isto corresponde, essencialmente, aos conjuntos de índice e aos parâmetros que estabelecem as dimensões do modelo. A figura a seguir ilustra estas construções para um modelo geral de fluxo de custo mínimo (adaptado de Fourer and Gay, 1993):

```
model transship;
set
   N;
   A := N cross N;
par
   L where (L > 0);
   b[i in N];
   lb[<i,j> in A] where (lb[i,j] >= 0);
   ub[<i,j> in A] where (ub[i,j] >= lb[i,j]);
   bp[<i,j> in A,1..L];
   sl[<i,j> in A,1..L+1];
```

Fig. 2.5 - Declaração de Conjuntos de Índice e Parâmetros

Observe que a presença de sinais de pontuação (como vírgula, ponto-e-vírgula e ponto) são importantes na descrição de um modelo NPL2, além de outros símbolos definidos pela linguagem como, por exemplo, := (símbolo de atribuição) e .. (símbolo de extensão de intervalo).

Deve-se observar que a declaração de conjuntos e parâmetros pode fazer uso de tantas seções `set` ou `par` quantas forem necessárias, segundo a conveniência do programador. Assim, o trecho de programa acima poderia ser escrito também como mostra a Figura 2.6:

```
model transship;
par
   L where (L > 0);
set
   N;
par
   b[i in N];
set
   A := N cross N;
par
   lb[<i,j> in A] where (lb[i,j] >= 0);
   ub[<i,j> in A] where (ub[i,j] >= lb[i,j]);
   bp[<i,j> in A,1..L];
   sl[<i,j> in A,1..L+1];
```

Fig. 2.6 - Declaração Alternativa de Conjuntos de Índice e Parâmetros

Observe na Figura 2.6 que algumas entidades do modelo (como o parâmetro `b`, por exemplo) são definidas em função de outras entidades (o conjunto `N`, no caso). Evidentemente, a ordem de declaração de entidades primárias (como `N`) e entidades derivadas (como `b`) em um modelo NPL2 é importante, devendo uma entidade primária ser declarada antes de qualquer uma de suas entidades derivadas.

Uma declaração de conjunto de índice estabelece, obrigatoriamente, o identificador do conjunto, podendo ainda estabelecer, opcionalmente, uma instância de conjunto e uma condição lógica necessária para a existência do conjunto.

Uma instância de conjunto corresponde a um conjunto específico, que pode ser representado na forma usual, como por exemplo:

{ 1, 2, 3, 4 }
{ a, b, c }

ou na forma de um intervalo, como por exemplo:

[1..5] (que corresponde ao conjunto { 1, 2, 3, 4, 5 })
[1..10 step 3] (que corresponde ao conjunto { 1, 4, 7, 10 })

unesp

Outra forma de especificar conjuntos de índice é por meio de expressões de domínio. Uma expressão de domínio estabelece identificadores para os índices de um conjunto. Isto é importante, por exemplo, quando é necessário estabelecer condições para a existência de elementos do conjunto, como nas declarações a seguir:

```
set
   Cities;
   Links := {<c1 in Cities, c2 in Cities>} where (c1 <> c2);
```

Neste caso, `c1 in Cities` e `c2 in Cities` são expressões de domínio necessárias para estabelecer que só haverá um elemento do conjunto `Links` no caso de seus dois índices serem diferentes. Assim, se `Cities` é o conjunto:

```
{ SP, RJ, BH },
```

o conjunto `Links` irá corresponder a:

```
Links = { <SP,RJ>, <SP,BH>, <RJ,SP>, <RJ,BH>, <BH,SP>, <BH,RJ> }
```

É importante observar as notações possíveis para um elemento de um conjunto de índices em um programa NPL2. Considere o conjunto `Links` acima, por exemplo. Um elemento deste conjunto pode ser escrito como:

```
<i,j> in Links
```

ou como:

```
Links[i,j]
```

Na notação `<i,j> in Links`, o termo `<i,j>` é denominado tupla. De um modo geral, uma tupla identifica um elemento de um conjunto de índices. Por conveniência, para uma tupla constituída por apenas um componente (como `<i>`), os símbolos delimitadores < e > não são necessários. Assim, a expressão:

```
Links := { <c1 in Cities, c2 in Cities> }
```

poderia ser escrita, equivalentemente, como:

```
Links := { <Cities[c1], Cities[c2]> }
```

As tuplas são importantes em uma expressão de domínio e para a terminologia adotada em NPL2. O conjunto `Links` acima, por exemplo, poderia ser definido também por meio das seguintes declarações:

```
set
   Cities;
   Routes := Cities cross Cities;
   Links := {<c1,c2> in Routes} where (c1 <> c2);
```

Dada a equivalência de notações `<c1,c2> in Routes` e `Routes[c1,c2]`, permite-se denominar `Routes` como um conjunto bidimensional.

Uma expressão de domínio pode conter ainda uma especificação de intervalo, escrita na forma: `constante-inteira..expressão-aritmética`, como ocorre, por exemplo, na declaração:

```
ts := {<i,j> in A, 1..L+1};
```

Neste caso, `ts` é um conjunto tridimensional, tendo o terceiro índice identificado por números inteiros 1, 2, ..., e assim sucessivamente até o valor da expressão L+1.

É possível também a declaração de conjuntos de índices como resultado de operações sobre conjuntos declarados previamente. É o que ocorre, por exemplo, com o conjunto `Routes` declarado acima, definido como o produto cartesiano (`cross`) do conjunto `Cities`. Além do produto cartesiano, a linguagem NPL2 admite as seguintes operações sobre conjuntos: união (`union`), interseção (`inter`) e diferença (`diff`).

Outra forma de especificar instâncias de conjuntos de índices e parâmetros é através de tabelas. A seção a seguir discute o relacionamento da linguagem NPL2 com um sistema de banco de dados relacional (Korth and Silberchatz, 1991).

2.4.2 - Conjuntos de Índice, Parâmetros e Relações

A integração de linguagens de modelagem a sistemas de gerenciamento de bancos de dados relacionais já foi reconhecida como uma necessidade por diversos autores (Chobineh, 1991a; Bhargava *et al.*, 1992; Kristjansson *et al.*, 1993; Moody, 1994). Com esta integração, os conjuntos de índice e os parâmetros de um modelo

podem ser vistos como relacionados a tabelas de um banco de dados, alcançando-se com isto independência entre modelo e dados, sem a necessidade de repetição, no ambiente de modelagem, de capacidades já presentes em sistemas de gerenciamento de bancos de dados. Tal integração, para a linguagem NPL2, é discutida a seguir, após uma breve apresentação de conceitos sobre o modelo de dados relacional (extraídos de Kristjansson *et al.*, 1993), importantes para o relacionamento deste modelo de dados com a linguagem NPL2.

No modelo de dados relacional a única construção visível ao usuário é a relação (ou tabela). As colunas de uma tabela são conhecidas como atributos da relação. O número de colunas de uma tabela é conhecido como o grau da relação. Considere, por exemplo, uma relação sobre termelétricas. Esta relação pode conter diversos atributos, como: nome da usina, tipo de combustível utilizado, capacidade de geração da usina, níveis mínimo e máximo tolerado de emissão de poluentes, tipo de controle sobre a usina: se provindo do poder público ou da iniciativa privada, dentre outros. Um exemplo de tal relação pode ser visto na tabela da figura a seguir.

Termelet

Nome da Usina	Tipo de Combustível	Capacidade de Geração	Mínimo de Emissão	Emissão Tolerada	Tipo de Controle
T1	1	270	0.2	0.3	Público
T2	2	240	0.1	0.1	Público
T3	1	350	0.1	0.2	Privado
T4	3	100	0.3	0.5	Privado
T5	2	600	0.2	0.3	Público

Fig. 2.7 - Tabela Termelet: Exemplo de Relação

No modelo de dados relacional, cada relação é definida como um conjunto de linhas de tabela. Cada linha da tabela Termelet, por exemplo, corresponde a uma usina termelétrica com seus atributos associados. Cada atributo da tabela possui um domínio, a partir do qual provêm os valores do atributo. Para a tabela acima, considere, por exemplo, que o domínio do atributo Tipo_de_Combustível é o conjunto { 1, 2, 3, 4 }. Portanto, o domínio corresponde ao conjunto de valores válidos para um atributo. Em uma relação, cada ocorrência de um atributo deve conter um único valor de seu domínio.

Em uma tabela relacional existe um atributo (ou combinação de atributos) que pode ser usado para identificar univocamente as linhas da tabela. Estes atributos são denominados chaves da relação. No caso de uma chave ser constituída por mais de um atributo, cada um destes atributos é denominado componente de chave. Estes atributos refletem as dependências funcionais da relação, ou seja, como o valor de cada atributo

depende do valor dos atributos-chave. Para a tabela da Figura 2.7, por exemplo, o atributo Nome_da_Usina pode ser considerado como atributo-chave. Observe que, dado o valor do atributo Nome_da_Usina, os valores de todos os demais atributos ficam determinados. Como, em geral, existe mais de uma possibilidade de escolha para atributos-chave de uma relação, é comum designar uma das chaves possíveis como chave principal.

Além destas estruturas básicas, o modelo de dados relacional possui três regras para garantir a integridade dos dados armazenados nas tabelas:

- integridade de entidade, que requer a existência de uma forma única de identificar cada linha da tabela, o que é obtido definindo-se uma chave principal onde nenhum componente pode ser vazio;
- integridade de domínio, que estipula aos atributos somente valores legítimos, tomados de seus domínios;
- integridade referencial, que garante que um valor que aparece em uma relação para um dado conjunto de atributos também aparece para um certo conjunto de atributos em outra relação.

Um atributo de uma relação que é a chave principal de outra relação é conhecido como chave estrangeira. Tais chaves representam referências que permitem a combinação de tabelas por meio da operação de junção, como se discute adiante.

As operações usuais sobre conjuntos (união, interseção, diferença e produto cartesiano) se aplicam também ao modelo relacional, uma vez que uma relação é definida como um conjunto de linhas de tabela. Além destas operações, a álgebra relacional inclui as operações de seleção, projeção e junção. A operação de seleção se aplica a uma única tabela e seleciona um subconjunto de linhas da relação. Por exemplo, da tabela da Figura 2.7 pode-se selecionar todas as termelétricas que utilizam combustível de tipo 2. Isto irá resultar na seguinte tabela:

Nome da Usina	Tipo de Combustível	Capacidade de Geração	Mínimo de Emissão	Emissão Tolerada	Tipo de Controle
T2	2	240	0.1	0.1	Público
T5	2	600	0.2	0.3	Público

Fig. 2.8 - Tabela Resultante de Operação de Seleção

Evidentemente, uma tabela resultante de uma operação de seleção possui os mesmos atributos da relação sobre a qual a operação foi executada. Pode-se também observar

que seleções são comutativas, de forma que para uma sequência de seleções a ordem de execução é irrelevante.

O operador de projeção também se aplica a uma única relação e realiza uma seleção vertical na tabela, escolhendo alguns atributos e descartando os demais. Por exemplo, pode-se aplicar o operador de projeção na tabela da Figura 2.7 para construir uma nova tabela apenas com o tipo de combustível utilizado e a capacidade de geração de cada usina, conforme mostra a Figura 2.9 a seguir.

Tccg

Nome da Usina	Tipo de Combustível	Capacidade de Geração
T1	1	270
T2	2	240
T3	1	350
T4	3	100
T5	2	600

Fig. 2.9 - Tabela Resultante de Operação de Projeção

O grau de uma tabela obtida com a operação de projeção é igual ao número de atributos especificados na lista de projeção. No caso acima, por exemplo, a lista de projeção é dada por:

```
< Nome_da_Usina, Tipo_de_Combustivel, Capacidade_de_Geracao >
```

e a tabela resultante é de grau 3. Se a lista de projeção não contém todos os atributos-chave da relação, é possível que ocorra a duplicação de linhas, o que precisa ser eliminada para garantir que a tabela projetada seja também uma relação (ou seja, um conjunto de linhas). Isto pode ser visto facilmente considerando-se, por exemplo, a aplicação da lista de projeção `< Tipo_de_Controle >` à tabela da Figura 2.7. Projeções são, evidentemente, não comutativas.

O operador de junção é usado para combinar duas tabelas relacionadas (por meio de atributos) em uma única tabela. Considere, por exemplo, a tabela da Figura 2.10.

Caso seja necessário saber quais são os tipos de combustível transportados pelas companhias distribuidoras e a capacidade de geração das usinas em cada uma das rotas, pode-se aplicar o operador de junção às tabelas Rotas (Figura 2.10) e Tccg (Figura 2.9).

unesp

Rotas

Número da Rota	Distribuidor	Nome da Usina	Custo da Rota
1	1	T1	100
2	1	T2	150
3	1	T5	120
4	2	T1	250
5	2	T4	180
6	3	T1	130
7	3	T2	200
8	3	T3	210
9	3	T4	140
10	3	T5	180

Fig. 2.10 - Tabela Rotas

Neste caso, a condição de junção será: Nome da usina na tabela Rotas = Nome da usina na tabela Tccg. Isto irá resultar na seguinte tabela:

Número da Rota	Distribuidor	Nome da Usina	Custo da Rota	Nome da Usina	Tipo de Combustível	Capacidade de Geração
1	1	T1	100	T1	1	270
2	1	T2	150	T2	2	240
3	1	T5	120	T5	2	600
4	2	T1	250	T1	1	270
5	2	T4	180	T4	3	100
6	3	T1	130	T1	1	270
7	3	T2	200	T2	2	240
8	3	T3	210	T3	1	350
9	3	T4	140	T4	3	100
10	3	T5	180	T5	2	600

Fig. 2.11 - Tabela Resultante de Operação de Junção

Pode-se perceber que esta operação de junção é equivalente a executar um produto cartesiano das linhas da tabela Rotas com as linhas da tabela Tccg, mas somente quando a combinação de linhas satisfaz a condição de junção. Em casos, como o acima, em que a condição de junção envolve uma comparação de igualdade, a operação é conhecida como equijunção. Observe que em uma junção sempre haverá repetição de atributos na tabela resultante (é o caso do atributo Nome_da_Usina, na tabela da Figura 2.11). Uma operação de junção que remove da tabela resultante os atributos supérfluos é conhecida como junção natural.

Estas operações da álgebra relacional são importantes para a especificação de instâncias de conjuntos de índices e parâmetros na linguagem NPL2. Para isto dispõe-se da construção `table`. Antes de discutir tal construção é importante observar que em tabelas relacionais, as instâncias de conjuntos de índices correspondem aos atributos-

chave da relação, enquanto os demais atributos podem ser considerados como instâncias de parâmetros associados àqueles conjuntos de índices. Assim, na tabela da Figura 2.7, por exemplo, o atributo Nome_da_Usina pode ser considerado como um conjunto de índices, enquanto os atributos Tipo_de_Combustível e Capacidade_de_Geração (dentre outros) podem ser considerados como parâmetros associados ao atributo Nome_da_Usina. Este relacionamento é descrito em NPL2 através das seguintes construções:

```
set
   Usina := table(Termelet, key Nome_da_Usina);
par
   tc[i in Usina] := table(Termelet, data Tipo_de_Combustivel);
   cg[i in Usina] := table(Termelet, data Capacidade_de_Geracao);
```

Neste caso, considerando a tabela da Figura 2.7, a declaração `set` define o conjunto:

```
Usina = { T1, T2, T3, T4, T5 }
```

e a declaração `par` define os parâmetros:

i	tc[i]
T1	1
T2	2
T3	1
T4	3
T5	2

i	cg[i]
T1	270
T2	240
T3	350
T4	100
T5	600

Deve-se observar, portanto, que tais declarações correspondem a operações de projeção sobre a relação especificada. Como a declaração de parâmetros pode incluir uma cláusula `where`, é possível estabelecer, na expressão lógica, uma condição de seleção. Seja:

```
par
   cg2[i in Usina] := table(Termelet, data Capacidade_de_Geracao)
                                   where (tc[i] = 2);
```

Neste caso, o parâmetro `cg2` será o resultado de uma operação de projeção na tabela Termelet, tendo `< Capacidade_de_Geracao >` como lista de projeção. Mas da tabela resultante serão mantidas somente as linhas da tabela onde o valor do parâmetro `tc` for

igual a 2, o que equivale a uma operação de seleção. Ou seja, esta declaração irá resultar no seguinte parâmetro:

i	cg2[i]
T2	240
T5	600

Outra forma aceita pela linguagem NPL2 para especificar conjuntos de índices e parâmetros é dada pela construção `forsome`, que especifica uma junção de tabelas. Sejam, por exemplo, as seguintes declarações:

```
set
   Tipo := table(Termelet, key Tipo_de_Controle);
   Rota := table(Rotas, key Numero_da_Rota);
   Tipo_Rota :=
   forsome(<t in Tipo, r in Rota>,
      Termelet key Nome_da_Usina = Rotas key Nome_da_Usina);
```

Neste caso, o conjunto `Tipo_Rota[t,r]` provém da junção das tabelas Termelet e Rotas, sendo a condição de junção dada pela igualdade dos índices referentes ao nome da usina nestas duas tabelas. Em termos das relações mostradas nas figuras 2.7 e 2.10, tem-se:

Tipo_Rota	
Publico	1
Publico	4
Publico	6
Publico	2
Publico	7
Privado	8
Privado	5
Privado	9
Publico	3
Publico	10

Este novo conjunto de índices poderá ser usado em NPL2, por exemplo, em expressões como:

```
sum(r in Tipo_Rota[t,r],
      Qtd_Transportada[r] >= Min_Necessario[t,c]
      where (c in Combustivel_Rota[c,r]));
```

onde `Qtd_Transportada` e `Min_Necessario` podem ser parâmetros e `Combustivel_Rota`, um conjunto de índices, definidos previamente.

A construção `forsome` pode ser usada também para a declaração de parâmetros, bastando para isto, incluir uma cláusula `data`. Sejam, por exemplo, a tabela:

Combust

Tipo de Combustível	Mínimo Transportado
1	50000
2	20000
3	10000
4	70000

Fig. 2.12 - Tabela Combust

e as declarações:

```
set
   Tipo := table(Termelet, key Tipo_de_Controle);
   Comb := table(Combust, key Tipo_de_Combustivel);
par
   Min_Necessario := forsome(<t in Tipo, c in Comb>,
      Termelet key Tipo_de_Combustivel =
            Combust key Tipo_de_Combustivel,
      data Minimo_Transportado);
```

Neste caso, define-se o parâmetro bidimensional `Min_Necessario[t,c]` como o atributo `Minimo_Transportado` da relação resultante da junção das tabelas Termelet e Combust, por meio do atributo comum Tipo_de_Combustivel. Observe que, considerando as tabelas das figuras 2.7 e 2.12, este parâmetro irá correponder à seguinte tabela:

t	c	Min_Necessario[t,c]
Publico	1	50000
Publico	2	20000
Privado	1	50000
Privado	3	10000

2.4.3 - Declaração de Estrutura de Rede Linear Por Partes

O benefício de se ter estruturas especiais em um modelo linear nem sempre é fácil de ser aproveitado. Para isto, o tradutor da linguagem de modelagem deve ser capaz de reconhecer a presença das estruturas no modelo e ser capaz de comunicar tais estruturas ao algoritmo de solução. O caso de modelos que incorporam termos lineares por partes, por exemplo, é típico. Embora sejam muitas as suas aplicações (Marins e Perin, 1996), nem sempre os modelos lineares por partes podem ser tratados diretamente em sistemas de modelagem. Em alguns sistemas, um modelo linear por partes para ser resolvido, precisa ser convertido, através de várias transformações, em um modelo linear equivalente. Tal conversão nem sempre é bem-vinda pois, além de aumentar o tamanho do modelo, com a introdução de novas variáveis ou restrições, a concepção original do modelo como uma formulação linear por partes fica desfeita, podendo levar a uma solução pouco natural.

Quando se dispõe de um algoritmo capaz de resolver modelos de fluxo em redes lineares por partes diretamente, isto é, sem a necessidade de qualquer transformação, parece interessante a possibilidade de expressar tais estruturas explicitamente no modelo. A linguagem NPL2 foi desenvolvida tendo em mente a inclusão de estruturas para expressar funções lineares por partes e para especificar uma rede através da declaração direta de seus componentes: nós e arcos.

Como discutiu-se na Seção 2.2, uma função contínua de uma variável é linear por partes se seu domínio puder ser particionado em intervalos nos quais a função é linear. Na linguagem NPL2 as funções lineares por partes, embora possam ser arbitrárias, devem ser definidas por um número finito de tais intervalos. Existem diversas formas de expressar uma função linear por partes (ver, por exemplo, Fourer and Gay, 1993, p. 5). A linguagem NPL2 adota a representação dada por um par, onde o primeiro elemento é uma lista de pontos de quebra (*breakpoints*), que identificam os limites dos intervalos, e o segundo elemento é uma lista de inclinações. Além deste par de listas, a linguagem fixa um ponto arbitrário, de modo a ter as funções lineares por partes estabelecidas univocamente.

Como discutido anteriormente, os termos lineares por partes de um modelo NPL2 correspondem aos coeficientes das variáveis (arcos) na função-objetivo. Assim tais termos serão expressos através da cláusula:

`obj` < lista-de-expressões : lista-de-expressões >

unesp

onde a primeira lista-de-expressões deve corresponder à lista de pontos de quebra e a segunda, à lista de inclinações da função que define o termo linear por partes. Além disto, a linguagem NPL2 adota as seguintes convenções:

- a primeira inclinação corresponde ao trecho linear da função entre $-\infty$ e o primeiro ponto de quebra;
- a última inclinação refere-se ao trecho linear da função entre o último ponto de quebra e $+\infty$;
- o valor da função no ponto zero é igual a zero.

Assim, funcões como as mostradas na Figura 2.13:

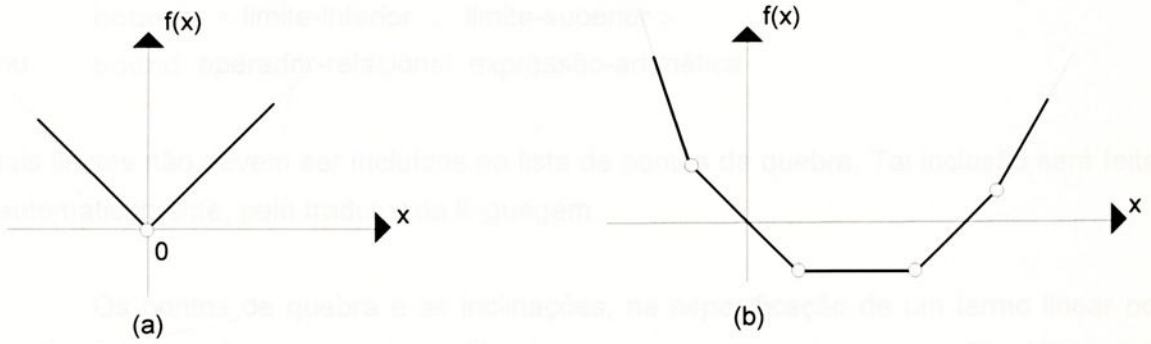


Fig. 2.13 - Funções Lineares por Partes

são representadas em NPL2 por:

(a)
< 0 : -1, 1 >

(b)
< -1, 1, 3, 5 : -5, -1, 0, 1, 3 >

Observe que, com as convenções adotadas, a lista de inclinações terá sempre um elemento a mais do que a lista de pontos de quebra. Isto deve ser mantido, mesmo se forem estabelecidos limites para a variável e tais limites forem considerados como pontos de quebra: o limite inferior como primeiro ponto de quebra e o limite superior como último ponto de quebra. Assim, se apenas o limite inferior for especificado, a lista de pontos de quebra não deverá incluir este ponto, como ilustra a figura a seguir, onde considera-se zero como limite inferior e, portanto, com a inclinação 0.5 referindo-se ao intervalo [0, 1].

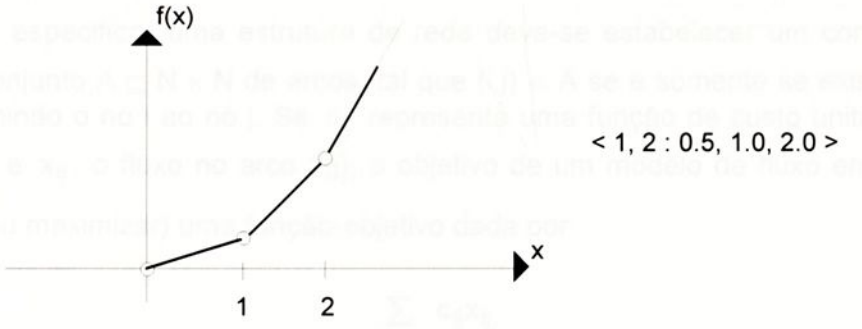


Fig. 2.14 - Limite Inferior como Ponto de Quebra

No caso geral, definidos os limites para uma variável, o que pode ser feito em NPL2 por meio das cláusulas:

`bounds` < limite-inferior : limite-superior >
ou `bound` operador-relacional expressão-aritmética

tais limites não devem ser incluídos na lista de pontos de quebra. Tal inclusão será feita, automaticamente, pelo tradutor da linguagem.

Os pontos de quebra e as inclinações, na especificação de um termo linear por partes, não precisam ser valores literais, como nos exemplos acima. Em NPL2, tais valores podem ser expressões aritméticas quaisquer, inclusive com o emprego de variáveis. Isto é conveniente, por exemplo, para especificar termos representando custos convexos crescentes, como os exigidos pelo algoritmo de solução utilizado. Assim, é possível, por exemplo, escrever um termo linear por partes da forma:

```
< bp[i,j] : sl[i,j] >
```

dadas as seguintes declarações:

```
par
   nbrk[<i,j> in A];
   bp[<i,j> in A, 1..nbrk[i,j]];
   sl[<i,j> in A, 1..nbrk[i,j]+1];
```

onde `nbrk[i,j]` representa o número de pontos de quebra de um arco `<i,j>` do conjunto A. Com isto, pode-se definir, no modelo, funções lineares por partes que dependem dos dados do problema.

Para especificar uma estrutura de rede deve-se estabelecer um conjunto N de nós e um conjunto $A \subseteq N \times N$ de arcos, tal que $(i,j) \in A$ se e somente se existe um arco orientado unindo o nó i ao nó j. Se c_{ij} representa uma função de custo unitário relativa ao arco (i,j) e x_{ij}, o fluxo no arco (i,j), o objetivo de um modelo de fluxo em rede será minimizar (ou maximizar) uma função-objetivo dada por:

$$\sum_{(i,j) \in A} c_{ij} x_{ij}$$

sujeito às limitações de fluxo nos arcos, ou seja:

$$lb_{ij} \leq x_{ij} \leq ub_{ij} \qquad (i,j) \in A$$

e ao balanço de fluxo nos nós, ou seja:

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = b_i \qquad i \in N$$

onde lb_{ij} representa o limite inferior de fluxo no arco (i,j), ub_{ij} corresponde ao limite superior de fluxo no arco (i,j), $\sum_{(i,j) \in A} x_{ij}$ representa o total de fluxo que sai do nó i, $\sum_{(j,i) \in A} x_{ji}$ indica o total de fluxo que entra em i, e b_i representa:

- a quantidade de fluxo produzido no nó i, se $b_i > 0$, ou
- o negativo da quantidade de fluxo consumido no nó i, se $b_i < 0$.

No caso de $b_i = 0$, então i é um nó de transbordo, onde o fluxo que entra iguala o fluxo que sai.

Para especificar tal estrutura a linguagem NPL2 dispõe das construções `node` e `arc`. A construção `node` é usada para declarar os nós da rede e caracterizar as restrições de balanço de fluxo que devem ocorrer nestes nós. A construção `arc` declara os arcos da rede, especificando os nós que o arco conecta e estabelecendo informações opcionais sobre limites de fluxo e coeficientes de custo. A Figura 2.15, a seguir, ilustra estas construções em um modelo de fluxo de custo mínimo já referenciado parcialmente na Seção 2.4.1.

```
model transship;
set
  N;
  A := N cross N;

par
  L where (L > 0);
  b[i in N];
  lb[<i,j> in A] where (lb[i,j] >= 0);
  ub[<i,j> in A] where (ub[i,j] >= lb[i,j]);
  bp[<i,j> in A, 1..L];
  sl[<i,j> in A, 1..L+1];

node
  Bal[i in N] where (net_out = b[i]);

arc
  Flow[<i,j> in A]
    from Bal[i] to Bal[j]
    bounds < lb[i,j] : ub[i,j] >
    obj < bp[i,j] : sl[i,j] >;

minimize Total_Cost.
```

Fig. 2.15 - Modelo de Fluxo de Custo Mínimo

Neste modelo os nós são denominados `Bal[i]`, onde i representa um valor do conjunto N. A construção `node`, neste caso, especifica ainda a condição de balanço de fluxo (`net_out = b[i]`), ou seja:

(total de fluxo que sai de i) - (total de fluxo que entra em i) = b[i]

Para especificar condições de balanço de fluxo, a linguagem NPL2 dispõe ainda da palavra reservada `net_in`, que resume a expressão:

(total de fluxo que entra) - (total de fluxo que sai)

do nó especificado. No caso geral, uma condição de balanço relaciona (através de um dos operadores relacionais disponíveis: =, <>, >, <, >= e <=) as palavras `net_in` ou `net_out` a uma expressão aritmética, podendo haver várias condições combinadas logicamente (através dos operadores lógicos: `not`, `and` e `or`). Por exemplo, a expressão:

```
(net_out >= a) and (net_out <= b)
```

estabelece que a diferença entre o fluxo que sai e o fluxo que entra em um nó deve estar no intervalo [a,b], onde a e b são expressões aritméticas quaisquer.

A Figura 2.15 apresenta também a especificação dos arcos da rede:

```
arc
  Flow[<i,j> in A]
    from Bal[i] to Bal[j]
    bounds < lb[i,j] : ub[i,j] >
    obj < bp[i,j] : sl[i,j] >;
```

Neste caso, `Flow[i,j]` é o nome do arco que liga o nó `Bal[i]` ao nó `Bal[j]`, para cada par (i,j) $\in$ A. É interessante observar, novamente, a distinção entre o conjunto de índices e o conjunto de componentes de rede. Com esta distinção pode-se representar, em modelos mais complexos, os arcos (e também os nós) de uma forma bem estruturada, não sendo necessário, por exemplo, que os índices de arcos correspondam a pares de elementos do conjunto de nós. É o caso do modelo Train, apresentado na Seção 2.4, onde se tem arcos declarados como:

```
X[<c1,t1,c2,t2> in Sched]
```

representando vagões de um trem que parte da cidade c1 no horário t1 e chega à cidade c2 no horário t2.

A declaração de arcos deve incluir, obrigatoriamente, pelo menos uma das cláusulas `from` ou `to` (é possível a existência de arcos para os quais tem-se especificado apenas o destino ou apenas a origem, como discutido inicialmente para o modelo de fluxo máximo). A declaração de arcos pode incluir também, opcionalmente, uma especificação de limites sobre o fluxo (através das cláusulas `bounds` ou `bound`, já comentadas) e uma especificação de coeficiente deste arco na função-objetivo (através da cláusula `obj`), o qual pode ser uma expressão aritmética simples:

`obj` expressão-aritmética

para o caso de modelos lineares, ou um par composto por uma lista de pontos de quebra e uma lista de inclinações:

`obj` < lista de pontos de quebra : lista de inclinações >

para modelos que incluem termos lineares por partes, como discutido anteriormente.

O tradutor NPL2 adota as seguintes convenções para o caso destas cláusulas serem omitidas:

- a ausência de uma cláusula sobre limites de fluxo é interpretada como: `bound >= 0`, ou seja, a quantidade de fluxo no arco pode ser um valor qualquer maior ou igual a zero;
- a ausência da cláusula de coeficiente do arco na função-objetivo é interpretada como: `obj 1`, ou seja, os arcos contribuem linear e unitariamente para o objetivo do modelo.

Um modelo NPL2 deve terminar com uma declaração `minimize` ou `maximize`, que estabelece um nome para o objetivo do modelo. Deve-se observar que o tradutor da linguagem irá considerar como função-objetivo a expressão:

$$\sum_i c_i x_i$$

para todo arco x_i declarado no modelo, onde c_i corresponde ao coeficiente estabelecido na cláusula `obj` para este arco. Portanto, considerando as convenções adotadas, é imprescindível a inclusão de uma cláusula `obj 0` para os arcos que não devem ser incluídos na função-objetivo do modelo.

Nesta seção foram discutidas, em essência, as construções da linguagem NPL2. Muitas variações destas construções podem ser escritas para formulação de situações específicas. Para ilustrar, apresenta-se na seção a seguir, dois modelos (adaptados de exemplos de Fourer and Gay, 1993) que mostram construções da linguagem NPL2 usadas para descrever situações bem estruturadas.

2.5 - Exemplos de Modelos NPL2

Apresenta-se a seguir dois exemplos de programas NPL2, escritos para modelar situações práticas.

O primeiro modelo (Dist), determina o planejamento da produção e da distribuição de um conjunto de produtos através de uma rede de produtores, distribuidores e consumidores. O segundo modelo (Train) é um modelo de fluxo em redes que, dados os horários diários de partida e de chegada de trens entre diversas cidades, aloca vagões de passageiros aos trens, de forma a atender a demanda e minimizar o número total de vagões necessários.

unesp

2.5.1 - Modelo Generalizado de Distribuição

```
model Dist;
/*
   Este modelo determina o planejamento da producao e da distribuicao
   de um dado produto, de modo a atender um conjunto de demandas ao
   menor custo possivel.
*/

! CONJUNTOS DE INDICES E PARAMETROS SOBRE A DISTRIBUICAO

set
  whse;                          ! Locais dos armazens.
  dctr where (dctr in whse);     ! Locais dos centros de distribuicao.
  dw := dctr cross whse;
par
  sc[<d,w> in dw]                ! Custo de transporte de um centro de
    where (sc[d,w] >= 0);        !   distribuicao (d) a um armazem (w).
  huge where (huge > 0);         ! Custo maximo de transporte.
  msr[<d,w> in dw];              ! Restricoes de tamanho minimo de lote
                                 !   a transportar na rota d -> w.
  dsr[d in dctr]                 ! Demanda total minima necessaria para
    where (dsr[d] >= 0);         !   um transporte em uma rota.

! CONJUNTOS DE INDICES E PARAMETROS SOBRE A PRODUCAO

set
  fact where (fact in dctr);     ! Locais das fabricas.
par
  rtmin where (rtmin >= 0);      ! Limite inferior do numero total de
                                 !   empregados regulares nas fabricas.
  rtmax where (rtmax >= rtmin);  ! Idem acima, mas para limite superior.
  otmin where (otmin >= 0);      ! Limite inferior do total de horas
                                 !   extras nas fabricas.
  otmax where (otmax >= otmin);  ! Idem acima, mas para limite superior.
  rmin[f in fact]                ! Limite inferior do numero de equipes
    where (rmin[f] >= 0);        !   regulares de trabalho na fabrica f.
  rmax[f in fact]                ! Idem acima, mas para limite superior.
    where (rmax[f] >= rmin[f]);
  omin[f in fact]                ! Limite inferior de horas-extras na
    where (omin[f] >= 0);        !   fabrica f.
  omax[f in fact]                ! Idem acima, mas para limite superior.
    where (omax[f] >= omin[f]);
  hd[f in fact]                  ! Total de horas regulares de trabalho
    where (hd[f] >= 0);          !   por dia, na fabrica f.
  dp[f in fact]                  ! Numero de dias de trabalho no periodo
    where (dp[f] > 0);           !   de planejamento.

! CONJUNTOS DE INDICES E PARAMETROS SOBRE PRODUTOS

set
  prd;                           ! Grupo de produtos.
par
  wt[p in prd]                   ! Peso de 1000 unidades do produto p.
    where (wt[p] > 0);
  cpp[p in prd]                  ! Caixas (1000 unidades) de produto p
    where (cpp[p] > 0);          !   por viagem.
  tc[p in prd]                   ! Custo de transporte por 1000 unidades
    where (tc[p] >= 0)           !   de produto p.;
  pt[p in prd,f in fact]         ! Tempo de producao de 1000 unidades
    where (pt[p,f] >= 0);        !   do produto p na fabrica f.
  rpc[p in prd,f in fact]        ! Custo de producao de 1000 unidades
    where (rpc[p,f] >= 0);       !   em horario regular de trabalho.
  opc[p in prd,f in fact]        ! Idem acima, mas para horas-extras.
    where (opc[p,f] >= 0);

! CONJUNTOS DE INDICES E PARAMETROS SOBRE DEMANDAS

par
  dt[p in prd]                   ! Demanda total pelo produto p.
    where (dt[p] >= 0);
```




```
  ds[p in prd,w in whse]           ! Fracao historica da demanda total do
    where (ds[p,w] >= 0.0) and     !   produto p atendida pelo armazem w.
          (ds[p,w] <= 1.0);

  dstot[p in prd] :=               ! Fracao total da demanda do produto p
    sum(w in whse,ds[p,w]);        !   atendida pelos armazens.
  dem[p in prd,w in whse] :=       ! Demandas projetadas a serem satisfeitas.
    dt[p]*ds[p,w]/dstot[p];        !
set
  srt                              ! Lista de pares ordenados que
    where (d <> w and              !   representam rotas permitidas.
           sc[d,w] < huge and
           ((w in dctr) or sum(p in prd,dem[p,w]) > 0) and
           not (msr[d,w] and
                sum(p in prd,1000*dem[p,w]/cpp[p]) < dsr[d]));

! NOS DA REDE

node
  RT where (net_out >= rtmin and   ! Origem de todas as equipes regulares
            net_out <= rtmax);     !   de trabalho alocadas.
  OT where (net_out >= otmin and   ! Origem de todas as equipes extras
            net_out <= otmax);     !   de trabalho alocadas.
  P_RT[f in fact];                 ! Origem das equipes regulares de
                                   !   trabalho da fabrica f.
  P_OT[f in fact];                 ! Idem acima, mas para equipes extras.
  MF[p in prd,f in fact];          ! Fontes de manufatura:
                                   !   envia a WF para demanda local;
                                   !   envia a DF para distribuicao.
  DF[p in prd,d in dctr];          ! Centros de distribuicao:
                                   !   recebe produtos vindos de centros WF;
                                   !   recebe produtos fabricados em MF;
                                   !   envia para nos WF.
  WF[p in prd,w in whse]           ! Localizacao dos armazens:
    where (net_in = dem[p,w]);     !   recebe de DF e MF para satisfazer
                                   !     a demanda local;
                                   !   envia a DF para distribuicao.

! ARCOS DA REDE

arc
  Work_RT[f in fact]               ! Horas de trabalho regulares alocadas
    from RT to P_RT[f]             !   a cada fabrica.
    bounds <rmin[f]:rmax[f]>
    obj 0;

  Work_OT[f in fact]               ! Horas-extras alocadas a cada fabrica.
    from OT to P_OT[f]
    where (Work_OT[f] >= omin[f] and
           Work_OT[f] <= omax[f])
    obj 0;

  Manu_RT[p in prd,f in fact]      ! Horas regulares alocadas para
    from P_RT[f] to M[p,f]         !   manufaturar cada produto em cada
    bound >= 0                     !   fabrica.
    obj (rpc[p,f]*dp[f]*hd[f]/pt[p,f])
    where (rpc[p,f] <> 0);

  Manu_OT[p in prd,f in fact]      ! Horas-extras alocadas para
    from P_OT[f] to M[p,f]         !   manufaturar cada produto em cada
    obj (opc[p,f]/pt[p,f])         !   fabrica.
    where (Manu_OT[p,f] >= 0 and opc[p,f] <> 0);

  Prod_L[p in prd,f in fact]       ! Total de producao de cada produto
    from M[p,f] to WLoc[p,f]       !   em cada fabrica para atender a
    bound >= 0                     !   demanda local (1000 unidades).
    obj 0;

  Prod_D[p in prd,f in fact]       ! Idem acima, mas para distribuicao.
    from M[p,f] to DSource[p,f]
    bound >= 0
    obj 0;
```

```
  Ship[p in prd,
       d in dctr,w in whse]          ! Total transportado de cada produto
    from DSource[p,d] to WLoc[p,w]   !   em cada uma das rotas permitidas.
    obj (sc[d,w]*wt[p])
    bound >= 0;

  Trans[p in prd,d in dctr]          ! Total distribuido de cada produto
    from WLoc[p,d] to DSource[p,d]   !   em cada centro de distribuicao.
    obj (tc[p])
    bound >= 0;

! OBJETIVO DO MODELO

minimize cost.                       ! Custo total: custos de producao
                                     !   mais custos de transporte.
```

2.5.2 - Modelo de Alocação

```
model Train;
/*
    Este modelo de fluxo em rede aloca vagoes de passageiros a trens
    de forma a minimizar o numero total de vagoes necessarios.
*/

! CONJUNTOS DE INDICES E PARAMETROS SOBRE HORARIOS

set
  cities;                                ! Locais de cidades
  links[c1 in cities,c2 in cities]       ! Ligacao entre cidades.
    where (c1 <> c2);
par
  integer last where (last > 0);

set
  times := [1..last];                    ! Intervalos de tempo por dia.

  sched[c1 in cities,t1 in times,c2 in cities,t2 in times]
    where <c1,c2> in links;              ! Indice de um trem que parte da
                                         !   cidade c1 no instante t1 e chega
                                         !   na cidade c2 no instante t2.

! PARAMETROS DE DEMANDA

par
  integer section                        ! Numero maximo de vagoes
    where section > 0;                   !   por secao de um trem.

  demand[<c1,t1,c2,t2> in sched]         ! Para cada trem:
    where demand[c1,t1,c2,t2] > 0;       !   o menor numero de vagoes para
                                         !   satisfazer a demanda deste trem.

  low[<c1,t1,c2,t2> in sched] :=         ! Numero minimo de vagoes para.
    ceil(demand[c1,t1,c2,t2]);           !   satisfazer a demanda.

  high[<c1,t1,c2,t2> in sched] :=        ! Numero maximo de vagoes no trem.
    max(2,
        min(ceil(2*demand[c1,t1,c2,t2]),
            section*ceil(demand[c1,t1,c2,t2]/section)));

/*  Observe que o numero maximo de vagoes permitidos em um trem e' 2,
    se a demanda exige menos de 1 vagao. Do contrario, o minimo esta
    entre o numero de vagoes necessarios para acomodar duas vezes a
    demanda e o numero de vagoes necessarios para se ter o minimo de
    secoes do trem.
*/
```


unesp

```
! PARAMETROS DE DISTANCIA

par
  dist_table[<c1,c2> in links]
    where dist_table[c1,c2] >= 0;
  distance[<c1,c2> in links] :=          ! Distancias entre cidades.
    if (dist_table[c1,c2] > 0) then
      dist_table[c1,c2]
    else
      dist_table[c2,c1];

! ESPECIFICACAO DOS NOS

node
  N[c1 in cities,t1 in times];

  ! Para cada cidade e horario:
  !   o numero de vagoes nao utilizados no intervalo de tempo atual sera
  !   igual ao numero de vagoes nao utilizados no intervalo anterior,
  !   mais o numero de vagoes em trens que estao chegando,
  !   menos o numero de vagoes em trens que estão partindo.

! ESPECIFICACAO DE ARCOS

arc
  U[c in cities,t in times]              ! Numero de vagoes nao usados
    from N[c,t] to N[c,next(t)]          !   estocados na cidade c, no
    obj if (t = last) then 1             !   inicio do intervalo de tempo
    bound >= 0;                          !   que comeca no instante t.

  X[<c1,t1,c2,t2> in sched]              ! Numero de vagoes de um trem que
    from N[c1,t1] to N[c2,t2]            !   parte de c1 no instante t1 e
    obj if (t2 < t1) then 1              !   chega em c2 no instante t2.
    bounds <low[c1,t1,c2,t2]:high[c1,t1,c2,t2]>;

! OBJETIVO

! Numero de vagoes no sistema:
!   soma dos vagoes nao usados e dos vagoes existentes em trens durante
!   o ultimo intervalo de tempo do dia.

minimize cars.
```

Estes exemplos mostram bem o benefício de dispor de uma linguagem de modelagem que impõe a formulação de um problema de otimização como um modelo de fluxo em redes (formulações alternativas para estes dois problemas podem ser vistos em Fourer *et al.*, 1993) e garanta que toda instância gerada pelo tradutor do modelo possua a estrutura correta de coeficientes de rede, de modo a ser possível submeter o problema a um algoritmo eficiente de solução.

unesp

Capítulo 3

Uma Proposta de Interface Inteligente de Modelagem

3.1 - Introdução

A formulação de um modelo de Programação Linear compreende a especificação de uma função-objetivo linear que deve ser minimizada ou maximizada e de um conjunto de restrições lineares da forma "≤", "=" ou "≥". Por exemplo:

$$\text{Max} \sum_{j=1}^{n} c_j x_j$$

$$\text{sujeito a: } \begin{cases} \sum_{j=1}^{n} a_{ij} x_j \leq b_i & (i = 1,\ldots,m) \\ x_j \geq 0 & (j = 1,\ldots,n) \end{cases}$$

onde x_j (j = 1, ..., n) são as *variáveis de decisão*, c_j (j = 1, ..., n) são os *coeficientes de custo* destas variáveis na função-objetivo, a_{ij} (j = 1, ..., n) são os coeficientes destas variáveis na i-ésima restrição (i = 1, ..., m) e b_i (i = 1, ..., m) são os *valores do lado direito* das restrições.

A formulação de problemas reais como modelos de PL tem resultado em classes de modelos típicos, como por exemplo, modelos de produção, modelos de composição de produto, modelos de distribuição, modelos de planejamento, dentre muitos outros. A modelagem de um problema real, geralmente, envolve uma combinação de tais modelos típicos. Assim, um aspecto importante da modelagem de PL é a conceituação do problema, ou seja, a verificação se o problema pertence a uma determinada categoria. Por exemplo, verificar se o problema pode ser visto como um problema de produção independentemente do tipo de produto, processo de fabricação, e outros fatores. Evidentemente, a habilidade de conceituação requer muito conhecimento e vem com a experiência em reconhecer situações típicas.

unesp

Segundo Lucas (1986), o desenvolvimento de um modelo de PL como um conjunto de condições algébricas requer os seguintes passos:

a) definir, a partir da categoria do problema, os subscritos das variáveis e seus intervalos de variação, ou seja, definir os conjuntos de objetos do modelo e suas dimensões;
b) definir as variáveis de decisão, os coeficientes e os grupos de restrições em termos dos subscritos definidos no passo (a);
c) especificar, a partir das restrições físicas do problema, os relacionamentos entre os itens definidos no passo (b), o que inclui a função-objetivo e as restrições do modelo.

Para ilustrar, considere um problema de produção. Neste caso, a formulação algébrica irá requerer:

Conjuntos de Objetos e Índices de Conjunto:

P - conjunto de produtos ($i \in P$, um produto).
R - conjunto de recursos ($j \in R$, um recurso).

Variável de Decisão:

x_{ij} - quantidade do produto i fabricado com o recurso j ($i \in P$; $j \in R$).

Coeficientes:

c_{ij} - lucro unitário do produto i fabricado com o recurso j.
d_i - demanda pelo produto i.
a_{ij} - quantidade do recurso j necessário para a produção unitária do produto i.
b_j - disponibilidade do recurso j.
($i \in P$; $j \in R$)

Relacionamentos Lineares para a Função-Objetivo e as Restrições:

$$\text{Max} \sum_{i \in P} \sum_{j \in R} c_{ij} x_{ij}$$

Sujeito a:

$$\sum_{i \in P} a_{ij} x_{ij} \leq b_j \quad (j \in R)$$

(a utilização de um recurso não pode ser maior do que a disponibilidade deste recurso);

$$\sum_{j \in R} x_{ij} \geq d_i \quad (i \in P)$$

(a quantidade fabricada de cada produto não pode ser menor que a demanda por este produto)

$x_{ij} \geq 0 \ (i \in P; j \in R)$.

(a quantidade fabricada não pode ser negativa).

Assim, o modelo:

$$\text{Max} \sum_{i \in P} \sum_{j \in R} c_{ij} x_{ij}$$

Sujeito a:

$$\sum_{i \in P} a_{ij} x_{ij} \leq b_j \quad (j \in R)$$

$$\sum_{j \in R} x_{ij} \geq d_i \quad (i \in P)$$

$$x_{ij} \geq 0 \ (i \in P; j \in R).$$

representa uma *classe* dos problemas de produção. Uma *instância* (problema específico) desta classe é estabelecida ao se definir os dados do problema.

Embora os passos definidos por Lucas constituam um procedimento sistemático e estruturado para a construção de um modelo, como ilustrado pelo exemplo acima, a tarefa de modelagem é frequentemente demorada, sujeita a erros e perfeitamente compreensível somente por especialistas. Além disto, como comentado no capítulo anterior, para que um modelo de PL possa ser resolvido por computador é necessário representá-lo em uma forma aceitável pelo algoritmo de solução. Assim, muitas ferramentas têm sido desenvolvidas para estreitar a distância entre a representação algébrica de uma classe de problemas (conveniente a especialistas) e a representação conveniente ao programa otimizador. Collaud and Boltuck (1994) classificam estas ferramentas em gerações sucessivas, conforme ilustra a Figura 3.1.

unesp

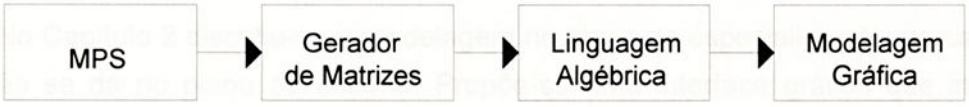


Fig. 3.1 - Gerações de Ferramentas de Modelagem

Nos últimos anos, o advento de sistemas de computação gráfica tem influenciado o projeto de interfaces para o usuário e alterado a forma de representação de modelos. Muitos autores têm discutido o uso de representações gráficas para facilitar a tarefa de formulação de modelos de PL (Ma *et al.*, 1989; Jones, 1990; Jones, 1991; Choobineh, 1991b; Steiger *et al.*, 1993; Jones, 1994; Collaud and Boltuck, 1994; Forster and Mevert, 1994).

O emprego de técnicas de Inteligência Artificial (IA), especialmente, sistemas especialistas, em problemas de Pesquisa Operacional tem merecido atenção nos últimos anos (Belhot e Moccellin, 1989; Doukidis and Paul, 1992). O uso destas técnicas em Sistemas de Apoio à Decisão tem recebido também grande atenção. Mentzas (1994), por exemplo, faz uma revisão geral dos Sistemas de Informação Inteligentes e procura mostrar em que sentido tais sistemas constituem ferramentas poderosas para o processo de tomada de decisão em organizações modernas. Connell and Powell (1990) discutem as aplicações de sistemas especialistas em sistemas de apoio à decisão. O uso de técnicas de IA para auxiliar as funções de um sistema de modelagem tem recebido também grande atenção nos últimos anos (Dolk and Konsynski, 1984; Dhar and Croker, 1988; Greenberg *et al.*, 1989; Angehrn and Lüthi, 1990; Raghunathan, 1992; El-Najdawi and Stylianou, 1993). Especificamente com relação à tarefa de formulação de modelos, o emprego de técnicas de representação de conhecimento também tem sido discutido (Murphy, 1988; Stohr, 1988; Krishnan *et al.*, 1992; Murphy *et al.*, 1992b; Mitra *et al.*, 1994).

Neste capítulo propõe-se uma interface gráfica inteligente para a formulação de modelos de PL. Esta proposta considera o problema de formulação de modelos em dois planos: o *plano do especialista* construtor do modelo, no qual é interessante dispor-se das características associadas às linguagens algébricas, ou seja, generalidade, declaratividade, pequeno (para especialistas) volume de trabalho e independência em relação aos dados e ao otimizador, e o *plano do usuário* não-especialista, onde é preciso uma visão mais próxima do problema real, a fim de alcançar as características de concretude e modularidade conceitual.

unesp

No Capítulo 2 discutiu-se a modelagem no plano do especialista. Neste capítulo a discussão se dá no plano do usuário. Propõe-se uma interface gráfica que incorpora técnicas de Inteligência Artificial para auxiliar na conceituação do problema e na formulação do modelo. Em linhas gerais, tal interface divide a tarefa de modelagem em três estágios, nos quais são construídos:

- o modelo de dados, onde estão estabelecidas as classes de objetos primitivos (classes primárias), as classes compostas em termos de relacionamentos entre as classes primárias e os atributos dos objetos destas classes (primárias e compostas), admitindo-se que os dados necessários ao modelo quantitativo estão armazenados em tabelas relacionais.
- o modelo conceitual, onde define-se o problema a ser modelado em termos das classes de objetos primitivos do domínio e de atividades que relacionam estes objetos. O modelo conceitual associa ainda, as variáveis de decisão e os atributos dos objetos primitivos e atividades através de um grafo de atributos.
- o modelo de Programação Linear, onde são estabelecidas a função-objetivo e as restrições do modelo.

Pretende-se com tal interface gráfica (e com os algoritmos de interpretação das construções gráficas) obter uma formulação algébrica de um problema de PL possível de ser executada por computador. Assim, com a interface proposta, a tarefa de formulação de modelos é vista segundo ilustra a Figura 3.2 a seguir.

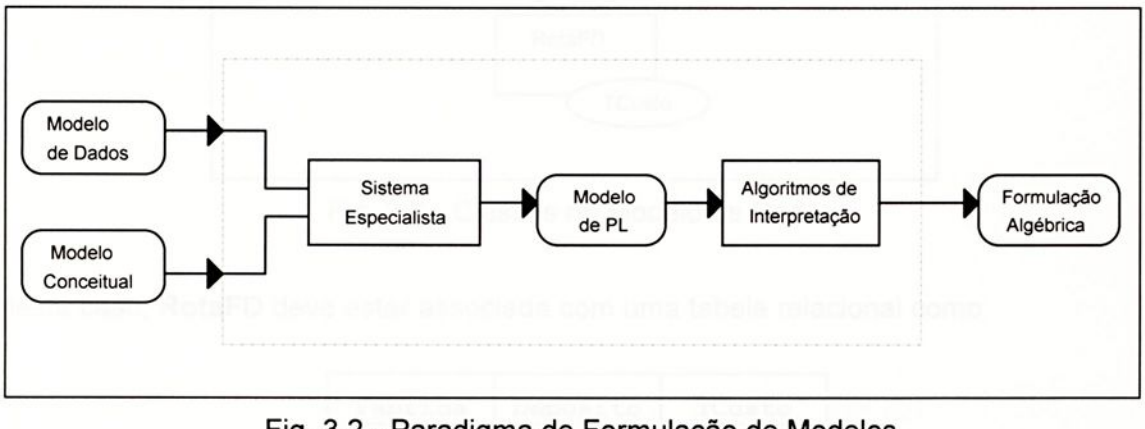


Fig. 3.2 - Paradigma de Formulação de Modelos

Nesta figura, o retângulo pontilhado inclui os passos automáticos do paradigma de formulação de modelos, ou seja, com a interface proposta neste trabalho o usuário deve descrever como os dados estão organizados (*Modelo de Dados*) e como é a

representação conceitual do problema (*Modelo Conceitual*) e deve receber como resultado, a formulação algébrica do problema escrita em uma linguagem algébrica de modelagem, pronta para execução em computador.

Tais modelos são discutidos nas seções a seguir, considerando que o objetivo é obter uma formulação algébrica na linguagem MPL (com as extensões propostas por Moody, 1994).

3.2 - O Modelo de Dados

Na construção do Modelo de Dados, presume-se que todos os dados necessários para uma instância do modelo estão armazenados em tabelas relacionais. Desta forma, cada um dos objetos do domínio do problema deve estar associado a, pelo menos, uma tabela relacional. Nestas tabelas de dados, as colunas correspondentes às chaves primárias estarão associadas às classes de objetos primitivos e as demais colunas da tabela serão denominadas atributos. Por exemplo, para um problema de distribuição, Fábrica e Depósito podem ser considerados como objetos primitivos do domínio e RotaFD, uma classe composta definida a partir destes dois objetos, como mostra a Figura 3.3 a seguir.

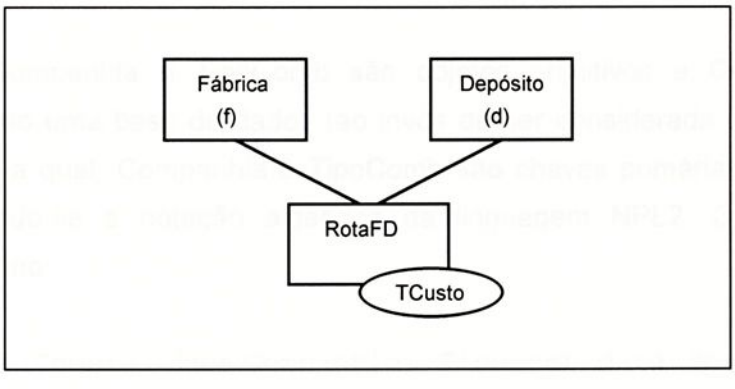


Fig. 3.3 - Classes no Modelo de Dados

Neste caso, **RotaFD** deve estar associada com uma tabela relacional como:

Fábrica	Depósito	TCusto
f1	d1	tc1
f2	d2	tc2
...	...	...

onde as colunas **Fábrica** e **Depósito** correspondem às chaves primárias da relação e a coluna **TCusto** corresponde a um atributo de RotaFD.

Na formulação algébrica, os objetos primitivos irão corresponder a conjuntos de índices unidimensionais e as classes compostas irão corresponder a conjuntos de índices multidimensionais, onde cada dimensão é definida por uma classe primária correspondente.

Admite-se que o Modelo de Dados inclua tabelas de dados que não são (ou não devem ser) associados a um objeto do domínio do problema. Neste caso, emprega-se um símbolo gráfico diferente, como mostra a Figura 3.4.

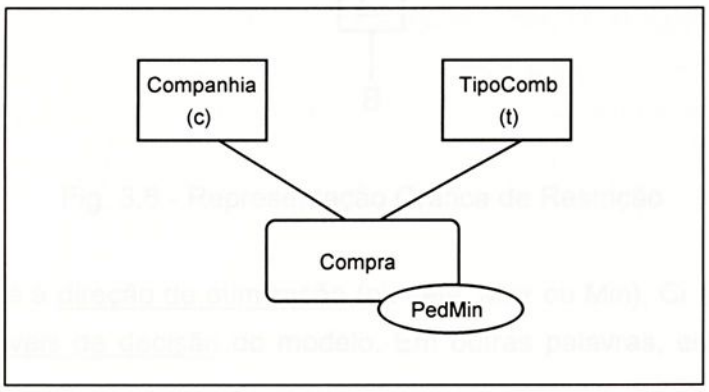


Fig. 3.4 - Objetos e Tabela de Dados no Modelo de Dados

Nesta figura, Companhia e TipoComb são objetos primitivos e **Compra** deve ser considerado como uma base de dados (ao invés de ser considerada como uma classe composta) para a qual, Companhia e TipoComb são chaves primárias e PedMin é um atributo. Adotando-se a notação algébrica da linguagem NPL2, Compra pode ser representado como:

```
table(Compra, key Companhia, TipoComb data PedMin);
```

3.3 - O Modelo de PL

O objetivo da interface gráfica proposta neste capítulo é a obtenção do modelo de PL formulado em uma linguagem algébrica de modelagem, conforme ilustra a Figura 3.2. Neste capítulo, para facilitar a apresentação, o Modelo de PL será representado graficamente. Assim, duas classes de símbolos são usadas na representação de um Modelo de PL, como ilustram as Figuras 3.5 e 3.6 a seguir.

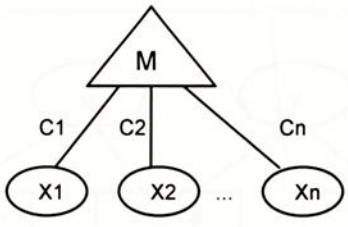


Fig. 3.5 - Representação Gráfica da Função-Objetivo

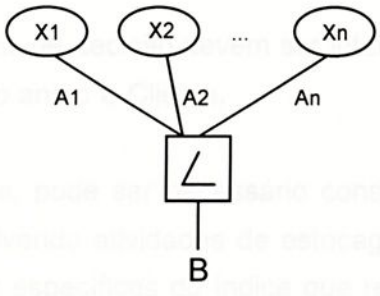


Fig. 3.6 - Representação Gráfica de Restrição

Na Figura 3.5, M é a direção de otimização (ou seja, Max ou Min), Ci são os coeficientes e Xi são as variáveis de decisão do modelo. Em outras palavras, esta figura deve ser interpretada como:

$$M \sum_i C_i X_i$$

Na Figura 3.6, Xi são as variáveis de decisão, $\angle$ é um operador relacional (ou seja, $\leq$, = ou $\geq$), e Ai e B são, respectivamente, o lado-esquerdo e o lado-direito da restrição. Portanto, a Figura 3.6 deve ser interpretada como:

$$\sum_i A_i X_i \angle B$$

De uma forma geral, os coeficientes Ci na funcão-objetivo e os coeficientes Ai e B nas restrições devem aparecer como atributos no Modelo de Dados. Para a representação algébrica do modelo, estes coeficientes (assim como as variáveis de decisão) devem ser interpretados a partir de seus relacionamentos com as classes primárias correspondentes. Por exemplo, considere a Figura 3.7 abaixo, onde as variáveis de decisão X e Y estão relacionadas com os objetos primitivos Depósito, Fábrica e Cliente.

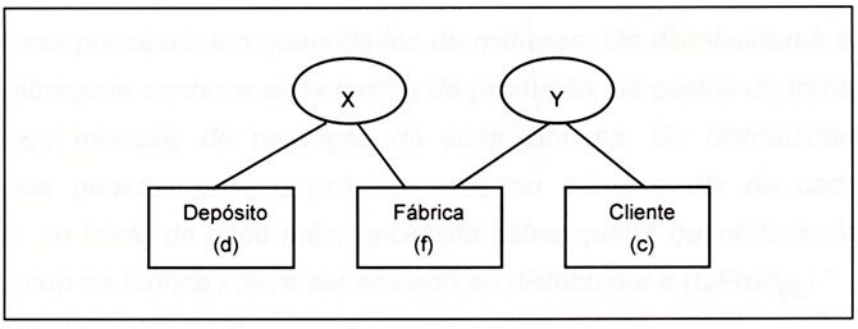


Fig. 3.7 - Variáveis de Decisão e Objetos Primitivos

Neste caso, as variáveis de decisão devem ser interpretadas como **X[f,d] e Y[f,c]** onde f ∈ Fábrica, d ∈ Depósito and c ∈ Cliente.

Para alguns problemas, pode ser necessário considerar valores particulares de índices. Em um modelo envolvendo atividades de estocagem, por exemplo, em geral é necessário considerar valores específicos do índice que representa períodos de tempo. O Modelo de PL, portanto, poderá incluir indices instanciados. Por exemplo, considere a variável de decisão X acima e seja um Modelo de PL contendo os elementos mostrados na Figura 3.8:

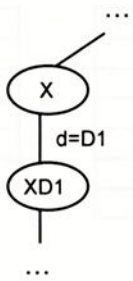


Fig. 3.8 - Variáveis de Decisão e Instâncias de Índices

Neste caso, X corresponde à variável de decisão X[f,d] (f ∈ Fábrica, d ∈ Depósito), ou seja, uma variável de decisão para cada possível par (Fábrica, Depósito), e XD1 denota a variável de decisão X[f,D1] (f ∈ Fábrica), instanciada para um depósito específico d = D1.

Para ilustrar estas construções, considere os seguintes exemplos.

Examplo 3.1. Problema de Distribuição (Lucas and Mitra, 1983):

> *"Um fabricante possui duas fábricas F1 e F2. Na fábrica F1 é possível fabricar os produtos P1 e P3, e na fábrica F2, os produtos P1, P2 e P3 podem ser manufaturados. O fabricante distribui estes produtos diretamente para três*

distribuidores principais em quantidades de milhares. Os distribuidores são D1, D2 e D3. O fabricante conhece seus custos de produção, os custos de transporte e as capacidades mensais de produção de suas fábricas. Os distribuidores devem enviar seus pedidos para o próximo mês no primeiro dia de cada mês. O fabricante, no início de cada mês, necessita saber qual a quantidade de produto j manufaturado na fábrica i deve ser enviado ao distribuidor k ($QProd_{ijk}$)."

Considere que os dados numéricos necessários ao modelo estão estabelecidos nas seguintes tabelas relacionais:

Fab_Capacidades

Fábrica	Produto	Capacidade
F1	P1	36
F1	P3	85
F2	P1	54
F2	P2	60
F2	P3	45

Distr_Pedidos

Produto	Distrib.	Pedido
P1	D1	50
P1	D2	10
P1	D3	30
P2	D1	15
P2	D2	15
P2	D3	20
P3	D1	20
P3	D2	60
P3	D3	30

Custos_Produção

Fábrica	Produtos	CustoProd
F1	P1	1.5
F1	P3	5.6
F2	P1	1.8
F2	P2	7.0
F2	P3	6.2

Custos_Transporte

Fábrica	Distrib.	CustoTransp
F1	D1	0.6
F1	D2	1.2
F1	D3	1.4
F2	D1	0.7
F2	D2	1.3
F2	D3	1.5

Este problema, formulado na linguagem MPL, pode ser escrito como:

```
TITLE Distr;
IMPORT
   Database("Fab_Capacidades",
            INDICES Fabrica = "Fabricas",
                    Produto = "Produtos",
            DATA    FCap    = "Capacidades");
   Database("Distr_Pedidos",
            INDICES Produto = "Produtos",
                    Distrib = "Distrib",
            DATA    DReq    = "Pedidos");
   Database("Custos_Producao",
            INDICES Fabrica = "Fabricas",
                    Produto = "Produtos",
            DATA    PCost   = "CustoProd");
   Database("Custos_Transporte",
            INDICES Fabrica = "Fabricas",
                    Distrib = "Distrib",
            DATA    TCost   = "CustoTransp");
DECISION VARIABLES
```

```
   QProd[Fabrica,Produto,Distrib];
MODEL
   MIN Custos = SUM(Fabrica,Produto,Distrib:
                  (PCost[Fabrica,Produto]+TCost[Fabrica,Distrib])*
                    QProd[Fabrica,Produto,Distrib]);
SUBJECT TO
   SUM(Distrib: QProd[Fabrica,Produto,Distrib]) <=
                  FCap[Fabrica,Produto];
   SUM(Fabrica: QProd[Fabrica,Produto,Distrib]) =
                  DReq[Produto,Distrib];
END
```

A representação gráfica deste problema será estabelecida em um Modelo de PL, como mostra a Figura 3.9 a seguir.

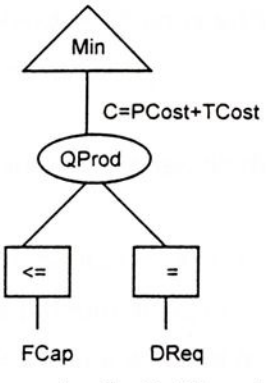


Fig. 3.9 - Representação Gráfica de Modelo de PL

Para que esta figura possa ser interpretada corretamente, seus elementos constituintes, como variáveis de decisão, coeficientes e atributos devem estar relacionados às correspondentes classes primárias no Modelo de Dados, conforme ilustra a Figura 3.10, que mostra a parte do Modelo de Dados para este problema, contendo tais informações.

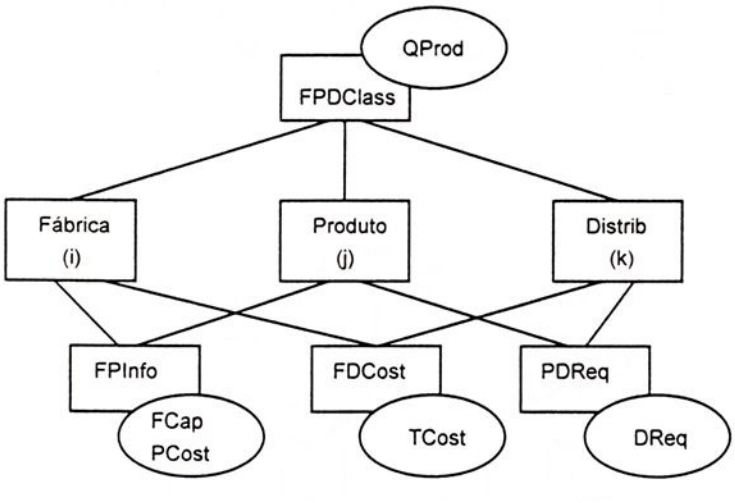


Fig. 3.10 - Parte de Modelo de Dados

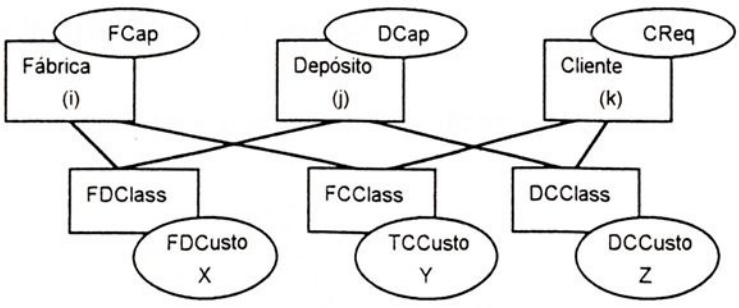
Exemplo 3.2. Problema de Distribuição de Produto Único, com Transbordo (Lucas *et al.*, 1993):

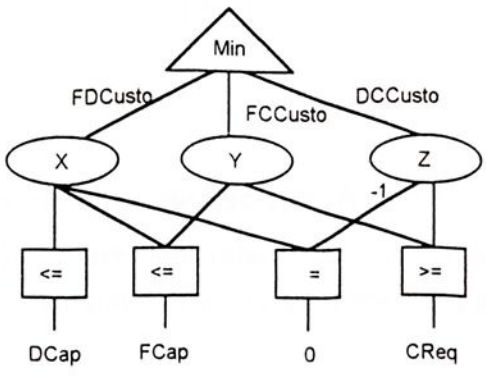
> *"Uma companhia possui i fábricas e j depósitos. Esta companhia vende seu produto para k clientes. A venda deste produto a cada um dos clientes pode ser feita por um depósito ou diretamente por uma fábrica. Todos os custos de distribuição: de uma Fábrica para um Depósito (FDCusto), de uma Fábrica para um Cliente (FCCusto) e de um Depósito para um Cliente (DCCusto) são pagos pela companhia. Cada fábrica possui uma capacidade mensal (FCap) de produção e cada depósito possui uma capacidade máxima de armazenamento (DCap) que não pode ser excedida. Além disto, a requisição mensal de cada cliente (CReq) precisa ser completamente atendida. A companhia deseja encontrar um padrão de distribuição que minimize seus custos."*

Neste caso, pode-se ter as seguintes variáveis de decisão:

X_{ij} - denotando a quantidade de produto enviada da fábrica i para o depósito j
Y_{ik} - denotando a quantidade de produto enviada da fábrica i para o cliente k
Z_{jk} - denotando a quantidade de produto enviada do depósito j para o cliente k

e os modelos de dados e de PL serem estabelecidos graficamente como:





Interpretado corretamente, este modelo pode ser escrito como:

$$\text{Min } (\sum_{i,j} \text{FDCost}_{ij} * X_{ij}) + (\sum_{i,k} \text{FCCost}_{ik} * Y_{ik}) + (\sum_{j,k} \text{DCCost}_{jk} * Z_{jk})$$

sujeito a:

$$\sum_{i} X_{ij} \leq \text{DCap}_j$$

$$\sum_{j} X_{ij} + \sum_{k} Y_{ik} \leq \text{FCap}_i$$

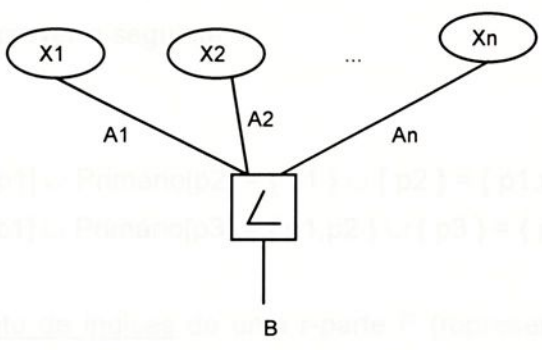
$$\sum_{i,j} X_{ij} - \sum_{j,k} Z_{jk} = 0$$

$$\sum_{i} Y_{ik} + \sum_{j} Z_{jk} \geq \text{CReq}_k$$

A questão é: *"como estas estruturas gráficas podem ser interpretadas de modo que seja possível traduzi-las para uma linguagem algébrica de modelagem?"*. Para resolver esta questão é necessário dispor de algoritmos capazes de interpretar o Modelo de Dados e interpretar tanto a função-objetivo como as restrições no Modelo de PL. A seção a seguir discute o algoritmo para interpretação das restrições no Modelo de PL e fornece algumas indicações como os demais algoritmos de interpretação devem ser especificados.

3.4 - Algoritmo de Interpretação de Restrições

Considere uma restrição R representada graficamente como:



onde X_i (i = 1, ..., n) são as **variáveis de decisão**, A_i (i = 1, ..., n) são os **coeficientes** tecnológicos, $\angle \in \{ \leq, =, \geq \}$ é o **operador relacional** e B é o valor do lado direito (ou **d-valor**, para resumir) de R. Para estabelecer formalmente o algoritmo de interpretação de restrições, admite-se que:

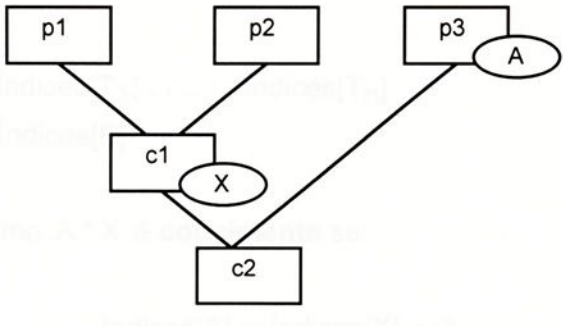
- as variáveis de decisão, os coeficientes e o d-valor serão referenciados coletivamente como partes de restrição (ou **r-parte**, para resumir);
- cada produto $A_i * X_i$ (i = 1, ..., n) será denominado termo de lado esquerdo (ou, resumidamente, **e-termo**);
- cada coeficiente A_i (i = 1, ..., n) e ld-valor B é uma constante ou um objeto do Modelo de Dados (ou **md-objeto**, para resumir).

Além disto, as seguintes definições serão necessárias:

Definição 3.1. O conjunto primário de um md-objeto D (representado por **Primário[D]**) é definido como:

Primário[D] = {D}, se D é um objeto primitivo;
Caso contrário, Primário[D] é a união dos conjuntos primários de todos os objetos relacionados diretamente com o objeto D no Modelo de Dados.

Por exemplo, considere que o Modelo de Dados inclui os seguintes objetos:



Neste caso, pode-se escrever o seguinte:

Primário[p1] = { p1 }
Primário[c1] = Primário[p1] ∪ Primário[p2] = { p1 } ∪ { p2 } = { p1,p2 }
Primário[c2] = Primário[c1] ∪ Primário[p3] = { p1,p2 } ∪ { p3 } = { p1,p2,p3 }

Definição 3.2. O conjunto de índices de uma r-parte P (representado por **Índices[P]**) é definido como:

Índices[P] = ∅, se P é uma constante;
Índices[P] = Primário[D], caso contrário

onde D é o md-objeto ao qual P está associado. Um elemento p $\in$ Índices[P] é denominado um subscrito de P.

Por exemplo, considere o Modelo de Dados acima:

Índices[A] = Primário[p3] = { p3 } (ou seja, o atributo A pode ser escrito como A_{p3})
Índices[X] = Primário[c1] = { p1,p2 } (ou seja, o atributo X pode ser escrito como $X_{p1,p2}$)

O conjunto de índices de um e-termo T é definido como a união dos conjuntos de índices das r-partes presentes em T.

Por exemplo, o conjunto de índices do e-termo A*X é:

$$\text{Índices[A*X]} = \text{Índices[A]} \cup \text{Índices[X]} = \{ p_3 \} \cup \{ p_1, p_2 \} = \{ p_1, p_2, p_3 \}.$$

Definição 3.3. Seja R uma restrição tal que T_i (i = 1, ..., n) são os e-termos e B é o d-valor de R. O conjunto de índices do lado esquerdo (representado por **E_Índices[R]**) e o conjunto de índices do lado direito (representado por **D_Índices[R]**) de R são definidos como:

E_Índices[R] = Índices[T_1] $\cup$ ... $\cup$ Índices[T_n]
D_Índices[R] = Índices[B]

Definição 3.4. Um e-termo A * X é **consistente** se:

$$\text{Índices[A]} \cap \text{Índices[X]} \neq \varnothing.$$

Por exemplo, o e-termo A*X, acima, é não-consistente, pois: Índices[A] $\cap$ Índices[X] = $\varnothing$.

Definição 3.5. Seja D um md-objeto tal que Primário[D] = {i_1,i_2,...,i_k}, k $\geq$ 2. Seja i $\in$ Primário[D]. A projeção do subscrito i em relação ao md-objeto D (representada por **Projeção[i,D]**), é definida como:

Projeção[i,D] = {"i in D[i_1,i_2,...,i_k]"}

Por exemplo, considerando o Modelo de Dados acima, {"p1 in c1[p1,p2]"} e {"p1 in c2[p1,p2,p3]"} são duas possíveis projeções do subscrito p1.

Definição 3.6. Seja T um e-termo não-consistente de uma restrição. Seja D um md-objeto tal que Índices[T] $\subseteq$ Primário[D]. Seja i $\in$ Índices[T]. Uma relação de consistência para o subscrito i (representado por **R_Consistência[i]**) é definido como:

$$R\_Consistência[i] = Projeção[i,D].$$

Definição 3.7. A forma consistente de subscrito i em relação a uma r-parte P (representada por **F_Consistente[i,P]**) é definida como:

F_Consistente[i,P] = $\varnothing$, se i $\notin$ Índices[P];
F_Consistente[i,P] = {i}, se i $\in$ Índices[P] e R_Consistência[i] = $\varnothing$;
F_Consistente[i,P] = R_Consistência[i], caso contrário.

A forma consistente de um conjunto de subscritos $\{i_1, ..., i_k\}$ ($k \geq 1$) em relação a uma r-parte P (representado por **F_Consistente[{i_1, ..., i_k},P]**) é definida como:

$$F\_Consistente[\{i_1, ..., i_k\},P] = F\_Consistente[i_1,P] \cup ... \cup F\_Consistente[i_k,P]$$

Definição 3.8. A expansão de uma r-parte P (representada por **Expansão[P]**), é definida como:

$$Expansão[P] = "P[p_1, ..., p_m]"$$

onde: $\{p_1, ..., p_m\}$ = F_Consistente[Índices[P],P].

A expansão de um e-termo T = A*X (representada por **Expansão[T]**) é definida como:

$$Expansão[T] = Concat(Expansão[A],"*",Expansão[X])$$

onde Concat é o operador de concatenação de cadeias de caracteres.

Por exemplo, a expansão do e-termo A*X em relação ao conjunto de subscritos {"p1 in c1[p1,p2]",p2,"p3 in c2[p1,p2,p3]"} é escrito como:

A["p3 in c2[p1,p2,p3]"]*X["p1 in c1[p1,p2]",p2].

Definição 3.9. Seja R uma restrição. O conjunto de índices não-compatíveis de R (representado por **N_Índices[R]**) é definido como:

$$N\_Índices[R] = D\_Índices[R] - E\_Índices[R].$$

A restrição R é **compatível** se:

$$N\_Índices[R] = \varnothing$$

Definição 3.10. Seja R uma restrição. O conjunto de índices projetados (representado por **Projetados[R]**), o conjunto de índices não-projetados (representado por **Livres[R]**), o conjunto de possíveis índices de somatórias (representado por **P_Soma[R]**) e o conjunto de índices consistentes (representado por **C_Índices[R]**) de R são definidos como:

Projetados[R] = $\{i_1, ..., i_k\} \subseteq$ E_Índices[R], tal que R_Consistência[i_j] $\neq \varnothing$ (j = 1, ..., k).
Livres[R] = E_Índices[R] - Projetados[R].
P_Soma[R] = Livres[R] - D_Índices[R]
C_Índices[R] = $\{i_1, ..., i_k\} \subseteq$ P_Soma[R], tal que R_Consistência[i_j] $\neq \varnothing$ (j = 1, ..., k).

Definição 3.11. Seja R uma restrição não-compatível. Seja $i \in$ N_Índices[R] e seja D um md-objeto, tal que:

a) $i \in$ Primário[D] = $\{i_1, ..., i_k\}$
b) Primário[D] $\cap$ Livres[R] $\neq \varnothing$

Uma relação de compatibilidade para o subscrito i (representado por **R_Compatibilidade[i]**) é definido como:

$$R\_Compatibilidade[i] = \{"i_j \text{ in } D[i_1, ..., i_k]"\}$$

onde $i_j \in$ Livres[R] (j = 1, ..., k).

Definição 3.12. Seja R uma restrição e seja $s \in$ N_Índices[R] tal que R_Compatibilidade[R,s] = $\{"i_s \text{ in } D_s[i_1,...,i_{k_s}]"\}$ ($k_s \geq 2$). A cláusula de seleção de R em relação ao subscrito s (representada por **S_Cláusula[R,s]**) é definida como:

$$S\_Cláusula[R,s] = \{"s = s \text{ in } D_s[i_1,...,i_{k_s}]"\}$$

A cláusula de seleção de R em relação a um conjunto $S = \{s_1, ..., s_k\}$ ($k \geq 0$) de subscritos (representada por **S_Cláusula[R,S]**) é definida como:

S_Cláusula[R,S] = ∅, se S = ∅;
S_Cláusula[R,S] = S_Cláusula[R,s_1] ∪ ... ∪ S_Cláusula[R,s_k], caso contrário.

Com estas definições, pode-se estabelecer o algoritmo de interpretação de restrições da maneira a seguir.

Algoritmo de Interpretação de Restrições

Seja R uma restrição.

Passo 1. [Estabelecer as relações de consistência para cada e-termo de R]
Para cada e-termo T de R, fazer:
Se T é não-consistente, então:
Seja i ∈ Índices[T].
Determinar R_Consistência[i].

Passo 2. [Estabelecer as relações de compatibilidade para R]
Para cada i ∈ N_Índices[R], fazer:
Determinar R_Compatibilidade[i].

Passo 3. [Determinar o conjunto S de índices de somatórias e o conjunto W de cláusulas de seleção de R]
Seja S = ∅.
Determinar P_Soma[R].
Se N_Índices[R] ≠ ∅, então:
Seja i ∈ N_Índices[R].
Seja R_Compatibilidade[i] = {"i_j in D[i_1, ..., i_k]"}.
Substituir S por (S ∪ R_Compatibilidade[i]).
Determinar S_Cláusula[R,N_Índices[R]-{i}].
Substituir P_Soma[R] por (P_Soma[R] - {i_1, ..., i_k}).
Para cada i ∈ C_Índices[R], fazer:
Seja R_Consistência[i] = {"i in D[i_1, ..., i_k]"}.
Substituir S por (S ∪ R_Consistência[i]).
Substituir P_Soma[R] por (P_Soma[R] - {i_1, ..., i_k}).
Substituir S por (S ∪ P_Soma[R]).

unesp

Para uma restrição R, os resultados produzidos por este algoritmo serão:

- as relações de consistência para cada e-termo de R;
- o conjunto S de índices de somatórias de R;
- o conjunto W de cláusulas de seleção de R.

A partir destes resultados, para gerar a **forma algébrica** da restrição, considere R escrita como:

$$R = T1 + ... + Tn \angle B$$

Sejam:

- e_i = Expansão[Ti] (i = 1, ..., n);
- b = Expansão[B];
- $S = \{s_1, ..., s_k\}$, o conjunto de índices de somatórias de R;
- $W = \{w_1, ..., w_m\}$, o conjunto de cláusulas de seleção de R;
- $S_i = S \cap$ Índices[Ti] $= \left\{s_{i1},...,s_{ik_i}\right\}$ (i = 1, ..., n; $k_i \geq 0$), o conjunto de índices de somatórias de R em relação ao e-termo Ti;

Com isto, a forma algébrica de R é escrita como:

[SumString i][WhereString] : e_1 + ... + [SumString n][WhereClause] : e_n $\angle$ b;

onde:

$$[\text{SumString i}] = \begin{cases} \text{"SUM } s_{i1},...,s_{ik_i}\text{"}, & \text{se } S_i \neq \phi \\ \text{vazio, se } S_i = \phi. \end{cases}$$

$$[\text{WhereString}] = \begin{cases} \text{"WHERE}(w_1,...,w_m)\text{"}, & \text{se } \{w_1,...,w_m\} \neq \phi \\ \text{vazio, se } \{w_1,...,w_m\} = \phi. \end{cases}$$

Para ilustrar a aplicação deste algoritmo, considere o problema enfrentado por companhias de geração de energia para alocar combustíveis às suas estações de potência (Kristjansson *et al.*, 1993). Cada companhia possui diversas estações de potência as quais requerem diversos tipos de combustível a partir de alguns postos de abastecimento. O problema é decidir quanto combustível deve ser transportado em cada

unesp

uma das rotas viáveis entre as estações de potência e os postos de abastecimento de modo a minimizar o custo total. As restrições a serem satisafeitas são as seguintes:

- casa estação de potência possui uma certa demanda de energia por satisfazer e níveis de produção de enxofre, cloro e cinza a serem respeitados;
- existem restrições ambientais sobre os níveis máximos de emissão de SO_2 para cada companhia;
- para cada companhia, as compras de cada tipo de combustivel necessário estão restritas a quantidades mínimas;
- existem limitações de capacidade sobre a disponibilidade de combustível de cada posto de abastecimento.

Considere que o Modelo de Dados para este problema inclui os objetos e atributos mostrados na Figura 3.11 abaixo.

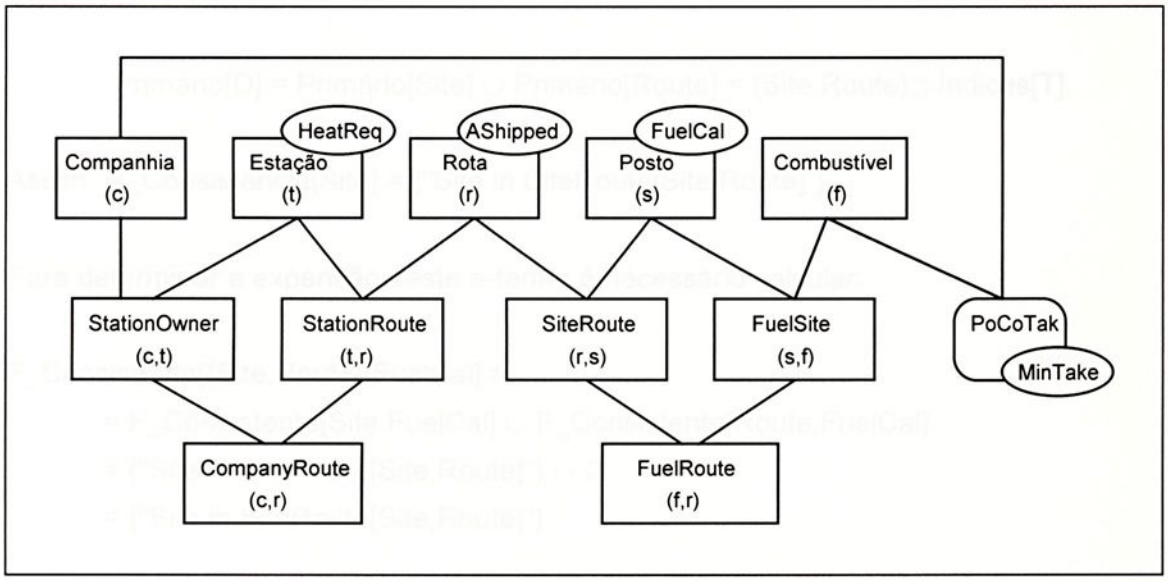
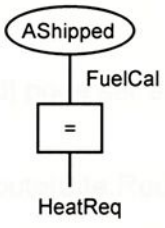


Fig. 3.11 - Modelo de Dados para o Problema das Estações de Potência

Exemplo 3.3. Seja uma restrição R representada graficamente como:



A seguir considera-se a execução passo-a-passo do algoritmo de interpretação de restrições para R.

Passo 1: Estabelecer as relações de consistência para cada e-termo de R.

Neste caso, o único e-termo é: T = FuelCal*AShipped.

Índices[FuelCal] = {Site};
Índices[AShipped] = {Route};
E_Índices[R] = {Site,Route};
D_Índices[R] = {Station};

Índices[FuelCal] ∩ Índices[AShipped] = ∅, ou seja, FuelCal*AShipped é não-consistente.

Índices[T] = {Site,Route}. Seja i = Site.

A partir do Modelo de Dados é possível encontrar D = SiteRoute, tal que:

$$\text{Primário}[D] = \text{Primário}[\text{Site}] \cup \text{Primário}[\text{Route}] = \{\text{Site},\text{Route}\} \supseteq \text{Índices}[T].$$

Assim, R_Consistência[Site] = {"Site in SiteRoute[Site,Route]"}.

Para determinar a expansão deste e-termo é necessário calcular:

F_Consistente[{Site,Route},FuelCal] =
= F_Consistente[Site,FuelCal] ∪ F_Consistente[Route,FuelCal]
= {"Site in SiteRoute[Site,Route]"} ∪ ∅
= {"Site in SiteRoute[Site,Route]"}

F_Consistente[{Site,Route},AShipped] =
= F_Consistente[Site,AShipped] ∪ F_Consistente[Route,AShipped]
= ∅ ∪ {Route}
= {Route}

Portanto, Expansão[FuelCal*AShipped] pode ser escrita como:

FuelCal["Site in SiteRoute[Site,Route]"]*AShipped[Route].

Observe que, como a escolha de i é arbitrária,

FuelCal[Site]*AShipped["Route in SiteRoute[Site,Route]"],

pode ser também considerada como uma expansão válida para este e-termo.

Passo 2: Determinar as relações de compatibilidade para R.

N_Índices[R] = D_Índices[R] - E_Índices[R] = {Station} - {Site,Route} = {Station}.

Portanto, N_Índices[R] ≠ ∅ e R é não-compatível.

Projetados[R] = {Site}.
Livres[R] = {Site,Route} - {Site} = {Route}.

Para determinar a relação de compatibilidade é necessário encontrar um md-objeto D tal que:

Station ∈ Primário[D];
Primário[D] ∩ {Route} ≠ ∅.

Do Modelo de Dados é possível encontrar D = StationRoute, pois:

Primário[StationRoute] = {Station,Route}

o que satisfaz as condições acima.

Portanto,

R_Compatibilidade[Station] = {"Route in StationRoutes[Station,Route]"}

Passo 3: Determinar o conjunto S de índices de somatórias e o conjunto W de cláusulas de seleção de R.

S = ∅
P_Soma[R] = {Route} - {Station} = {Route}

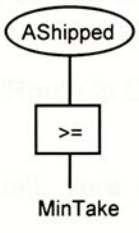
Pelo Passo 2, N_Índices[R] = {Station}. Assim, i = Station.

S = ∅ ∪ {"Route in StationRoute[Station,Route]"}
S_Cláusula[R, N_Índices[R] - {Station}] = S_Cláusula[R,∅] = ∅.
P_Soma[R] = {Route} - {Station,Route} = ∅.
C_Índices[R] = ∅.
S = {"Route in StationRoute[Station,Route]"} ∪ ∅.

Portanto, a forma algébrica da restrição FuelCal*AShipped = HeatReq será:

SUM("Route in StationRoute[Station,Route]" :
FuelCal["Site in SiteRoute[Site,Route]"]*AShipped[Route]) = HeatReq[Station];

Exemplo 3.4. Considere R como a seguinte restrição:



Neste caso a aplicação do algoritmo de interpretação de restrições será:

Passo 1: Estabelecer as relações de consistência para cada e-termo de R.

Neste caso o único e-termo é T = AShipped, e portanto não há necessidade de relação de consistência. Portanto, Expansão[AShipped] será:

AShipped[Route].

Passo 2: Determinar as relações de compatibilidade para R.

E_Índices[R] = {Route}
D_Índices[R] = Índices[MinTake] = {Company,Fuel}
N_Índices[R] = {Company,Fuel} - {Route} = {Company,Fuel}.

Ou seja, N_Índices[R] ≠ ∅ e portanto, R é não-compativel.

Neste caso, será necessário determinar:

R_Compatibilidade[Company], e
R_Compatibilidade[Fuel].

Projetados[R] = ∅
Livres[C] = {Route} - ∅ = {Route}

Para determinar R_Compatibilidade[Company], será necessário encontar D ∈ Modelo de Dados tal que:

Company ∈ Primário[D];
Primário[D] ∩ {Route} ≠ ∅.

A partir do Modelo de Dados pode-se encontrar D = CompanyRoute. Portanto:

R_Compatibilidade[Company] = {"Route in CompanyRoute[Company,Route]"}

Para determinar R_Compatibilidade[Fuel], será necessário encontrar D ∈ Modelo de Dados tal que:

Fuel ∈ Primário[D];
Primário[D] ∩ {Route} ≠ ∅.

Pode-se encontrar, no Modelo de Dados, D = FuelRoute. Portanto:

R_Compatibilidade[Fuel] = {"Route in FuelRoute[Fuel,Route]"}

Passo 3: Determinar o conjunto S de índices de somatórias e o conjunto W de cláusulas de seleção de R.

S = ∅
P_Soma[R] = {Route} - {Company,Fuel} = {Route}
Pelo Passo 2, N_Índices[R] = {Company,Fuel}. Seja i = Company.
S = ∅ ∪ {"Route in CompanyRoute[Company,Route]"}.
S_Cláusula[R, N_Índices[R] - {Company}] = S_Cláusula[R,{Fuel}] = S_Cláusula[R,Fuel] = {"Fuel = Fuel in FuelRoute[Fuel,Route]"}.

P_Soma[R] = {Route} - {Company,Route} = ∅.
C_Índices[R] = ∅.
S = {"Route in CompanyRoute[Company,Route]"}.

Portanto, a forma algébrica da restrição AShipped ≥ MinTake será:

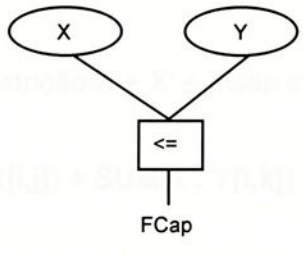
SUM("Route in CompanyRoute[Company,Route]"
WHERE ("Fuel = Fuel in FuelRoute[Fuel,Route]") :
AShipped[Route]) ≥ MinTake[Company,Fuel]

Deve ser observado que se a escolha de i no Passo 3 fosse: i = Fuel, a forma algébrica de AShipped ≥ MinTake seria:

SUM("Route in FuelRoute[Fuel,Route]"
WHERE ("Company = Company in CompanyRoute[Company,Route]") :
AShipped[Route]) ≥ MinTake[Company,Fuel]

que pode, portanto, ser considerada também como uma interpretação válida desta restrição.

Exemplo 3.5. Considere o Problema de Distribuição de Produto Único com Transbordo (ver Exemplo 3.2). Seja R a seguinte restrição:



A interpretação desta restrição transcorre da seguinte maneira:

Passo 1: Estabelecer as relações de consistência para cada e-termo de R.

Neste caso, existem dois e-termos: T1 = X e T2 = Y. Como estes e-termos são simples, não há necessidade de relações de consistência para eles. Logo:

Índices[T1] = {i,j}
Expansão[T1] = X[i,j]

Índices[T2] = {i,k}
Expansão[T2] = Y[i,k]

Além disto:

E_Índices[C] = {i,j} ∪ {i,k} = {i,j,k}
D_Índices[C] = Índices[FCap] = {i}

Passo 2: Determinar as relações de compatibilidade para R.

N_Índices[C] = {i} - {i,j,k} = ∅
Assim, neste caso, a restrição é compatível.

Projetados[C] = ∅
Livres[C] = {i,j,k} - ∅ = {i,j,k}

Passo 3: Determinar o conjunto S de índices de somatórias e o conjunto W de cláusulas de seleção de R.

S = ∅
P_Soma[C] = {i,j,k} - {i} = {j,k}
Pelo Passo 2: N_Índices[C] = ∅.
C_Índices[C] = ∅.
S = ∅ ∪ {j,k} = {j,k}

Portanto, a forma algébrica da restrição X + X' ≤ FCap será:

$$\mathbf{SUM(j : X[i,j]) + SUM(k : Y[i,k]) \leq FCap[i]}$$

Apresenta-se no Apêndice B a listagem de um programa Lisp que implementa este algoritmo de interpretação de restrições.

3.5 - O Modelo Conceitual

O Modelo Conceitual define o problema de Programação Linear em termos das classes primárias de objetos (conjuntos primários de índices de objetos do domínio de

aplicação do problema) e de atividades que estão relacionadas com estas classes. No Modelo Conceitual, as classes primárias (representadas por retângulos) e as atividades (representadas por retângulos com bordas em negrito) são conectadas por arcos que possuem atributos. Os atributos de um arco são classificados como limitações (limitação de capacidade, limitação de demanda, ...) ou como coeficientes. O Modelo Conceitual pode conter, também, arcos orientados, representando entradas e saídas de alguma entidade em um objeto. Neste caso, admite-se que um balanço deve ser verificado entre as entradas e saídas do objeto ao qual os arcos orientados se referem (que pode ser uma classe primária ou uma atividade). As atividades do Modelo Conceitual também possuem seus próprios atributos: variáveis de decisão e coeficientes (os quais são classificados como coeficientes de perdas ou de ganhos) associados a cada uma das atividades.

Como sugere o trabalho de Murphy (1988), os objetos primitivos do domínio (classes de índices) podem ser classificados em termos da dimensão às quais se referem, por exemplo: material (que indica **o que** é o objeto), local (que indica **onde** está o objeto), modo (que indica **como** é obtido o objeto) e tempo (que indica **quando** existe o objeto). As atividades também podem ser classificadas em categorias (por exemplo, atividade de transporte, atividade de armazenagem, atividade de composição, ...) para as quais pode-se estabelecer algumas características específicas. O Modelo Conceitual desprovido dos nomes específicos dos seus elementos constituintes (atividades, variáveis de decisão, ...) e no qual os objetos do domínio são representados somente pelas suas dimensões correspondentes e as atividades são representadas apenas por suas categorias denomina-se **Esquema Conceitual** para esta classe particular de problemas.

Considere, por exemplo, o "Problema de Distribuição de Produto Único, com Transbordo" (ver Exemplo 3.2). Para este problema, o Modelo Conceitual será representado como mostra a Figure 3.12 a seguir.

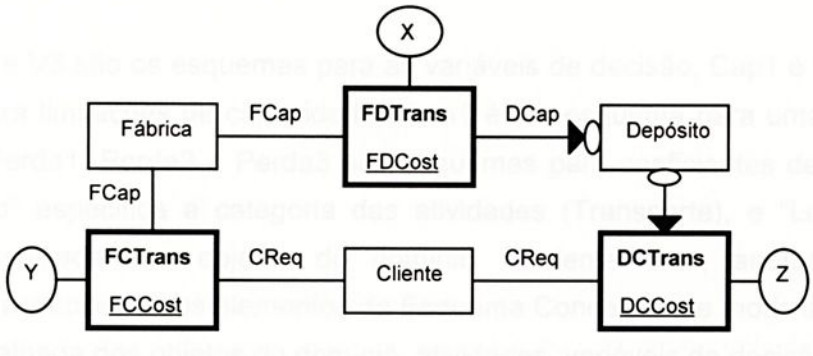


Fig. 3.12 - Modelo Conceitual: Problema de Distribuição com Transbordo

Nesta figura,

- Fábrica, Depósito e Cliente são as classes primárias (objetos primitivos do domínio), as quais referem-se todas à dimensão "local";
- FDTrans, FCTrans e DCTrans são as atividades;
- X (associado com a atividade FDTrans), representando a quantidade de produto enviada de uma Fábrica para um Depósito, Y (associado com a atividade FCTrans), representando a quantidade de produto enviada de uma Fábrica para um Cliente, e Z (associado com a atividade DCTrans), representando a quantidade de produto enviada de um Depósito para um Cliente, são as variáveis de decisão;
- FCap e DCap são limitações de capacidade e CReq é uma limitação de demanda;
- FCCost, FDCost e DCCost, os quais são mostrados sublinhados dentro dos retângulos representando as atividades, são coeficientes de perda;
- A entrada (relacionada com X) e a saída (relacionada com Z) da classe Depósito (note que os arcos orientados estão associados à classe Depósito) devem ser balanceadas.

Observe também que, a partir desta figura, é possível estabelecer o Esquema Conceitual para o Problema de Distribuição com Transbordo como mostra a Figura 3.13.

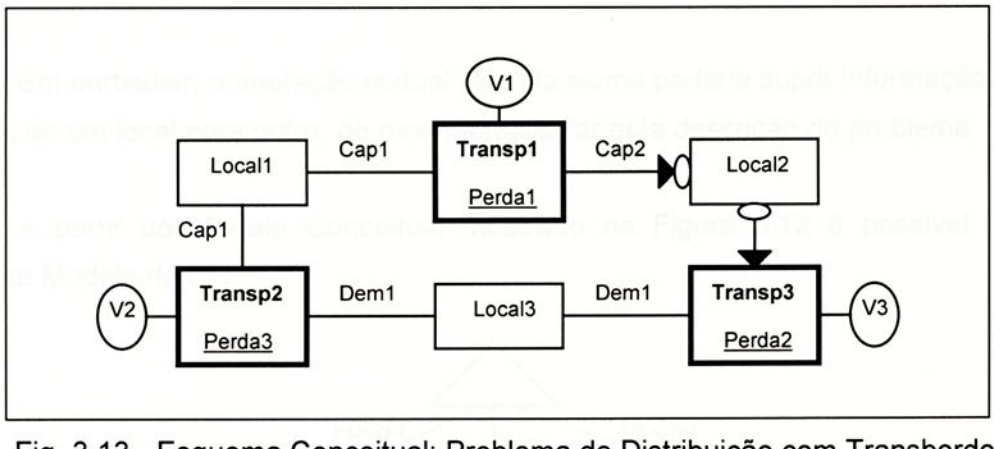


Fig. 3.13 - Esquema Conceitual: Problema de Distribuição com Transbordo

onde, V1, V2 e V3 são os esquemas para as variáveis de decisão, Cap1 e Cap2 indicam esquemas para limitações de capacidade, Dem1 é um esquema para uma limitação de demanda e Perda1, Perda2 e Perda3 são esquemas para coeficientes de perda. Além disto, "Transp" especifica a categoria das atividades (Transporte), e "Local" indica a dimensão associada aos objetos do domínio. Evidentemente, anotações textuais poderiam ser associadas aos elementos do Esquema Conceitual de modo a permitir uma descrição detalhada dos objetos do domínio, atividades, variáveis de decisão e assim por

diante. Observe que a partir da Figura 3.12 (e de informações presentes no Modelo de Dados) é possível gerar uma "descrição" para o problema, ou seja:

"O problema consiste das seguintes atividades:

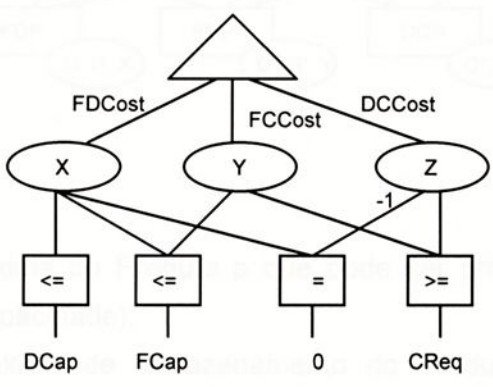
1. *Uma atividade de transporte de Fábrica i para Depósito j, sujeita a:*
   - *limitação de capacidade FCap[i];*
   - *limitação de capacidade DCap[j].*
2. *Uma atividade de transporte de Fábrica i para Cliente k, sujeita a:*
   - *limitação de capacidade FCap[i];*
   - *limitação de demanda CReq[k].*
3. *Uma atividade de transporte de Depósito j para Cliente k, sujeita a:*
   - *limitação de demanda CReq[k].*

O objetivo é determinar X[i,j], Y[i,k] e Z[j,k] (i ∈ Fábrica, j ∈ Depósito, k ∈ Cliente) levando em consideração:

- *o coeficiente de perda FDCost[i,j],*
- *o coeficiente de perda FCCost[i,k], e*
- *o coeficiente de perda DCCost[j,k]."*

Em particular, a anotação textual referida acima poderia suprir informação sobre o que flui de um local para outro, de modo a melhorar esta descrição do problema.

A partir do Modelo Conceitual mostrado na Figura 3.12 é possível inferir o seguinte Modelo de PL:



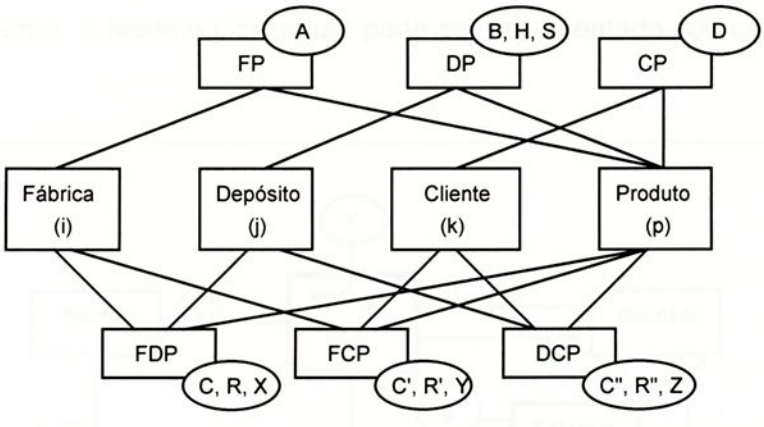
Ou seja, neste caso, é necessário apenas informar a direção de otimização do modelo (Min) para completar o Modelo de PL. As suposições sobre as quais se apoiam este processo de inferência são as seguintes:

- Cada atributo de arco do tipo limitação gera uma restrição no Modelo de PL. O lado esquerdo desta restrição deve levar em conta todos os arcos em que este atributo aparece e, consequentemente, incluir as variáveis de decisão associadas a estes arcos. Uma limitação de capacidade gera uma restrição do tipo "≤" e uma limitação de demanda gera uma restrição do tipo "≥";
- Numa restrição de balanço, as entradas são positivas e as saídas, negativas.

Observe que, como os coeficientes das atividades são classificados como ganhos ou perdas, é possível melhorar o processo de inferência incluindo-se uma nova "regra":

A direção de otimização do modelo deve ser "Min" se existem somente coeficientes de perda na função-objetivo. Caso contrário, a direção de otimização deve ser "Max". Se a direção de otimização do modelo é "Max" então os coeficientes de perda (caso existam) devem ser considerados com o sinal negativo.

Considere, a seguir, a extensão do problema de distribuição visto anteriormente para o caso de múltiplos produtos. Neste caso, o Modelo de Dados pode ser representado como:



onde:

A quantidade máxima do Produto p que pode ser produzida na Fábrica i (limitação de capacidade);

B capacidade máxima de armazenamento do Produto p no Depósito j (limitação de capacidade);

H custo de armazenagem do Produto p no Depósito j (coeficiente de perda);

D demanda do Cliente k pelo Produto p (limitação de demanda);

C custo unitário de distribuição do Produto p da Fábrica i para o Depósito j (coeficiente de perda);

C' custo unitário de distribuição do Produto p da Fábrica i para o Cliente k (coeficiente de perda);

C'' custo unitário de distribuição do Produto p do Depósito j para o Cliente k (coeficiente de perda);

R quantidade do Produto p distribuído da Fábrica i para o Depósito j (coeficiente);

R' quantidade do Produto p distribuído da Fábrica i para o Cliente k (coeficiente);

R'' quantidade do Produto p distribuído do Depósito j para o Cliente k (coeficiente);

X número de veículos com o Produto p em uma rota da Fábrica i para o Depósito j (variável de decisão);

Y número de veúculos com o Produto p em uma rota da Fábrica i para o Cliente k (variável de decisão);

Z número de veículos com o Produto p em uma rota do Depósito j para o Cliente k (variável de decisão);

S quantidade do Produto p armazenado no Depósito j (variável de decisão).

Para este problema, o Modelo Conceitual pode ser representado como mostra a Figura 3.14 a seguir.

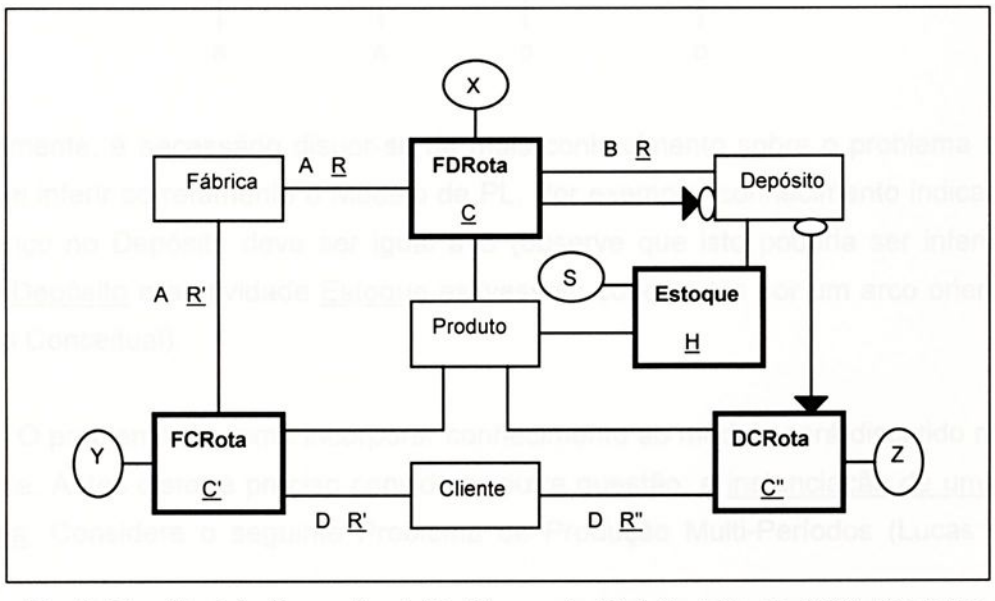
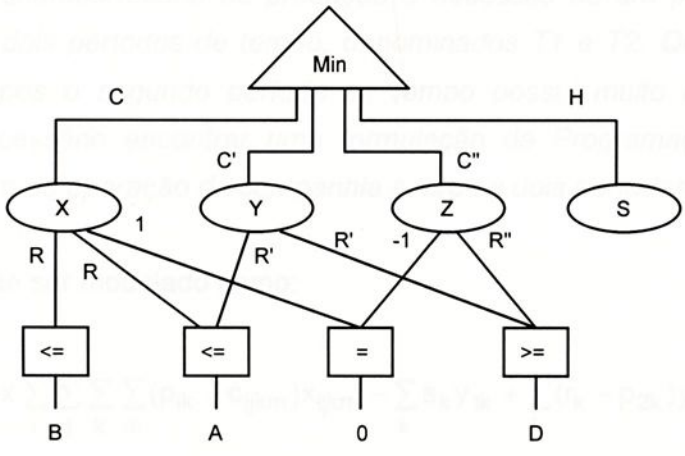
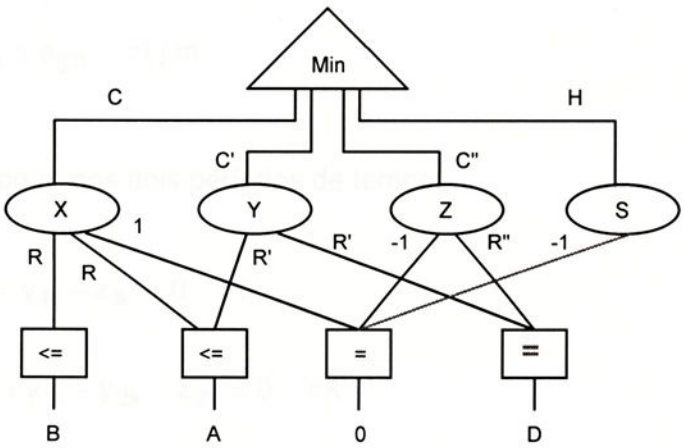


Fig. 3.14 - Modelo Conceitual: Problema de Distribuição de Múlti-Produtos

A partir deste Modelo Conceitual pode-se inferir o seguinte Modelo de PL:



Na verdade (ver Lucas *et al.*, 1993), o Modelo de PL para este problema deveria ser representado como:



Basicamente, é necessário dispor-se de mais conhecimento sobre o problema para ser possível inferir corretamente o Modelo de PL. Por exemplo, conhecimento indicando que o balanço no Depósito deve ser igual a S (observe que isto poderia ser inferido se o objeto Depósito e a atividade Estoque estivessem conectados por um arco orientado no Modelo Conceitual).

O problema de como incorporar conhecimento ao modelo será discutido na seção seguinte. Antes disto, é preciso considerar outra questão: a instanciação de uma classe primária. Considere o seguinte Problema de Produção Multi-Períodos (Lucas e Mitra, 1983):

"Uma companhia manufatura três produtos P1, P2 e P3 e possui, para isto, três máquinas M1, M2 e M3. A companhia pode operar em regime de produção normal

ou em regime sobrecarrecado de produção e necessita de um planejamento da produção para dois períodos de tempo, denominados T1 e T2. Qualquer produto não vendido após o segundo período de tempo possui muito pouco valor de revenda. É necessário encontrar uma formulação de Programação Linear que maximize o lucro de operação da companhia sobre os dois períodos."

Este problema pode ser modelado como:

$$\text{Max} \sum_i \sum_j \sum_k \sum_m (p_{ik} - c_{ijkm}) x_{ijkm} - \sum_k s_k y_{1k} + \sum_k (r_k - p_{2k}) y_{2k}$$

sujeito a:

a. disponibilidade de máquina:

$$\sum_k t_{ijkm} x_{ijkm} \leq a_{ijm} \quad \forall i, j, m$$

b. balanço de estoque nos dois períodos de tempo:

$$\sum_j \sum_m x_{1jkm} - y_{1k} - z_{1k} = 0 \quad \forall k$$

$$\sum_j \sum_m x_{2jkm} + y_{1k} - y_{2k} - z_{2k} = 0 \quad \forall k$$

c. demanda minima a ser satisfeita:

$$z_{ik} \geq d_{ik} \quad \forall i, k$$

d. limitante superior na armazenagem:

$$y_{1k} \leq h_k \quad \forall k$$

onde:

x_{ijkm} quantidade de produto k produzido no modo de produção j na máquina m no período de tempo i

c_{ijkm} custo de produção do produto k produzido no modo de produção j na máquina m no período de tempo i

y_{ik} quantidade de produto k armazenado no período i

z_{ik} quantidade de produto k vendido no período i

t_{ijkm} número de horas requiridas para produzir uma unidade do produto k na máquina m, no período de tempo i, usando o modo de produção j

a_{ijm} disponibilidade de máquina em horas para a máquina m no período i e modo de produção j

p_{ik} preço de venda do produto k no período de tempo i

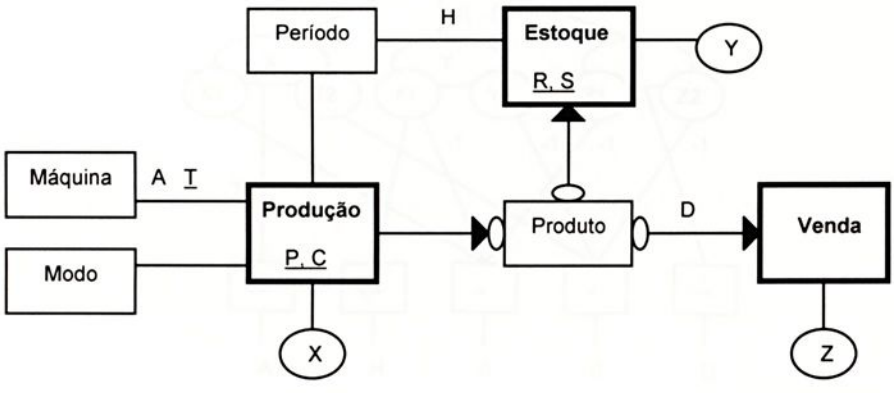
d_{ik} demanda pelo produto k no período de tempo i

s_k custo de armazenagem do produto k em um período de tempo

h_k capacidade de armazenagem para o produto k

r_k valor final de revenda do produto k

A dificuldade neste caso deve-se ao período, que precisa aparecer instanciado para ser possível levar em conta períodos específicos de tempo. Portanto, um Modelo Conceitual como o seguinte:



não será conveniente pois, neste caso, não é possível distinguir os períodos de tempo. Para este problema, é mais conveniente que o Modelo Conceitual seja representado como mostra a Figura 3.15.

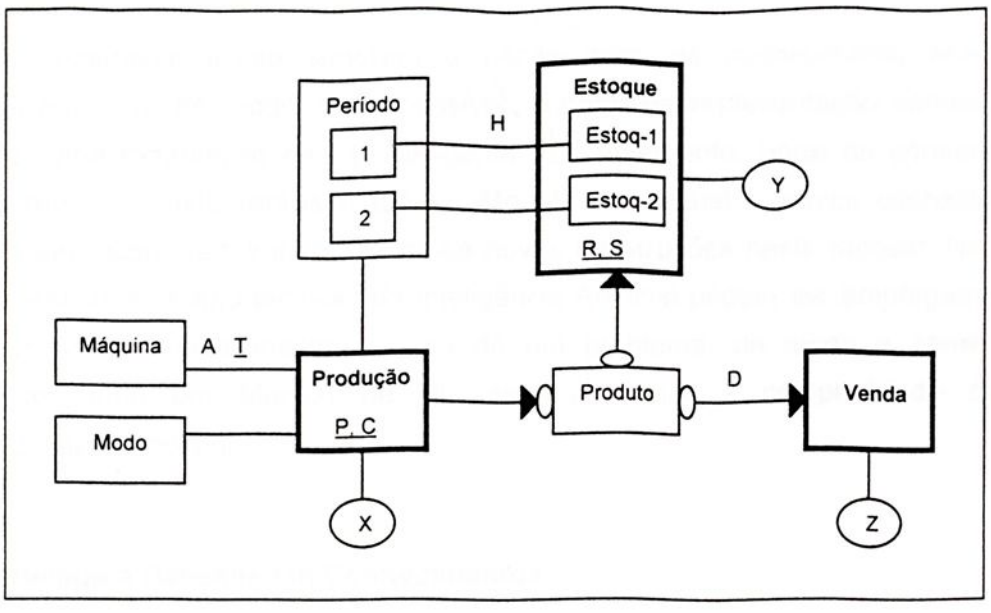
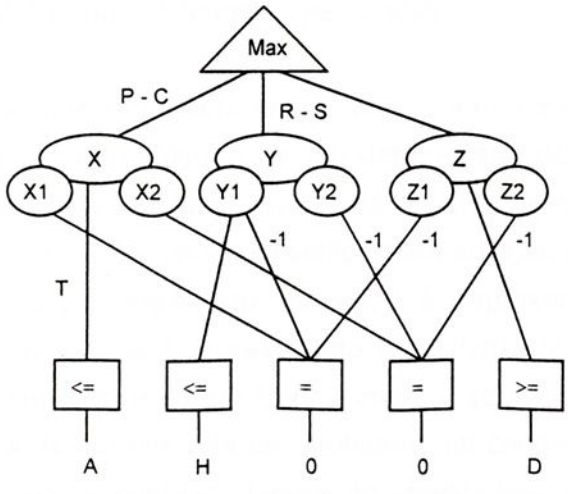


Fig. 3.15 - Modelo Conceitual: Problema de Produção Multi-Períodos

Nesta figura, o objeto Período aparece instanciado como Período = 1 e Período = 2 e existem sub-atividades Estoq-1 e Estoq-2 da atividade Estoque associadas a estes dois períodos, respectivamente. Da Figura 3.15, o Modelo de PL inferido será:



onde uma variável de decisão Vi corresponde à instanciação da variável V em relação ao período de tempo i. É fácil de ver, do modelo algébrico acima, que mesmo com os períodos de tempo instanciados não é possível inferir corretamente todo o Modelo de PL a partir do Modelo Conceitual. Neste caso, não foi possível inferir corretamente os coeficientes das variáveis de decisão na função-objetivo (porque o Modelo Conceitual é falho em representar tais coeficientes) nem o balanço de estoque no período 2. Claramente, é necessário dispor de mais conhecimento acerca do problema.

Os exemplos acima sugerem a necessidade de conhecimento acerca do problema em foco, de modo a ser possível, a partir da representação conceitual do problema, uma formulação de PL inteligente. Evidentemente, parte do conhecimento sobre o problema está representado no Modelo Conceitual e novos conhecimentos poderiam ser incorporados introduzindo-se novas construções neste modelo. Na seção seguinte discute-se como técnicas de Inteligência Artificial podem ser empregadas para incorporar novos conhecimentos acerca de um problema, de modo a permitir sua formulação como um Modelo de PL, sem aumentar a complexidade de sua representação conceitual.

3.6 - Modelagem Baseada em Conhecimentos

Um modelo é uma representação (abstrata) simbólica de um sistema real (concreto). No caso de modelos matemáticos tal representação envolve conceitos como constantes, variáveis, funções, relações e restrições. Neste contexto, a formulação de modelos é sinônimo de representação de conhecimento. Tópicos relacionados com representação de conhecimento têm sido estudados na área de Inteligência Artificial, em especial, na pesquisa relativa aos Sistemas Especialistas.

Sistemas Especialistas são programas de computador capazes de exibir um alto nível de perícia na solução de problemas de um determinado domínio de aplicação. A principal diferença entre os Sistemas Especialistas e outros sistemas de computação está na forma com que o conhecimento específico para solução de problemas (fatos e heurísticas obtidos de especialistas do domínio) é representado. Nos Sistemas Especialistas tal conhecimento está representado explicitamente e armazenado numa entidade separada, denominada Base de Conhecimento, ao invés de estar implícito na codificação do programa. Na solução de um problema, tal conhecimento é manipulado por uma estrutura de controle também claramente identificável, denominada Motor de Inferência (Senne, 1988). A Figura 3.16 apresenta a estrutura básica de um Sistema Especialista (Hayes-Roth *et al.*, 1985).

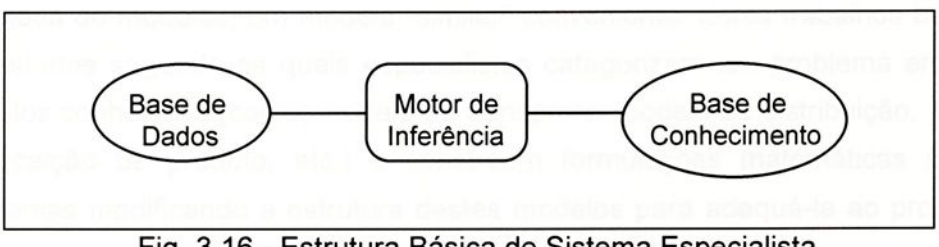


Fig. 3.16 - Estrutura Básica de Sistema Especialista

Existem muitas abordagens para a representação de conhecimento, por exemplo, o cálculo de predicados, as redes semânticas, as regras de produção e os objetos estruturados (*frames*). Destas representações, uma das mais empregadas em sistemas de formulação de modelos é a de regras de produção.

Como já adiantado no Capítulo 1, existem duas abordagens predominantes na construção de sistemas capazes de dar suporte à tarefa de formulação de modelos. Uma abordagem concentra-se apenas no conhecimento sintático ou estrutural dos modelos matemáticos, enquanto a outra abordagem combina conhecimento sintático de modelagem com conhecimento semântico do domínio de aplicação.

Usando o enfoque sintático, Stohr (1988), Murphy (1988) e Murphy *et al.* (1992b) desenvolvem um sistema capaz de auxiliar um especialista na construção de modelos de Programação Linear. O enfoque de combinar conhecimento sintático com conhecimento semântico foi usado, por exemplo, em Binbasiouglu and Jarke (1986), para construir um sistema capaz de formular modelos de PL para o domínio de planejamento de produção, e em Santhanam and Schniederjans (1993), para desenvolver um Sistema de Apoio à Decisão no domínio de seleção de projetos de sistemas de informação. Uma das vantagens deste segundo enfoque está na possibilidade de construir sistemas capazes de orientar um usuário não-especialista na formulação de modelos de decisão, por exemplo, na determinação de quais restrições são relevantes.

Nos últimos anos, muitos estudos têm sido feitos a respeito dos Sistemas de Modelagem Baseados em Conhecimento. Tais estudos procuram, principalmente, desenvolver ferramentas para automatizar a construção de modelos a partir de uma descrição qualitativa do problema, considerando a modelagem como um processo de transformação de representações. O trabalho de Bhargava and Krishnan (1993) apresenta uma revisão sobre as ferramentas baseadas em conhecimentos para a construção de modelos de PL e relaciona como principais contribuições:

- Obtenção da formulação matemática para um problema identificando, em uma biblioteca de modelos, um modelo "similar" conveniente. Estes trabalhos baseiam-se em estudos segundo os quais especialistas categorizam um problema em tipos de modelos conhecidos (como, modelo de transporte, modelo de distribuição, modelo de composição de produto, etc.) e constroem formulações matemáticas para seus problemas modificando a estrutura destes modelos para adequá-la ao problema em questão.

- Síntese da estrutura de um modelo a partir de uma representação do problema em termos de blocos de atividades (tais como: produção, transporte, composição, ...) e transformações de lugar, forma e tempo. Os blocos de atividades podem ser refinados de modo a tornarem-se cada vez mais detalhados. Esta representação é sintetizada em um modelo de PL por meio de regras que codificam os relacionamentos entre os tipos conhecidos de atividades. Uma característica importante desta abordagem é a habilidade em especificar grandes modelos a partir de uma combinação de sub-modelos.
- Síntese da estrutura matemática do modelo a partir de uma representação do problema em termos de conceitos do domínio e de seus relacionamentos, e de um conjunto de regras de construção de modelos independentes de domínio, tais como:

  SE:
  F é o fluxo de produtos de S para H, &
  A capacidade de produção em S é C, &
  A demanda requerida em H é D,
  ENTÃO:
  Soma(I) $F(I,J) \leq C$, &
  Soma(J) $F(I,J) \geq D$.

- Transformação da descrição do problema (em termos de conceitos do domínio e seus relacionamentos) em um modelo qualitativo de restrições, tais como:

  Utilização de um recurso ≤ Disponibilidade do recurso

  a partir de um conjunto de regras para gerar restrições sobre os objetos presentes no problema, e transformação do modelo qualitativo em um modelo matemático, a partir de um conjunto de regras de construção de modelos independentes de domínio e de informações semânticas associadas a atividades.

Discute-se a seguir, o desenvolvimento de um sistema especialista em formulação de modelos, capaz de interpretar modelos de dados e modelos conceituais conforme estabelecidos acima.

Como sugerido pelos exemplos do Seção 3.5, é necessário dispor de conhecimentos sobre o problema de modo a ser possível uma formulação inteligente de um modelo de PL. É importante que este conhecimento específico sobre modelagem possa ser incorporado ao sistema sempre que se mostrar necessário, como em um

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

processo evolutivo. Assim, é interessante que tal conhecimento possa ser acomodado em um sistema com arquitetura de Inteligência Artificial, como um Sistema Especialista.

Técnicas de Inteligência Artificial podem ser usadas para auxiliar outras funções de um sistema de modelagem e isto tem sido discutido por muitos autores, como já discutido anteriormente. A Base de Conhecimento de Modelos (ver Figura 1.1), por exemplo, pode ser estruturada de modo a armazenar tanto esquemas conceituais (e algumas informações relacionadas como, por exemplo, estruturas de atividades) para problemas típicos, visando auxiliar a construção de modelos com base em modelos e sub-modelos já conhecidos, como fatos e regras a serem usadas pelo Motor de Inferência para interpretar um modelo conceitual, visando auxiliar na tarefa de formulação de modelos de PL.

O trabalho discutido neste capítulo, entretanto, concentra-se no sistema de interface com o usuário e assim, considera o uso de conhecimento apenas na representação conceitual do problema e na tarefa de formulação de modelos.

Na construção do sistema especialista discutido a seguir adota-se um esquema baseado em regras e a linguagem OPS5 (Brownston *et al.*, 1985) para a representação de conhecimento. Assim, nesta implementação, o motor de inferência do sistema especialista é, na verdade, um interpretador OPS5 (Velasco, 1988). Mostra-se, a seguir, as estruturas OPS5 usadas para representar o Modelo de Dados, o Modelo Conceitual e o Modelo de PL, assim como alguns exemplos de regras da base de conhecimentos.

As seguintes estruturas são usadas para a descrição da parte do Modelo de Dados necessária para o sistema especialista:

(Object name class type) - descreve um md-objeto (coeficiente, variável de decisão, ...), onde name é a identificação do objeto e class é a classe (primária ou composta) relacionada a este objeto.

(Class name index) - descreve uma classe de objetos, onde name é a identificação da classe e index é o nome de um índice pertencente a esta classe.

(Index name type) - descrive um índice, onde name é a identificação do índice e type identifica a categoria do índice (como local, material, tempo, ...).

Para ilustrar, o Modelo de Dados para o Problema de Produção Multi-Períodos discutido na Seção 3.5 é representado da seguinte maneira:

unesp

```
(Object ^name A ^class TIME-MODE-PROD-MACH)
(Object ^name D ^class TIME-PROD)
(Object ^name H ^class PROD)
(Object ^name C ^class TIME-MODE-PROD-MACH)
(Object ^name S ^class PROD)
(Object ^name P ^class TIME-PROD)
(Object ^name R ^class PROD)
(Object ^name T ^class TIME-MODE-PROD-MACH)
(Object ^name X ^class TIME-MODE-PROD-MACH)
(Object ^name Y ^class TIME-PROD)
(Object ^name Z ^class TIME-PROD)

(Class ^name TIME ^index I)
(Class ^name MODE ^index J)
(Class ^name PROD ^index K)
(Class ^name MACH ^index M)
(Class ^name TIME-MODE-PROD-MACH ^index I)
(Class ^name TIME-MODE-PROD-MACH ^index J)
(Class ^name TIME-MODE-PROD-MACH ^index K)
(Class ^name TIME-MODE-PROD-MACH ^index M)
(Class ^name TIME-PROD ^index I)
(Class ^name TIME-PROD ^index K)

(Index ^name I ^type TIME)
(Index ^name J ^type MODE)
(Index ^name K ^type MATERIAL)
(Index ^name M ^type PLACE)
```

Para a descrição do Modelo Conceitual empregam-se as seguintes estruturas:

(Activity name class type) - descreve uma atividade, onde name é a identificação da atividade, class é a classe (primária ou composta) relacionada a esta atividade e type é a categoria da atividade (como transporte, transformação, ...).

(SubActv name actv index value) - descreve uma sub-atividade, onde name é a identificação da sub-atividade, actv é a atividade que contém esta sub-atividade, index é o nome de um índice pertencente a esta classe o qual está relacionado a esta sub-atividade e value é uma instância deste índice.

(Attribute name type actv sub) - descreve um atributo, onde name é a identificação do atributo, type é a categoria do atributo (como variável, coeficiente de perda, limitante, ...), actv é a atividade associada a este atributo e sub é o nome de uma sub-atividade desta atividade.

(Edge name actv class inst bal type) - descreve um arco que conecta objetos do Modelo Conceitual, onde name é a identificação do arco, actv e class são, respectivamente, os nomes da atividade e da classe (primária) que são conectados por este arco, inst é uma instância de índice associado a esta classe, bal indica qual é o objeto (atividade ou classe primária) que "possui" este arco, caso este seja um arco orientado e type é o

sentido de um arco orientado (+1, para arcos que entram no objeto e -1, para arcos que saem do objeto).

(Value name edge type) - descreve um valor existente em um arco, onde name é a identificação do valor, edge é o arco que contém este valor e type identifica a categoria deste valor (como limitação de capacidade, limitação de demanda, coeficiente, ...).

Para o Problema de Produção Multi-Períodos (ver Seção 3.5), o Modelo Conceitual é representado como:

```
(Activity ^name PRODUCTION ^class TIME-MODE-PROD-MACH ^type TRANSFORM)
(Activity ^name STORING ^class TIME-PROD ^type STORAGE)
(Activity ^name SELLING ^class TIME-PROD ^type SELL)

(SubActv ^name STORE-1 ^actv STORING ^index I ^value 1)
(SubActv ^name STORE-2 ^actv STORING ^index I ^value 2)

(Attribute ^name P ^type GAIN ^actv PRODUCTION)
(Attribute ^name C ^type LOSS ^actv PRODUCTION)
(Attribute ^name S ^type LOSS ^actv STORING ^sub STORE-1)
(Attribute ^name R ^type GAIN ^actv STORING ^sub STORE-2)
(Attribute ^name P ^type LOSS ^actv STORING ^sub STORE-2)
(Attribute ^name X ^type VARIABLE ^actv PRODUCTION)
(Attribute ^name Y ^type VARIABLE ^actv STORING)
(Attribute ^name Z ^type VARIABLE ^actv SELLING)

(Edge ^name E1 ^actv PRODUCTION ^class MACH)
(Edge ^name E2 ^actv PRODUCTION ^class MODE)
(Edge ^name E3 ^actv PRODUCTION ^class TIME)
(Edge ^name E4 ^actv PRODUCTION ^class PROD ^bal PROD ^type 1)
(Edge ^name E5 ^actv STORING ^class PROD ^bal PROD ^type -1)
(Edge ^name E6 ^actv STORING ^class TIME ^inst 1)
(Edge ^name E7 ^actv STORING ^class TIME ^inst 2)
(Edge ^name E8 ^actv SELLING ^class PROD ^bal PROD ^type -1)
(Edge ^name E9 ^actv SELLING ^class TIME)

(Value ^name A ^edge E1 ^type CAPACITY)
(Value ^name T ^edge E1 ^type COEFFICIENT)
(Value ^name H ^edge E6 ^type CAPACITY)
(Value ^name D ^edge E8 ^type DEMAND)
```

A representação do Modelo de PL irá empregar as seguintes estruturas:

(Objective type) - descreve a direção de otimização da função-objetivo, onde type é "Minimização" ou "Maximização".

(ObjFunct name sign coef dvar actv sub) - descreve os termos da função-objetivo, onde name é a identificação de um termo, sign indica se este termo é positivo (+1) ou negativo (-1), coef é o coeficiente deste termo, dvar é o nome da variável de decisão deste termo, actv é a atividade relacionada a esta variável de decisão e sub é uma sub-atividade desta atividade (caso exista).

unesp

(LhsTerm name edge sign coef dvar) - descreve um e-termo de uma restrição, onde name é a identificação da restrição, edge é o nome do arco do qual origina-se este e-termo, sign indica se este e-termo é positivo (+1) ou negativo (-1), coef é o coeficiente e-termo e dvar é o nome da variável de decisão deste e-termo.

(RhsTerm name oper rhs) - descreve o lado direito de uma restrição, onde name é a identificação da restrição, oper é o operador relacional (ou seja, ≤, =, ≥, ...) deste lado direito de restrição e rhs é o d-valor deste lado direito de restrição.

(GInstance name index value) - descreve uma instanciação global de índices em uma restrição, onde name é a identificação da restrição, index é o nome de um índice que deve aparecer globalmente instanciado nesta restrição e value é o valor da instância deste índice nesta restrição.

(LInstance name ident index value) - descreve uma instanciação local de índices em uma restrição, onde name é a identificação da restrição, ident é a identificação do e-termo desta restrição para o qual esta instanciação local se aplica, index é o nome do índice que deve aparecer localmente instanciado neste e-termo e value é o valor da instância deste índice neste e-termo.

Por exemplo, a análise do sistema especialista do Modelo de Dados e do Modelo Conceitual mostrados acima para o Problema de Produção Multi-Períodos resulta nas seguintes estruturas OPS5 representando o Modelo de PL:

```
 55: (LhsTerm ^dvar Z ^coef 1 ^sign 1 ^edge E8 ^name ?0)
 56: (LInstance ^ident E8 ^index K ^name ?0)
 57: (RhsTerm ^oper >= ^rhs D ^name ?0)
 59: (LhsTerm ^dvar Y ^coef 1 ^sign 1 ^edge E6 ^name ?1)
 60: (LInstance ^ident E6 ^value 1 ^index I ^name ?1)
 61: (RhsTerm ^oper <= ^rhs H ^name ?1)
 66: (LInstance ^ident E1 ^index M ^name ?3)
 68: (LhsTerm ^dvar X ^coef T ^sign 1 ^edge E1 ^name ?3)
 69: (RhsTerm ^oper <= ^rhs A ^name ?3)
 72: (LInstance ^ident E8 ^index K ^name ?4)
 74: (LInstance ^ident E5 ^index K ^name ?4)
 76: (LInstance ^ident E4 ^index K ^name ?4)
 80: (ObjFunct ^dvar Y ^coef P ^sign -1 ^sub STORE2 ^actv STORING ^name ?5)
 81: (LInstance ^ident OFUNCT ^value 2 ^index I ^name ?5)
 82: (ObjFunct ^dvar Y ^coef R ^sign 1 ^sub STORE2 ^actv STORING ^name ?6)
 83: (LInstance ^ident OFUNCT ^value 2 ^index I ^name ?6)
 85: (ObjFunct ^dvar Y ^coef S ^sign -1 ^sub STORE1 ^actv STORING ^name ?7)
 86: (LInstance ^ident OFUNCT ^value 1 ^index I ^name ?7)
 88: (ObjFunct ^dvar X ^coef C ^sign -1 ^actv PRODUCTION ^name ?8)
 89: (ObjFunct ^dvar X ^coef P ^sign 1 ^actv PRODUCTION ^name ?9)
 91: (Objective ^type Max)
113: (GInstance ^value 2 ^index I ^name ?10)
118: (LhsTerm ^dvar X ^coef 1 ^sign 1 ^name ?10)
120: (LhsTerm ^dvar Y ^coef 1 ^sign -1 ^name ?10)
122: (LhsTerm ^dvar Z ^coef 1 ^sign -1 ^name ?10)
123: (RhsTerm ^oper = ^rhs 0 ^name ?10)
```


unesp

```
139: (GInstance ^value 1 ^index I ^name ?11)
144: (LhsTerm ^dvar X ^coef 1 ^sign 1 ^name ?11)
146: (LhsTerm ^dvar Y ^coef 1 ^sign -1 ^name ?11)
148: (LhsTerm ^dvar Z ^coef 1 ^sign -1 ^name ?11)
149: (RhsTerm ^oper = ^rhs 0 ^name ?11)
164: (LhsTerm ^dvar Y ^coef 1 ^sign 1 ^edge STORE-TIME ^name ?10)
165: (LInstance ^ident STORE-TIME ^value 1 ^index I ^name ?10)
```

Observe, por exemplo, que as estruturas 55, 56 e 57 foram construídas porque existe uma limitação de demanda (D) no arco E8. Assim, estas três estruturas podem ser "lidas" como uma restrição denominada "?0" para a qual existe apenas um e-termo (1 * Z) e cujo lado direito possui "≥" como operador relacional e "D" como d-valor. Note que não existe instanciação local nesta restrição pois na estrutura 56, value é igual a "nil". Além disto, é interessante observar que não existe uma ordem predefinida para a construção dos componentes do modelo uma vez que as regras do sistema especialista são executadas segundo um esquema orientado pelos dados. Observe também, que é fácil traduzir estas estruturas OPS5 em uma representação algébrica esquemática. Para ilustrar, o Modelo de LP acima pode ser escrito como:

```
Max P * X - C * X - S * Y ( I = 1 ) + R * Y ( I = 2 ) - P * Y ( I = 2 )
Subject to:
   1 * Y ( I = 1 ) - 1 * Z - 1 * Y + 1 * X = 0 [ I = 2 ]
   1 * X - 1 * Z - 1 * Y = 0 [ I = 1 ]
   T * X <= A
   1 * Y ( I = 1 ) <= H
   1 * Z >= D
```

onde os índices instanciados localmente aparecem entre parênteses e as instanciações globais são escritas entre colchetes. Note, do Modelo de PL acima, que o sistema especialista foi capaz de inferir corretamente o balanço de estoque para cada período instanciado de tempo. Isto é possível porque a base de conhecimentos do sistema inclui a seguinte regra de balanço de estoque:

```
(p stock-balance
     (s-constr ^actv <a> ^type STORAGE)
     (Attribute ^name <v> ^type VARIABLE ^actv <a>)
     (LhsTerm ^name <n1> ^edge <> STORE-TIME ^sign <s> ^coef <c>
              ^dvar <v>)
     (LhsTerm ^name { <n2> <> <n1> } ^edge <> STORE-TIME ^dvar <v>)
     (GInstance ^name <n1> ^index <i> ^value <v1>)
     (GInstance ^name { <n2> <> <n1> } ^index <i> ^value { <v2> > <v1> })
     - (GInstance ^index <i> ^value { > <v1> < <v2> })
     (Index ^name <i> ^type TIME)
-->
     (make LhsTerm
           ^name <n2>
           ^edge STORE-TIME
           ^sign (compute -1 * <s>)
           ^coef <c>
           ^dvar <v>)
     (make LInstance
           ^name <n2>
           ^ident STORE-TIME
           ^index <i>
           ^value <v1>))
```


unesp

Essencialmente, esta regra estabelece:

SE:

- Existe uma atividade de estoque "a", e
- A variável "v" está associada a esta atividade "a", e
- Existe uma restrição para esta atividade incluindo um e-termo "(s)c * v", onde "s" é o sinal do coeficiente "c", para o qual o índice "i" possui um valor de instância global igual a "v1", e
- Existe uma outra restrição (denominada "n2") com um e-termo que inclui a mesma variável "v" globalmente instanciada para um valor "v2", tal que "v2" é maior do que "v1", e
- Não existe outro valor de instância global para a variável "v" que seja simultaneamente maior que "v1" e menor que "v2", e
- O índice "i" representa a dimensão tempo,

ENTÃO:

- Um novo e-termo, dado por: "(-1*s)c * v" deve ser incluído na restrição "n2".

A possibilidade de incluir novos conhecimentos específicos (como o ilustrado pela regra acima), além do conhecimento estrutural sobre como construir modelos de Programação Linear incorporado no Modelo Conceitual, dá ao sistema especialista de melhorar seu desempenho na formulação de modelos.

A listagem do protótipo de sistema especialista em formulação de modelos descrito neste capítulo encontra-se no Apêndice C. As representações OPS5 do Modelo de Dados e do Modelo Conceitual para os problemas discutidos neste capítulo, as respectivas sessões com o sistema especialista e os Modelos de PL resultantes também estão incluídos neste apêndice.

Capítulo 4

Algoritmos Heurísticos para Problemas de Localização

4.1 - Introdução

Um dos componentes mais importantes de um ambiente de modelagem, principalmente quando se considera tais ambientes como ferramentas de apoio à tomada de decisões, é a Base de Algoritmos de Solução (ver Figura 1.1). Na essência, são os algoritmos desta base que definem a utilidade da ferramenta na solução de problemas. Assim, para um ambiente inteligente de modelagem, é importante dispor de algoritmos eficientes.

No caso de modelos relacionados à existência de uma rede subjacente, embora para algumas classes de problemas existam algoritmos eficientes, grande parte dos problemas pertence à classe NP-difícil (*NP-hard*), para os quais uma solução ótima é obtida somente devido às custas de um grande esforço computacional (Garey and Johnson, 1979). Situações práticas de tais problemas irão exigir que decisões sejam tomadas rapidamente e, portanto, nem sempre uma solução ótima será necessária (ou possível), bastando dispor-se de boas soluções e da possibilidade de interagir facilmente com elas. Daí a importância de se ter algoritmos heurísticos para tais problemas.

Neste capítulo discute-se o desenvolvimento de algoritmos heurísticos para uma classe de problemas deste tipo: os problemas de localização. Em um problema de localização deseja-se estabelecer os locais onde serão sediados servidores para atender, da melhor maneira possível, um conjunto espacialmente distribuído de pontos de demanda (identificados como clientes ou consumidores). Pode-se imaginar que os locais dos servidores e dos consumidores constituem o conjunto $V = \{ 1, 2, ..., n \}$ de vértices de uma rede e os caminhos entre os servidores e os consumidores constituem o conjunto $A = \{ a_1, a_2, ..., a_m \}$ de arcos da rede, com $a_k = (i, j)$ e $i, j \in V$. Pode-se admitir também que, associado a cada arco $a_k = (i, j)$, $k = 1,...,m$, da rede, existe um valor não-negativo c_{ij}, o qual pode ser interpretado como a distância entre os vértices i e j, ou como o custo do caminho de i para j. Assim, soluções para tais problemas podem ser obtidas através

unesp

de modelos de otimização em redes.

Problemas de localização, dada a sua variedade e importância prática, vem sendo estudados por muitos pesquisadores, existindo uma extensa literatura a respeito. O estudo realizado por Brandeau and Chiu (1989) apresenta um levantamento de mais de 50 problemas representativos de localização. Estes autores apresentam como tais problemas foram formalmente estabelecidos, relacionam vários trabalhos e levantamentos de aplicações práticas sobre classes de problemas de localização já abordados e apresentam uma taxonomia para distinguir os diferentes tipos de problemas de localização.

Existem também alguns trabalhos que tratam problemas de localização vinculados a problemas de roteamento, ou seja, problemas onde as decisões relativas à escolha de locais não pode ser feito independentemente do estabelecimento de rotas entre estes locais. O trabalho de Laporte (1988), por exemplo, faz uma revisão desta classe de problemas, discute modelos nos quais as decisões de localização e roteamento são inter-relacionadas e apresenta algoritmos exatos e aproximativos de solução.

Neste capítulo discute-se uma nova abordagem para a solução de alguns problemas clássicos de localização. Esta abordagem baseia-se na combinação de duas heurísticas bem conhecidas - a relaxação lagrangeana e a relaxação *surrogate* - aliadas a um processo de otimização por subgradientes e leva a algoritmos heurísticos muito eficientes. Discute-se tal abordagem para problemas de p-medianas, localização simples e localização capacitada, e apresentam-se testes computacionais que demostram a eficiência dos algoritmos desenvolvidos. Antes de discutir os algoritmos heurísticos para estas classes de problemas, apresenta-se um resumo das técnicas lagrangeana e *surrogate*.

4.2 - As Relaxações Lagrangeana e *Surrogate*

Apresenta-se, formalmente, nesta seção, conceitos relativos às relaxações lagrangeana e surrogate importantes para este trabalho. O material apresentado aqui baseia-se nos trabalhos de Geoffrion (1974) e Parker and Rardin (1988), onde podem ser encontrados mais detalhes, incluindo provas de teoremas e provas de convergência de algoritmos.

O problema de programação linear inteira pode ser escrito como:

unesp

$$\min\ c\,x$$

sujeito a:

$$A\,x \geq b$$
$$B\,x \geq d \qquad x \geq 0,\ x \in I.$$

onde b, c e d são vetores, A e B são matrizes e x representa um vetor de variáveis inteiras. A separação das restrições do problema em dois conjuntos é para enfatizar que uma parte das restrições, $B\,x \geq d$, apresenta uma estrutura especial que pode ser explorada na busca de soluções. Seja este problema reescrito como:

$$\min\ c\,x$$

(P) sujeito a:

$$A\,x \geq b$$
$$x \in T$$

onde $T = \{ x \in I, x \geq 0 \mid B\,x \geq d \}$.

Seja (P) um problema de programação linear como estabelecido acima. Denota-se por:

($\overline{P}$), o problema que resulta de (P) quando são retiradas as condições de integralidade sobre suas variáveis;

F(P), o conjunto de soluções viáveis de (P);

v(P), o valor ótimo de (P).

Definição 4.1. Um problema (Q) é uma relaxação de um problema de minimização (P) se $F(Q) \supseteq F(P)$ e se a função objetivo de (Q) é menor ou igual à função objetivo de (P) no domínio F(P).

Definição 4.2. Seja (P) como estabelecido acima. Define-se a relaxação lagrangeana de (P) com relação às restrições $A\,x \geq b$ e a um vetor $t \geq 0$ como o problema:

$$\min\ c\,x + t\,(b - A\,x)$$

(L_tP) sujeito a:

$$x \in T$$

Claramente, (L_tP) é uma relaxação de (P) para todo $t \geq 0$ pois, $F(L_tP) \supseteq F(P)$ e o termo $t\,(b - A\,x)$ na função objetivo de (L_tP) é não positivo quando $A\,x \geq b$. Dado que

as restrições $B\,x \geq d$ possuem uma estrutura especial, espera-se com a relaxação lagrangeana que o problema (L_tP) possa ser resolvido eficientemente.

A utilidade de uma relaxação de (P) (e, em particular, da relaxação lagrangeana) é determinada por quão próximo seu valor ótimo está de v(P). Isto pode ser imaginado como um critério para medir a qualidade de uma escolha particular de t. O ideal seria escolher t como uma solução ótima do seguinte problema (denomindado dual lagrangeano):

$$\text{(DL)} \qquad \begin{aligned} &\max\ v(L_tP) \\ &\text{sujeito a:} \\ &t \geq 0. \end{aligned}$$

Teorema 4.1. Sejam (P), (L_tP) e (DL) como definidos acima. Então, para todo $t \geq 0$, $v(L_tP) \leq v(P)$ e, consequentemente, $v(DL) \leq v(P)$.

Teorema 4.2. Sejam (P), (L_tP) e (DL) como definidos acima. Se x é solução de (L_tP) para algum $t \geq 0$, e se $A\,x \geq b$ e $t\,(b - A\,x) = 0$, então x é solução ótima de (P).

É interessante observar, dos Teoremas 4.1 e 4.2, que a relaxação lagrangeana é de interesse não apenas para fornecer limitantes inferiores de v(P) mas também pela possibilidade de fornecer uma solução ótima (ou quase-ótima) de (P). No Teorema 4.2, se $t\,(b - A\,x) = \varepsilon$, então x é uma solução ε-ótima de (P). Na verdade, a qualidade das soluções obtidas pela relaxação lagrangeana é limitada pela diferença (*gap*) que existe entre v(P) e v(DL). O teorema a seguir caracteriza o valor ótimo do dual lagrangeano.

Teorema 4.3. Sejam (P), (L_tP) e (DL) como definidos acima. Então:

$$v(DL) = v\left(\begin{array}{l}\min cx \\ \text{sujeito a: } Ax \geq b,\ x \in [\,T\,]\end{array}\right)$$

onde [T] denota a envoltória convexa de T.

Corolário 4.1. Sejam (P), $(\overline{P})$ e (DL) como definidos acima. Então $v(DL) \geq v(\overline{P})$.

Este corolário mostra que o valor ótimo de (DL) é um limitante inferior, pelo menos, tão bom quanto o obtido pela relaxação de programação linear $(\overline{P})$. O corolário a seguir trata do caso limite.

Definição 4.3. Seja T como definido acima. Seja $\overline{T}$ a relaxação de programação linear de T. Certamente $[\ T\ ] \subseteq T$. Diz-se que T possui a propriedade de integralidade se $[\ T\ ] = \overline{T}$, ou seja, se $v(L_tP) = v(\overline{L_tP})$ para todo $t \geq 0$.

Corolário 4.2. Sejam $(\overline{P})$ e (DL) como definidos acima. Se T possui a propriedade de integralidade então $v(DL) = v(\overline{P})$.

Pode-se observar, deste corolário, que a utilidade da relaxação lagrangeana depende da escolha das restrições $B\ x \geq d$, ou seja, da escolha do conjunto T. Se tais restrições são tão simples a ponto de $[\ T\ ] = \overline{T}$, então a relaxação lagrangeana não conseguirá melhorar o que seria obtido pela relaxação de programação linear simplesmente.

Outro ponto importante para a efetividade da relaxação lagrangeana refere-se à escolha dos multiplicadores $t \geq 0$. Como comentado anteriormente, para determinar multiplicadores ótimos deve-se resolver o problema (DL). A dificuldade, neste caso, é que para avaliar a função objetivo de (DL) deve-se, para cada $t \geq 0$, resolver uma relaxação lagrangeana (L_tP). Entretanto, $v(L_tP)$ como função de t possui propriedades interessantes, como estabelece o teorema a seguir.

Teorema 4.4. Sejam (P) e (L_tP) como definidos acima. Então $v(L_tP)$, como função de t, é côncava e linear por partes.

Um dos métodos de solução de problemas não-lineares é a busca de gradientes. A idéia deste método é, partindo-se de um determinado ponto, avançar ao longo do gradiente da função até um ponto em que não se consiga mais um progresso significativo. Calcula-se, então, o gradiente neste último ponto e a busca continua. Se o problema a ser resolvido é de maximização e a função a ser maximizada é côncava, tal processo permite uma aproximação do ótimo. Do Teorema 4.4 tem-se que $v(L_tP)$ em função de t é côncava. A dificuldade aqui é que, como esta função é linear por partes, não é possível garantir que ela seja sempre diferenciável.

Para tratar a falta de diferenciabilidade é necessário generalizar o conceito de gradiente, o que é feito pela definição a seguir.

Definição 4.4. Um vetor s é um subgradiente de uma função (côncava) $\theta(t)$ no ponto t^* se, para todo t, $\theta(t^*) + s\ (t - t^*) \geq \theta(t)$.

Teorema 4.5. Seja (L_tP) como definido acima, com T não vazio e finito. Seja $S = \{ x \in T \mid x \text{ resolve } (L_tP) \}$. Então a coleção de subgradientes de $v(L_tP)$ para qualquer t tem a forma: $\{ \sum_{x \in S} \pi_x (b - Ax) \mid \sum_{x \in S} \pi_x = 1; \pi_x \geq 0, \text{ para todo } x \in S \}$.

O teorema a seguir estabelece que tais subgradientes aproximam (no sentido euclidiano de distância) as soluções de um problema à solução ótima.

Teorema 4.6. Seja $\theta(t)$ uma função côncava e contínua de t. Seja, para um dado α, o conjunto $T(\alpha) = \{ t \mid \theta(t) \geq \alpha \}$. Então, dados α, $t^* \notin T(\alpha)$ e s (subgradiente de θ no ponto t^*), existe δ tal que: $\| t - (t^* + \pi s) \| < \| t - t^* \|$ para todo $t \in T(\alpha)$ e $\pi \in (0, \delta)$.

Usando as idéias discutidas acima, pode-se estabelecer o seguinte algoritmo para a solução de um problema (P):

Algoritmo de Busca Lagrangeana por Subgradientes

1. Seja $t \geq 0$. Fazer $k \leftarrow 1$; $v \leftarrow -\infty$.
2. Resolver a relaxação lagrangeana (L_tP). Seja $x \in T$ a solução ótima obtida.
   Se $(b - A x) \leq 0$ e $t (b - A x) = 0$, então parar; x resolve o problema (P) e $v(L_tP) = v(DL) = v(P)$.
3. Se $v < v(L_tP) = c x + t (b - A x)$, fazer $v \leftarrow v(L_tP)$.
4. Fazer $t \leftarrow t + \pi_k (b - A x) / \| b - Ax \|$, onde π_k é um tamanho de passo que satisfaz as seguintes condições:
   $\pi_k > 0$ para todo k;
   $\lim_{k \to \infty} \pi_k = 0$;
   $\sum_{k=1}^{\infty} \pi_k = \infty$.
5. Fazer $t_j \leftarrow \max (0, t_j)$ para todo j. Fazer $k \leftarrow k + 1$ e voltar ao passo 2.

Deve-se observar neste algoritmo que é necessário manter o valor v de uma solução incumbente (passo 3) porque os passos de subgradiente não garantem melhorias em $v(L_tP)$. Outra observação refere-se aos tamanhos de passo π_k, que devem satisfazer as condições estabelecidas acima para garantir a convergência do algoritmo (Shapiro, 1979).

A relaxação lagrangeana é a mais familiar na Programação Matemática, mas existem várias outras. Outra relaxação possível do problema (P) é a relaxação *surrogate*,

unesp

estabelecida para um dado vetor $\lambda \geq 0$ como:

$$\begin{aligned} & \min \ c\,x \\ (SP^{\lambda}) \quad & \text{sujeito a:} \\ & \lambda\,(b - A\,x) \leq 0 \\ & x \in T \end{aligned}$$

Nota-se, portanto, que na relaxação surrogate o vetor $\lambda \geq 0$ é usado como multiplicador da restrição $A\,x \geq b$ do problema (ao invés de transferir estas restrições para a função objetivo, como no caso da relaxação lagrangeana).

Os conceitos apresentados para a relaxação lagrangeana podem ser estendidos para a relaxação surrogate. Assim, define-se o dual surrogate como o seguinte problema:

$$\begin{aligned} & \max \ v(SP^{\lambda}) \\ (DS) \quad & \text{sujeito a:} \\ & \lambda \geq 0. \end{aligned}$$

Da mesma forma, os teoremas da dualidade fraca e da dualidade forte apresentados para a relaxação lagrangeana (Teoremas 4.1 e 4.2, respectivamente) podem ser estabelecidos para a relaxação surrogate como:

Teorema 4.7. Sejam (P), (SP^{λ}) e (DS) como definidos acima. Então, para todo $\lambda \geq 0$, $v(SP^{\lambda}) \leq v(P)$ e, consequentemente, $v(DS) \leq v(P)$.

Teorema 4.8. Sejam (P), (SP^{λ}) e (DS) como definidos acima. Se x é solução de (SP^{λ}) para algum $\lambda \geq 0$, e se $A\,x \geq b$, então x é solução ótima de (P) e $v(DS) = v(P)$.

Os subproblemas (SP^{λ}) são geralmente mais difíceis de resolver do que os correspondentes lagrangeanos. Entretanto, os limitantes (inferiores) obtidos pela relaxação surrogate podem ser melhores que os correspondentes lagrangeanos, como estabelece o teorema a seguir.

Teorema 4.9. Sejam (P), (L_tP), (DL), (SP^{λ}) e (DS) como definidos acima. Então $v(DL) \leq v(DS)$. Além disto, se $v(DL) = v(DS)$, então para todo t^* resolvendo (DL) deve existir $x^* \in T$ tal que $t^*\,(b - A\,x^*) = 0$.

Dos teoremas da dualidade forte para as relaxações lagrangeana e surrogate (Teoremas 4.2 e 4.8, respectivamente), pode-se perceber que a versão lagrangeana requer uma hipótese adicional: t (b - A x) = 0. Pelo Teorema 4.9 percebe-se que esta condição aparece novamente na comparação de v(DL) e v(DS). Este teorema, portanto, garante que os limitantes obtidos pela relaxação surrogate serão sempre melhores (ou seja, maiores) do que os correspondentes lagrangeanos, a menos que toda solução para (DL) tenha uma complementar $x \in T$. Como é improvável que tais soluções complementares sempre existam, pode-se esperar que V(DS) > v(DL) para a maioria das formulações. Evidentemente, o esforço maior para resolver (SP^{λ}) pode sobrepujar a vantagem obtida com relação aos limitantes.

Da mesma forma que para a relaxação lagrangeana, a escolha de bons multiplicadores $\lambda \geq 0$ é crucial para a relaxação surrogate. Contudo, as similaridades entre o dual lagrangeano e o dual surrogate podem ser exploradas para generalizar o método de busca apresentado anteriormente (Parker and Rardin, 1988).

Nas seções a seguir, as idéias e os conceitos apresentados nesta seção são empregados para estabelecer uma nova abordagem para a solução de três classes de problemas de localização: p-medianas, localização simples e localização capacitada.

4.3 - Problema das P-Medianas

O objetivo do problema das p-medianas é determinar os locais de p servidores (denominados medianas) em uma rede de n vértices de modo a minimizar a soma das distâncias entre cada vértice consumidor e a mediana mais próxima. Um exemplo de problema das p-medianas é o seguinte: localizar, em uma cidade, p caixas de correio de modo a minimizar a distância total percorrida pelos usuários, imaginando-se que cada usuário prefere colocar suas correspondências na caixa de correio mais próxima.

Hakimi (1964; 1965) foi o primeiro a formular o problema para um único servidor, generalizando seu trabalho depois para multi-medianas. Este autor propôs também um procedimento simples de enumeração para o problema. O problema é bem conhecido como sendo NP-difícil e várias heurísticas têm sido desenvolvidas para resolvê-lo. Algumas destas heurísticas são usadas para obter boas soluções iniciais ou para calcular soluções intermediárias em métodos de solução mais sofisticados que exploram uma busca em árvore. Maranzana (1964) e Teitz and Bart (1968), por exemplo, propõem uma heurística de troca simples. Abordagens melhores aparecem em Efroymson and Ray

BIBLIOTECA unesp

(1966), Jarniven and Rajala (1972), Neebe (1978), Christofides and Beasley (1982), Beasley (1985), Galvão and Raggi (1989) e Beasley (1993). Os enfoques mais bem sucedidos consideram a relaxação lagrangeana e a otimização por subgradientes em uma abordagem primal-dual (Christofides and Beasley, 1982; Galvão and Raggi, 1989; Beasley, 1993).

Beasley (1993) descreve heurísticas muito eficientes para algumas classes de problemas de localização. Tais heurísticas são denominadas heurísticas lagrangeanas e usam a relaxação lagrangeana e a otimização por subgradientes. A cada iteração de subgradientes, as soluções lagrangeanas são tornadas (primal) viáveis aplicando-se heurísticas simples, as quais são substituídas por heurísticas de troca após um certo número de iterações. Lorena and Narciso (1996) têm introduzido heurísticas de relaxação para problemas generalizados de atribuição, usando um algoritmo geral de subgradientes. A nova relaxação apresentada é uma relaxação *surrogate*, usada anteriormente em outras aplicações, tais como problemas de cobertura de conjuntos (Lorena and Lopes, 1994) e problemas da mochila multidimensional (Freville *et al.*, 1990).

Nesta seção discute-se a aplicação de heurísticas de relaxação a problemas de p-medianas, introduzindo uma nova relaxação para o problema. A nova relaxação - denominada *relaxação LS* - combina as duas relaxações bem conhecidas para o problema das p-medianas. A relaxação lagrangeana e a relaxação surrogate são combinadas relaxando-se, no sentido surrogate, as restrições de atribuição na formulação do problema. Em seguida, a relaxação lagrangeana da restrição surrogate é obtida e otimizada de forma aproximada por um mecanismo dual unidimensional. As relaxações LS são mais estáveis (oscilam menos) e alcançam mais rapidamente os mesmos bons resultados das heurísticas puramente lagrangeanas.

Discute-se, a seguir, a relaxação LS e alguns aspectos teóricos para explicar seu bom comportamento. O problema de p-medianas considerado neste trabalho é modelado como o seguinte problema (P) de programação inteira binária:

$$v(P) = \min \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\, x_{ij}$$

sujeito a:

$$\sum_{i=1}^{n} x_{ij} = 1 \quad ; \quad j \in N \qquad (4.3.1)$$

FACULDADE DE ENGENHARIA
DE GUARATINGUETÁ
BIBLIOTECA

345

$$\sum_{i=1}^{n} x_{ii} = p \qquad (4.3.2)$$

$$x_{ij} \leq x_{ii} \quad ; \quad i,j \in N \qquad (4.3.3)$$

$$x_{ij} \in \{0,1\} \quad ; \quad i,j \in N \qquad (4.4.4)$$

onde:

$\left[d_{ij}\right]_{n\times n}$ é uma matriz simétrica de custos (ou distâncias), com $d_{ii} = 0$, $\forall i \in N$;

$\left[x_{ij}\right]_{n\times n}$ é uma matriz de alocação, com $x_{ij} = 1$ se o nó i é alocado ao nó j, e $x_{ij} = 0$, caso contrário; $x_{ii} = 1$ se o nó i é uma mediana e $x_{ii} = 0$, caso contrário;

p é o número de servidores (medianas) a serem localizados;

n é o número de nós da rede, e N = {1, ..., n}.

As restrições (4.3.1) e (4.3.3) garantem que cada nó j é alocado a apenas um nó i, o qual deve ser uma mediana. A restrição (4.3.2) determina o número exato de medianas a ser localizado (p), e a restrição (4.3.4) corresponde às condições de integralidade.

O objetivo é empregar heurísticas de relaxação para resolver o problema (P) de forma aproximada. A relaxação surrogate e a relaxação LS são definidas da forma a seguir.

Como proposto por Glover (1968), para um dado vetor $\lambda \in R^n$, uma relaxação surrogate de (P) - denotada por (SP^{λ}) - pode ser definida como:

$$v(SP^{\lambda}) = \min \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\, x_{ij}$$

sujeito a:

$$\sum_{j=1}^{n}\sum_{i=1}^{n} \lambda_j\, x_{ij} = \sum_{j=1}^{n} \lambda_j \qquad (4.3.5)$$

e (4.3.2), (4.3.3) e (4.4.4).

O valor ótimo $v(SP^{\lambda})$ é menor ou igual a v(P), e seu melhor valor pode resultar de um dual surrogate $\max_{\lambda \geq 0} v(SP^{\lambda})$. A função surrogate $s: R^n \rightarrow R$, $s(\lambda, v(SR^{\lambda}))$, possui algumas propriedades que dificultam encontrar a solução dual. Alguns métodos relatados na literatura procuram encontar uma solução aproximada do dual surrogate, como os métodos propostos por Dyer (1980) e Karwan and Rardin (1979). Observe que o problema (SP^{λ}) não pode ser resolvido facilmente, pois é um problema linear inteiro sem uma estrutura especial que possa ser explorada (Parker and Rardin, 1988).

unesp

Devido às dificuldades com a relaxação (SP^{λ}) propõe-se, neste trabalho, relaxar o problema novamente, agora no sentido lagrangeano. Para um dado $t \geq 0$, relaxa-se a restrição (4.3.5), obtendo a relaxação LS - denotada por (L_tSP^{λ}) - que é dada por:

$$v(L_tSP^{\lambda}) = \min \sum_{j=1}^{n} \sum_{i=1}^{n} (d_{ij} - t\lambda_j) + t \sum_{j=1}^{n} \lambda_j$$

sujeito a:

(4.3.2), (4.3.3) e (4.4.4).

Para um dado $t \geq 0$ e $\lambda \in R^n$, $v(L_tSP^{\lambda}) \leq v(SP^{\lambda}) \leq v(P)$. O problema ($L_tSP^{\lambda}$) é resolvido considerando a restrição (4.3.2) implicitamente, e sua decomposição para o índice i, o que leva aos seguintes n sub-problemas:

$$\min \sum_{j=1}^{n} (d_{ij} - t\lambda_j) x_{ij}$$

sujeito a:

(4.3.3) e (4.3.4).

Cada um destes n sub-problemas pode ser resolvido facilmente fazendo:

$$\beta_i = \sum_{j=1}^{n} \{ \min (0, d_{ij} - t\lambda_j) \},$$

e escolhendo I como o conjunto dos índices dos p menores β_i. Neste caso, a solução x_{ij}^{λ} para o problema (L_tSP^{λ}) é dada por: $x_{ii}^{\lambda} = 1$, se $i \in I$ ($x_{ii}^{\lambda} = 0$, caso contrário) e, para todo $i \neq j$, $x_{ij}^{\lambda} = 1$, se $i \in I$ e $d_{ij} - t\lambda_j < 0$ ($x_{ij}^{\lambda} = 0$, caso contrário).

Assim, a solução da relaxação LS é dada por:

$$v(L_tSP^{\lambda}) = \sum_{i=1}^{n} \beta_i \, x_{ii} + t \sum_{j=1}^{n} \lambda_j$$

Uma característica interessante da relaxação (L_tSP^{λ}) é que para t = 1 tem-se a relaxação lagrangeana usual, usando o multiplicador λ. Para um multiplicador λ fixo, o melhor valor de t pode resultar de (D_t^{λ}), o seguinte dual lagrangeano:

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

$$v(D_t^{\lambda}) = \max_{t \geq 0} v(L_t SP^{\lambda}).$$

É imediato que $v(SP^{\lambda}) \geq v(D_t^{\lambda}) \geq v(L_1 SP^{\lambda})$. É bem conhecido também que a função LS (lagrangeana/surrogate) $l : R+ \rightarrow R$, $l(t, v(L_t SP^{\lambda}))$, é côncava e linear por partes (Parker and Rardin, 1988). Uma solução exata para o problema D_t^{λ} (que fornece um limitante melhor do que o obtido pela relaxação lagrangeana usual) pode ser obtida por meio de uma busca sobre diferentes valores de t (ver Minoux, 1975 e Handler and Zang 1980). Contudo, em geral, existe um intervalo de valores $t_0 \leq t \leq t_1$ (com $t_0 = 1$ ou $t_1 = 1$) no qual tem-se limitantes melhores, como ilustra a Figura 4.1 (para o caso de $t_1 = 1$).

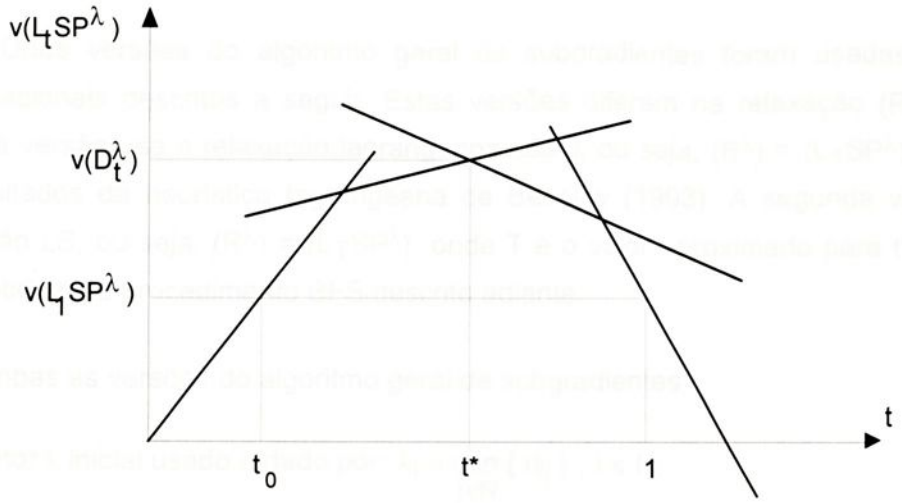


Fig. 4.1 - Limitantes na Função LS

Assim, para obter um limitante melhor do que o obtido pela relaxação lagrangeana usual não é necessário determinar o melhor valor t*, sendo suficiente encontrar um valor T tal que $t_0 \leq T \leq t_1$.

Apresenta-se, a seguir, o algoritmo geral de subgradientes que é usado como base para as heurísticas de relaxação propostas neste capítulo. Neste algoritmo os conjuntos O e C são definidos como: $O = \{ i \mid x_{ii} = 1 \}$ e $C = \{ i \mid x_{ii} = 0 \}$, ou seja, O é o conjunto dos nós já fixados como medianas e C é o conjunto dos nós já fixados como não-medianas.

Algoritmo Geral de Subgradientes

Dados $\lambda \geq 0$, $\lambda \neq 0$;
Fazer lb = -∞, ub = +∞, O = ∅, C = ∅;

Repetir

Resolver a relaxação (R^{λ}) obtendo x^{λ} e $v(R^{\lambda})$;

Obter a solução viável x^f e atualizar o valor v^f correspondente;

Atualizar lb = max [lb, $v(R^{\lambda})$];

Atualizar ub = min [ub, v^f];

Aplicar testes visando fixar variáveis, atualizando os conjuntos O e C;

Fazer $g_i^{\lambda} = 1 - \sum x_{ij}^{\lambda}$, $i \in N$;

Atualizar o tamanho do passo θ;

Fazer $\lambda_i = \lambda_i + \theta . g_i^{\lambda}$, $i \in N$;

Enquanto (as condições de parada não forem verificadas).

Duas versões do algoritmo geral de subgradientes foram usadas nos testes computacionais descritos a seguir. Estas versões diferem na relaxação (R^{λ}) usada. A primeira versão usa a relaxação lagrangeana usual, ou seja, $(R^{\lambda}) = (L_1SP^{\lambda})$, e reproduz os resultados da heurística lagrangeana de Beasley (1993). A segunda versão usa a relaxação LS, ou seja, $(R^{\lambda}) = (L_TSP^{\lambda})$, onde T é o valor aproximado para t* (ver Figura 4.1), obtido pelo procedimento BFS descrito adiante.

Para ambas as versões do algoritmo geral de subgradientes:

- o vetor λ inicial usado é dado por: $\lambda_i = \min_{j \in N} \{ d_{ij} \}$, $i \in N$;
- os tamanhos de passo usados são: $\theta = \pi$ (ub - lb) / $\| g^{\lambda} \|^2$, onde o controle do parâmetro π é como o proposto por Held and Karp (1971), ou seja, $0 \leq \pi \leq 2$, iniciando com $\pi = 2$ e dividindo π ao meio sempre que lb não aumenta por 30 iterações sucessivas;
- as condições de parada são:
  a) número de iterações maior do que 1000;
  b) $\pi \leq 0.005$;
  c) ub - lb < 1 ou $\left\| g^{\lambda} \right\|^2 = 0$ (solução ótima);
  d) todos as medianas foram fixadas;

A solução x^{λ} não é necessariamente viável para o problema (P), mas o conjunto I identifica quais nós devem ser considerados como medianas para produzir soluções viáveis para (P). Duas heurísticas são usadas para tornar x^{λ} primal viável. A primeira calcula o limite superior (ub) a cada iteração do algoritmo geral de subgradientes enquanto π não é dividido ao meio. Esta heurística faz simplesmente:

$$v^f = \sum_{j=1}^{n} (\min_{i \in I} d_{ij})$$

unesp

A segunda heurística, como sugerido por Beasley (1993), é uma heurística de troca que é usada todas as vezes em que π é atualizado para $\pi/2$. Considerando β_i ($i \in N$) ordenado em ordem crescente, esta heurística aplica o seguinte procedimento:

Heurística de Troca para P-Medianas

Fazer $U = \sum_{j=1}^{n} (\min_{i \in I} d_{ij})$ correspondente à solução x^{λ} associada com o maior limitante inferior (lb) já obtido.

Fazer m = max (25, n/10);
Fazer m = min (m, n-p);
Para j = p+1 até p+m ; $j \notin C$ fazer:
  Para i = 1 até p ; $i \notin O$ fazer:
    Intercambiar βi com βj, atualizando I correspondentemente;
    Calcular $v^f = \sum_{j=1}^{n} (\min_{i \in I} d_{ij})$;
    Se $v^f < U$ então fazer $U = v^f$; Do contrário, intercambiar β_i com β_j e atualizar I.
If U < ub então atualizar ub = U.

Para determinar o fator surrogate T (aproximação do valor ótimo t*) aplica-se o seguinte procedimento:

Algoritmo de Busca do Fator Surrogate (BFS) para P-Medianas

Dados S (tamanho de passo inicial) e M (número máximo de iterações), sejam t, s, n e z os valores atuais de: o fator surrogate, o tamanho do passo, o número de iterações e o melhor limitante inferior, respectivamente.

Fazer t = 0, s = S, z = 0, n = 0;
Repetir
  Atualizar t = t + s e n = n + 1;
  Se n > M então parar. Caso contrário, resolver (L_tSP^{λ}).
  Se $v(L_tSP^{\lambda}) > z$ então fazer:
    T = t e z = $v(L_tSP^{\lambda})$;
    Calcular a inclinação ω^{λ} da função LS como: $\omega^{\lambda} = \sum_{j=1}^{n} \lambda_j - \sum_{j=1}^{n} \sum_{i \in I} \lambda_j x_{ij}$;

Se $\omega^{\lambda} < 0$ então tentar melhorar o valor atual do fator surrogate resolvendo o problema ($L_{t-s/2}SP^{\lambda}$), atualizando T se necessário. Em seguida, parar.

Se $v(L_tSP^{\lambda}) < z$, fazer:

Tentar melhorar o fator surrogate atual resolvendo o problema ($L_{t-s/2}SP^{\lambda}$), atualizando T se necessário;

Se $n \leq \lceil M/2 \rceil$ então o tamanho inicial do passo é muito grande e deve ser reduzido à metade (S = S/2) para as próximas aplicações deste procedimento de busca;

Parar.

Enquanto (as condições de parada não forem verificadas).

Deste procedimento resulta o multiplicador T usado na relaxação LS. Entretanto, este procedimento não é usado durante todo o processo de solução. No caso desta busca produzir o mesmo multiplicador T para P aplicações consecutivas, então as próximas relaxações LS no processo de solução irá usar este valor T fixo e o procedimento de busca não será mais utilizado. Nos testes computacionais descritos a seguir foram usados os seguintes parâmetros: S = 0.5, M = 5, P = 10.

As heurísticas lagrangeana (L_1SP^{λ}) e LS (L_TSP^{λ}) para o problema das p-medianas (e também para os demais problemas, discutidos nas seções seguintes) foram programadas na linguagem C, compiladas sob o sistema operacional AIX, usando o compilador xlC com a opção de otimização -O2 e os testes computacionais foram realizados em uma estação de trabalho IBM RS/6000 modelo 3AT para problemas-teste obtidos da biblioteca OR-Library (Beasley, 1990), para os quais a solução ótima é conhecida. Os resultados são mostrados nas tabelas a seguir (todos os tempos computacionais mostrados nestas e nas próximas tabelas deste capítulo excluem o tempo necessário para a leitura de dados e o estabelecimento do problema).

As Tabelas 4.1 e 4.2 mostram os resultados obtidos pela relaxação lagrangeana usual e pela relaxação LS, respectivamente. Em cada uma destas tabelas mostra-se:

- D_UB, o desvio percentual do limitante superior encontrado (melhor solução viável) para a solução ótima, ou seja: 100*[ub - optimal] / optimal;
- D_LB, o desvio percentual da solução ótima para o limitante inferior encontrado (melhor valor da relaxação), ou seja: 100*[optimal - lb] / optimal;
- NRL, o número de relaxações lagrangeanas resolvidas. É importante observar que para a heurística lagrangeana (Tabela 4.1), o número de relaxações lagrangeanas resolvidas é também o número de iterações de subgradientes. Para a heurística LS (Tabela 4.2), entretanto, o número de relaxações lagrangeanas resolvidas inclui as relaxações resolvidas pelo procedimento BFS discutido acima, sendo portanto maior

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

do que o número de iterações de subgradientes correspondente.

- T_TOTAL, o tempo total de CPU (em segundos) gasto na solução.

Tabela 4.1 - Problema de P-Medianas: Resultados computacionais da heurística lagrangeana

Problema	n	p	D_UB	D_LB	NRL	T_TOTAL
1	100	5	-	-	337	0.93
2		10	-	0.124	417	1.34
3		10	-	0.241	555	1.79
4		20	-	-	331	1.58
5		33	-	-	280	2.07
6	200	5	-	0.525	468	4.75
7		10	-	-	454	5.45
8		20	-	-	285	5.03
9		40	-	-	328	10.76
10		67	-	-	387	17.04
11	300	5	-	0.055	460	10.10
12		10	-	0.134	438	11.67
13		30	-	-	327	16.38
14		60	-	0.037	454	45.85
15		100	-	-	385	65.30
16	400	5	0.012	0.871	496	19.11
17		10	-	0.447	536	24.00
18		40	-	0.025	473	58.01
19		80	-	-	354	102.93
20		133	-	-	314	181.77
21	500	5	-	-	362	20.60
22		10	0.058	0.433	507	35.45
23		50	-	-	401	100.43
24		100	-	-	373	233.14
25		167	-	-	373	436.51
26	600	5	-	0.659	440	37.34
27		10	0.036	0.090	488	48.65
28		60	-	-	448	224.90
29		120	-	-	311	454.03
30		200	-	-	411	1120.69
31	700	5	-	0.609	499	56.60
32		10	-	0.089	456	61.84
33		70	-	0.028	511	429.77
34		140	-	-	464	952.05
35	800	5	-	0.959	488	72.37
36		10	-	1.042	464	83.33
37		80	-	-	538	726.09
38	900	5	0.099	1.043	561	104.60
39		10	-	0.652	474	107.97
40		90	0.039	0.011	439	1091.27



Tabela 4.2 - Problema de P-Medianas: Resultados computacionais da heurística LS

Problema	n	p	D_UB	D_LB	NRL	T_TOTAL
1	100	5	-	-	259	0.76
2		10	0.293	0.113	444	1.57
3		10	-	0.228	421	1.47
4		20	-	-	237	1.18
5		33	-	-	147	1.14
6	200	5	-	0.523	540	5.61
7		10	-	-	456	5.87
8		20	-	-	313	5.30
9		40	-	-	344	9.98
10		67	-	-	293	12.33
11	300	5	-	0.046	551	12.94
12		10	-	0.131	528	14.43
13		30	-	-	361	15.74
14		60	-	-	353	37.19
15		100	-	-	359	57.73
16	400	5	-	0.866	442	18.48
17		10	-	0.440	503	24.23
18		40	-	-	380	46.62
19		80	-	-	358	79.02
20		133	-	-	454	231.51
21	500	5	-	-	442	27.16
22		10	0.105	0.426	508	37.39
23		50	-	-	281	73.37
24		100	-	-	347	174.65
25		167	-	-	330	377.14
26	600	5	-	0.652	540	49.47
27		10	-	0.082	596	61.34
28		60	-	-	371	169.93
29		120	-	-	273	342.60
30		200	-	-	346	879.95
31	700	5	-	0.604	506	63.06
32		10	-	0.079	546	75.52
33		70	-	-	416	382.35
34		140	-	-	405	730.48
35	800	5	-	0.956	577	89.47
36		10	0.070	1.036	525	96.41
37		80	0.040	0.009	413	705.65
38	900	5	0.099	1.039	446	91.52
39		10	-	0.643	520	123.28
40		90	-	-	422	1090.27

As Tabelas 4.3 e 4.4 mostram o tempo computacional necessário para estas heurísticas alcançarem soluções quase-ótimas (em relação ao melhor valor da relaxação), ou seja, para alcançarem desvios percentuais D_LB pequenos. Cada uma das tabelas a seguir mostra os tempos necessários (em segundos) para atingir desvios

unesp

percentuais D_LB que correspondem a 9%, 7%, 5%, 3% e 1%.

Tabela 4.3 - Problema das P-Medianas: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação lagrangeana

			D_LB				
Problema	n	p	9%	7%	5%	3%	1%
1	100	5	0.26	0.28	0.34	0.48	0.59
2		10	0.16	0.17	0.29	0.30	0.52
3		10	0.47	0.59	0.61	0.80	0.88
4		20	0.60	0.67	0.84	0.84	1.07
5		33	0.66	0.96	0.96	0.98	1.36
6	200	5	0.50	0.53	0.62	1.26	2.22
7		10	1.45	1.45	1.50	1.93	2.44
8		20	2.04	2.15	2.96	2.98	3.47
9		40	3.62	5.03	5.03	5.07	6.56
10		67	8.74	8.76	8.79	9.11	11.56
11	300	5	1.72	1.87	2.27	3.00	4.64
12		10	1.24	1.31	3.11	3.15	4.30
13		30	6.08	6.18	8.62	8.64	11.37
14		60	16.70	16.70	16.76	16.91	22.81
15		100	26.98	26.98	27.26	37.07	39.19
16	400	5	1.73	1.80	2.02	5.46	13.50
17		10	4.24	5.45	8.42	8.57	12.42
18		40	12.65	12.69	12.87	20.29	28.04
19		80	45.53	45.63	45.63	46.46	60.38
20		133	78.71	78.77	78.89	104.09	131.03
21	500	5	2.66	2.71	2.83	3.22	5.81
22		10	3.23	3.41	3.75	7.08	18.04
23		50	28.51	28.58	28.71	42.06	57.81
24		100	101.61	101.69	101.77	101.93	136.01
25		167	184.86	184.86	188.34	249.60	312.61
26	600	5	4.04	4.20	4.68	8.41	18.00
27		10	4.89	5.38	5.70	9.99	17.36
28		60	56.21	56.29	56.49	86.31	114.60
29		120	196.58	196.81	196.81	199.46	265.83
30		200	376.40	376.40	376.67	506.33	629.57
31	700	5	5.70	5.80	6.34	15.74	24.62
32		10	6.90	7.23	15.44	15.76	27.23
33		70	99.90	100.57	152.43	152.57	204.61
34		140	361.46	361.62	361.93	363.75	490.16
35	800	5	6.71	7.12	7.67	17.11	59.62
36		10	1.08	18.71	19.28	22.09	⊗
37		80	1.16	86.66	87.00	167.08	245.05
38	900	5	1.54	2.56	13.80	20.99	⊗
39		10	2.58	2.58	14.96	25.13	60.60
40		90	1.45	127.90	128.34	251.29	376.58

⊗ significa "não disponível"

Tabela 4.4 - Problema das P-Medianas: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação LS

			D_LB				
Problema	n	p	9%	7%	5%	3%	1%
1	100	5	0.08	0.09	0.14	0.20	0.35
2		10	0.43	0.43	0.48	0.68	0.71
3		10	0.20	0.21	0.22	0.38	0.60
4		20	0.12	0.12	0.20	0.39	0.57
5		33	0.40	0.41	0.41	0.42	0.74
6	200	5	0.80	1.51	1.52	1.62	2.73
7		10	1.17	1.17	1.19	1.40	2.28
8		20	1.28	1.34	1.41	2.37	2.65
9		40	2.63	2.64	2.65	2.68	4.20
10		67	3.03	3.03	3.05	3.13	5.70
11	300	5	2.69	2.76	2.95	4.14	4.61
12		10	3.09	3.14	3.23	4.23	4.37
13		30	0.76	1.04	1.62	3.82	4.20
14		60	7.97	7.97	8.02	8.11	13.28
15		100	10.36	10.36	10.43	11.13	20.68
16	400	5	1.72	4.19	4.23	4.34	10.86
17		10	3.23	3.27	3.34	3.57	7.66
18		40	7.83	7.91	8.58	15.82	22.33
19		80	4.29	19.55	19.60	19.91	36.73
20		133	30.72	30.72	30.90	56.23	57.07
21	500	5	9.11	9.17	9.34	9.73	14.34
22		10	7.44	7.56	7.79	11.99	16.96
23		50	17.46	17.46	30.08	30.21	45.85
24		100	38.89	38.89	39.05	39.58	75.28
25		167	65.05	65.15	65.34	127.29	188.46
26	600	5	6.84	7.00	7.16	10.23	21.57
27		10	10.94	11.02	11.26	11.51	15.86
28		60	28.81	29.01	56.67	56.77	58.05
29		120	7.12	75.04	75.15	75.27	141.68
30		200	137.89	137.89	138.17	142.77	269.44
31	700	5	11.17	11.49	12.24	16.82	24.59
32		10	14.88	14.99	15.21	15.54	25.78
33		70	55.46	56.66	102.84	103.37	157.03
34		140	21.01	137.81	137.97	139.56	258.80
35	800	5	0.71	0.99	6.30	7.69	59.65
36		10	0.72	7.41	7.83	19.98	⊗
37		80	1.22	91.31	171.75	250.21	334.98
38	900	5	1.59	13.48	13.65	25.66	⊗
39		10	1.27	9.74	10.64	22.02	48.72
40		90	1.54	122.28	247.35	370.72	490.78

⊗ significa "não disponível"



Das Tabelas 4.1 e 4.2 pode-se ver que a heurística LS alcança os mesmos bons resultados que a heurística lagrangeana usual. Entretanto, os desvios percentuais médios obtidos pela heurística LS são menores que os obtidos pela heurística lagrangeana usual. Em termos de tempos computacionais, pode-se notar das Tabelas 4.3 e 4.4 que a heurística LS obtém soluções quase-ótimas mais rapidamente do que a heurística lagrangeana. Além disto, pode-se notar das Tabelas 4.1 e 4.2 que, para um dado número de nós (n), os problemas que demandam os maiores tempos de processamento correspondem a p = n/3 e, nestes casos, o desempenho computacional da heurística LS é significativamente superior ao da heurística lagrangeana usual (ver Tabela 4.5).

Para comparar o desempenho computacional destas heurísticas mostra-se a seguir (ver Figura 4.2) um gráfico onde estão plotados os valores de $v(L_tSP^{\lambda})$ obtidos a cada iteração pela heurística lagrangeana e pela heurística LS para o problema 30 (o que demandou mais tempo de processamento nos testes acima). Pode-se observar que a sequência de relaxações LS é mais estável que as correpondentes relaxações lagrangeanas.

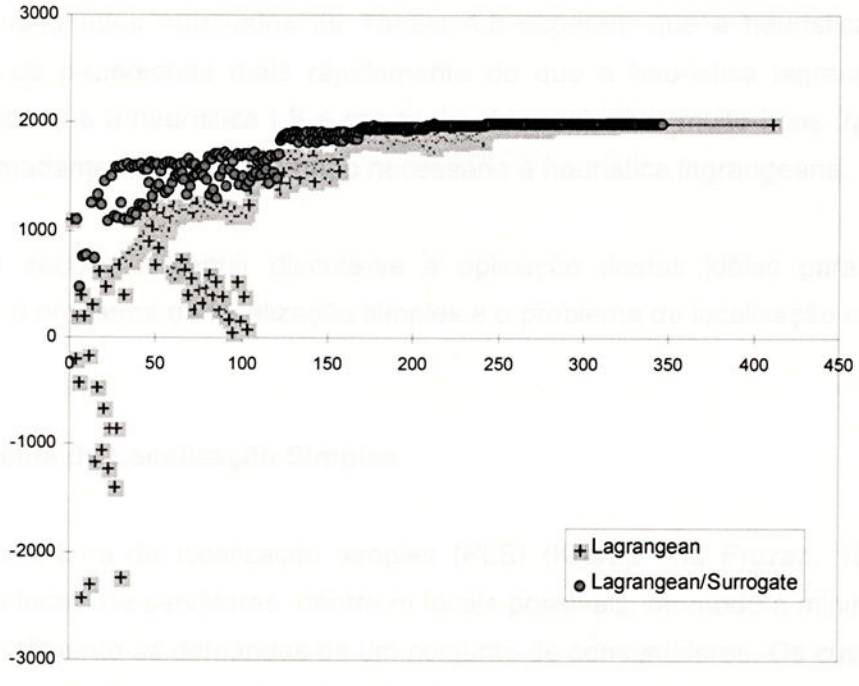


Fig. 4.2 - Comportamento Computacional de Heurísticas

A tabela a seguir mostra as razões de tempo de processamento necessárias para a obtenção de soluções quase-ótimas e de tempo de processamento total para os problemas correspondentes a p = n/3 para cada uma das classes de problemas. Para

esta tabela incluiu-se os problemas (n = 700, p = 233), (n = 800, p = 267) e (n = 900, p = 300) não disponíveis na biblioteca OR-Library. A Tabela 4.5 mostra as razões (tempo para a heurística LS)/(tempo para a heurística lagrangeana).

Tabela 4.5 - Problema das P-Medianas: Razões de tempo para as heurísticas LS e lagrangeana

			D_LB					
n	p	Ótimo	9%	7%	5%	3%	1%	T_TOTAL
100	33	1355	0.66	0.45	0.45	0.45	0.55	0.50
200	67	1255	0.40	0.41	0.41	0.71	0.57	0.72
300	100	1729	0.44	0.44	0.44	0.35	0.57	0.87
400	133	1789	0.39	0.39	0.39	0.54	0.43	1.27
500	167	1828	0.38	0.38	0.38	0.53	0.62	0.86
600	200	1989	0.36	0.36	0.36	0.51	0.61	0.78
700	233	1847	0.34	0.34	0.34	0.26	0.51	0.63
800	267	2035	0.34	0.34	0.34	0.35	0.51	0.63
900	300	2106	0.34	0.34	0.34	0.34	0.50	1.00
Valores médios			0.41	0.38	0.38	0.45	0.54	0.81

Os resultados mostrados na Tabela 4.5 sugerem que a heurística LS resolve problemas de p-medianas mais rapidamente do que a heurística lagrangeana usual. Sugere ainda que a heurística LS é capaz de obter soluções muito boas (quase-ótimas) em, aproximadamente, 43% do tempo necessário à heurística lagrangeana.

Nas seções a seguir discute-se a aplicação destas idéias para dois outros problemas: o problema de localização simples e o problema de localização capacitado.

4.4 - Problema de Localização Simples

O problema de localização simples (PLS) (Krarup and Pruzan, 1983) objetiva escolher os locais de servidores, dentre m locais possíveis, de modo a minimizar o custo total de atendimento às demandas de um conjunto de consumidores. Os custos, no caso, incluem um custo fixo para a localização de cada servidor e um custo de atendimento aos consumidores. O termo simples, no caso, refere-se à ausência de restrições de capacidade dos servidores. Um exemplo de PLS é dado pelo seguinte problema: deseja-se estabelecer locais para a instalação de fábricas de um determinado produto para o atendimento das demandas de n centros de distribuição deste produto, de modo a minimizar os custos de produção em cada fábrica e os custos de transporte das fábricas

para os centros de distribuição.

É interessante observar para o PLS que, como não existe restrição de capacidade dos servidores, na situação ótima cada consumidor será inteiramente atendido pelo servidor melhor localizado, ou seja, o de menor custo associado. Isto pode ser importante em problemas práticos como, por exemplo, quando deseja-se que cada centro de distribuição seja atendido por uma única fábrica, a fim de garantir um bom serviço de atendimento. A suposição de um único produto pode também não ser restritiva, se vários produtos compartilharem recursos de manufatura e distribuição, ou seja, quando o produto único estiver referenciando um conjunto padrão de produtos.

O PLS é bem conhecido com sendo NP-difícil. Vários algoritmos têm sido desenvolvidos e aplicados com sucesso ao PLS. Alguns destes algoritmos aparecem nos trabalhos de Erlenkotter (1978), Guinard (1988), Galvão and Raggi (1989), Korkel (1989) e Beasley (1993).

Nesta seção apresenta-se uma nova abordagem para o PLS, baseada, como já discutido na seção anterior, na combinação das bem conhecidas relaxações lagrangeana e surrogate para o problema. Esta nova abordagem - a heurística LS - consegue resultados tão bons (e muitas vezes, melhores) quanto os obtidos pela heurística lagrangeana tradicional, mas em menor tempo computacional, principalmente para problemas de grande porte. Nesta seção discute-se esta nova abordagem e apresenta-se vários testes computacionais realizados com problemas disponíveis na literatura, com os quais pretende-se demonstrar a efetividade desta nova abordagem.

O problema de localização simples considerado nesta seção pode ser modelado como o seguinte problema (P) de programação inteira binária:

$$v(P) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{i=1}^{m} F_i\, y_i$$

sujeito a:

$$\sum_{i=1}^{m} x_{ij} = 1 \quad ; \quad j \in N = \{ 1, 2, \ldots, n \} \tag{4.4.1}$$

$$x_{ij} \le y_i \quad ; \quad i \in M = \{ 1, 2, .., m \}, j \in N \tag{4.4.2}$$

$$x_{ij}\, , y_i \in \{ 0, 1 \} \quad ; \quad i \in M, j \in N \tag{4.4.3}$$

unesp

onde:

$[c_{ij}]_{m \times n}$ é uma matriz simétrica de custos de transporte;

$[x_{ij}]_{m \times n}$ é a matriz de alocação, onde $x_{ij} = 1$ se a demanda do consumidor j é satisfeita pelo servidor i; $x_{ij} = 0$, caso contrário;

$[y_i]_m$ é o vetor indicando servidores abertos ou fechados, com $y_i = 1$ se o servidor i está aberto; $y_i = 0$, caso contrário;

$[F_i]_m$ é o vetor de custos fixos para a abertura de servidores;

m é o número de locais potenciais para os servidores e n é o número de consumidores.

Neste caso, para um dado $\lambda \in R^n$, uma relaxação surrogate de (P) - denotada por (SP^{λ}) - pode ser definida por:

$$v(SP^{\lambda}) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{i=1}^{m} F_i\, y_i$$

sujeito a:

$$\sum_{j=1}^{n} \sum_{i=1}^{m} \lambda_j\, x_{ij} = \sum_{j=1}^{n} \lambda_j \qquad (4.4.4)$$

$$x_{ij} \le y_i \quad ; \quad i \in M = \{ 1, 2, .., m \}, j \in N$$

$$x_{ij}\, , y_i \in \{ 0, 1 \} \quad ; \quad i \in M, j \in N$$

Para um dado $t \ge 0$ a restrição (4.4.4) pode, então, ser relaxada, resultando na relaxação LS - denotada por L_tSP^{λ} - que é dada por:

$$v(L_tSP^{\lambda}) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{i=1}^{m} F_i\, y_i + t \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{m} x_{ij} - 1 \right)$$

sujeito a (4.4.2) e (4.4.3).

O problema L_tSP^{λ} pode ser resolvido facilmente observando-se que ao fazer $y_i = 1$ (ou seja, ao considerar o servidor i como aberto) a contribuição à função objetivo é dada por:

$$\gamma_i = F_i + \sum_{j=1}^{n} \min \left[0, (c_{ij} + t\, \lambda_j) \right], \quad i \in M$$

O conjunto I de servidores abertos é, portanto, definido como $I = \{ i \in M \mid \gamma_i \leq 0 \}$. As variáveis de alocação são, então, facilmente determinadas, fazendo $x_{ij} = 1$ se $(c_{ij} + t\,\lambda_j) \leq 0$ e $y_i = 1$. Do contrário, $x_{ij} = 0$.

Como já observado para o problema das p-medianas, para um dado $t \geq 0$ e um dado $\lambda \in R^n$, tem-se que $v(L_t SP^\lambda) \leq v(SP^\lambda) \leq v(P)$, sendo que para t = 1, a relaxação $(L_t SP^\lambda)$ corresponde a relaxação lagrangeana usual para o multiplicador λ. Fixado o multiplicador λ, o melhor valor de t pode ser obtido do dual lagrangeano $v(D_t^\lambda) = \max_{t \geq 0} v(L_t SP^\lambda)$, sendo imediato que $v(D_t^\lambda) \geq v(L_1 SP^\lambda)$.

Embora, como já comentado anteriormente, o valor exato t* de $v(D_t^\lambda)$ possa ser obtido por um processo de busca, em geral, existe um intervalo de valores $t_0 \leq t^* \leq t_1$ com $t_0 = 1$ ou $t_1 = 1$, no qual é possível obter bons limitantes (ver Figura 4.1). Assim, é possível determinar, por um processo de busca, um valor T tal que a seguinte sequência de desigualdades seja válida:

$$v(L_1 SP^\lambda) \leq v(L_T SP^\lambda) \leq v(D_t^\lambda) \leq v(SP^\lambda) \leq v(P)$$

Para uma sequência de diferentes multiplicadores λ, tem-se $v(L_T SP^\lambda)$ como os melhores valores a serem usados no algoritmo geral de subgradientes discutido anteriormente (ver Seção 4.3).

Deve-se observar que, no caso específico do PLS, são considerados os seguintes pontos no algoritmo geral de subgradientes:

- o valor inicial de λ é tal que: $\lambda_j = \min_{j \in N} \{ c_{ij} \}$, $i \in M$.
- os conjuntos O e C são definidos como $O = \{ i \in M \mid y_i = 1 \}$ e $C = \{ i \in M \mid y_i = 0 \}$. Para cada λ, fixa-se uma variável y_i ($i \in M$) ao valor α ($\alpha \in \{0, 1\}$) se: $v(L_T SP^\lambda \mid y_i = 1 - \alpha) \geq ub$.
- as condições de parada são:
  a) $\pi \leq 0.005$;
  b) $ub - lb < 0.01$;
  c) $\left\| g^\lambda \right\|^2 = 0$;
  d) todos os servidores foram fixados;
  e) número de iterações maior do que 1000.

Para a maioria das iterações a solução x^{λ} é tornada viável conservando-se os servidores abertos e alterando-se as alocações $x^{f}_{ij^*} = 1$ se $i \in I$, $j^* \in N$ e $c_{ij^*} = \min \{ cij \}$.

Entretanto, quando o parâmetro π é dividido ao meio, uma heurística de troca é usada para procurar por soluções viáveis melhores. Esta heurística de troca é dada pelo seguinte algoritmo:

Heurística de Troca para o PLS

Seja o vetor γ ordenado segundo valores crescentes de γ_i ($i \in M$) e sejam dados $q = v^f$ (valor da solução viável x^f atual) e $p = |I|$ (número de servidores abertos).

Fazer k = max (25, m/10);
Fazer k = min (k, m);
Para j = p+1 até p+k ; $j \notin C$ fazer:
  Para i = 1 até p ; $i \notin O$ fazer:
    Considerar uma das seguintes alternativas, atualizando I correspondentemente:
    a) Excluir γ_i do conjunto de servidores abertos;
    b) Incluir γ_j no conjunto de servidores abertos;
    c) Intercambiar γ_i com γ_j;
    Calcular $v^f = \sum_{j=1}^{n} (\min_{i \in I} c_{ij})$;
    Se $v^f < q$ então fazer $q = v^f$. Caso contrário, desfazer a alternativa selecionada e atualizar I.
Se q < ub então fazer ub = q.

A heurística de busca do fator surrogate (T), usado na solução do problema ($L_T SP^{\lambda}$), usado para o problema de localização simples é dado pelo seguinte algoritmo:

Busca do Fator Surrogate para o PLS

Dados s = 0.50 (tamanho inicial do passo) e max_it = 5 (número máximo de iterações), fazer:
  t = 0.0;
  z = 0.0,
  k = 0;
  Repetir
    t = t + s;
    k = k + 1;

Se k > max_it então parar. Caso contrário, resolver $(L_t SP^{\lambda})$.

Se $v(L_t SP^{\lambda}) > z$ então fazer o seguinte:

Fazer T = t ;

Fazer $z = v(L_t SP^{\lambda})$;

Calcular a inclinação ω^{λ} da função LS como: $\omega^{\lambda} = \sum_{j=1}^{n} \lambda_j - \sum_{j=1}^{n} \sum_{i \in I} \lambda_j x_{ij}$

Se $\omega^{\lambda} < 0$ então tentar melhorar o fator surrogate atual resolvendo o problema $(L_{t-s/2} SP^{\lambda})$, atualizando T = t - s/2, se necessário, e, em seguida, parar.

Caso contrário, isto é, se $\omega^{\lambda} > 0$:

Tentar melhorar o fator surrogate atual resolvendo o problema $(L_{t-s/2} SP^{\lambda})$, atualizando T = t - s/2, se necessário.

Se $k < \lceil max\_it / 2 \rceil$ então o passo inicial é muito grande e deve ser dividido ao meio para as próximas aplicações do algoritmo. Neste caso, fazer s = s/2.

Parar.

Enquanto (as condições de parada não forem verificadas).

Como no caso do problema das p-medianas, o algoritmo de busca do fator surrogate não é utilizado durante todo o processo de solução. Para o PLS, se esta busca resultar no mesmo fator T para 3 aplicações consegutivas do algoritmo, então este fator torna-se fixo para os demais problemas $(L_T SP^{\lambda})$.

Os resultados dos testes computacionais são mostrados nas tabelas a seguir, onde D_UB, D_LB, NRL e T_TOTAL são como definidos anteriormente, NSA corresponde ao número de servidores abertos e NSI, o número de servidores que ainda ficaram indecididos (se abertos ou não) ao final da aplicação da heurística. Nestas tabelas, os problemas estão divididos em duas classes: a dos problemas pequenos, com instâncias de tamanhos (n = 50, m = 16), (n = 50, m = 25) e (n = 50, m = 50); e a dos problemas grandes, com instâncias de tamanho (n = 1000, m = 100).

Nas tabelas 4.6 e 4.7, a seguir, mostram-se os resultados obtidos pelas heurísticas lagrangeana usual e LS, respectivamente.

Para evitar o efeito de condições de parada espúrias e demonstrar que as sequências de resultados da heurística LS são mais estáveis e obtidas mais rapidamente do que as da heurística lagrangeana usual mostra-se, nas Tabelas 4.8 a 4.11, o tempo computacional necessário (em segundos) para estas heurísticas alcançarem soluções quase-ótimas, da forma como definido anteriormente, ou seja, para alcançarem desvios percentuais D_LB bem pequenos. Em cada uma destas tabelas a seguir mostram-se os

unesp

tempos que estas heurísticas gastam para atingir desvios D_LB que correspondem a 0.5%, 0.4%, 0.3%, 0.2% e 0.1%, para os problemas pequenos, e a 5%, 4%, 3%, 2% e 1%, para os problemas grandes.

Tabela 4.6 - Problema de Localização Simples: Resultados computacionais da heurística lagrangeana

Problema	n	m	D_UB	D_LB	NRL	NSA	NSI	T_TOTAL
7-1	50	16	-	-	381	11	-	0.26
7-2			-	-	265	9	-	0.18
7-3			-	-	323	5	-	0.21
7-4			-	-	202	4	-	0.12
10-1	50	25	-	-	279	15	-	0.49
10-2			-	-	260	11	-	0.37
10-3			-	-	304	8	-	0.37
10-4			-	-	302	4	-	0.29
13-1	50	50	-	-	297	15	-	1.16
13-2			-	-	393	11	-	1.13
13-3			-	-	384	8	-	0.82
13-4			-	-	336	4	-	0.57
A	1000	100	-	-	102	4	-	8.24
B			-	-	315	7	-	25.43
C			0.033	0.057	603	1	21	64.17

Tabela 4.7 - Problema de Localização Simples: Resultados computacionais da heurística LS

Problema	n	m	D_UB	D_LB	NRL	NSA	NSI	T_TOTAL
7-1	50	16	-	-	245	11	-	0.18
7-2			-	-	234	9	-	0.17
7-3			-	-	466	5	-	0.25
7-4			-	-	234	4	-	0.12
10-1	50	25	-	-	281	15	-	0.34
10-2			-	-	262	11	-	0.30
10-3			-	-	340	8	-	0.35
10-4			-	-	284	4	-	0.22
13-1	50	50	-	-	313	15	-	0.84
13-2			-	-	343	11	-	0.70
13-3			-	-	440	8	-	0.84
13-4			-	-	332	4	-	0.52
A	1000	100	-	-	100	4	-	6.80
B			-	-	187	7	-	16.63
C			0.033	0.051	578	1	21	64.71

unesp

Tabela 4.8 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação lagrangeana para problemas pequenos

			D_LB				
Problema	n	m	0.5%	0.4%	0.3%	0.2%	0.1%
7-1	50	16	0.08	0.10	0.11	0.12	0.12
7-2			0.12	0.15	0.15	0.15	0.18
7-3			0.13	0.13	0.13	0.14	0.17
7-4			0.09	0.09	0.10	0.10	0.10
10-1	50	25	0.28	0.37	0.37	0.44	0.44
10-2			0.21	0.22	0.22	0.33	0.35
10-3			0.23	0.23	0.23	0.23	0.27
10-4			0.16	0.20	0.20	0.20	0.22
13-1	50	50	0.85	0.85	0.85	0.91	1.07
13-2			0.62	0.79	0.80	0.83	1.00
13-3			0.46	0.51	0.51	0.63	0.70
13-4			0.27	0.27	0.27	0.38	0.40

Tabela 4.9 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação LS para problemas pequenos

			D_LB				
Problema	n	m	0.5%	0.4%	0.3%	0.2%	0.1%
7-1	50	16	0.11	0.12	0.12	0.12	0.15
7-2			0.12	0.12	0.12	0.13	0.16
7-3			0.14	0.15	0.16	0.17	0.19
7-4			0.11	0.11	0.11	0.11	0.11
10-1	50	25	0.24	0.24	0.25	0.25	0.25
10-2			0.23	0.23	0.23	0.23	0.23
10-3			0.17	0.18	0.18	0.22	0.33
10-4			0.14	0.14	0.15	0.15	0.17
13-1	50	50	0.79	0.79	0.80	0.80	0.80
13-2			0.50	0.51	0.51	0.51	0.52
13-3			0.47	0.48	0.49	0.50	0.54
13-4			0.34	0.34	0.35	0.35	0.41

Tabela 4.10 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação lagrangeana para problemas grandes

			D_LB				
Problema	n	m	5%	4%	3%	2%	1%
A	1000	100	5.32	5.32	5.32	6.17	6.17
B			7.08	7.08	7.70	7.95	11.41
C			7.13	7.13	9.63	9.92	19.16



Tabela 4.11 - Problema de Localização Simples: Tempos de CPU para soluções quase-ótimas obtidas pela relaxação LS para problemas grandes

			D_LB				
Problema	n	m	5%	4%	3%	2%	1%
A	1000	100	1.00	1.01	1.83	1.83	2.32
B			1.83	2.42	2.42	2.42	8.70
C			2.52	2.52	2.52	7.05	7.24

Examinando os resultados mostrados nas tabelas acima pode-se concluir que a heurística LS é capaz de resolver problemas de localização simples, de forma aproximada, mais rapidamente do que a heurística lagrangeana usual. Os resultados são mais evidentes para problemas grandes. Para facilitar esta análise, mostra-se nas Tabelas 4.12 e 4.13, a seguir, as razões (tempo para a heurística LS) / (tempo para a heurística lagrangeana) para cada classe de problemas.

Tabela 4.12 - Problema de Localização Simples: Razões de tempo para as heurísticas LS e lagrangeana na solução de pequenas instâncias

	Tempo para soluções quase-ótimas (D_LB)						Tempo Total	
Problema	0.5%	0.4%	0.3%	0.2%	0.1%	Média	Razão	Média
7-1	1.38	1.20	1.09	1.00	1.25		0.69	
7-2	1.00	0.80	0.80	0.87	0.89		0.94	
7-3	1.08	1.15	1.23	1.21	1.12		1.19	
7-4	1.22	1.22	1.10	1.10	1.10		1.00	
10-1	0.86	0.65	0.68	0.57	0.57		0.69	
10-2	1.10	1.05	1.05	0.70	0.66		0.81	
10-3	0.74	0.78	0.78	0.96	1.22		0.95	
10-4	0.88	0.70	0.75	0.75	0.77		0.76	
13-1	0.93	0.93	0.94	0.88	0.75		0.72	
13-2	0.81	0.65	0.64	0.61	0.52		0.62	
13-2	1.02	0.94	0.96	0.79	0.77		1.02	
13-4	1.26	1.26	1.30	0.92	1.03	0.93	0.91	0.86

Tabela 4.13 - Problema de Localização Simples: Razões de tempo para as heurísticas LS e lagrangeana na solução de grandes instâncias

	Tempo para soluções quase-ótimas (D_LB)						Tempo Total	
Problema	5%	4%	3%	2%	1%	Média	Razão	Média
A	0.19	0.19	0.34	0.30	0.38		0.83	
B	0.26	0.34	0.31	0.30	0.76		0.65	
C	0.35	0.35	0.26	0.71	0.38	0.36	1.01	0.83

unesp

4.5 - Problema de Localização Capacitado

O problema de localização discutido na seção anterior torna-se consideravelmente mais complexo quando existem restrições sobre a demanda total que pode ser atendida em cada servidor. O problema de localização capacitado (PLC) é também conhecido como sendo NP-difícil. Este problema, e suas diversas variações, tem sido muito estudado, existindo muitos resultados relatados na literatura. Cornuejols *et al.* (1991) apresentam uma comparação computacional e teórica de várias relaxações propostas na literatura, cobrindo as principais formulações matemáticas para o problema. Uma revisão do problema pode ser encontrada em Domschke and Drexl (1985). Algoritmos exatos e heurísticas para o PLC aparecem em Nauss (1978), Baker (1982), Chistofides and Beasley (1983), Guignard-Spielberg and Kim (1983), Jacobsen (1983), Beasley (1988), Barceló *et al.* (1991) e Beasley (1993).

Nesta seção apresenta-se a heurística LS para o PLC, baseada, como já discutido anteriormente, na combinação das relaxações lagrangeana e surrogate para o problema. Discute-se aqui esta nova abordagem e também os testes computacionais realizados com problemas disponíveis na literatura, com os quais pretende-se demonstrar a efetividade desta nova abordagem.

O problema de localização capacitado discutido nesta seção é modelado como o seguinte problema de programação inteira mista:

$$v(P) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{i=1}^{m} F_i\, y_i$$

sujeito a:

$$\sum_{i=1}^{m} x_{ij} = 1 \quad ; \quad j \in N = \{1, \ldots, n\}$$

$$\sum_{j=1}^{n} d_j\, x_{ij} \le s_i\, y_i \quad ; \quad i \in M = \{1, \ldots, m\}$$

$$x_{ij} \le y_i \quad ; \quad i \in M, j \in N$$

$$\sum_{i=1}^{m} s_i\, y_i \ge \sum_{j=1}^{n} d_j$$

$$0 \le x_{ij} \le 1, \; y_i \in \{0, 1\} \quad ; \quad i \in M, j \in N$$

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

onde:

m, n, $[c_{ij}]_{m \times n}$, $[y_i]_m$ e $[F_i]_m$ são como para o problema de localização simples, discutido na seção anterior;

$[x_{ij}]_{m \times n}$ é a matriz de alocação, onde x_{ij} é a fração de demanda do cliente j atendida pelo servidor i;

$[s_i]_m$ é o vetor de capacidades dos servidores (ou seja, o limite superior da demanda total que pode ser suprida pelo servidor i);

$[d_j]_n$ é o vetor de demandas dos clientes;

Deve-se observar que a restrição $\sum_{i=1}^{m} s_i\, y_i \geq \sum_{j=1}^{n} d_j$ é redundante para o problema primal, mas leva a limites melhores para as relaxações (Cornuejols *et al.*, 1991). Da mesma forma como discutido para o PLS (Seção 4.4), dados $\lambda \in R^n$ e $t \geq 0$, pode-se estabelecer a relaxação LS $(L_t SP^{\lambda})$ para este problema. No caso do PLC, entretanto, a solução de $(L_t SP^{\lambda})$ é obtida separando o problema em m sub-problemas $(L_t SP^{\lambda})_i$, com $i \in M$, definidos como:

$$v(L_t SP^{\lambda})_i = \min \sum_{j=1}^{n} (c_{ij} + t\lambda_j)\, x_{ij} + F_i\, y_i$$

sujeito a:

$$\sum_{j=1}^{n} d_j\, x_{ij} \leq s_i\, y_i$$

$$x_{ij} \leq y_i \quad ; \quad j \in N$$

$$0 \leq x_{ij} \leq 1,\ y_i \in \{0, 1\} \quad ; \quad j \in N$$

Fazendo $y_i = 1$, cada problema $(L_t SP^{\lambda})_i$ é um problema da mochila de restrição contínua (correspondente ao benefício de abrir um servidor), que pode ser resolvido em tempo linear (Martello and Toth, 1990). Após resolver os m subproblemas, seja $z_i = v(L_t SP^{\lambda})_i$. Então, o conjunto de servidores a serem abertos pode ser determinado resolvendo-se o seguinte problema da mochila 0-1 (Horowitz and Sahni, 1974):

$$v(L_t SP^{\lambda}) = \min \sum_{i=1}^{m} z_i\, y_i - t \sum_{j=1}^{n} \lambda_j$$

sujeito a:

$$\sum_{i=1}^{m} s_i \, y_i \geq \sum_{j=1}^{n} d_j$$

$$y_i \in \{0,1\} \quad ; \quad i \in M$$

Uma vez determinado o conjunto I de servidores abertos, a alocação de clientes aos servidores é feita da mesma forma que para o PLS, ou seja, fazendo $x_{ij} = 1$ se $(c_{ij} + t\lambda_j) \leq 0$ e $i \in I$ (do contrário, $x_{ij} = 0$).

No caso do PLC, o algoritmo geral de subgradientes é aplicado considerando um limite de 2000 iterações e com um novo controle de passo. Para este problema, o parâmetro π começa com o valor $2 / T_{inicial}$, onde $T_{inicial}$ é o valor resultante do processo de busca para resolver de forma aproximada o problema (D_t^{λ}) na primeira iteração do algoritmo geral de subgradientes. Se após 30 iterações sucessivas o limite superior ub não aumentar, π é atualizado para $\pi = \pi/\kappa$, $\kappa \neq 0$. Para a heurística lagrangeana usual $t = T_{inicial} = 1$ e $\kappa = 2$, e para a heurística LS κ é calculado de modo a tornar π menor do que 0.005 após 6 divisões de π.

Da mesma forma que para o PLS, na maioria das iterações a solução x^{λ} é feita viável conservando-se os servidores abertos, alterando-se as alocações $x^{f}_{ij^*} = 1$ se $i \in I$, $j^* \in N$ e $c_{ij^*} = \min \{ cij \}$ e utilizando-se uma heurística de troca para procurar por soluções viáveis melhores, sempre que o parâmetro π é dividido. No caso do PLC, entretanto, um passo adicional é executado, no qual são resolvidos problemas de transporte que resultam do problema (P) quando os servidores são fixados como abertos ($y_i = 1$) ou como fechados ($y_i = 0$), como comenta-se a seguir.

Uma vez estabelecidos os servidores abertos, deve-se resolver o seguinte problema para determinar as alocações x_{ij}:

$$\min \sum_{i=1}^{ma} \sum_{j=1}^{n} c_{ij} x_{ij}$$

sujeito a:

$$\sum_{i=1}^{ma} x_{ij} = 1 \quad ; \quad j \in N$$

$$\sum_{j=1}^{n} d_j x_{ij} \leq s_i \quad ; \quad i \in M_a = \{ 1, \ldots, ma \}$$

$$0 \leq x_{ij} \leq 1 \quad ; \quad i \in M_a, j \in N$$

onde ma corresponde ao número de servidores abertos. Nesta formulação x_{ij} corresponde à fração da demanda do cliente j atendida pelo servidor i. Imaginando-se uma outra variável de decisão w_{ij} representando a quantidade de produto transportada do servidor aberto i para o cliente j, tem-se o seguinte modelo de transporte:

$$\min \sum_{i=1}^{ma} \sum_{j=1}^{n} c_{ij} w_{ij}$$

sujeito a:

$$\sum_{j=1}^{n} w_{ij} = s_i \quad ; \quad i \in M_a$$

$$\sum_{i=1}^{ma} w_{ij} = d_j \quad ; \quad j \in N$$

$$w_{ij} \geq 0 \quad ; \quad i \in M_a, j \in N$$

É importante observar que, como são abertos tantos servidores quantos forem necessários para atender toda a demanda dos clientes, tem-se, no caso geral, que: $\sum_{i=1}^{ma} s_i > \sum_{j=1}^{n} d_j$. Neste caso, para o modelo de transporte, cria-se um cliente-fictício cuja demanda é igual ao excesso de oferta dos servidores, de modo a se ter o balanço das ofertas e demandas.

Estes modelos de transporte são resolvidos usando o algoritmo Primal Simplex para redes discutido no Capítulo 2 (ver Seção 2.3). Uma vez resolvidos tais modelos, tem-se w_{ij}, o fluxo de produtos do servidor aberto i para o cliente j. Para determinar a fração de demanda do cliente j atendido pelo servidor i basta fazer: $x_{ij} = \frac{w_{ij}}{d_j}$.

Para o problema de localização capacitada, os algoritmos referentes à heurística de troca e à busca do fator surrogate são ligeiramente diferentes do que os estabelecidos para o PLS. Apresenta-se, a seguir, tais algoritmos para o PLC.

Heurística de Troca para o PLC

Seja dado, $q = v^f$ (valor da solução viável x^f atual).

Fazer k = max (25, m/10);

Fazer k = min (k, m);

Para i = 1 até m, com $i \notin O$ e $y_i = 1$, fazer:

p = 0;

Para j = 1 até m, com $j \notin C$, $y_j = 0$ e $p < k$, fazer:

p = p + 1;

Intercambiar i com j, atualizando I correspondentemente;

Calcular $v^f = \sum_{j=1}^{n} (\min_{i \in I} c_{ij})$;

Se $v^f < q$ então fazer $q = v^f$. Do contrário, intercambiar i com j e atualizar I.

Se q < ub então fazer ub = q.

Busca do Fator Surrogate para o PLC

Sejam dados: s = 0.35 (tamanho de passo inicial) e max_it = 10 (número máximo de iterações).

Fazer t = 1.5; r = 0, k = 0;

Repetir

k = k + 1;

Se k > max_it então parar. Caso contrário, resolver $(L_t SP^{\lambda})$.

Se $v(L_t SP^{\lambda}) > r$ então fazer o seguinte:

Fazer T = t ;

Fazer $r = v(L_t SP^{\lambda})$;

Calcular a inclinação ω^{λ} da função LS usando: $\omega^{\lambda} = \sum_{j=1}^{n} \lambda_j - \sum_{j=1}^{n} \sum_{i \in I} \lambda_j x_{ij}$

Se $\omega^{\lambda} < 0$ então fazer:

$t^- = t$;

Se $t^+ > 0$ então tentar melhorar o fator surrogate atual resolvendo o problema $(L_{(t^- + t^+)/2} SP^{\lambda})$, atualizando $T = \frac{t^- + t^+}{2}$, se necessário, e parar. Caso contrário, fazer t = t - s.

Caso contrário, ou seja, se $\omega^{\lambda} > 0$, fazer:

$t^+ = t$;

Se $t^- > 0$ então tentar melhorar o fator surrogate atual resolvendo o problema $(L_{(t^- + t^+)/2} SP^{\lambda})$, atualizando $T = \frac{t^- + t^+}{2}$, se necessário, e parar. Caso contrário,

fazer t = t + s.

Caso contrário, ou seja, se $v(L_t\, SP^{\lambda})$ < r, tentar melhorar o fator surrogate atual resolvendo o problema $(L_{t-s/2}\, SP^{\lambda})$, atualizando T = t - s/2, se necessário, e parar.

Enquanto (as condições de parada não forem verificadas).

Este procedimento de busca resulta no fator T usado para resolver o problema $(L_T\, SP^{\lambda})$ na relaxação LS. Entretanto, se esta busca resultar no mesmo fator T para 2 aplicações consecutivas do algoritmo, adota-se este fator como fixo para os demais problemas $(L_T\, SP^{\lambda})$, tornando o processo de busca não mais necessário.

Os resultados dos testes computacionais são mostrados nas tabelas a seguir. Nestas tabelas, da mesma forma que para o PLS, os problemas estão divididos em duas classes: a dos problemas pequenos, com instâncias de tamanhos (n = 50, m = 16), (n = 50, m = 25) e (n = 50, m = 50); e a dos problemas grandes, com instâncias de tamanho (n = 1000, m = 100).

Nas tabelas a seguir mostram-se os resultados obtidos pelas heurísticas lagrangeana usual e LS para o PLC. Para estas tabelas valem as mesmas observações já feitas na apresentação dos resultados dos testes computacionais para o PLS (ver Seção 4.4).

Tabela 4.14 - Problema de Localização Capacitado: Resultados computacionais da heurística lagrangeana na solução de pequenas instâncias

Problema	n	m	D_UB	D_LB	NRL	NSA	NSI	T_TOTAL
4-1	50	16	1.331	0.008	460	9	7	4.17
4-2			1.254	0.011	544	8	8	4.85
4-3			1.194	0.034	663	8	8	5.71
4-4			1.114	0.044	568	8	8	5.14
5-1			0.397	0.043	559	3	12	5.10
6-1			-	-	92	11	0	0.67
6-2			-	-	178	8	1	1.48
6-3			0.002	0.132	491	3	6	4.06
6-4			0.750	0.002	382	2	11	3.28
7-1			-	-	106	11	0	0.63
7-2			-	-	146	9	0	1.20
7-3			-	-	101	5	1	0.75
7-4			-	-	87	4	0	0.61
8-1	50	25	0.884	0.179	555	5	20	10.88
8-2			1.128	0.044	757	4	21	14.03
8-3			1.565	0.052	661	1	24	13.53
8-4			3.683	0.190	663	0	25	14.17
9-1			-	-	121	7	12	1.86
9-2			-	0.078	564	6	7	7.28
9-3			0.129	0.196	634	2	14	9.25
9-4			0.295	0.404	543	1	17	9.32
10-1			-	-	184	7	15	3.00
10-2			-	-	195	11	0	2.33
10-3			-	-	49	8	0	0.47
10-4			-	-	242	4	1	2.59
11-1	50	50	-	0.036	699	13	5	23.93
11-2			-	0.135	746	6	15	26.29
11-3			0.035	0.221	1146	4	16	39.06
11-4			1.004	0.261	673	2	39	38.26
12-1			-	-	305	6	22	12.39
12-2			-	0.078	440	6	9	17.36
12-3			-	0.108	646	3	11	21.45
12-4			0.174	0.417	739	1	21	24.09
13-1			-	-	374	15	0	16.37
13-2			-	-	430	10	1	14.72
13-3			-	-	204	8	0	7.14
13-4			-	-	270	4	0	6.73

unesp

Tabela 4.15 - Problema de Localização Capacitado: Resultados computacionais da heurística LS na solução de pequenas instâncias

Problema	n	m	D_UB	D_LB	NRL	NSA	NSI	T_TOTAL
4-1	50	16	1.331	0.008	439	8	8	3.63
4-2			1.254	0.030	442	8	8	3.67
4-3			1.194	0.051	498	8	8	3.96
4-4			1.114	0.017	470	8	8	3.87
5-1			0.643	0.044	437	3	12	3.61
6-1			-	-	102	11	0	0.61
6-2			-	-	181	9	0	1.47
6-3			0.002	0.132	372	3	11	3.00
6-4			0.750	0.001	416	3	10	3.10
7-1			-	-	100	11	0	0.60
7-2			-	-	75	9	0	0.43
7-3			-	-	221	5	0	1.55
7-4			-	-	88	4	0	0.62
8-1	50	25	0.884	0.177	484	5	20	7.69
8-2			1.128	0.058	617	3	22	9.91
8-3			2.187	0.051	442	0	25	8.55
8-4			3.561	0.185	562	0	25	9.96
9-1			-	-	130	5	17	2.38
9-2			-	0.080	267	8	5	3.18
9-3			0.194	0.197	408	2	15	5.78
9-4			0.295	0.403	329	2	16	5.47
10-1			-	-	123	5	17	1.96
10-2			-	-	164	6	8	2.07
10-3			-	-	122	5	5	1.29
10-4			-	-	167	4	1	1.74
11-1	50	50	-	0.036	468	13	6	18.18
11-2			-	0.125	482	6	14	18.23
11-3			0.035	0.243	661	6	15	22.62
11-4			2.514	0.278	618	0	50	29.70
12-1			-	-	219	5	20	10.60
12-2			-	0.081	300	4	10	12.84
12-3			-	0.107	521	3	11	16.10
12-4			0.174	0.417	525	1	41	16.93
13-1			-	-	319	2	42	12.42
13-2			-	-	185	5	9	6.50
13-3			-	-	435	7	1	11.45
13-4			-	-	155	5	0	3.86

unesp

Tabela 4.16 - Problema de Localização Capacitado: Resultados computacionais da heurística lagrangeana na solução de grandes instâncias

Problema	n	m	D_UB	D_LB	NRL	NSA	NSI	T_TOTAL
A-1	1000	100	2.761	0.065	597	0	87	1484.94
A-2			3.759	0.052	683	0	89	1463.76
A-3			-	0.208	506	0	65	1135.72
A-4			0.008	-	706	1	4	1042.80
B-1			0.351	-	490	0	37	1752.42
B-2			4.600	0.300	673	0	100	1753.72
B-3			3.950	0.340	590	0	100	1600.45
B-4			2.265	0.109	803	0	90	1750.28
C-1			0.105	0.161	1136	0	28	2387.25
C-2			3.566	0.438	670	0	100	1763.89
C-3			1.340	0.095	709	0	81	1744.86
C-4			0.027	0.050	508	0	33	1493.86

Tabela 4.17 - Problema de Localização Capacitado: Resultados computacionais da heurística LS na solução de grandes instâncias

Problema	n	m	D_UB	D_LB	NRL	NSA	NSI	T_TOTAL
A-1	1000	100	0.263	0.070	505	0	97	1023.98
A-2			4.123	0.058	454	0	92	884.00
A-3			-	0.202	475	2	16	813.53
A-4			0.008	-	360	3	2	548.38
B-1			0.351	-	569	0	38	1300.43
B-2			4.892	0.302	735	0	100	1396.15
B-3			3.061	0.366	469	0	100	1035.73
B-4			2.265	0.114	780	0	89	1311.43
C-1			0.105	0.164	993	0	29	1706.54
C-2			0.388	0.451	655	0	59	1339.78
C-3			2.092	0.338	614	0	96	1193.15
C-4			0.027	0.050	601	0	48	1186.83



Tabela 4.18 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de pequenas instâncias obtidas pela relaxação lagrangeana

			D_LB				
Problema	n	m	5%	4%	3%	2%	1%
4-1	50	16	0.99	0.99	0.99	1.00	1.02
4-2			0.97	0.97	1.00	1.03	1.08
4-3			0.92	0.92	0.93	0.94	1.18
4-4			1.52	1.52	1.54	1.57	2.01
5-1			0.39	0.43	0.67	1.13	1.15
6-1			0.24	0.25	0.26	0.27	0.28
6-2			0.20	0.20	0.25	0.26	0.30
6-3			0.26	0.30	0.30	0.39	0.50
6-4			0.23	0.24	0.24	0.36	0.74
7-1			0.20	0.20	0.20	0.22	0.24
7-2			0.19	0.21	0.21	0.22	0.26
7-3			0.21	0.22	0.27	0.29	0.33
7-4			0.22	0.22	0.23	0.25	0.59
8-1	50	25	1.89	1.91	1.91	1.95	1.99
8-2			2.10	2.13	2.17	2.27	2.53
8-3			2.51	2.56	2.59	2.62	3.24
8-4			2.75	2.75	2.81	2.92	4.60
9-1			0.52	0.55	0.57	0.61	0.66
9-2			0.36	0.38	0.67	1.87	1.88
9-3			0.46	0.59	0.69	0.82	1.02
9-4			1.39	1.41	1.41	1.42	2.82
10-1			0.38	0.50	0.56	1.84	1.86
10-2			0.32	0.37	0.37	0.50	0.84
10-3			0.30	0.32	0.32	0.34	0.36
10-4			0.28	0.31	0.37	0.49	0.99
11-1	50	50	7.07	7.07	7.09	7.14	7.24
11-2			4.64	4.86	5.21	5.25	5.53
11-3			4.64	5.46	5.46	5.88	9.99
11-4			6.33	6.49	6.88	11.96	12.10
12-1			2.85	2.91	6.72	6.72	6.76
12-2			4.36	4.36	4.46	4.64	4.96
12-3			3.81	3.83	3.83	4.02	4.14
12-4			5.24	5.24	5.29	5.48	9.10
13-1			1.67	2.20	2.46	6.05	6.15
13-2			0.84	0.90	0.99	2.11	4.71
13-3			0.90	1.06	1.12	1.18	3.78
13-4			1.11	1.11	1.16	1.92	3.59

unesp

Tabela 4.19 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de pequenas instâncias obtidas pela relaxação LS

			D_LB				
Problema	n	m	5%	4%	3%	2%	1%
4-1	50	16	1.43	1.44	1.45	1.46	1.52
4-2			1.39	1.40	1.42	1.50	1.57
4-3			1.35	1.37	1.39	1.43	1.46
4-4			1.09	1.10	1.11	1.13	1.16
5-1			0.36	0.36	0.82	0.82	0.85
6-1			0.22	0.22	0.25	0.27	0.28
6-2			0.22	0.23	0.23	0.25	0.30
6-3			0.25	0.27	0.28	0.35	0.79
6-4			0.34	0.34	0.35	0.42	0.85
7-1			0.20	0.23	0.24	0.25	0.28
7-2			0.23	0.23	0.24	0.26	0.28
7-3			0.21	0.22	0.24	0.26	0.28
7-4			0.19	0.20	0.20	0.23	0.57
8-1	50	25	1.72	1.74	1.75	1.78	1.84
8-2			2.34	2.35	2.36	2.38	2.44
8-3			1.97	1.98	1.99	2.02	2.08
8-4			2.27	2.27	2.33	2.45	2.85
9-1			0.39	0.39	0.41	0.45	1.63
9-2			0.36	0.40	0.42	0.49	0.60
9-3			0.70	0.75	0.78	0.78	1.70
9-4			1.17	1.18	1.20	1.26	1.35
10-1			0.35	0.35	0.41	1.70	1.71
10-2			0.31	0.36	0.42	0.57	0.60
10-3			0.37	0.37	0.38	0.41	0.43
10-4			0.27	0.28	0.28	0.31	0.59
11-1	50	50	7.53	7.56	7.64	7.77	7.92
11-2			6.38	6.40	6.43	6.51	6.78
11-3			8.12	8.18	8.42	8.61	8.97
11-4			8.46	8.54	8.91	9.22	9.64
12-1			1.09	1.15	1.55	2.59	6.59
12-2			4.03	4.03	4.08	4.16	4.31
12-3			4.30	4.32	4.34	4.34	4.38
12-4			4.28	4.28	4.33	4.39	4.72
13-1			1.02	1.34	2.07	3.18	4.00
13-2			0.83	2.78	2.80	2.82	2.86
13-3			0.92	1.04	1.53	4.04	4.06
13-4			0.66	0.66	0.66	0.95	2.19

unesp

Tabela 4.20 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de grandes instâncias obtidas pela relaxação lagrangeana

			D_LB				
Problema	n	m	5%	4%	3%	2%	1%
A-1	1000	100	302.29	306.92	337.17	458.09	475.51
A-2			304.98	310.74	322.74	435.12	449.11
A-3			309.99	313.48	313.48	347.30	495.94
A-4			195.29	250.83	290.52	290.52	391.21
B-1			406.73	410.15	418.27	641.01	642.16
B-2			344.45	349.07	387.63	531.72	605.51
B-3			326.43	329.94	523.40	533.77	716.63
B-4			334.62	359.37	411.78	547.34	713.83
C-1			371.09	533.98	535.16	540.94	726.70
C-2			367.31	379.83	459.62	494.31	808.29
C-3			327.23	330.69	341.88	553.37	556.78
C-4			328.52	330.87	333.08	465.84	481.65

Tabela 4.21 - Problema de Localização Capacitado: Tempos de CPU para soluções quase-ótimas de grandes instâncias obtidas pela relaxação LS

			D_LB				
Problema	n	m	5%	4%	3%	2%	1%
A-1	1000	100	172.72	184.63	197.50	235.54	361.99
A-2			149.30	155.14	156.30	167.88	202.63
A-3			135.73	138.12	142.83	147.54	266.19
A-4			112.30	114.57	116.89	123.96	133.57
B-1			206.36	212.42	219.44	228.76	285.09
B-2			188.54	196.85	207.47	226.23	473.50
B-3			196.45	205.92	228.31	395.40	403.66
B-4			174.82	176.01	198.99	321.12	324.45
C-1			233.80	254.57	282.99	340.57	352.42
C-2			215.28	233.30	256.99	458.53	464.38
C-3			187.43	218.94	247.93	387.76	399.14
C-4			176.36	178.68	183.14	195.33	228.00

unesp

Tabela 4.22 - Problema de Localização Capacitado: Razões de tempo para as heurísticas LS e lagrangeana na solução de pequenas instâncias

	Tempo para soluções quase-ótimas (D_LB)						Tempo Total	
Problema	5%	4%	3%	2%	1%	Média	Razão	Média
4-1	1.44	1.45	1.46	1.46	1.49		0.87	
4-2	1.43	1.44	1.42	1.46	1.45		0.76	
4-3	1.47	1.49	1.49	1.52	1.24		0.69	
4-4	0.72	0.72	0.72	0.72	0.58		0.75	
5-1	0.92	0.84	1.22	0.73	0.74		0.71	
6-1	0.92	0.88	0.96	1.00	1.00		0.91	
6-2	1.10	1.15	0.92	0.96	1.00		0.99	
6-3	0.96	0.90	0.93	0.90	1.58		0.74	
6-4	1.48	1.42	1.46	1.17	1.15		0.95	
7-1	1.00	1.15	1.20	1.14	1.17		0.95	
7-2	1.21	1.10	1.14	1.18	1.08		0.36	
7-3	1.00	1.00	0.89	0.90	0.85		2.07	
7-4	0.86	0.91	0.87	0.92	0.97		1.02	
8-1	0.91	0.91	0.92	0.91	0.92		0.71	
8-2	1.11	1.10	1.09	1.05	0.96		0.71	
8-3	0.78	0.77	0.77	0.77	0.64		0.63	
8-4	0.83	0.83	0.83	0.84	0.62		0.70	
9-1	0.75	0.71	0.72	0.74	2.47		1.28	
9-2	1.00	1.05	0.63	0.26	0.32		0.44	
9-3	1.52	1.27	1.13	0.95	1.67		0.62	
9-4	0.84	0.84	0.85	0.89	0.48		0.59	
10-1	0.92	0.70	0.73	0.92	0.92		0.65	
10-2	0.97	0.97	1.14	1.14	0.71		0.89	
10-3	1.23	1.16	1.19	1.21	1.19		2.74	
10-4	0.96	0.90	0.76	0.63	0.60		0.67	
11-1	1.07	1.07	1.08	1.09	1.09		0.76	
11-2	1.38	1.32	1.23	1.24	1.23		0.69	
11-3	1.75	1.50	1.54	1.46	0.90		0.58	
11-4	1.34	1.32	1.30	0.77	0.80		0.78	
12-1	0.38	0.40	0.23	0.39	0.97		0.86	
12-2	0.92	0.92	0.91	0.90	0.87		0.74	
12-3	1.13	1.13	1.13	1.08	1.06		0.75	
12-4	0.82	0.82	0.82	0.80	0.52		0.70	
13-1	0.61	0.61	0.84	0.53	0.65		0.76	
13-2	0.99	3.09	2.83	1.34	0.61		0.44	
13-3	1.02	0.98	1.37	3.42	1.07		1.60	
13-4	0.59	0.59	0.57	0.49	0.61	1.03	0.57	0.85

unesp

Tabela 4.23 - Problema de Localização Capacitado: Razões de tempo para as heurísticas LS e lagrangeana na solução de grandes instâncias

	Tempo para soluções quase-ótimas (D_LB)						Tempo Total	
Problema	5%	4%	3%	2%	1%	Média	Razão	Média
A-1	0.57	0.60	0.59	0.51	0.76		0.69	
A-2	0.49	0.50	0.48	0.39	0.45		0.60	
A-3	0.44	0.44	0.46	0.42	0.54		0.72	
A-4	0.58	0.46	0.40	0.43	0.34		0.53	
B-1	0.51	0.52	0.52	0.36	0.44		0.74	
B-2	0.55	0.56	0.54	0.43	0.78		0.80	
B-3	0.60	0.62	0.44	0.74	0.56		0.65	
B-4	0.52	0.49	0.48	0.59	0.45		0.75	
C-1	0.63	0.48	0.53	0.63	0.48		0.71	
C-2	0.59	0.61	0.56	0.93	0.57		0.76	
C-3	0.57	0.66	0.73	0.70	0.72		0.68	
C-4	0.54	0.54	0.55	0.42	0.47	0.54	0.79	0.70

Das tabelas 4.22 e 4.23 podem ser retiradas as seguintes conclusões para o PLC:

- Para os problemas pequenos e que demandam pouco tempo de processamento, a razão média de 1.03 para os tempos associados às soluções quase-ótimas mostram que, praticamente, não há melhoria em termos de tempo computacional quando se usa a heurística lagrangeana usual ou a heurística LS. Entretanto, a razão média de 0.85 para os tempos totais expendidos na solução dos problemas confirma uma melhor eficiência computacional da heurística LS.

- Para os problemas grandes, para os quais os tempos de processamento são significativamente maiores, o ganho da heurística LS, em termos de tempo computacional, é mais evidente. Os resultados mostram que a heurística LS é quase duas vezes mais rápida do que a heurística lagrangeana usual para a obtenção de soluções quase-ótimas e cerca de 30% melhor quando se considera o tempo total gasto na solução dos problemas testados.

Portanto, com base nos resultados obtidos com os testes computacionais realizados para as classes de problemas de p-medianas, localização simples e localização capacitada pode-se concluir que o emprego da heurística LS apresenta vantagens em relação à aplicação da heurística lagrangeana usual. A economia em termos de tempo computacional é bastante significativa para grandes problemas. Isto

unesp

encoraja o uso desta nova relaxação em outros problemas combinatórios e pode ser muito importante em um ambiente de modelagem em redes, principalmente quando usado como ferramenta de apoio à tomada de decisões em situações práticas.

Capítulo 5

Conclusão

5.1 - O Trabalho Atual

As contribuições discutidas nos capítulos anteriores, evidentemente, ainda não são suficientes para o desenvolvimento completo do ambiente de modelagem em redes pretendido (Figura 1.1), o que justifica o título deste trabalho.

Tais contribuições têm sido implementadas e vem sendo reunidas no sistema MoNet (ver Apêndice D). A figura a seguir destaca os módulos e os relacionamentos entre módulos para os quais já existe algum desenvolvimento.

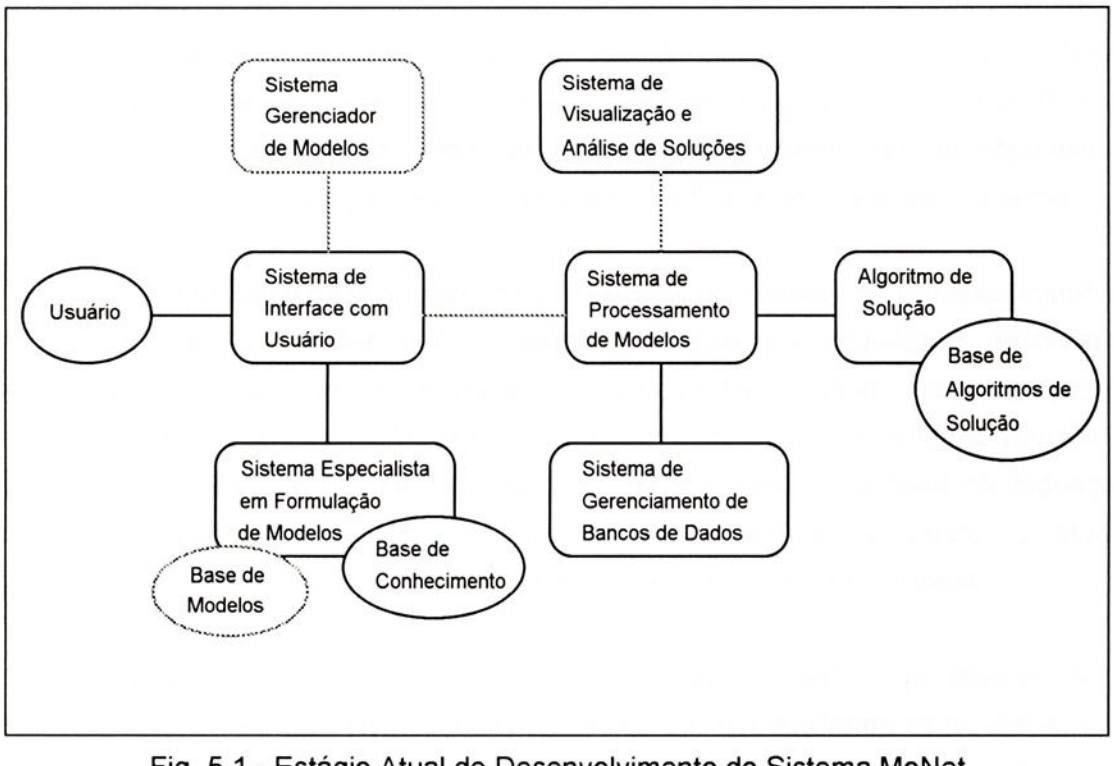


Fig. 5.1 - Estágio Atual de Desenvolvimento do Sistema MoNet

Os desenvolvimentos já realizados sobre cada um destes módulos correspondem a vários trabalhos, discutidos a seguir.

O projeto completo do Sistema de Interface com Usuário encontra-se discutido em Senne *et al.* (1996). Sobre este projeto implementou-se uma interface gráfica (Marchezan *et al.*, 1995; Nakagawa *et al.*, 1996) que permite ao usuário especificar o modelo conceitual de um problema (através das primitivas gráficas discutidas no Capítulo 3) e traduzir este modelo em construções da linguagem OPS5, que por sua vez são interpretadas pelo Sistema Especialista em Formulação de Modelos (ver Apêndice C). A Figura 5.2 destaca os desenvolvimentos já realizados (ainda que parcialmente).

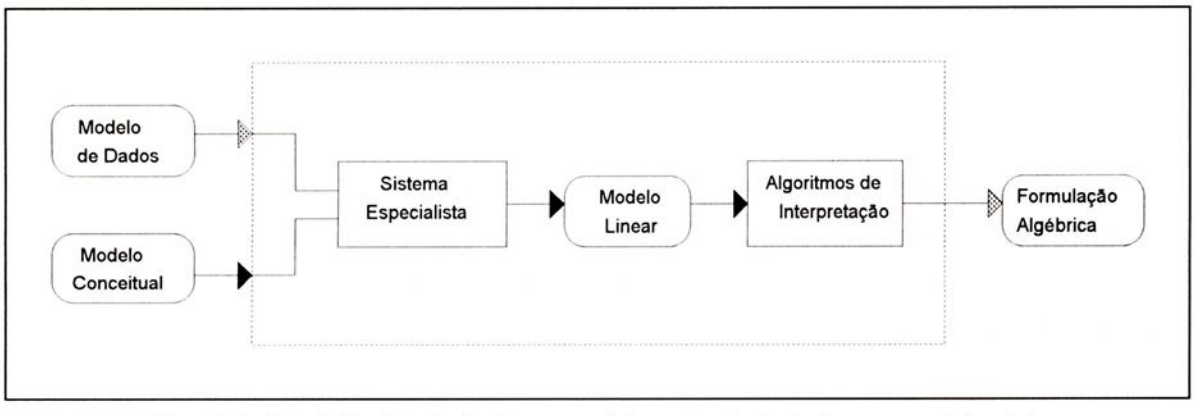


Fig. 5.2 - Estágio Atual de Desenvolvimento da Interface com Usuário

O desenvolvimento de um protótipo de Sistema de Visualização e Análise de Soluções vem sendo desenvolvido como projetos de Iniciação Científica. No momento dispõe-se de um editor gráfico de redes, acoplado ao algoritmo Primal Simplex apresentado no Capítulo 2. Isto permite, portanto, dado um modelo de rede, estabelecido graficamente, obter e visualizar (por meio de cores) os fluxos em cada um dos arcos.

A implementação da linguagem NPL2 resultará no Sistema de Processamento de Modelos. Atualmente somente o analisador sintático da linguagem encontra-se implementado. A conclusão desta implementação resultará também na integração do Sistema de Processamento de Modelos a um Sistema de Gerenciamento de Bancos de Dados que deverá utilizar o módulo DBE (*Data Base Engine*) disponível na linguagem Delphi (Borland, 1995). Com isto, pretende-se adotar o enfoque proposto por Moody (1994) de reutilização, no ambiente de modelagem, de um SGBD já existente.

Sobre os algoritmos de solução existem vários trabalhos já desenvolvidos. Dispõe-se de um sistema para teste de algoritmos em redes (Senne *et al.*, 1994) e de estudo comparativo da eficiência de algoritmos para solução de modelos de fluxos em redes lineares por partes (Marins *et al.*, 1997). Dispõe-se também das implementações dos algoritmos de localização discutidos no Capítulo 4 (Lorena and Senne, 1996a; Lorena and Senne, 1996b; Senne and Lorena, 1997a; Senne and Lorena, 1997b).

unesp

É importante observar que a abordagem discutida no Capítulo 4 já foi aplicada, também com bons resultados, ao problema generalizado de atribuição (Narciso, 1994) e ao problema do caixeiro viajante, e vem sendo formalizada, desde então, como tese de doutorado no curso de Computação Aplicada do Instituto Nacional de Pesquisas Espaciais (INPE), por Marcelo Gonçalves Narciso, com orientação do Dr. Luís Antonio Nogueira Lorena.

5.2 - O Trabalho Futuro

Com este trabalho, foram dados os primeiros passos na direção de um ambiente inteligente de modelagem em redes. Embora as linhas gerais de desenvolvimento deste ambiente já estarem estabelecidas, muito trabalho ainda existe por fazer, mesmo para os módulos destacados na Figura 5.1. Portanto, são significativas as oportunidades de desenvolvimentos futuros.

Sobre o Sistema de Interface com Usuário será necessário desenvolver a representação de Modelos de Dados e sua tradução para construções OPS5. Além disto, será preciso desenvolver novos algoritmos de interpretação, de forma a ser possível obter a formulação algébrica de um modelo em rede segundo o esquema mostrado na Figura 5.2. Será necessário ainda, providenciar a ligação deste sistema de interface com o Sistema de Processamento de Modelos, permitindo que a modelagem em redes acompanhe um paradigma de programação visual.

Sobre o Sistema Especialista em Formulação de Modelos é necessário ainda uma grande atenção. Em primeiro lugar, é necessário explorar como os esquemas conceituais podem ser organizados na Base de Modelos (ver Figura 5.1), de modo a permitir tanto a construção de novos modelos como combinação de sub-modelos já existentes, quanto a particularização de modelos já conhecidos para situações particulares. Além disto, é necessário expor o sistema especialista a um número grande de aplicações, visando enriquecer sua base de conhecimento e aumentar sua funcionalidade.

Aprimorar o Sistema de Processamento de Modelos é também uma tarefa por fazer. Antes de tudo, é necessário concluir a implementação da linguagem NPL2. Um outro passo, deverá ser o enriquecimento da linguagem, introduzindo novas construções que permitam sua utilização para modelos lineares em geral, o que vai implicar na

unesp

necessidade de inclusão na base, de novos algoritmos de solução. Com tais extensões pode-se ter a linguagem NPL2 como alvo dos algoritmos de interpretação de modelos lineares discutidos anteriormente (ver Figura 5.2). Será necessário, também, providenciar a ligação do SPM ao Sistema de Visualização e Análise de Soluções, de modo a permitir a conexão das representações algébrica e gráfica de partes de um modelo.

O enriquecimento da Base de Algoritmos de Solução com o desenvolvimento de novos algoritmos é também uma tarefa a fazer. Existem vários outros problemas que também ocorrem com frequência e merecem uma abordagem apropriada, como os problemas de roteamento, por exemplo.

A utilização de um Sistema de Informações Geográficas (Fischbeck, 1994), como parte do Sistema de Visualização e Análise de Soluções, parece interessante e merece atenção. Um ambiente que disponha de um sofisticado Sistema de Visualização e Análise de Soluções parece ser muito útil no processo de análise de uma cidade com vistas ao planejamento de transporte público, por exemplo. Mapas de uso do solo indicando a distribuição espacial das atividades urbanas no espaço da cidade, a compartimentação da cidade em setores, a estimativa da população residente e a sua caracterização sócio-econômica poderiam ser armazenados em um Sistema de Informações Geográficas e usados para estimar e localizar espacialmente a demanda por transporte coletivo. Algoritmos de roteamento poderiam obter soluções levando em conta tais dados, como ocorre no trabalho de Li and Eglese (1996).

Afora tudo isto, ainda será necessário um grande esforço para desenvolver o Sistema de Gerenciamento de Modelos e, desta forma, completar a arquitetura pretendida para o ambiente de modelagem em redes.

Referências Bibliográficas

AHUJA, R.K.; MAGNANTI, T.L.; ORLIN, J.B. *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ, Prentice Hall, 1993.

ANDREI, N.; BARBULESCU, M. Balance Constraints Reduction of Large-Scale Linear Programming Problems. *Annals of Operations Research*, 43 (1-4): 149-170, 1993.

ANGEHRN, A.A.; LÜTHI, H.J. Intelligent Decision Support Systems: A Visual Interactive Approach, *Interfaces*, 20 (6): 17-28, Nov.-Dec. 1990.

BABAYEV, D.A.; MARDANOV, S.S Reducing the Number of Variables in Integer and Linear Programming Problems. *Computational Optimization and Applications*, 3 (2): 99-109, May, 1994.

BAKER, B.M., Linear Relaxations of the Capacitated Warehouse Location Problem, *Journal of the Operational Research Society*, 33: 475-479, 1982.

BARCELÓ, J.; FERNANDEZ, E.; JÖRNSTEN, K.O. Computational Results from a New Lagrangean Relaxation Algorithm for the Capacitated Plant Location Problem, *European Journal of Operational Research*, 53: 38-45, 1991.

BARR, R.S., GLOVER, F.; KLINGMAN, D. The Alternating Path Basis Algorithm for the Assignment Problem. *Mathematical Programming*, 13 (1): 1-13, 1977.

BASTON, V.D.; RAHMOUNI, M.K.; WILLIAMS, H.P. The Practical Conversion of Linear Programmes to Network Flow Models. *European Journal of Operational Research*, 50: 325-335, 1991.

BEASLEY, J.E. A Note on Solving Large p-Median Problems. *European Journal of Operational Research*, 21: 270-273, 1985.

BEASLEY, J.E. An Algorithm for Solving Large Capacitated Warehouse Location Problems. *European Journal of Operational Research*, 33: 314-325, 1988.

BEASLEY, J.E. Lagrangean Heuristics for Location Problems. *Europen Journal of Operational Research*, 65: 383-399, 1993.

BEASLEY, J.E. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41 (11): 1069-1072, 1990.

BELHOT, R.V.; MOCCELLIN, J.V. Uma Análise de Relações entre a Pesquisa Operacional e Sistemas Especialistas. In: *Simpósio Brasileiro de Pesquisa Operacional*, 22. Fortaleza, CE, Anais, p. 505-511, Out. 1989.

BHARADWAJ, A.; CHOOBINEH, J.; SHETTY, B. Model Management Systems: A Survey. *Annals of Operations Research*, 38 (1-4): 17-67, Dec. 1992.

unesp

BHARGAVA, H.K.; KRISHNAN, R.; MUKHERJEE, S. On the Integration of Data and Mathematical Modeling Languages. *Annals of Operations Research*, 38 (1-4): 69-95, Dec. 1992.

BHARGAVA, H.K.; KRISHNAN, R. Computer-Aided Model Construction. *Decision Support Systems*, 9 (1): 91-111, Jan. 1993.

BINBASIOUGLU, M.; JARKE, M. Domain Specific DSS Tools for Knowledge Based Model Building. *Decision Support Systems*, 2: 213-223, 1986.

BISSCHOP, J.J; KUIP, C.A.C. Hierarchical Sets in Mathematical Programming Modeling Languages. *Computational Optimization and Applications*, 1: 415-438, 1993(a).

BISSCHOP, J.J; KUIP, C.A.C. Compound Sets in Mathematical Programming Modeling Languages. *Management Science*, 39 (6): 746-756, June, 1993(b).

BISSCHOP, J.J.; FOURER, R. New Constructs for the Description of Combinatorial Optimization Problems in Algebraic Modelling Languages. *Computational Optimization and Applications*, 1994

BLANNING, R.W. Model Management Systems: An Overview. *Decision Support Systems*, 9 (1): 9-18, Jan. 1993.

BORLAND Inc. *Borland Delphi for Windows - User's Guide*. Borland International, Inc., Scotts Valley, CA, 1995.

BRANDEAU, M.L.; CHIU, S.S. An Overview of Representative Problems in Location Research, *Management Science,* 35 (6): 645-674, 1989.

BROWNSTON, L.; FARREL, R.; KANT, E.; MARTIN, N. *Programming Expert Systems in OPS5 - An Introduction to Rule-Based Programming*. Reading, MA, Addison-Wesley, 1985.

CHANG, A.M.; HOLSAPPLE, C.W.; WHINSTON, A.B. Model Management Issues and Directions. *Decision Support Systems*, 9 (1): 19-37, Jan. 1993.

CHOOBINEH, J. A Diagramming Technique for Representation of Linear Models. *OMEGA*, 19 (1): 43-51, 1991(b).

CHOOBINEH, J. SQLMP: A Data Sublanguage for Representation and Formulation of Linear Mathematical Models, *ORSA Journal on Computing*, 3 (4): 358-375, Fall 1991(a).

CHRISTOFIDES, N.; BEASLEY, J.E. A Tree Search Algorithm for the P-Median Problems. *European Journal of Operational Research*, 10: 196-204, 1982.

CHRISTOFIDES, N.; BEASLEY, J.E. Extensions to a Lagrangean Relaxation Approach for the Capacitated Warehouse Location Problem. *European Journal of Operational Research*, 12: 19-28, 1983.

CHUNG, Q.B.; O'KEEFE, R.M. A Formal Analysis of the Model Management Literature. *Annals of Operations Research*, 38 (1-4): 137-176, Dec. 1992.

COLLAUD, G.; BOLTUCK, J.P. gLPS: A Graphical Tool for the Definition and Manipulation of Linear Problems. *European Journal of Operational Research*, 72 (2): 277-286, Jan. 1994.

unesp

CONNELL, N.A.D.; POWELL, P.L. A Comparison of Potential Applications of Expert Systems and Decision Support Systems. *Journal of the Operational Research Society*, 41 (5): 431-439, 1990.

CORNUEJOLS, G.; SRIDHARAN, R.; THIZY, J.M. A Comparison of Heuristics and Relaxations for the Capacitated Plant Location Problem. *European Journal of Operational Research*, 50: 280-297, 1991.

CUNNINGHAM, W.H. A Network Simplex Method. *Mathematical Programming*, 11 (2): 105-116, 1976.

CUNNINGHAM, W.H. Theoretical Properties of the Network Simplex Method. *Mathematics of Operations Research*, 4 (2): 196-208, 1979.

DANTZIG, G.B. *Linear Programming and Extensions*. Princeton, NJ, Princeton University Press, 1963.

DARBY-DOWMAN, K. *The Development of Modelling Skills in Linear Programming*. Technical Report 11/93, Department of Mathematics and Statistics, Brunel University, England, Oct. 1993.

DHAR, V.; CROKER, A. Knowledge-Based Decision Support in Business: Issues and a Solution. *IEEE Expert*, 3 (1): 53-62, Spring 1988.

DOLK, D.R.; KONSYNSKI, B.R. Knowledge Representation for Model Management Systems. *IEEE Transactions on Software Engineering*, SE-10 (6): 619-628, Nov. 1984.

DOLK, D.R.; KOTTEMANN, J.E. Model Integration and a Theory of Models. *Decision Support Systems*, 9 (1): 51-63, Jan. 1993.

DOLK, D.R. An Introduction to Model Integration and Integrated Modeling Environments. *Decision Support Systems*, 10: 249-254, 1993.

DOLK, D.R. Model Management Systems for Operations Research: A Prospectus. In: Mitra, G. (ed.), *Mathematical Models for Decision Support*, NATO ASI Series, vol. 48, Springer-Verlag, Berlin, p. 348-373, 1988(a).

DOLK, D.R. Relational Data Management and Modeling Systems: A Tutorial. In: Mitra, G. (ed.), *Mathematical Models for Decision Support*, NATO ASI Series, vol. 48, Springer-Verlag, Berlin, p. 323-345, 1988(b).

DOMSCHKE, W.; DREXL, A. Location and Layout Planning: An International Bibliography. In: *Notes in Economics and Mathematical Systems*, vol. 238, Springer-Verlag, Berlin, 1985.

DOUKIDIS, G.I.; PAUL, R.J. (eds.) *Artificial Intelligence in Operational Research*. The MacMillan Press, London, UK, 1992.

DRUD, A.S. Interfaces Between Modeling Systems and Solution Algorithms. In: Mitra, G. (ed.), *Mathematical Models for Decision Support*, NATO ASI Series, vol. 48, Springer-Verlag, Berlin, 1988. p. 187-196.

DYER, M.E. Calculating Surrogate Constraints. *Mathematical Programming*, 19: 255-278, 1980.

unesp

EFROYMSON, M.A.; RAY, T.L. A Branch-and-Bound Algorithm for Plant Location. *Operations Research*, 14: 361-368, 1966.

EL-NAJDAWI, M.K.; STYLIANOU, A.C. Expert Support Systems: Integrating AI Technologies. *Communications of the ACM*, 36 (12): 55-65(103), Dec. 1993.

ERLENKOTTER, D. A Dual-Based Procedure for Uncapacitated Facitity Location. *Operations Research*, 26: 992-1009, 1978.

FISCHBECK, P. GIS: More than a Map. *OR/MS Today*, 42-45, Aug. 1994.

FORSTER, M.; MEVERT, P. A Tool for Network Modeling. *European Journal of Operational Research*, 72 (2): 287-299, Jan. 1994.

FOURER, R.; GAY, D.M.; KERNIGHAN, B.W. A Modeling Language for Mathematical Programming. *Management Science*, 36 (5): 519-554, May, 1990.

FOURER, R.; GAY, D.M. *Expressing Special Structures in an Algebraic Modeling Language for Mathematical Programming*. Technical Report 91-09, Department of Industrial Engineering and Management Sciences, Northwestern University, May, 1993.

FOURER, R.; GAY, D.M.; KERNIGHAN, B.W. *AMPL - A Modeling Language for Mathematical Programming*. The Scientific Press, South San Francisco, CA, 1993.

FOURER, R. A Simplex Algorithm for Piecewise-Linear Programming I: Derivation and Proof. *Mathematical Programming*, 33 (2): 204-233, 1985.

FOURER, R. A Simplex Algorithm for Piecewise-Linear Programming II: Finiteness, Feasibility and Degeneracy. *Mathematical Programming*, 41 (3): 281-315, 1988.

FOURER, R. A Simplex Algorithm for Piecewise-Linear Programming III: Computational Analysis and Applications. *Mathematical Programming*, 53 (2): 213-235, 1992.

FREVILLE, A.; LORENA, L.A.N.; PLATEAU, G. *New Subgradient Algorithms for the 0-1 Multiknapsack Lagrangian and Surrogate Duals*. Pre-publication #90. LIPN-Universite Paris Nord, 1990.

GALVÃO, R.D.; RAGGI, L.A. A Method for Solving to Optimality Uncapacitated Location Problems. *Annals of Operations Research*, 18: 225-244, 1989.

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

GEOFFRION, A.M. *An Informal Annotated Bibliography on Structured Modeling*. Working Paper No. 390, Western Management Science Institute, University of California, Los Angeles, Feb. 1994.

GEOFFRION, A.M. An Introduction to Structured Modeling. *Management Science,* 33 (5): 547-588, May, 1987.

GEOFFRION, A.M. Indexing in Modeling Languages for Mathematical Programming. *Management Science*, 38 (3): 325-344, Mar. 1992(c).

GEOFFRION, A.M. Lagrangean Relaxation for Integer Programming. *Mathematical Programming Study*, 2: 82-114, 1974.

GEOFFRION, A.M. The Formal Aspects of Structured Modeling. *Operations Research,* 37 (1): 30-51, Jan.-Feb. 1989.

unesp

GEOFFRION, A.M. The SML Language for Structured Modeling: Levels 1 and 2. *Operations Research*, 40 (1): 38-57, Jan.-Feb. 1992(a).

GEOFFRION, A.M. The SML Language for Structured Modeling: Levels 3 and 4. *Operations Research*, 40 (1): 58-75, Jan.-Feb. 1992(b).

GLOVER, F.; KLINGMAN, D.; PHILIPS, N. Netform Modeling and Applications. *Interfaces*, 20 (4): 7-27, Jul.-Aug. 1990.

GLOVER, F.; KLINGMAN, D.; PHILLIPS, N.V. *Network Models in Optimization and Their Applications in Practice*. New York, NY, John Wiley & Sons, 1992.

GLOVER, F. Surrogate Constraints. *Operations Research*, 16 (4): 741-749, 1968.

GREENBERG, H.J.; LUCAS, C.; MITRA, G. Computer-Assisted Modelling and Analysis of Linear Programming Problems: Towards a Unified Framework, *IMA Journal of Mathematics in Management,* 1: 251-265, 1987.

GREENBERG, H.J.; LUNDGREN, J.R.; MAYBEE, J.S. Extensions of Graph Inversion to Support an Artificially Intelligent Modeling Environment. *Annals of Operations Research*, 21: 127-142, 1989.

GREENBERG, H.J.; MURPHY, F.H. A Comparison of Mathematical Programming Modeling Systems, *Annals of Operations Research*, 38: 177-238, 1992.

GREENBERG, H.J. A Functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models, *ACM Transactions on Mathematical Software,* 9 (1): 18-56, 1983.

GREENBERG, H.J. Enhancements of ANALYZE: A Computer-Assisted Analysis System of Linear Programming. *ACM Transactions on Mathematical Software*, 19 (2): 233-256, June, 1993.

GREENBERG, H.J. Syntax-Directed Report Writing in Linear Programming using ANALYZE. *European Journal of Operations Research*, 72 (2): 300-311, Jan. 1994.

GUIGNARD-SPIELBERG, M.; KIM, S. *A Strong Lagrangean Relaxation for Capacitated Plant Location Problems*. Department of Statistics Technical Report #56, The Wharton School, University of Pennsylvania, PA, 1983.

GUINARD, M. A Lagrangean Dual Ascent Algorithm for the Simple Plant Location Problem. *European Journal of Operational Research*, 35: 193-200, 1988.

HADJICONSTANINOU, E.; MITRA, G. A Linear and Discrete Programming Framework for Representing Qualitative Knowledge. *Journal of Economic Dynamics and Control*, 18: 273-297, 1994.

HAKIMI, S.L. Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems. *Operations Research*, 13: 462-475, 1965.

HAKIMI, S.L. Optimum Location of Switching Centers and the Absolute Centers and the Medians of a Graph. *Operations Research*, 12: 450-459, 1964.

HANDLER, G.; ZANG, I. A Dual Algorithm for the Constrained Shortest Path Problem. *Networks*, 10: 293-310, 1980.

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

HAYES-ROTH, F.; WATERMAN, D.A.; LENAT, D.B. *Building Expert Systems*. Reading, MA, Addison-Wesley, 1985.

HILLIER, F.S.; LIEBERMAN, G.J. *Introdução à Pesquisa Operacional*. Rio de Janeiro, RJ, Campus, 1988.

HOROWITZ, E.; SAHNI, S. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM*, 21: 277-292, 1974.

IBM Corporation. *IBM Mathematical Programming System Extended /370* (MPSX/370). Program Reference Manual, SH19-1095-1, New York, NY, 1976.

JACOBSEN, S.K. Heuristics for the Capacitated Plant Location Model. *European Journal of Operational Research*, 12: 253-261, 1983.

JARVINEN, P.J.; RAJALA, J.A Branch and Bound Algorithm for Seeking the P-Median. *Operations Research*, 20: 173-178, 1972.

JONES, C.V. An Introduction to Graph-Based Modeling Systems, Part I: Overview. *ORSA Journal on Computing*, 2 (2): 136-151, Spring 1990.

JONES, C.V. An Introduction to Graph-Based Modeling Systems, Part II: Graph-Grammars and the Implementation. *ORSA Journal on Computing*, 3 (3): 180-206, Summer 1991.

JONES, C.V. Visualization and Optimization. *ORSA Journal on Computing*, 6 (3): 221-257, Summer 1994.

KARWAN, M.L.; RARDIN, R.L. Some Relationships Between Lagrangean and Surrogate Duality in Integer Programming. *Mathematical Programming*, 17: 320-334, 1979.

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

KORTH, H.F.; SILBERSCHATZ, A. Database System Concepts. Singapore, McGraw-Hill, 1991.

KOTTEMANN, J.E.; DOLK, D.R. Model Integration and Modeling Languages: A Process Perspective. *Information Systems Research*, 3 (1): 1-16, Mar. 1992.

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

KRISHNAN, R.; LI, X.; STEIER, D. A Knowledge-Based Mathematical Model Formulation System. *Communications of the ACM*, 35 (9): 138-146, Sep. 1992.

KRISTJANSSON, B.; LUCAS, C.; MITRA, G.; MOODY, S. *Sets and Indices in Linear Programming Modelling and their Integration with Relational Data Models*. Technical Report 02/93, Department of Mathematics and Statistics, Brunel University, May, 1993.

KRISTJANSSON, B. *MPL Modelling System - Release 3.0*, Maximal Software Ltd., Iceland, 1993.

KUIP, C.A.C. Algebraic Languages for Mathematical Programming. *European Journal of Operational Research*, 67 (1): 25-51, May 1993.

LAPORTE, G. Location-Routing Problems. In: GOLDEN, B.L.; ASSAD, A.A. *Vehicle Routing: Methods and Studies*. Amsterdam, Netherlands, Elsevier Science, p. 163-197, 1988.

unesp

LENARD, M.L. An Object-Oriented Approach to Model Management. *Decision Support Systems*, 9 (1): 67-73, Jan. 1993.

LENARD, M.L. Structured Model Management. In: Mitra, G. (ed.), *Mathematical Models for Decision Support*, NATO ASI Series, vol. 48, Springer-Verlag, Berlin, p. 375-391, 1988.

LI, L.Y.O.; EGLESE, W. An Interactive Algorithm for Vehicle Routeing for Winter-Gritting. *Jounal of the Operational Research Society*, 47 (2): 217-228, 1996.

LORENA, L.A.N.; LOPES, F.B. A Surrogate Heuristic for Set Covering Problems. *European Journal of Operational Research*, 79: 138-150, 1994.

LORENA, L.A.N.; NARCISO, M.G. Relaxation Heuristics for Generalized Assignment Problems. *European Journal of Operational Research,* 91: 600-610, 1996.

LORENA, L.A.N.; SENNE, E.L.F. A Lagrangean/Surrogate Heuristic for Uncapacitated Facility Location Problems. In: *Latin-Iberian-American Congress on Operations Research and System Engineering, 8 / Simpósio Brasileiro de Pesquisa Operacional, 28*. Rio de Janeiro, RJ, Anais, p. 854-859 (Vol. 2), Ago. 1996(a).

LORENA, L.A.N.; SENNE, E.L.F. A Lagrangean/Surrogate Algorithm for a Capacitated Facility Location Problem. Submetido para publicação em: *Location Science*. Dez. 1996(b).

LUCAS, C.; MITRA, G. Computer-Assisted Mathematical Programming (Modelling) System: CAMPS, *The Computer Journal,* 31 (4): 364-375, Aug. 1988.

LUCAS, C.; MITRA, G. *Modelling of Mathematical Programs: An Analysis of Strategy and an Outline Description of a Computer Assisted System*. Student Notes, Department of Mathematics and Statistics, Brunel University, England, UK, Oct. 1983.

LUCAS, C.; MITRA, G.; MOODY, S.; KRISTJANSSON, B. *Introducing New Constructs for Data Modelling and Column Generation in LP Modelling Languages*. Technical Report 11/93, Department of Mathematics and Statistics, Brunel University, England, UK, Oct. 1993.

LUCAS, C.A. *An Investigation of Computer Based Tools for Mathematical Programming Modelling*, PhD Thesis, Department of Mathematics and Statistics, Brunel University, England, UK, 1986.

MA, P.; MURPHY, F.; STOHR, E. A Graphics Interface for Linear Programming. *Communications of the ACM*, 32 (8): 996-1012, 1989.

MARANZANA, F.E. On the Location of Supply Points to Minimize Transport Costs. *Operations Research Quarterly*, 15: 261-27, 1964.

MARCHEZAN, A.R.; NAKAGAWA, E.C.; SENNE, E.L.F. Interface Gráfica para Especificação de Modelos de Programação Linear. In: *Simpósio de Iniciação Científica da Universidade de São Paulo, 3*. São Carlos, SP, Resumos, p. 246 (Vol. 2), Dez. 1995.

MARINS, F.A.S.; PERIN, C. Strong Feasibility in Network Piecewise-Linear Programs. In: *Latin-Iberian-American Congress on Operations Research and System Engineering, 4*. Proceedings, p. 364-383, Rio de Janeiro, RJ, 1988.

unesp

MARINS, F.A.S.; PERIN, C. Programação Linear por Partes: Revisão Teórica e Aplicações. *Produção*, 6 (2): 146-163, Dez. 1996.

MARINS, F.A.S.; SENNE, E.L.F.; DARBY-DOWMAN, K.; MACHADO, A.F.; PERIN, C. Algorithms for Network Piecewise-Linear Programs: A Comparative Study. *European Journal of Operational Research*, 97: 183-199, 1997.

MARINS, F.A.S. *Algoritmos para Redes Lineares por Partes: Um Estudo Comparativo Estatisticamente Planejado*. Tese de Livre Docência, Unesp, Guaratinguetá, SP, Ago. 1996.

MARINS, F.A.S. *Estudos sobre Programas de Redes Lineares por Partes*. Tese de Doutorado, Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, SP, 1987.

MARINS, F.A.S., DARBY-DOWMAN, K.; SENNE, E.L.F.; MACHADO, A.F.; PERIN, C. *Algorithms for Network Piecewise-Linear Programs: A Statistically Designed Study*. Technical Report TR/20/94, Department of Mathematics and Statistics, Brunel University, England, UK, 1994.

MAROS, I. Performance Evaluation of the MINET - Minimum Cost Netflow Solver. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 12: 199-217, 1993.

MARTELLO, S.; TOTH, P. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, New York, 1990.

MATURANA, S.V. Issues in the Design of Modeling Languages for Mathematical Programming. *European Journal of Operational Research*, 72 (2): 243-261, Jan. 1994.

McBRIDE, R.D.; O'LEARY, D.E. The Use of Mathematical Programming with Artificial Intelligence and Expert Systems. *European Journal of Operations Research*, 70 (1): 1-15, Oct. 1993.

MENTZAS, G. Towards Intelligent Organisational Information Systems. *International Transactions in Operational Research*, 1 (2): 169-187, 1994

MINOUX, M. Plus Courts Chemins avec Constraints: Algorithmes et Applications. *Annals of Telecommunications*, 30: 383-394, 1975.

MITRA, G.; LUCAS, C.; MOODY, S.; HADJICONSTANINOU, E. Tools for Reformulating Logical Forms into Zero-One Mixed Integer Programs. *European Journal of Operations Research*, 72 (2): 262-276, Jan. 1994.

MITRA, G. Models for Decison Making: An Overview of Problems, Tools and Major Issues. In: Mitra, G. (ed.), *Mathematical Models for Decision Support*, NATO ASI Series, vol. 48, Springer-Verlag, Berlin, p. 17-53, 1988.

MOODY, S.A. *Methods and Tools for Modelling Linear and Integer Programming Problems*, PhD Thesis, Department of Mathematics and Statistics, Brunel University, England, UK, 1994.

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

MOOKERJEE, V.S.; CHATURVEDI, A.R. A Blackboard Control Architecture for Model Selection and Sequencing. *European Journal on Information Systems*, 2 (1): 3-14, Jan. 1993.

MUHANNA, W.A. SYMMS: A Model Management System that Supports Model Reuse, Sharing, and Integration. *European Journal of Operational Research*, 72 (2): 214-243, Jan. 1994.

MURPHY, F.H.; STOHR, E.A.; ASTHANA, A. Representation Schemes for Linear Programming Models. *Management Science,* 38 (7): 964-991, July 1992(a).

MURPHY, F.H.; STOHR, E.A.; MA, P.C. Composition Rules for Building Linear Programming Models from Component Models. *Management Science,* 38 (7): 948-963, July 1992(b).

MURPHY, F.H. A KnowledgeBase for Formulating Linear Programs. In: Mitra, G. (ed.), *Mathematical Models for Decision Support*, NATO ASI Series, vol. 48, Springer-Verlag, Berlin, p. 451-470, 1988.

MURTY, K.G. *Network Programming*. Englewood Cliffs, NJ, Prentice Hall, 1992.

NAKAGAWA, E.C.; MARCHEZAN, A.R.; SENNE, E.L.F. IGPL: Uma Interface Gráfica para Especificação de Modelos de Programação Linear. In: *Congresso de Iniciação Científica da Unesp, 8*. Guaratinguetá, SP. Resumos, p. 12, Dez. 1996.

NARCISO, M.G. *Novas Heurísticas para o Problema Generalizado de Atribuição*. Dissertação de Mestrado, INPE, São José dos Campos, SP, 1994.

NAUSS, R.M. An Improved Algorithm for the Capacitated Facility Location Problem. *Journal of the Operational Research Society*, 29: 1195-1201, 1978.

NEEBE, A.W. A Branch and Bound Algorithm for the P-Median Transportation Problem. *Journal of the Operational Research Society*, 29: 989-995, 1978.

PARKER, R.G.; RARDIN, R.L. *Discrete Optimization*. Academic Press, New York, 1988.

POTTER, W.D.; BYRD, T.A.; MILLER, J.A.; KOCHUT, K.J. Extending Decision Support Systems: The Integration of Data, Knowledge and Model Management. *Annals of Operations Research*, 38 (1-4): 501-527, Dec. 1992.

RAGHUNATHAN, S. A Planning Aid: An Intelligent Modeling Systems for Planning Problems Based on Constraint Satisfaction. *IEEE Transactions on Knowledge and Data Engineering*, 4 (4): 317-335, Aug. 1992.

SANTHANAM, R.; SCHNIEDERJANS, M.J. A Model Formulation System for Information System Project Selection. *J. Computers & Ops. Res.*, 20 (7): 755-767, Sep. 1993.

SEN, A.; BHARADWAJ, A.S.; VINZE, A.S. Use of Reason Maintenance in Model Formulation. *OMEGA*, 21 (1): 123-133, Jan. 1993.

SEN, A.; LIOU, S.T.; VINZE, A.S. AEROBA: A Blackboard Approach to Model Formulation. *JMIS*, 9 (3): 123-143, Winter 1992-93.

SENNE, E.L.F. *Ambiente de Inteligência Artificial para Solução de Modelos de Otimização em Redes*. Relatório Final de Atividades (Fapesp proc. 92/2860-9), Unesp, Guaratinguetá, SP, 1995.

unesp

SENNE, E.L.F.; LUCAS, C.; TAYLOR, S. Towards an Intelligent Graphical Interface for Linear Programming Modelling. *Journal of Intelligent Systems*, 6 (1): 63-94, 1996.

SENNE, E.L.F.; MARINS, F.A.S.; PERIN, C. *NplSys - Network Piecewise-Linear Testing System: User's Guide*. Relatório Técnico TR/19/94, Department of Mathematics and Statistics, Brunel University, England, UK, Nov. 1994.

SENNE, E.L.F.; LORENA, L.A.N Lagrangean/Surrogate Heuristics for Facility Location Problems. Aceito para apresentação em: *EURO XV-INFORMS XXXIV*, Barcelona, Spain, July, 1997(a).

SENNE, E.L.F.; LORENA, L.A.N A Lagrangean/Surrogate Aproach to p-Median Problems. Submetido para publicação em: *European Journal of Operational Research*, May, 1997(b).

SENNE, E.L.F. *Motor de Inferência para Sistemas Especialistas*. Tese de Doutorado em Computação Aplicada, Relatório Técnico INPE-4465-TDL/321, São José dos Campos, SP, Jan. 1988.

SHAPIRO, J.F. *Mathematical Programming: Structures and Algorithms*. New York, NY, Wiley-Interscience, 1979.

STEIGER, D.; SHARDA, R. LP Modeling Languages for Personal Computers: A Comparison. *Annals of Operations Research*, 43 (1-4): 195-216, 1993.

STEIGER, D.; SHARDA, R.; LECLAIRE, B. Graphical Interfaces for Network Modeling: A Model Management System Perspective. *ORSA Journal on Computing*, 5 (3): 275-291, Summer 1993.

STOHR, E.A. *Automated Support for Formulating Linear Programs*. Working Paper GBA 88-4, Graduate School of Business Administration, New York University, Feb. 1988.

SUHL, U.H. MOPS - Mathematical OPtimization System. *European Journal of Operations Research*, 72 (2): 312-322, Jan. 1994.

TEITZ, M.B.; BART, P. Heuristic Methods for Estimating the Vertex Median of a Weighted Graph. *Operations Research*, 16: 955-961, 1968.

VELASCO, F.R.D. *Linguagem OPS5 para Microcomputadores - Versão 1.0: Manual do Usuário*. S.J.Campos, SP, INPE/DPI, Out. 1988.

WILLIAMS, H.P. *Model Building in Mathematical Programming*. Chichester, UK, John Wiley & Sons, 1993.

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

Apêndice A

Gramática da Linguagem NPL2

Apresenta-se a seguir, a gramática da linguagem NPL2 (*Network Piecewise-Linear Programming Language*). Deve-se observar que na descrição desta gramática são adotadas as seguintes convenções:

- símbolos terminais da linguagem são escritos em fonte de caracteres igualmente espaçados e em negrito (como em: **`model`**);

- símbolos não-terminais da linguagem são escritos em itálico (como em: *modelo*);

- as partes da gramática delimitadas por colchetes são opcionais. Assim, a produção:

  declaração-conjunto → *identificador* [**:=** *instância-conjunto*]

  deve ser entendida como: a declaração de um conjunto deve se dar por meio de um identificador, o qual pode ser seguido, opcionalmente, pelo símbolo de atribuição (**:=**) e da declaração de uma instância de conjunto.

- as partes da gramática separadas pelo símbolo | correspondem a produções alternativas do símbolo não terminal que aparece à esquerda do símbolo de derivação (→).

NPL2 - Gramática da Linguagem

modelo → **`model`** *identificador* **;**
lista-de-declarações
lista-de-componentes
objetivo **.**

lista-de-declarações → *declaração* **;**
| *declaração* **;** *lista-de-declarações*

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

lista-de-componentes → *declaração-de-componente* **;**
| *declaração-de-componente* **;** *lista-de-componentes*

objetivo → **minimize** *identificador*
| **maximize** *identificador*

declaração → **set** *declaração-de-conjuntos*
| **par** *declaração-de-parâmetros*

declaração-de-conjuntos → *declaração-conjunto* **;**
| *declaração-conjunto* **;** *declaração-de-conjuntos*

declaração-de-parâmetros → *declaração-parâmetro* **;**
| *declaração-parâmetro* **;** *declaração-de-parâmetros*

declaração-conjunto → *identificador* [**:=** *instância-conjunto*]
[**where** *expressão-lógica*]

instância-conjunto → *identificador*
| **{** *lista-de-objetos* **}**
| **[** *expressão-aritmética* **..** *expressão-aritmética*
[**step** *expressão-aritmética*] **]**
| *tabela*
| **read**
| *valor-conjunto* *operador-conjunto* *valor-conjunto*
| **(** *valor-conjunto* **)**

declaração-parâmetro → [*tipo-parâmetro*] *identificador-componente*
[**:=** *valor-parâmetro*] [**where** *expressão-lógica*]

tipo-parâmetro → **integer** | **real** | **logical**

identificador-componente → *identificador* [**[** *expressão-domínio* **]**]

valor-parâmetro → *expressão-aritmética*
| **(** *lista-de-expressões* **)**
| *coluna-de-tabela*
| **read**

lista-de-objetos → *identificador* [**[** *lista-de-identificadores* **]**]
| *constante-numérica*
| *tupla*

tupla → **<** *lista-de-indexação* **>**

lista-de-domínios → *expressão-domínio*

cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

| *expressão-domínio* **,** *lista-de-domínios*

expressão-domínio → *expressão-de-indexação*
| *constante-inteira* **..** *expressão-aritmética*

expressão-de-indexação → *identificador* **in** *identificador-componente*
| **<** *lista-de-identificadores* **>in** *identificador-componente*

lista-de-identificadores → *identificador*
| *identificador* **,** *lista-de-identificadores*

tabela → **table (** *identificador [,]* **key** *lista-de-identificadores* **)**
| **forsome (** *tupla , identificador* **key** *lista-de-identificadores*
operador-relacional identificador **key** *lista-de-identificadores* **)**

lista-de-indexação → *expressão-de-indexação*
| *expressão-de-indexação* **,** *lista-de-indexação*

coluna-de-tabela → **table (** *identificador [,]* **data** *identificador* **)**
| **forsome (** *tupla , identificador* **key** *lista-de-identificadores*
operador-relacional identificador **key** *lista-de-identificadores*
[,] **data** *identificador* **)**

declaração-de-componente → *declaração-de-nós*
| *declaração-de-arcos*

declaração-de-nós → **node** *lista-de-nós*

lista-de-nós → *declaração-nó* **;**
| *declaração-nó* **;** *lista-de-nós*

declaração-de-arcos → **arc** *lista-de-arcos*

lista-de-arcos → *declaração-arco* **;**
| *declaração-arco* **;** *lista-de-arcos*

declaração-nó → *identificador-componente [* **where** *condição-nó]*

condição-nó → *expressão-balanço operador-relacional expressão-aritmética*
| **not** *condição-nó*
| *condição-nó* **and** *condição-nó*
| *condição-nó* **or** *condição-nó*
| **(** *condição-nó* **)**

expressão-balanço → *operador-balanço*
| *operador-balanço operador-aditivo expressão-aritmética*

unesp

declaração-arco → *identificador-componente expressão-arco [opções-arco]*
[**where** *expressão-lógica]*

expressão-arco → **from** *expressão-componente*
| **to** *expressão-componente*
| **from** *expressão-componente* **to** *expressão-componente*

opções-arco → *limites-arco*
| **obj** *coeficiente-objetivo*

limites-arco → **bounds <** *expressão-aritmética* **:** *expressão-aritmética* **>**
| **bound** *operador-relacional expressão-aritmética*

coeficiente-objetivo → *expressão-aritmética*
| **<** *lista-de-expressões* **:** *lista-de-expressões* **>**

lista-de-expressões → *expressão-aritmética*
| *expressão-aritmética* **,** *lista-de-expressões*

expressão-lógica → *expressão-aritmética operador-relacional expressão-aritmética*
| **not** *expressão-lógica*
| *expressão-componente* **in** *valor-conjunto*
| *expressão-lógica* **and** *expressão-lógica*
| *expressão-lógica* **or** *expressão-lógica*
| **(** *expressão-lógica* **)**

expressão-componente → *identificador [* **[** *lista-de-expressões* **]** *]*
| **<** *lista-de-expressões* **>**

expressão-aritmética → *operador-aditivo termo*
| *termo operador-aditivo termo*

termo → *fator operador-multiplicativo fator*

fator → *primário* **^** *primário*

primário → *constante-numérica*
| *expressão-componente*
| **if (** *expressão-lógica* **) then** *expressão-aritmética*
[**else** *expressão-aritmética]*
| *somatória*
| *função-intrínsica* **(** *lista-de-expressões* **)**
| **(** *expressão-aritmética* **)**

somatória → **sum (** *expressão-de-indexação [* **,** *] expressão-aritmética*
[**where** *expressão-lógica]* **)**

função-intrínsica → **max** | **min** | **ceil** | **floor** | **next** | **pred**

operador-conjunto → **union** | **inter** | **diff** | **cross**

operador-relacional → **=** | **>** | **<** | **>=** | **<=** | **<>**

operador-balanço → **net_in** | **net_out**

operador-aditivo → **+** | **-**

operador-multiplicativo → ***** | **/**

Apêndice B

Algoritmo de Interpretação de Restrições

Apresenta-se, a seguir, a listagem de um programa Lisp que implementa o Algoritmo de Interpretação de Restrições.

```
;;; Auxiliary Functions
;;; ===================

(defun difference(X Y)
;;;     ==========
;;; X and Y are lists representing sets.
;;; Returns a list of elements of X which are not in Y.
  (cond
    ((null X) NIL)
    ((member (car X) Y)(difference (cdr X) Y))
    (T (cons (car X)(difference (cdr X) Y)))))

(defun contains(X Y)
;;;     ========
;;; X and Y are lists representing sets.
;;; Returns T if X contains Y.
  (cond
    ((null Y) T)
    ((member (car Y) X)(contains X (cdr Y)))
    (T NIL)))

(defun list-to-string(X)
;;;     ==============
;;; X is a list.
;;; Returns a string from the elements of X separated by commas.
  (let ((S ""))
    (loop
      (cond
        ((null X)(return S))
        (T (progn
            (setq S (concatenate 'string S (string (car X))))
            (setq X (cdr X))
            (cond ((not (null X))
                   (setq S (concatenate 'string S ","))))))))))

(defun all-but-first(X)
;;;     =============
;;; X is a string "compatibility relationship".
;;; Returns X without the first name before the word "IN".
  (coerce
    (subseq (coerce X 'list)(position #\Space (coerce X 'list)))
    'string))

;;; Main Functions
;;; ==============
```

unesp

```
(defun primary(P)
;;;     =======
;;; P is a constraint piece.
  (let ((a (assoc P Data-Model)))
    (cond
      ((null a) NIL)
      ((null (cdr a)) (list P))
      (T (remove-duplicates (mapcan 'primary (cdr a)))))))

(defun index(P)
;;;     =====
;;; P is a constraint piece.
  (let ((a (assoc P DM-Attributes)))
    (cond
      ((null a) (primary P))
      (T (primary (cadr a))))))

(defun index-t(L)
;;;     =======
;;; L is a lhs-term of a constraint.
;;; Returns Index[L].
  (remove-duplicates (union (index (car L))(index (cadr L)))))

(defun l-index(C)
;;;     =======
;;; C is a constraint.
  (remove-duplicates
    (mapcan '(lambda(p)
        (append (index (car p)) ;; Indices of a lhs-term.
                (index (cadr p))))
      (car C)))) ;; Left-hand side of the constraint.

(defun r-index(C)
;;;     =======
;;; C is a constraint.
  (index
    (caddr C))) ;; Right-hand side of the constraint.

(defun n-index(C)
;;;     =======
;;; C is a constraint.
  (difference
    (r-index C)(l-index C)))

(defun rconsistency(I D)
;;;     ============
;;; I is a subscript, D is a DM-object.
  (progn
    (setf (get I 'd-rconsistency) D)
    (setf (get I 'rconsistency)
      (list
        (concatenate 'string
          (string I) " IN " (string D)
          "[" (string-right-trim " "
              (list-to-string (primary D))) "]")))))

(defun dm-consistency(L)
;;;     ==============
;;; L is a lhs-term of a constraint.
;;; Returns a DM-object D such as Primary[D] contains Index[L].
  (let
    ((DM Data-Model)
     (IL (index-t L)))
    (loop
      (cond
```

```
        ((null DM)
         (error (concatenate 'string
             "No DM-object for consistency of "
             (string (car L)) " & " (string (cadr L)))))
        ((contains (primary (caar DM)) IL)(return (caar DM))))
      (setq DM (cdr DM)))))

(defun f-cons(I P)
;;;    ======
;;; I is a subscript; P is a constraint piece.
  (cond
    ((not (member I (index P))) NIL)
    ((null (get I 'rconsistency))(list i))
    (T (get I 'rconsistency))))

(defun fconsistent(S P)
;;;    ===========
;;; S is a set of subscripts; P is a constraint piece.
  (cond
    ((null S) NIL)
    (T (union (f-cons (car S) P)(fconsistent (cdr S) P)))))

(defun expansion(P)
;;;    =========
;;; P is a constraint piece.
  (cond
    ((null (index P))(write-to-string P))
    (T (concatenate 'string
        (string P) "[" (string-right-trim " "
          (list-to-string (fconsistent (index P) P))) "]"))))

(defun expansion-t(L)
;;;    ===========
;;; L is a list (A X) representing a lhs-term of a constraint.
;;; Returns expansion(A)*expansion(X).
  (concatenate 'string
    (expansion (car L)) "*" (expansion (cadr L))))

(defun p-index(C)
;;;    =======
;;; C is a constraint.
  (mapcan
    '(lambda(x)
      (cond
        ((null (get x 'rconsistency)) NIL)
        (T (list x)))) (l-index C)))

(defun l-free(C)
;;;    ======
;;; C is a constraint.
  (difference
    (l-index C)(p-index C)))

(defun p-summed(C)
;;;    ========
;;; C is a constraint.
  (difference
    (l-free C)(r-index C)))

(defun c-index(C)
;;;    =======
;;; C is a constraint.
  (mapcan
    '(lambda(x)
      (cond
        ((null (get x 'rconsistency)) NIL)
```

```
        (T (list x)))) (p-summed C)))

(defun rcompatibility(I D J)
;;;    ==============
;;; I is a subscript; D is a DM-object; J belongs to Primary[D].
  (progn
    (setf (get I 'd-rcompatibility) D)
    (setf (get I 'j-rcompatibility) J)
    (setf (get I 'rcompatibility)
      (list
        (concatenate 'string
          (string J) " IN " (string D)
          "[" (string-right-trim " "
              (list-to-string (primary D))) "]")))))

(defun d-compatible(I C)
;;;    ============
;;; I is a subscript; C is a constraint.
;;; Returns a list (J D) where:
;;;   J belongs to L-Free[C];
;;;   D is a DM-object such as the intersection of Primary[D] and
;;;   L-Free[C] is not empty.
  (let
    ((DM Data-Model)
     (LF (l-free C)))
    (loop
      (cond
        ((null DM)
         (error "No DM-object for constraint compatibility."))
        ((and
           (member I (primary (caar DM)))
           (intersection (primary (caar DM)) LF))
         (return (list (j-compatible (caar DM)(l-free C))(caar DM)))))
      (setq DM (cdr DM)))))

(defun j-compatible(D F)
;;;    ============
;;; D is a DM-object; F is the set L-Free of a constraint.
;;; Selects the first element of the intersection of Primary[D] and F.
  (car (intersection (primary D) F)))

(defun w-clause(S)
;;;    ========
;;; S is a subscript.
  (list
    (concatenate 'string
      (string S) " = " (string S)
      (all-but-first (car (get S 'rcompatibility))))))

(defun where-clause(S)
;;;    ============
;;; S is a set of subscripts.
  (cond
    ((null S) NIL)
    (T (progn
        (setf (get (car S) 'projected)(car (w-clause (car S))))
        (where-clause (cdr S))))))

(defun cis-step1(C)
;;;    =========
;;; C is a constraint.
;;; Performs the Step 1 of the Constraint Interpretation Algorithm.
  (mapc
    '(lambda(L) ;; L is a lhs-term of C.
      (cond
        ((null (index (car L))) NIL)
```

```
        ((null (intersection (index (car L))(index (cadr L))))
         (let
           ((I (car (index-t L))))
           (rconsistency I (dm-consistency L))))))
    (car C)))

(defun cis-step2(C)
;;;    =========
;;; C is a constraint.
;;; Performs the Step 2 of the Constraint Interpretation Algorithm.
  (let ((NI (n-index C)))
    (loop
      (cond
        ((null NI)(return T))
        (T (let ((DJ (d-compatible (car NI) C)))
            (rcompatibility (car NI)(cadr DJ)(car DJ))
            (setq NI (cdr NI))))))))

(defun cis-step3(C)
;;;    =========
;;; C is a constraint.
;;; Performs the Step 3 of the Constraint Interpretation Algorithm.
;;; Returns a list (S W), where:
;;;   S is the list of summation indices of C;
;;;   W is the list of where-clauses indices of C.
  (let (
     (S NIL)
     (W NIL)
     (PS (p-summed C))) ;; PS is the set of possible summed indices of C.
    (let ((NI (n-index C)))
      (cond
        ((null NI) T)
        (T (let (
             (I (car NI))
             (NP NIL)) ;; Stores the non-projected form of an index.
            (setq NP (get I 'j-rcompatibility))
            (setq S (union S (list NP)))
            (setf (get NP 'projected) (car (get I 'rcompatibility)))
            (setq W (difference NI (list I)))
            (where-clause W)
            (setq PS (difference PS
                       (primary (get I 'd-rcompatibility))))))))
    (loop
      (let ((CI (c-index C)))
        (cond
          ((null CI)(return T))
          (T (progn
              (setq S (union S (get (car CI) 'rconsistency)))
              (setq PS (difference PS
                  (primary (get (car CI) 'd-rconsistency))))))
        (setq CI (cdr CI))))
    (setq S (union S PS))
    (list S W)))

(defun sum-string(X L)
;;;    ==========
;;; X is a list of indices; L is a lhs-term of a constraint;
;;; Returns the Sum-String for the lhs-term L.
  (let (
     (S "SUM(")
     (XL (intersection X (index-t L))))
    (cond ((null XL) "")
      (T (loop
          (cond ((null XL)(return S)))
          (cond
            ((get (car XL) 'projected)
```

```
              (setq S (concatenate 'string S (get (car XL) 'projected))))
             (T (setq S (concatenate 'string S (string (car XL))))))
            (setq XL (cdr XL))
            (cond ((not (null XL))
                   (setq S (concatenate 'string S ","))))))))))

(defun where-string(X)
;;;    ============
;;; X is a list of indices.
;;; Returns the Where-String.
  (let
    ((W " WHERE "))
    (cond ((null X) "")
      (T (loop
           (cond ((null X)(return W)))
           (cond
             ((get (car X) 'projected)
              (setq W (concatenate 'string W (get (car X) 'projected))))
             (T (setq W (concatenate 'string W (string (car X))))))
           (setq X (cdr X))
           (cond ((not (null X))
                  (setq W (concatenate 'string W ","))))))))))

(defun algebraic-form(C)
;;;    ==============
;;; C is a constraint.
;;; Returns the algebraic form of C.
  (prog (
     SW  ;; SW is the list (S R W) returned by cis-step3;
     LH) ;; LH is the left-hand side of C;
    (cis-step1 C)
    (cis-step2 C)
    (setq SW (cis-step3 C))
    (write-line "Algebraic Form:")
    (setq LH (car C))
    (loop
      (and (null LH)(return T))
      (write-string
        (concatenate 'string
          (sum-string (car SW)(index-t (car LH)))
          (where-string (cadr SW))
          ":" (expansion-t (car LH)) ")"))
      (setq LH (cdr LH))
      (cond ((not (null LH))(write-string "+"))))
    (write-string (cadr C))
    (write-string (expansion (caddr C)))
    (write-line ";")
    (return 'OK)))

;;; Data Structures
;;; ===============

(setq DM-1
  '((company)(station)(route)(site)(fuel)
    (station-owner company station)(station-route station route)
    (site-route route site)(fuel-site site fuel)(pocotak fuel company)
    (company-route station-owner station-route)
    (fuel-route site-route fuel-site)))

(setq DMA-1
  '((heatreq station)(ashipped route)(fuelcal site)(mintake pocotak)))

(setq DM-2
  '((i)(j)(k)
    (fdroute i j)(fcroute i k)(dcroute j k)))
```

```
(setq DMA-2
  '((faccap i)(depcap j)(custreq k)(fd-cost fdroute)(x fdroute)
    (tc-cost fcroute)(x1 fcroute)(dc-cost dcroute)(x2 dcroute)))

(setq Data-Model DM-1)
(setq DM-Attributes DMA-1)

(setq C1-1
  '(((FuelCal AShipped)) "=" HeatReq))

(setq C1-2
  '(((1 AShipped)) ">" MinTake))

(setq C2-1
  '(((1 X)(1 x1)) "<" faccap))
```

Apêndice C

Sistema Especialista em Formulação de Modelos

Apresenta-se, a seguir, a listagem de um protótipo de sistema especialista em formulação de Modelos de PL (a partir de informações do Modelo de Dados e do Modelo Conceitual), escrito na linguagem OPS5.

```
; DECLARATION SECTION
; ===================

; Global Declarations
; -------------------

(vector-attribute
            elem-list)  ; implements lists of elements (arrays), used by
                        ; the Replication Algorithm. Note that a special
                        ; element (NULL) is included as the first element
                        ; in arrays, so that the indices can vary from 1
                        ; to n.


; Classes to Describe the Conceptual Model
; ----------------------------------------

(literalize Activity
            name        ; name of the activity
            class       ; class of the activity
            type)       ; type of the activity

(literalize SubActv
            name        ; name of the sub-activity
            actv        ; activity which contains this sub-activity
            index       ; index related to this sub-activity
            value       ; instanced value of this index
            status)     ; control element

(literalize Attribute
            name        ; name of the attribute
            type        ; type of the attribute
            actv        ; activity for which the attribute belongs to
            sub)        ; sub-activity for which the attribute belongs to

(literalize Edge
            name        ; name of the edge
            actv        ; activity associated with the edge
            class       ; primary class associated with the edge
            inst        ; value of an instance for this class
            bal         ; for which object a balance is required (if any)
            type)       ; type of the oriented edge (1/-1) for balance
```

unesp

```
(literalize Value
            name        ; name of the value
            edge        ; edge which holds the value
            type)       ; type of the value


; Classes to Describe the Data Model (for the Expert System)
; ----------------------------------------------------------

(literalize Object
            name        ; name of the object
            class       ; class of the object
            type)       ; type of the object

(literalize Class
            name        ; name of the class
            index)      ; an index which belongs to this class

(literalize Index
            name        ; name of the index
            type)       ; type of this index

; Classes to Describe the LP Model
; --------------------------------

(literalize Model
            name        ; name of the model
            type        ; type of the model
            descr)      ; description of the model

(literalize Objective
            type)       ; type of the objective-function

(literalize ObjFunct
            name        ; name of a term of the objective-function
            sign        ; sign (1/-1) of this term
            coef        ; coefficient of this term
            dvar        ; variable of this term
            actv        ; activity for which this variable belongs
            sub)        ; sub-activity (locally instanced indices)

(literalize LhsTerm
            name        ; name of the constraint which owns this lhs-term
            edge        ; name of the edge this lhs-term comes from
            sign        ; sign (1/-1) of this lhs-term
            coef        ; coefficient of this lhs-term
            dvar        ; variable of this lhs-term
            status)     ; controls the replication

(literalize RhsTerm
            name        ; name of the rhs-term of a constraint
            oper        ; relational operator of this rhs-term
            rhs)        ; value of this rhs-term

(literalize GInstance
            name        ; constraint this global instance applies to
            index       ; name of the instanced index
            value)      ; value of the instance

(literalize LInstance
            name        ; constraint this local instance applies to
            ident       ; identity of this local instantiation
            index       ; name of the instanced index
            value)      ; value of the instance
```




```
; Classes for the Replication Algorithm
; -------------------------------------

(literalize IGlobal
            cstr        ; name of the constraint (global instantiation)
            name        ; name of the instanced index
            addr        ; element address (initially set out as zero)
            card        ; number of instance values in the array
            elem-list)  ; array of the instance values (begins with NULL)

(literalize Stack
            cstr        ; constraint's identification
            elem-list)  ; list of the array's names

(literalize Contr
            cstr        ; constraint's identification
            num         ; total of arrays to be considered
            top         ; controls the change of instance values
            get)        ; address of the next element (instance value)

(literalize VName
            cstr        ; constraint's identification
            array)      ; name of the current array


; Auxiliary Data Structures for the Expert System's Implementation
; ----------------------------------------------------------------

(literalize a-mpiece name actv)
(literalize b-constr name into status)
(literalize c-constr name attrib)
(literalize s-constr name actv type)
(literalize p-term name actv edge coef dvar status step)
(literalize c-show name term)
(literalize i-local name ident)
(literalize r-clone name cstr)
(literalize c-term name)


; PRODUCTION SECTION: LP MODEL FORMULATION EXPERT SYSTEM
; ======================================================


; Rules to Interpret the Conceptual and Data Models.
; --------------------------------------------------

(p objective-function
;  ------------------
;  Enables to create objective-function terms.
;
     (Attribute ^type << LOSS GAIN >>)
     - (ObjFunct)
-->
     (make o-funct))

(p ofunction-term
;  -------------
;  Creates a term of the objective function.
;
     (o-funct)
     (Attribute ^name <c> ^type << LOSS GAIN >> ^actv <a> ^sub <s>)
     (Attribute ^name <v> ^type VARIABLE ^actv <a>)
-->
     (make ObjFunct
           ^name (genatom)
           ^sign 1
```

```
          ^coef <c>
          ^dvar <v>
          ^actv <a>
          ^sub <s>))

(p ofunction-set-sign
;  -----------------
;  Corrects the sign of a term of the objective function,
;  if there are loss and gain attributes.
;
     (o-funct)
     {(ObjFunct ^sign 1 ^coef <c> ^actv <a>) <obfunct>}
     (Attribute ^name <c> ^type LOSS ^actv <a>)
     (Attribute ^type GAIN)
-->
     (modify <obfunct>
             ^sign -1))

(p ofunction-instance
;  -----------------
;  Searches for locally instanced indices.
;
     (o-funct)
     (ObjFunct ^name <n> ^sub { <s> <> nil })
     (SubActv ^name <s> ^index <i> ^value <v>)
     - (LInstance ^name <n> ^ident OFUNCT ^index <i> ^value <v>)
-->
     (make LInstance
           ^name <n>
           ^ident OFUNCT
           ^index <i>
           ^value <v>))

(p objective-type
;  -------------
;  The objective type is "Minimization" by default.
;  If there is at least one gain coefficient then
;  the objective type is replaced to "Maximization".
;
     {(Objective ^type Min) <obj>}
     (Attribute ^type GAIN)
-->
     (make after-all)
     (modify <obj>
             ^type Max))

(p a-modelpart
;  -----------
;  Enables to create p-terms, that is, parts of the model.
;  There should be a coefficient related to each p-term.
;
     (Value ^name <c> ^edge <e> ^type COEFFICIENT)
     (Edge ^name <e> ^actv <a>)
     - (a-mpiece ^actv <a>)
-->
     (make a-mpiece
           ^name (genatom)
           ^actv <a>))

(p a-p-term
;  --------
;  Creates a model piece (p-term).
;
     (a-mpiece ^name <n> ^actv <a>)
     (Edge ^name <e> ^actv <a>)
     (Value ^name <c> ^edge <e> ^type COEFFICIENT)
```


unesp

```
      (Object ^name <c> ^class <x>)
      (Attribute ^name <v> ^type VARIABLE)
      (Object ^name <v> ^class <x>)
 -->
      (make p-term
            ^name <n>
            ^actv <a>
            ^edge <e>
            ^coef <c>
            ^dvar <v>
            ^status N
            ^step 0))

(p b-constraint
;  ------------
;  Enables to create a balance constraint.
;
      (Edge ^name <e> ^bal { <b> <> nil })
      - (b-constr ^into <b>)
 -->
      (make b-constr
            ^name (genatom)
            ^into <b>))

(p b-constr-actv
;  -------------
;  Creates a left-hand-side term of a balance constraint, for which
;  the balance is required for an activity.
;
      (b-constr ^name <b> ^into <a>)
      (Edge ^name <e> ^actv <a> ^class <pc> ^inst <x> ^bal <a> ^type <s>)
      (Class ^name <pc> ^index <i>)
      (Activity ^name <a> ^class <c1>)
      (Attribute ^name <v> ^type VARIABLE)
      (Object ^name <v> ^class <c2>)
      (Class ^name <c1> ^index <i>)
      (Class ^name <c2> ^index <i>)
 -->
      (make LhsTerm
            ^name <b>
            ^edge <e>
            ^sign <s>
            ^coef 1
            ^dvar <v>)
      (make LInstance
            ^name <b>
            ^ident <e>
            ^index <i>
            ^value <x>))

(p b-constr-class
;  --------------
;  Creates a left-hand-side term of a balance constraint, for which
;  the balance is required for a class.
;
      (b-constr ^name <b> ^into <pc>)
      (Edge ^name <e> ^actv <a> ^class <pc> ^inst <x> ^bal <pc> ^type <s>)
      (Class ^name <pc> ^index <i>)
      (Activity ^name <a> ^class <c1>)
      (Attribute ^name <v> ^type VARIABLE ^actv <a>)
      (Object ^name <v> ^class <c2>)
      (Class ^name <c1> ^index <i>)
      (Class ^name <c2> ^index <i>)
 -->
      (make LhsTerm
            ^name <b>
```


unesp

```
            ^edge <e>
            ^sign <s>
            ^coef 1
            ^dvar <v>)
      (make LInstance
            ^name <b>
            ^ident <e>
            ^index <i>
            ^value <x>))

(p b-right
;  -------
;  Creates a right-hand-side term of a balance constraint.
;
      (b-constr ^name <b>)
-->
      (make RhsTerm
            ^name <b>
            ^oper "="
            ^rhs 0))

(p c-constraint
;  ------------
;  Enables to create a capacity/demand constraint.
;
      (Value ^name <n> ^type << CAPACITY DEMAND >>)
      - (c-constr ^attrib <n>)
-->
      (make c-constr
            ^name (genatom)
            ^attrib <n>))

(p c-constr
;  --------
;  Creates a left-hand-side term of a capacity/demand constraint.
;  The coefficient is initially set out as the unity.
;
      (c-constr ^name <n> ^attrib <a>)
      (Value ^name <a> ^edge <e> ^type << CAPACITY DEMAND >>)
      (Edge ^name <e> ^actv <d> ^class <pc> ^inst <x>)
      (Class ^name <pc> ^index <i>)
      (Attribute ^name <v> ^type VARIABLE ^actv <d>)
-->
      (make LhsTerm
            ^name <n>
            ^edge <e>
            ^sign 1
            ^coef 1
            ^dvar <v>)
      (make LInstance
            ^name <n>
            ^ident <e>
            ^index <i>
            ^value <x>))

(p c-right-cap
;  -----------
;  Creates a right-hand-side term of a capacity constraint.
;
      (c-constr ^name <c> ^attrib <a>)
      (Value ^name <a> ^type CAPACITY)
      - (RhsTerm ^name <c>)
-->
      (make RhsTerm
            ^name <c>
            ^oper "<="
```

```
          ^rhs <a>))

(p c-right-dem
;  -----------
;  Creates a right-hand-side term of a demand constraint.
;
     (c-constr ^name <c> ^attrib <a>)
     (Value ^name <a> ^type DEMAND)
     - (RhsTerm ^name <c>)
-->
     (make RhsTerm
           ^name <c>
           ^oper ">="
           ^rhs <a>))

(p fix-coeff-edge
;  --------------
;  Fixes the unity coefficient of a lhs-term if there is a p-term
;  for the same activity and primary class (and consequently, the
;  same variable).
;
     {(LhsTerm ^edge <e> ^coef 1 ^dvar <v>) <lhs-term>}
     (Attribute ^name <v> ^type VARIABLE ^actv <a>)
     {(p-term ^actv <a> ^edge <e> ^coef <c> ^dvar <v> ^status << N F >>)
              <p-term>}
-->
     (modify <p-term>
             ^status F)
     (modify <lhs-term>
             ^coef <c>))

(p instance-lookup
;  ---------------
;  Searches for indices in LhsTerms which should be globally instanced.
;
     (after-all)
     (LhsTerm ^name <n> ^edge <e>)
     (Edge ^name <e> ^actv <a> ^inst nil)
     (SubActv ^actv <a> ^index <i> ^status 0)
     - (IGlobal ^cstr <n> ^name <i>)
     - (r-clone ^cstr <n> ^name <i>)
-->
     (make IGlobal
           ^cstr <n>
           ^name <i>
           ^addr 0
           ^card 0
           ^elem-list NULL))

(p array-instance
;  --------------
;  Sets out the indices in LhsTerms which should be globally instanced
;  (updating the arrays IGlobal).
;
     (LhsTerm ^name <n> ^edge <e>)
     (Edge ^name <e> ^actv <a> ^inst nil)
     {(SubActv ^actv <a> ^index <i> ^value <v> ^status 0) <sub>}
     {(IGlobal ^cstr <n> ^name <i> ^card <c>) <ginst>}
     - (r-clone ^cstr <n> ^name <i>)
-->
     (modify <sub>
             ^status 1)
     (modify <ginst>
             ^card (compute <c> + 1)
             ^elem-list (substr <ginst> elem-list inf) <v>))
```

```
(p array-instance-end
;  ------------------
;  There is no more index to be globally instanced.
;
     {(after-all) <after>}
     (IGlobal ^cstr <n> ^name <i>)
     - (SubActv ^index <i> ^status 0)
     - (r-clone ^cstr <n> ^name <i>)
-->
     (remove <after>)
     (make r-clone
           ^cstr <n>
           ^name <i>))

(p subactv-restore
;  ---------------
;  Sets the sub-activities status attribute back to zero.
;
     (IGlobal ^cstr <c> ^name <i>)
     (r-clone ^cstr <c> ^name <i>)
     {(SubActv ^index <i> ^status 1) <sub>}
-->
     (modify <sub>
           ^status 0))

; Rules to Implement the Replication Algorithm
; ============================================
; If there are structures IGlobal (globally instanced indices) for a
; given constraint, it is necessary to replicate this constraint for
; each combination of the instanced indices. The rules prefixed "cmb"
; do this replication taking all the index combinations into account.

(p cmb-initialize
;  --------------
;  Initializes the stack of array's names.
;
     (r-clone ^cstr <c>)
     (IGlobal ^cstr <c> ^addr 0)
     - (Stack ^cstr <c>)
-->
     (make VName
           ^cstr <c>)
     (make Stack
           ^cstr <c>
           ^elem-list NULL)
     (make Contr
           ^cstr <c>
           ^num 0))

(p cmb-push-name
;  -------------
;  Pushes an array name into the stack.
;
     (r-clone ^cstr <c>)
     {(Stack ^cstr <c>) <stack>}
     {(Contr ^cstr <c> ^num <n>) <contr>}
     {(IGlobal ^cstr <c> ^name <a> ^addr 0) <array>}
-->
     (modify <array>
           ^addr 1)
     (modify <contr>
           ^num (compute <n> + 1))
     (modify <stack>
           ^elem-list (substr <stack> elem-list inf) <a>))

(p cmb-push-name-end
```

```
;  -----------------
;  There is no more array name to stack.
;
     (r-clone ^cstr <c>)
     (Stack ^cstr <c>)
     - (IGlobal ^cstr <c> ^addr 0)
-->
     (make init-contr))

(p cmb-init-contr
;  --------------
;  Initializes the search for combinations of array elements.
;
     (r-clone ^cstr <c>)
     {(init-contr) <init>}
     {(Contr ^cstr <c> ^num <n>) <contr>}
-->
     (remove <init>)
     (make c-term
           ^name (genatom))
     (modify <contr>
             ^top <n>
             ^get 1)
     (make get-name))

(p cmb-get-name
;  ------------
;  Gets an array name from the stack.
;
     (r-clone ^cstr <c>)
     {(get-name) <get>}
     {(Stack ^cstr <c>) <stack>}
     {(Contr ^cstr <c> ^num <n> ^get { <g> <= <n> }) <contr>}
     {(VName ^cstr <c>) <vname>}
-->
     (remove <get>)
     (bind <base> (litval elem-list))
     (bind <addr> (compute <base> + <g>))
     (modify <contr>
             ^get (compute <g> + 1))
     (modify <vname>
             ^array (substr <stack> <addr> <addr>))
     (make get-value))

(p cmb-get-value
;  -------------
;  Gets the current element (instance value) from the array under
;  analysis.
;
     (r-clone ^cstr <s>)
     {(get-value) <get>}
     (VName ^cstr <s> ^array <a>)
     {(IGlobal ^cstr <s> ^name <a> ^addr <c> ^card { <d> >= <c> })
               <array>}
     (c-term ^name <n>)
-->
     (remove <get>)
     (bind <base> (litval elem-list))
     (bind <addr> (compute <base> + <c>))
     (make GInstance
           ^name <n>
           ^index <a>
           ^value (substr <array> <addr> <addr>))
     (make test-empty))

(p cmb-test-empty
```

```
;  --------------
;  All the array's current elements have been considered.
;  The next step is to change the current element of the top array.
;  Before a replica of the constraint on focus will be made.
;
     (r-clone ^cstr <c>)
     {(test-empty) <test>}
     {(Contr ^cstr <c> ^num <n> ^top <t> ^get { <g> > <n> }) <contr>}
-->
     (remove <test>)
     (modify <contr>
             ^get <t>)
     (make replica))

(p cmb-replica-lhs
;  ---------------
;  Replicates the lhs-terms of the focused constraint.
;
     (replica)
     (r-clone ^cstr <r>)
     (c-term ^name <n>)
     {(LhsTerm ^name <r> ^sign <s> ^coef <c> ^dvar <d> ^status nil)
               <lhs>}
-->
     (modify <lhs>
             ^status R)
     (make LhsTerm
           ^name <n>
           ^sign <s>
           ^coef <c>
           ^dvar <d>))

(p cmb-replica-rhs
;  ---------------
;  Replicates the rhs-term of the focused constraint.
;
     {(replica) <repl>}
     (r-clone ^cstr <r>)
     (c-term ^name <n>)
     - (LhsTerm ^name <r> ^status nil)
     (RhsTerm ^name <r> ^oper <o> ^rhs <v>)
-->
     (remove <repl>)
     (make RhsTerm
           ^name <n>
           ^oper <o>
           ^rhs  <v>)
     (make restore-status))

(p cmb-restore-status
;  ------------------
;  Restores the status of replicated lhs-terms so that new replications
;  can be made if necessary.
;
     (restore-status)
     (r-clone ^cstr <r>)
     {(LhsTerm ^name <r> ^status R) <lhs>}
-->
     (modify <lhs>
             ^status nil))

(p cmb-resume-test-empty
;  --------------------
;  Resumes the rule cmb-test-empty.
;
     {(restore-status) <rest>}
```

```
     (r-clone ^cstr <r>)
     - (LhsTerm ^name <r> ^status R)
-->
     (remove <rest>)
     (make top-name))

(p cmb-test-empty-fail
;  -------------------
;  There are other arrays to consider.
;
     (r-clone ^cstr <c>)
     {(test-empty) <test>}
     (Contr ^cstr <c> ^num <n> ^get { <g> <= <n> })
-->
     (remove <test>)
     (make get-name))

(p cmb-top-name
;  ------------
;  Gets the next array for which is still possible to consider an
;  element.
;
     (r-clone ^cstr <c>)
     {(top-name) <top>}
     {(Stack ^cstr <c>) <stack>}
     (Contr ^cstr <c> ^get <g>)
     {(VName ^cstr <c>) <vname>}
-->
     (remove <top>)
     (bind <base> (litval elem-list))
     (bind <addr> (compute <base> + <g>))
     (modify <vname>
             ^array (substr <stack> <addr> <addr>))
     (make new-addr))

(p cmb-new-addr
;  ------------
;  Sets the address variable so that the next element can be considered.
;
     (r-clone ^cstr <s>)
     {(new-addr) <new>}
     (VName ^cstr <s> ^array <a>)
     {(IGlobal ^cstr <s> ^name <a> ^addr <c> ^card { <d> > <c>}) <array>}
-->
     (remove <new>)
     (modify <array>
             ^addr (compute <c> + 1))
     (make init-contr))

(p cmb-new-addr-fail
;  -----------------
;  All the elements of this array have been considered, so an other
;  array should be looked for.
;
     (r-clone ^cstr <s>)
     {(new-addr) <new>}
     (VName ^cstr <s> ^array <a>)
     {(IGlobal ^cstr <s> ^name <a> ^addr <c> ^card { <d> <= <c> })
               <array>}
-->
     (remove <new>)
     (modify <array>
             ^addr 1)
     (make new-top))

(p cmb-new-top
```

```
;  ----------
;  Decrements the stack's top in order to consider the next array.
;
     (r-clone ^cstr <c>)
     {(new-top) <new>}
     {(Contr ^cstr <c> ^top <t>) <contr>}
-->
     (remove <new>)
     (modify <contr>
             ^top (compute <t> - 1))
     (make test-repeat))

(p cmb-test-repeat
;  ---------------
;  There is still a combination of elements to consider.
;
     (r-clone ^cstr <c>)
     {(test-repeat) <test>}
     {(Contr ^cstr <c> ^top { <t> > 0 }) <contr>}
-->
     (remove <test>)
     (modify <contr>
             ^get <t>)
     (make top-name))

(p cmb-test-repeat-fail
;  --------------------
;  All the combinations have been scanned.
;
     (r-clone ^cstr <c>)
     {(test-repeat) <test>}
     (Contr ^cstr <c> ^top { <t> <= 0 })
-->
     (remove <test>)
     (make cmb-delete))

(p cmb-delete
;  ----------
;  Deletes the lhs-terms of the constraint which has been replicated.
;
     (cmb-delete)
     (r-clone ^cstr <c>)
     {(LhsTerm ^name <c>) <lhs>}
-->
     (remove <lhs>))

(p cmb-delete-end
;  --------------
;  Deletes the rhs-term of the constraint which has been replicated.
;
     {(cmb-delete) <delete>}
     (r-clone ^cstr <c>)
     {(RhsTerm ^name <c>) <rhs>}
-->
     (remove <delete>)
     (remove <rhs>)
     (make cmb-clear))

(p cmb-clear-start
;  ---------------
;  Deletes the control structures used to implement
;  the Replication Algorithm.
;
     (cmb-clear)
     (r-clone ^cstr <c>)
     {(Stack ^cstr <c>) <stack>}
```

```
     {(VName ^cstr <c>) <vname>}
     {(Contr ^cstr <c>) <contr>}
-->
     (remove <stack>)
     (remove <vname>)
     (remove <contr>))

(p cmb-clear-iglobal
;  -----------------
;  Deletes the arrays IGlobal used to implement the Replication
;  Algorithm.
;
     (cmb-clear)
     (r-clone ^cstr <c>)
     {(IGlobal ^cstr <c>) <iglobal>}
-->
     (remove <iglobal>))

(p cmb-clear-end
;  -------------
;  No more structures used to implement the Replication Algorithm.
;
     {(cmb-clear) <clear>}
     (r-clone ^cstr <c>)
     - (IGlobal ^cstr <c>)
-->
     (remove <clear>))

(p storing
;  -------
;  Storage activity.
;
     (Activity ^name <a> ^type STORAGE)
     - (s-constr ^actv <a> ^type STORAGE)
-->
     (make s-constr
           ^actv <a>
           ^type STORAGE))

(p stock-balance
;  -------------
;  The stock balance should consider the amount in stock at successive
;  periods of time.
;
     (s-constr ^actv <a> ^type STORAGE)
     (Attribute ^name <v> ^type VARIABLE ^actv <a>)
     (LhsTerm ^name <n1> ^edge <> STORE-TIME ^sign <s> ^coef <c>
              ^dvar <v>)
     (LhsTerm ^name { <n2> <> <n1> } ^edge <> STORE-TIME ^dvar <v>)
     (GInstance ^name <n1> ^index <i> ^value <v1>)
     (GInstance ^name { <n2> <> <n1> } ^index <i> ^value { <v2> > <v1> })
     - (GInstance ^index <i> ^value { > <v1> < <v2> })
     (Index ^name <i> ^type TIME)
-->
     (make LhsTerm
           ^name <n2>
           ^edge STORE-TIME
           ^sign (compute -1 * <s>)
           ^coef <c>
           ^dvar <v>)
     (make LInstance
           ^name <n2>
           ^ident STORE-TIME
           ^index <i>
           ^value <v1>))
```


unesp

```
; The following rules aim:
; ========================
;   - to show the model,
;   - to recover the working memory in order to rebuild
;     the LP Model data structures.
; -----------------------------------------------------

(p show-model
;  ----------
;  Shows the model's header.
;
     {(show) <show>}
     {(o-funct) <o-funct>}
     (Model ^name <n> ^type <t> ^descr <d>)
-->
     (remove <show>)
     (remove <o-funct>)
     (write (crlf)
            "Model ......" <n> (crlf)
            "Type ......." <t> (crlf)
            "Description:" <d> (crlf)
            (crlf))
     (make t-show))

(p show-objective
;  --------------
;  Shows the type of the objective-function.
;
     {(t-show) <t-show>}
     {(Objective ^type <t>) <obj>}
-->
     (remove <t-show>)
     (remove <obj>)
     (write <t>)
     (make o-show1))

(p show-ofunction1
;  ---------------
;  Shows the first term of the objective-function.
;
     {(o-show1) <o-show1>}
     {(ObjFunct ^name <n> ^sign 1 ^coef <c> ^dvar <d>) <ofunct>}
-->
     (remove <o-show1>)
     (remove <ofunct>)
     (write <c> "*" <d>)
     (make o-show)
     (make i-local
           ^name <n>
           ^ident OFUNCT))

(p ofunction-show-instance
;  -----------------------
;  Shows the locally instanced indices in a term of the objective
;  function.
;
     {(i-local ^name <n> ^ident OFUNCT) <local>}
     {(LInstance ^name <n> ^ident OFUNCT ^index <i> ^value <v>) <linst>}
-->
     (remove <linst>)
     (remove <local>)
     (write "(" <i> "=" <v> ")"))

(p ofunction-instance-end
;  ----------------------
```


cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

```
;  Deletes working memory element if there is no locally instanced index
;  in a term of the objective-function.
;
     {(i-local ^name <n> ^ident OFUNCT) <local>}
     - (LInstance ^name <n> ^ident OFUNCT)
-->
     (remove <local>))

(p show-ofunction-plus
;  -------------------
;  Shows the other positive terms of the objective-function.
;
     (o-show)
     {(ObjFunct ^name <n> ^sign 1 ^coef <c> ^dvar <d>) <ofunct>}
-->
     (remove <ofunct>)
     (write "+" <c> "*" <d>)
     (make i-local
           ^name <n>
           ^ident OFUNCT))

(p show-ofunction-minus
;  --------------------
;  Shows the other negative terms of the objective-function.
;
     (o-show)
     {(ObjFunct ^name <n> ^sign -1 ^coef <c> ^dvar <d>) <ofunct>}
-->
     (remove <ofunct>)
     (write "-" <c> "*" <d>)
     (make i-local
           ^name <n>
           ^ident OFUNCT))

(p show-subject-to
;  ---------------
;  Enables to show the constraints.
;
     {(o-show) <o-show>}
     - (ObjFunct)
-->
     (remove <o-show>)
     (write (crlf) "Subject to:" (crlf))
     (make show-c))

(p show-constraint
;  ---------------
;  Enables to show the left-hand-side of the constraints.
;
     {(show-c) <show-c>}
     (LhsTerm ^name <n>)
     - (c-show ^name <n>)
-->
     (remove <show-c>)
     (make c-show
           ^name <n>
           ^term 1))

(p show-lhs1
;  ---------
;  Shows the first term of the left-hand-side of a constraint.
;
     {(c-show ^name <n> ^term 1) <c-show>}
     {(LhsTerm ^name <n> ^edge <e> ^sign 1 ^coef <c> ^dvar <d>) <lhs>}
-->
     (remove <lhs>)
```




```
     (write "  " <c> "*" <d>)
     (modify <c-show>
             ^term 2)
     (make i-local
           ^name <n>
           ^ident <e>))

(p local-instance
;  --------------
;  Shows the locally instanced lhs-term indices of the constraint.
;
     (i-local ^name <n> ^ident <e>)
     {(LInstance ^name <n> ^ident <e> ^index <i> ^value { <v> <> nil })
                 <linst>}
-->
     (remove <linst>)
     (write "(" <i> "=" <v> ")"))

(p local-instance-fail
;  -------------------
;  There is only local instances with the value equals to nil.
;
     (i-local ^name <n> ^ident <e>)
     {(LInstance ^name <n> ^ident <e> ^value nil) <linst>}
-->
     (remove <linst>))

(p local-instance-end
;  ------------------
;  There is no locally instanced lhs-term indice in the constraint.
;
     {(i-local ^name <n> ^ident <e>) <local>}
     - (LInstance ^name <n> ^ident <e>)
-->
     (remove <local>))

(p show-lhs-plus
;  -------------
;  Shows the additional positive terms of the constraint's
;  left-hand-side.
;
     (c-show ^name <n> ^term 2)
     {(LhsTerm ^name <n> ^edge <e> ^sign 1 ^coef <c> ^dvar <d>) <lhs>}
-->
     (remove <lhs>)
     (write "+" <c> "*" <d>)
     (make i-local
           ^name <n>
           ^ident <e>))

(p show-lhs-minus
;  --------------
;  Shows the additional negative terms of the constraint's
;  left-hand-side.
;
     (c-show ^name <n> ^term 2)
     {(LhsTerm ^name <n> ^edge <e> ^sign -1 ^coef <c> ^dvar <d>) <lhs>}
-->
     (remove <lhs>)
     (write "-" <c> "*" <d>)
     (make i-local
           ^name <n>
           ^ident <e>))

(p show-rhs
;  --------
```

```
;  Shows the right-hand-side of a constraint.
;
     (c-show ^name <n>)
     - (LhsTerm ^name <n>)
     {(RhsTerm ^name <n> ^oper <o> ^rhs <r>) <rhs>}
-->
     (remove <rhs>)
     (write <o> <r>))

(p global-instance
;  --------------
;  Shows the indices of the constraint which are globally instanced.
;
     (c-show ^name <n>)
     - (LhsTerm ^name <n>)
     - (RhsTerm ^name <n>)
     {(GInstance ^name <n> ^index <i> ^value <v>) <ginst>}
-->
     (remove <ginst>)
     (write "[" <i> "=" <v> "]"))

(p show-constraint-end
;  -------------------
;  Enables to show other constraints.
;
     {(c-show ^name <n>) <c-show>}
     - (LhsTerm ^name <n>)
     - (RhsTerm ^name <n>)
     - (GInstance ^name <n>)
-->
     (remove <c-show>)
     (write (crlf))
     (make show-c))

(p clear-show-c
;  ------------
;  Enables to show parts of constraints which was not possible to
;  use in the LP Model (p-terms).
;
     {(show-c) <show-c>}
     - (LhsTerm)
-->
     (remove <show-c>)
     (make show-p))

(p show-pterms
;  -----------
;  Starts the showing of parts of constraints not used (p-terms).
;
     {(show-p) <show-p>}
     (p-term ^status N)
-->
     (write (crlf))
     (write "It was not possible to use the following term(s):" (crlf))
     (remove <show-p>)
     (make p-show))

(p clear-show-p
;  ------------
;  There is no p-term to show.
;
     {(show-p) <show-p>}
     - (p-term ^status N)
-->
     (remove <show-p>)
     (make clear))
```


unesp

```
(p pterms-show
;  -----------
;  Shows parts of constraints (p-terms).
;
     (p-show)
     {(p-term ^coef <c> ^dvar <v> ^status N) <lterm>}
-->
     (write "  " <c> "*" <v> (crlf))
     (remove <lterm>))

(p clear-p-show
;  ------------
;  Enables to delete working memory elements so that
;  the model can be rebuilt.
;
     {(p-show) <p-show>}
     - (p-term ^status N)
-->
     (remove <p-show>)
     (make clear))

(p clear-a-mpiece
;  --------------
;  Enables to rebuild the model pieces.
;
     (clear)
     {(a-mpiece) <a-mpiece>}
-->
     (remove <a-mpiece>))

(p clear-b-constr
;  --------------
;  Enables to rebuild the balance constraints.
;
     (clear)
     {(b-constr) <b-constr>}
-->
     (remove <b-constr>))

(p clear-c-constr
;  --------------
;  Enables to rebuild the demand/capacity constraints.
;
     (clear)
     {(c-constr) <c-constr>}
-->
     (remove <c-constr>))

(p clear-s-constr
;  --------------
;  Enables to rebuild constraints for special activities.
;
     (clear)
     {(s-constr) <s-constr>}
-->
     (remove <s-constr>))

(p clear-p-terms
;  -------------
;  Enables to rebuild p-terms.
;
     (clear)
     {(p-term) <p-term>}
-->
     (remove <p-term>))
```


unesp

```
(p clear-c-terms
;  -------------
;  Enables to rebuild copies of constraints.
;
     (clear)
     {(c-term) <c-term>}
-->
     (remove <c-term>))

(p clear-clone
;  -----------
;  Enables to rebuild replications.
;
     (clear)
     {(r-clone) <r-clone>}
-->
     (remove <r-clone>))

(p clear-clear
;  -----------
;  Removes the permission to delete working memory elements and so,
;  enables to rebuid the LP Model.
;
     {(clear) <clear>}
     - (a-mpiece)
     - (b-constr)
     - (c-constr)
     - (s-constr)
     - (p-term)
     - (c-term)
     - (r-clone)
-->
     (remove <clear>)
     (make Objective
           ^type Min))


; INITIALIZATION SECTION
; ======================

; The Objective Type is set out as "Minimization" by default.
; -----------------------------------------------------------

(make Objective
      ^type Min)
```

Apresenta-se, a seguir, exemplos de sessões com o sistema especialista na formulação de Modelos de PL, para alguns problemas clássicos discutidos anteriormente. Para cada problema mostra-se: a representação OPS5 do Modelo de Dados, a representação OPS5 do Modelo Conceitual, as estruturas OPS5 resultantes da análise dos modelos conceitual e de dados representados anteriormente e o resultado da sessão.

C.1 - Problema de Distribuição de Produto Único, com Transbordo

C.1.1 - Modelo de Dados

```
(Object ^name FCAP ^class FACTORY)
(Object ^name DCAP ^class DEPOT)
(Object ^name CREQ ^class CUSTOMER)
(Object ^name FDCOST ^class FACTORY-DEPOT)
(Object ^name FCCOST ^class FACTORY-CUSTOMER)
(Object ^name DCCOST ^class DEPOT-CUSTOMER)
(Object ^name X ^class FACTORY-DEPOT)
(Object ^name Y ^class FACTORY-CUSTOMER)
(Object ^name Z ^class DEPOT-CUSTOMER)

(Class ^name FACTORY ^index I)
(Class ^name DEPOT ^index J)
(Class ^name CUSTOMER ^index K)
(Class ^name FACTORY-DEPOT ^index I)
(Class ^name FACTORY-DEPOT ^index J)
(Class ^name FACTORY-CUSTOMER ^index I)
(Class ^name FACTORY-CUSTOMER ^index K)
(Class ^name DEPOT-CUSTOMER ^index J)
(Class ^name DEPOT-CUSTOMER ^index K)

(Index ^name I ^type PLACE)
(Index ^name J ^type PLACE)
(Index ^name K ^type PLACE)
```

C.1.2 - Modelo Conceitual

```
(Activity ^name FDTRANS ^class FACTORY-DEPOT ^type TRANSPORT)
(Activity ^name FCTRANS ^class FACTORY-CUSTOMER ^type TRANSPORT)
(Activity ^name DCTRANS ^class DEPOT-CUSTOMER ^type TRANSPORT)

(Attribute ^name FDCOST ^type LOSS ^actv FDTRANS)
(Attribute ^name FCCOST ^type LOSS ^actv FCTRANS)
(Attribute ^name DCCOST ^type LOSS ^actv DCTRANS)
(Attribute ^name X ^type VARIABLE ^actv FDTRANS)
(Attribute ^name Y ^type VARIABLE ^actv FCTRANS)
(Attribute ^name Z ^type VARIABLE ^actv DCTRANS)

(Edge ^name E1 ^actv FDTRANS ^class FACTORY)
(Edge ^name E2 ^actv FDTRANS ^class DEPOT ^bal DEPOT ^type 1)
(Edge ^name E3 ^actv DCTRANS ^class DEPOT ^bal DEPOT ^type -1)
(Edge ^name E4 ^actv DCTRANS ^class CUSTOMER)
(Edge ^name E5 ^actv FCTRANS ^class CUSTOMER)
(Edge ^name E6 ^actv FCTRANS ^class FACTORY)

(Value ^name FCAP ^edge E1 ^type CAPACITY)
(Value ^name FCAP ^edge E6 ^type CAPACITY)
(Value ^name DCAP ^edge E2 ^type CAPACITY)
(Value ^name CREQ ^edge E4 ^type DEMAND)
(Value ^name CREQ ^edge E5 ^type DEMAND)
```

C.1.3 - Estruturas Resultantes da Análise

```
44: (c-constr ^attrib CREQ ^name ?0)
45: (LhsTerm ^dvar Y ^coef 1 ^sign 1 ^edge E5 ^name ?0)
46: (LInstance ^ident E5 ^index K ^name ?0)
47: (RhsTerm ^oper >= ^rhs CREQ ^name ?0)
48: (LhsTerm ^dvar Z ^coef 1 ^sign 1 ^edge E4 ^name ?0)
```


cm 1 2 3 4 5 6 7 unesp 1 12 13 14 15 16 17 18

```
49: (LInstance ^ident E4 ^index K ^name ?0)
50: (c-constr ^attrib DCAP ^name ?1)
51: (LhsTerm ^dvar X ^coef 1 ^sign 1 ^edge E2 ^name ?1)
52: (LInstance ^ident E2 ^index J ^name ?1)
53: (RhsTerm ^oper <= ^rhs DCAP ^name ?1)
54: (c-constr ^attrib FCAP ^name ?2)
55: (LhsTerm ^dvar Y ^coef 1 ^sign 1 ^edge E6 ^name ?2)
56: (LInstance ^ident E6 ^index I ^name ?2)
57: (RhsTerm ^oper <= ^rhs FCAP ^name ?2)
58: (LhsTerm ^dvar X ^coef 1 ^sign 1 ^edge E1 ^name ?2)
59: (LInstance ^ident E1 ^index I ^name ?2)
60: (b-constr ^into DEPOT ^name ?3)
61: (LhsTerm ^dvar Z ^coef 1 ^sign -1 ^edge E3 ^name ?3)
62: (LInstance ^ident E3 ^index J ^name ?3)
63: (LhsTerm ^dvar X ^coef 1 ^sign 1 ^edge E2 ^name ?3)
64: (LInstance ^ident E2 ^index J ^name ?3)
65: (RhsTerm ^oper = ^rhs 0 ^name ?3)
66: (o-funct)
67: (ObjFunct ^dvar Z ^coef DCCOST ^sign 1 ^actv DCTRANS ^name ?4)
68: (ObjFunct ^dvar Y ^coef FCCOST ^sign 1 ^actv FCTRANS ^name ?5)
69: (ObjFunct ^dvar X ^coef FDCOST ^sign 1 ^actv FDTRANS ^name ?6)
```

C.1.4 - Sessão de Formulação de Modelo de PL

```
C:\OPS5>ops5
ops5: (load expert.ops)
ops5: (load distr.mod)
ops5: (run)
ops5: (make show)
ops5: (run)

Model ..... M01
Type ..... DISTR-TS
Description: Single Product Distribution Problem with Transshipment

Min FDCOST * X + FCCOST * Y + DCCOST * Z
Subject to:
   1 * X - 1 * Z = 0
   1 * X + 1 * Y <= FCAP
   1 * X <= DCAP
   1 * Z + 1 * Y >= CREQ

ops5: (exit)
C:\OPS5>
```

C.2 - Problema de Distribuição de Múltiplos Produtos

C.2.1 - Modelo de Dados

```
(Object ^name A ^class FACTORY-PRODUCT)
(Object ^name B ^class DEPOT-PRODUCT)
(Object ^name H ^class DEPOT-PRODUCT)
(Object ^name D ^class CUSTOMER-PRODUCT)
(Object ^name C0 ^class FACTORY-DEPOT-PRODUCT)
(Object ^name C1 ^class FACTORY-CUSTOMER-PRODUCT)
(Object ^name C2 ^class DEPOT-CUSTOMER-PRODUCT)
(Object ^name R0 ^class FACTORY-DEPOT-PRODUCT)
(Object ^name R1 ^class FACTORY-CUSTOMER-PRODUCT)
(Object ^name R2 ^class DEPOT-CUSTOMER-PRODUCT)
(Object ^name X ^class FACTORY-DEPOT-PRODUCT)
(Object ^name Y ^class FACTORY-CUSTOMER-PRODUCT)
(Object ^name Z ^class DEPOT-CUSTOMER-PRODUCT)
```


unesp

```
(Object ^name S ^class DEPOT-PRODUCT)

(Class ^name FACTORY ^index I)
(Class ^name DEPOT ^index J)
(Class ^name CUSTOMER ^index K)
(Class ^name PRODUCT ^index P)
(Class ^name FACTORY-PRODUCT ^index I)
(Class ^name FACTORY-PRODUCT ^index P)
(Class ^name DEPOT-PRODUCT ^index J)
(Class ^name DEPOT-PRODUCT ^index P)
(Class ^name CUSTOMER-PRODUCT ^index K)
(Class ^name CUSTOMER-PRODUCT ^index P)
(Class ^name FACTORY-DEPOT-PRODUCT ^index I)
(Class ^name FACTORY-DEPOT-PRODUCT ^index J)
(Class ^name FACTORY-DEPOT-PRODUCT ^index P)
(Class ^name FACTORY-CUSTOMER-PRODUCT ^index I)
(Class ^name FACTORY-CUSTOMER-PRODUCT ^index K)
(Class ^name FACTORY-CUSTOMER-PRODUCT ^index P)
(Class ^name DEPOT-CUSTOMER-PRODUCT ^index J)
(Class ^name DEPOT-CUSTOMER-PRODUCT ^index K)
(Class ^name DEPOT-CUSTOMER-PRODUCT ^index P)

(Index ^name I ^type PLACE)
(Index ^name J ^type PLACE)
(Index ^name K ^type PLACE)
(Index ^name P ^type MATERIAL)
```

C.2.2 - Modelo Conceitual

```
(Activity ^name FDROUTE ^class FACTORY-DEPOT-PRODUCT ^type TRANSPORT)
(Activity ^name FCROUTE ^class FACTORY-CUSTOMER-PRODUCT ^type TRANSPORT)
(Activity ^name DCROUTE ^class DEPOT-CUSTOMER-PRODUCT ^type TRANSPORT)
(Activity ^name STORING ^class DEPOT-PRODUCT ^type STORAGE)

(Attribute ^name H ^type LOSS ^actv STORING)
(Attribute ^name C0 ^type LOSS ^actv FDROUTE)
(Attribute ^name C1 ^type LOSS ^actv FCROUTE)
(Attribute ^name C2 ^type LOSS ^actv DCROUTE)
(Attribute ^name X ^type VARIABLE ^actv FDROUTE)
(Attribute ^name Y ^type VARIABLE ^actv FCROUTE)
(Attribute ^name Z ^type VARIABLE ^actv DCROUTE)
(Attribute ^name S ^type VARIABLE ^actv STORING)

(Edge ^name E1 ^actv FDROUTE ^class FACTORY)
(Edge ^name E2 ^actv FDROUTE ^class DEPOT)
(Edge ^name E3 ^actv FDROUTE ^class DEPOT ^bal DEPOT ^type 1)
(Edge ^name E4 ^actv FDROUTE ^class PRODUCT)
(Edge ^name E5 ^actv FCROUTE ^class FACTORY)
(Edge ^name E6 ^actv FCROUTE ^class CUSTOMER)
(Edge ^name E7 ^actv FCROUTE ^class PRODUCT)
(Edge ^name E8 ^actv DCROUTE ^class DEPOT ^bal DEPOT ^type -1)
(Edge ^name E9 ^actv DCROUTE ^class CUSTOMER)
(Edge ^name E10 ^actv DCROUTE ^class PRODUCT)
(Edge ^name E11 ^actv STORING ^class DEPOT ^bal DEPOT ^type -1)
(Edge ^name E12 ^actv STORING ^class PRODUCT)

(Value ^name A ^edge E1 ^type CAPACITY)
(Value ^name R0 ^edge E1 ^type COEFFICIENT)
(Value ^name B ^edge E2 ^type CAPACITY)
(Value ^name R0 ^edge E2 ^type COEFFICIENT)
(Value ^name A ^edge E5 ^type CAPACITY)
(Value ^name R1 ^edge E5 ^type COEFFICIENT)
(Value ^name D ^edge E6 ^type DEMAND)
(Value ^name R1 ^edge E6 ^type COEFFICIENT)
```




```
(Value ^name D ^edge E9 ^type DEMAND)
(Value ^name R2 ^edge E9 ^type COEFFICIENT)
```

C.2.3 - Estruturas Resultantes da Análise

```
74: (a-mpiece ^actv DCROUTE ^name ?0)
76: (c-constr ^attrib D ^name ?1)
78: (LInstance ^ident E9 ^index K ^name ?1)
79: (p-term ^step 0 ^dvar Z ^coef R2 ^edge E9 ^status F ^actv DCROUTE ^name ?0)
80: (LhsTerm ^dvar Z ^coef R2 ^sign 1 ^edge E9 ^name ?1)
81: (RhsTerm ^oper >= ^rhs D ^name ?1)
83: (LInstance ^ident E6 ^index K ^name ?1)
84: (a-mpiece ^actv FCROUTE ^name ?2)
86: (p-term ^step 0 ^dvar Y ^coef R1 ^edge E6 ^status F ^actv FCROUTE ^name ?2)
87: (LhsTerm ^dvar Y ^coef R1 ^sign 1 ^edge E6 ^name ?1)
89: (c-constr ^attrib A ^name ?3)
91: (LInstance ^ident E5 ^index I ^name ?3)
92: (p-term ^step 0 ^dvar Y ^coef R1 ^edge E5 ^status F ^actv FCROUTE ^name ?2)
93: (LhsTerm ^dvar Y ^coef R1 ^sign 1 ^edge E5 ^name ?3)
94: (RhsTerm ^oper <= ^rhs A ^name ?3)
96: (LInstance ^ident E1 ^index I ^name ?3)
97: (a-mpiece ^actv FDROUTE ^name ?4)
100: (p-term ^step 0 ^dvar X ^coef R0 ^edge E1 ^status F ^actv FDROUTE ^name ?4)
101: (LhsTerm ^dvar X ^coef R0 ^sign 1 ^edge E1 ^name ?3)
102: (c-constr ^attrib B ^name ?5)
104: (LInstance ^ident E2 ^index J ^name ?5)
105: (p-term ^step 0 ^dvar X ^coef R0 ^edge E2 ^status F ^actv FDROUTE ^name ?4)
106: (LhsTerm ^dvar X ^coef R0 ^sign 1 ^edge E2 ^name ?5)
107: (RhsTerm ^oper <= ^rhs B ^name ?5)
108: (b-constr ^into DEPOT ^name ?6)
109: (LhsTerm ^dvar S ^coef 1 ^sign -1 ^edge E11 ^name ?6)
110: (LInstance ^ident E11 ^index J ^name ?6)
111: (LhsTerm ^dvar Z ^coef 1 ^sign -1 ^edge E8 ^name ?6)
112: (LInstance ^ident E8 ^index J ^name ?6)
113: (LhsTerm ^dvar X ^coef 1 ^sign 1 ^edge E3 ^name ?6)
114: (LInstance ^ident E3 ^index J ^name ?6)
115: (RhsTerm ^oper = ^rhs 0 ^name ?6)
116: (o-funct)
117: (ObjFunct ^dvar S ^coef H ^sign 1 ^actv STORING ^name ?7)
118: (ObjFunct ^dvar Z ^coef C2 ^sign 1 ^actv DCROUTE ^name ?8)
119: (ObjFunct ^dvar Y ^coef C1 ^sign 1 ^actv FCROUTE ^name ?9)
120: (ObjFunct ^dvar X ^coef C0 ^sign 1 ^actv FDROUTE ^name ?10)
121: (s-constr ^type STORAGE ^actv STORING)
```

C.2.4 - Sessão de Formulação de Modelo de PL

```
C:\OPS5>ops5
ops5: (load expert.ops)
ops5: (load multi.mod)
ops5: (run)
ops5: (make show)
ops5: (run)

Model ..... M03
Type ..... DISTR-MC
Description: Multi-Commodity Distribution Problem

Min C0 * X + C1 * Y + C2 * Z + H * S
Subject to:
   1 * X - 1 * Z - 1 * S = 0
   R0 * X <= B
   R0 * X + R1 * Y <= A
   R1 * Y + R2 * Z >= D
```

Apêndice D

MoNet - Ambiente de Modelagem em Redes

Apresenta-se a seguir, sucintamente, o estágio atual de desenvolvimento do sistema MoNet (Senne, 1995) de modelagem em redes. os módulos deste sistema vêm sendo desenvolvidos através de diversos projetos de pesquisa e de iniciação científica. A integração destes módulos é, em vários casos, uma tarefa ainda por fazer. Neste ambiente o usuário dispõe das seguintes interfaces gráficas:

- NPL2, para estabelecer modelos na linguagem algébrica discutida no Capítulo 2. Nesta interface o usuário dispõe de um editor de programas, com opções de compilação (tradução da representação algébrica para uma representação conveniente ao algoritmo de solução) e edição de dados como tabelas de um banco de dados relacional;
- IGPL, para estabelecer modelos conceituais de um problema (através das primitivas gráficas discutidas no Capítulo 3) e traduzir este modelo em construções da linguagem OPS5;
- VIEW, para edição de redes e visualização de soluções (fluxos em cada um dos arcos obtidos pelo algoritmo Primal Simplex apresentado no Capítulo 2, ou localização de servidores e alocação de clientes a servidores obtidas pelos algoritmos discutidos no Capítulo 4, para problemas de localização).

Nestas interfaces, todas as funções básicas relativas à janela gráfica, tais como: criação, movimentação, redimensionamento, fechamento, transformação em ícone, maximização, rolagem horizontal e vertical, são executadas através de chamadas a procedimentos da linguagem *Delphi* (Borland, 1995). A exibição de menus e quadros de diálogo com as opções de utilização do programa e o tratamento de entradas de teclado e de *mouse*, também utilizam recursos disponíveis nesta linguagem.

Na interface NPL2 o usuário dispõe de um ambiente como mostra a Figura D1, a seguir.



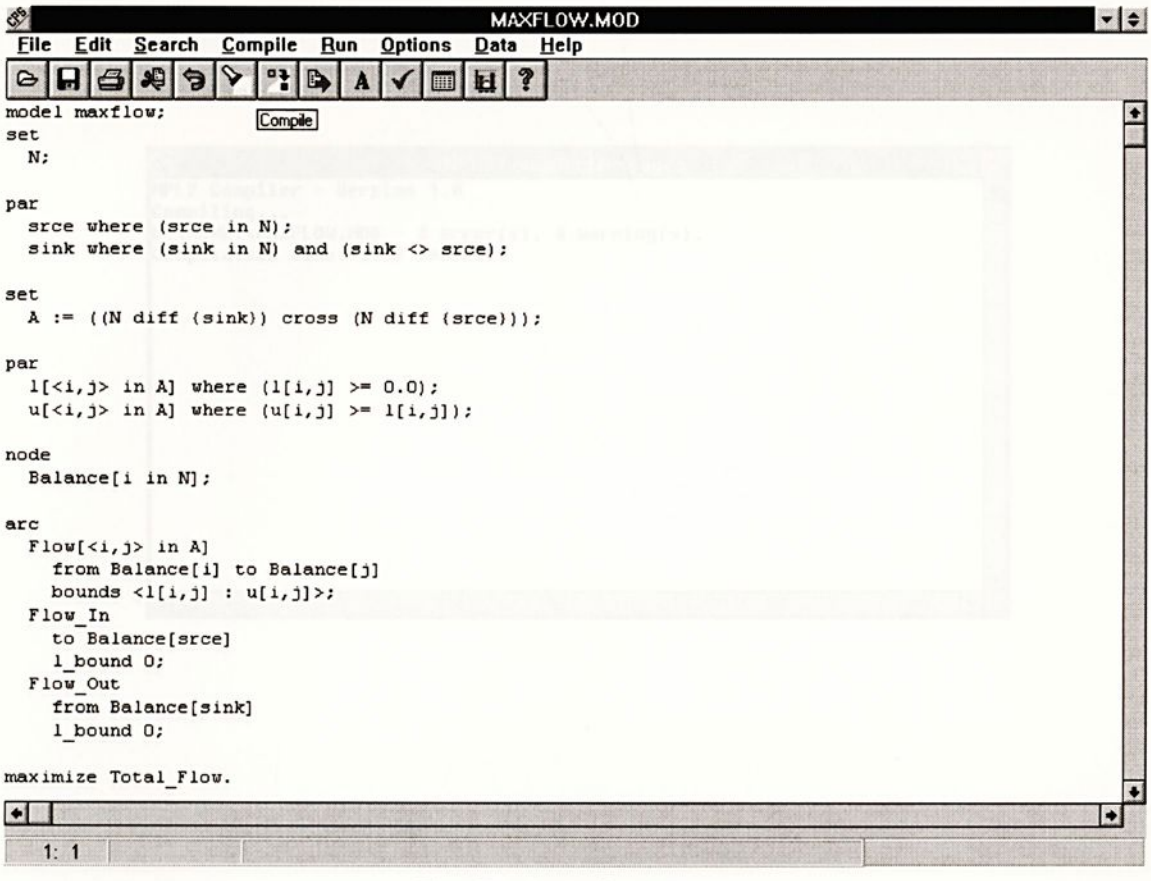


Fig. D1 - Interface NPL2

Neste ambiente o usuário dispõe de diversas opções para carregar, editar, salvar, compilar, imprimir e executar um programa escrito na linguagem NPL2. Algumas destas opções aparecem tanto na barra de menu principal como na barra de ferramentas, na forma de ícones.

Na implementação atual, as opções mostradas na Figura D1 estão implementadas, com exceção das opções Compile (parcialmente) e Run. A implementação atual da fase de compilação (Figura D2) resume-se apenas à análise sintática do programa NPL2, com a execução de rotinas semânticas apenas para a manutenção da tabela de símbolos. É preciso ainda um esforço no sentido de implementar as rotinas semânticas para tradução das construções algébricas em estruturas convenientes ao algoritmo Primal Simplex usado para resolver o modelo. Somente após esta implementação será possível dispor da opção Run, que poderá vir a ter uma ligação com a interface VIEW.

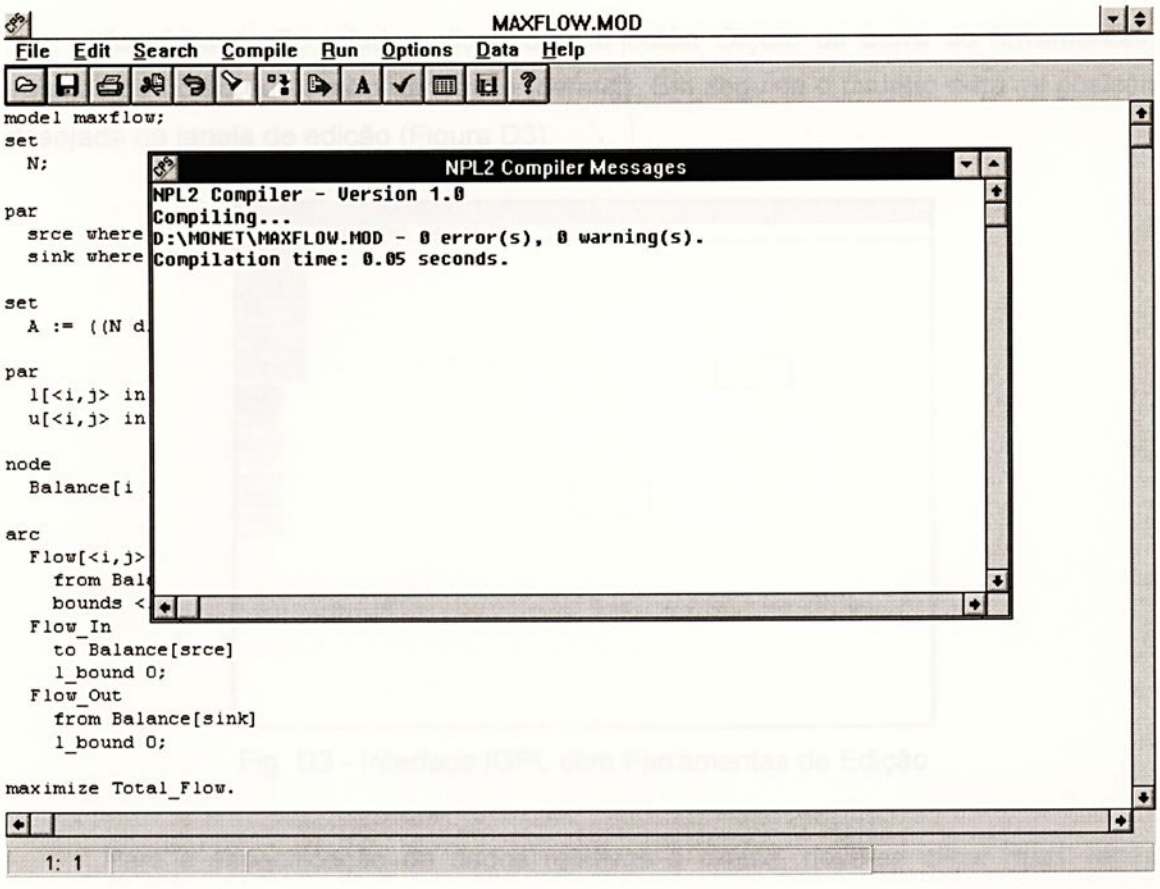


Fig. D2 - Fase de Compilação de Programa NPL2

Ao acessar a interface IGPL o usuário dispõe de uma tela gráfica contendo uma barra de menu principal com as seguintes opções e sub-opções:

File	Model	Compile	Tools	Help
New	Edit		Show grid	Index
Open	Clear all		Snap to grid	Using
Save as			Color	About
Exit				

Para a edição do modelo conceitual de um problema dispõe-se de uma barra de ferramentas de edição de modelos no canto superior esquerdo da janela de edição. Esta barra de ferramentas contém botões para a criação de objetos (*classes* ou *atividades*) e de criação de relacionamentos (*normais* ou representando *balanços*).

As classes são criadas clicando-se o botão *Objeto* da barra de ferramentas, seguido do botão que simboliza classe (*default*). Em seguida o usuário clica na posição desejada na janela de edição (Figura D3).

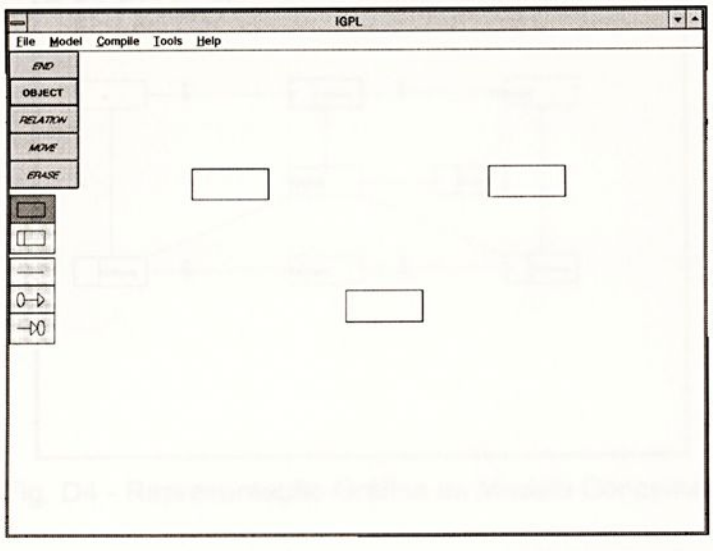


Fig. D3 - Interface IGPL com Ferramentas de Edição

Para a especificação de dados relativos à classe, deve-se clicar duas vezes sobre o objeto, o que provoca a abertura de um quadro de diálogo onde o usuário especifica o nome, o índice, o tipo da classe, além de características visuais deste objeto. Para a edição dos objetos relativos a atividades, clica-se no botão *Objeto*, seguido do botão que simboliza atividade. Em seguida, deve-se clicar na janela de edição a fim de escolher a posição onde esta atividade deve ser exibida. Clicando duas vezes sobre o objeto atividade abre-se um quadro de diálogo para o preenchimento dos campos relativos à atividade: nome, variável de decisão, coeficiente (especificado como ganho ou perda) e o tipo da atividade, além das características visuais.

Para relacionar dois objetos (uma atividade a uma classe ou vice-versa), clica-se sobre o botão *Relation* na barra de ferramentas, escolhe-se uma das opções de relação e, em seguida, clica-se sobre os objetos a serem relacionados. Um clique duplo sobre a relação abre um quadro de diálogo para a especificação de dados relativos à relação, os quais poderão ser correspondentes a uma restrição (caracterizada como de demanda ou de capacidade) ou correspondentes a coeficientes.

A Figura D4 ilustra o resultado da fase de edição para um modelo de distribuição.

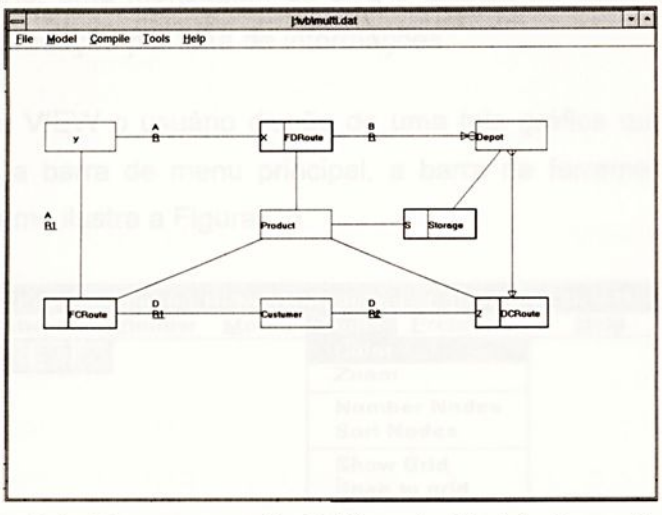


Fig. D4 - Representação Gráfica de Modelo Conceitual

Com a opção *Compile* tem-se a tradução das estruturas gráficas do modelo conceitual para sua representação simbólica na linguagem OPS5. Com a tradução cria-se um arquivo em disco com o mesmo nome do modelo mas com a extensão *.mod*, o qual será entrada para o Sistema Especialista desenvolvido para auxiliar na tarefa de construção de modelos (ver Apêndice C). Esta opção abre também uma janela (ver Figura D5) onde as construções OPS5 podem ser visualizadas.

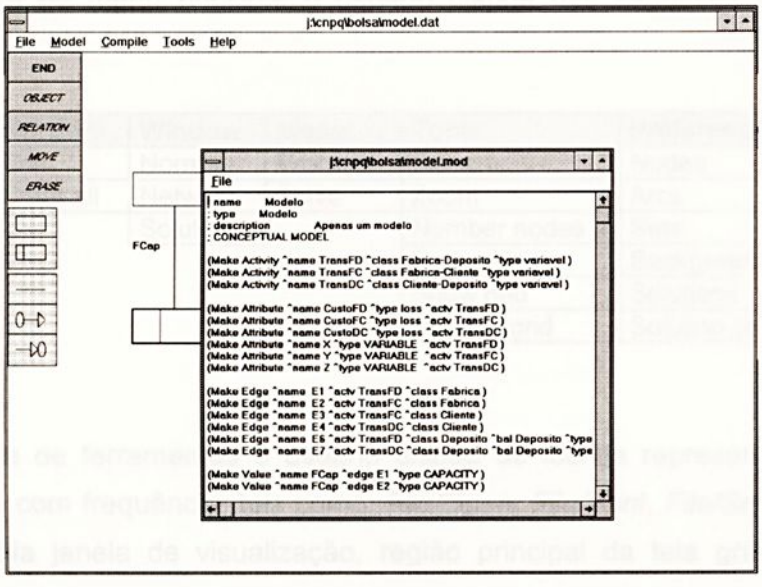


Fig. D5 - Compilação de Modelo Conceitual

Deve-se ressaltar que este processo de tradução só se completa caso todas as informações obrigatórias relativas aos objetos e relações tenham sido preenchidas pelo usuário. Do contrário, uma mensagem de erro é exibida ao usuário alertando para a impossibilidade de tradução por falta de informações.

Na interface VIEW o usuário dispõe de uma tela gráfica que compreende três regiões principais: a barra de menu principal, a barra de ferramentas e a janela de visualização, conforme ilustra a Figura D6.

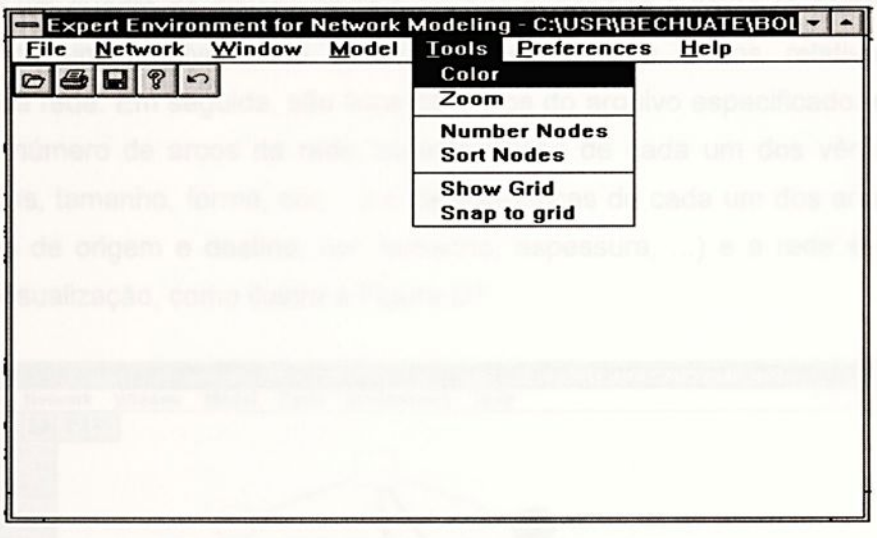


Fig. D6 - Tela Principal da Interface VIEW

Na barra de menu principal o usuário dispõe das seguintes opções e sub-opções:

File	Network	Window	Model	Tools	Preferences	Help
New	Edit	Normal	Problem	Color	Nodes	Index
Open	Clear all	Network	Solve	Zoom	Arcs	Using
Save		Solution		Number nodes	Sets	About
Save as				Sort nodes	Background	
Print				Show grid	Solutions	
Printer setup				Snap to grid	Solution colors	
Exit						

Na barra de ferramentas o usuário dispõe de ícones representando algumas opções usadas com frequência, tais como: *File/Open*, *File/Print*, *File/Save*, *Help/Using*, *Network/Edit*. Na janela de visualização, região principal da tela gráfica, o usuário define as características da rede e visualiza as soluções encontradas para os problemas.

Discute-se a seguir, de forma sucinta, algumas das opções do menu principal da interface VIEW.

File/New. Com esta opção o usuário habilita a criação de um novo arquivo no qual serão armazenados os dados relativos a uma determinada rede.

File/Open. Esta opção (acessível também por ícone na barra de ferramentas) apresenta ao usuário um quadro de diálogo onde o usuário especifica o nome (e correspondente diretório) de um arquivo onde estão armazenados os dados relativos a uma determinada rede. Em seguida, são lidos os dados do arquivo especificado: número de vértices e número de arcos da rede, características de cada um dos vértices (como coordenadas, tamanho, forma, cor, ...) e características de cada um dos arcos da rede (como, nós de origem e destino, cor, tamanho, espessura, ...) e a rede é exibida na janela de visualização, como ilustra a Figura D7.

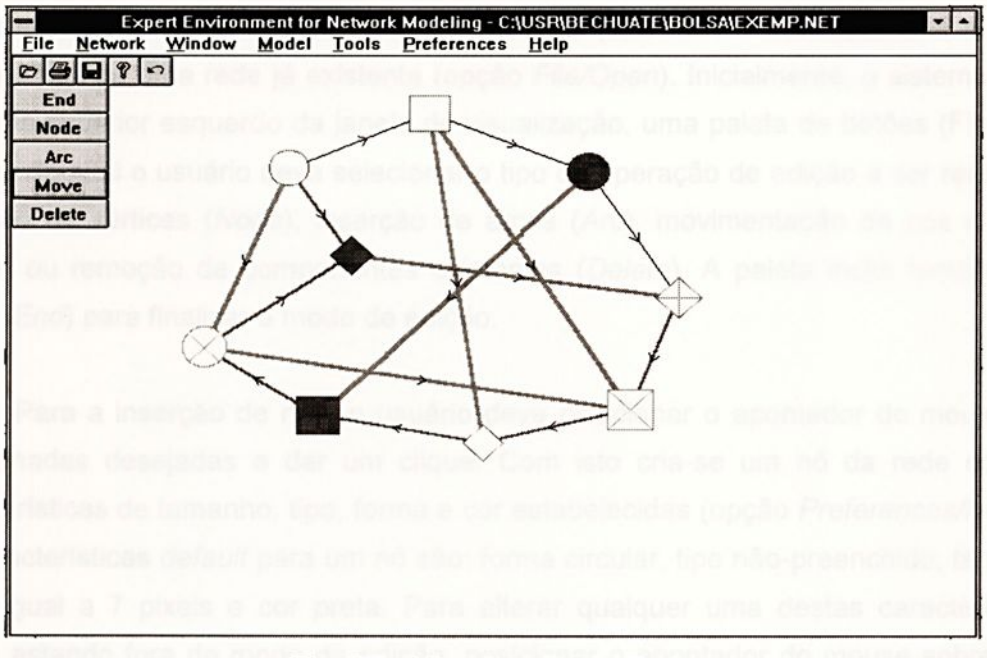


Fig. D7 - Visualização de Rede

File/Save. Salva em arquivo os dados relativos à rede apresentada na janela de visualização. Caso esta rede corresponda a um novo arquivo, ou seja, tenha sido construída através da opção *File/New*, invoca-se a opção *File/Save as*. Esta opção pode ser acessada também através de ícone na barra de ferramentas.

File/Save as. Esta opção apresenta um quadro de diálogo, espera que o usuário especifique o nome (e correspondente diretório) do arquivo onde serão armazenados os dados relativos à rede exibida na janela de visualização e, em seguida, invoca a opção *File/Save*.

File/Print. Esta opção imprime a rede que está sendo exibida na janela de visualização.

File/Printer setup. Configura a impressora e os parâmetros de impressão a serem usados pela opção *File/Print*.

File/Exit. Esta opção permite sair da interface VIEW, retornando à interface principal do ambiente MoNet. Antes de deixar a interface, caso a rede sendo exibida na janela de visualização tenha sido modificada, mostra-se ao usuário um quadro de diálogo com opções para salvamento e, se necessário, invoca-se a opção *File/Save*.

Network/Edit. Permite a edição de uma nova rede (caso tenha sido escolhida a opção *File/New*) ou de uma rede já existente (opção *File/Open*). Inicialmente, o sistema exibe no canto superior esquerdo da janela de visualização, uma paleta de botões (Figura 2) através da qual o usuário deve selecionar o tipo de operação de edição a ser realizada: inserção de vértices (*Node*), inserção de arcos (*Arc*), movimentação de nós e arcos (*Move*) ou remoção de componentes existentes (*Delete*). A paleta inclui também um botão (*End*) para finalizar o modo de edição.

Para a inserção de nós, o usuário deve posicionar o apontador do mouse nas coordenadas desejadas e dar um clique. Com isto cria-se um nó da rede com as características de tamanho, tipo, forma e cor estabelecidas (opção *Preferences/Nodes*). As características *default* para um nó são: forma circular, tipo não-preenchido, tamanho (raio) igual a 7 pixels e cor preta. Para alterar qualquer uma destas características basta, estando fora do modo de edição, posicionar o apontador do mouse sobre o nó em questão e dar um clique duplo. Com isto, o sistema exibe o quadro de diálogo mostrado na Figura D8, no qual o usuário define as características desejadas.

unesp

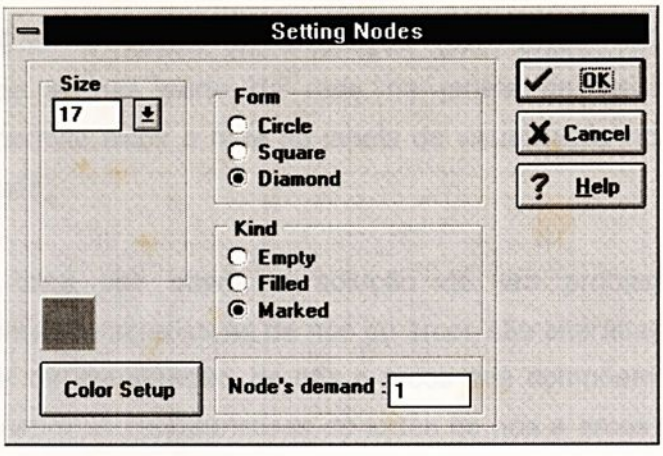


Fig. D8 - Quadro de Diálogo de Características de Nós

Para a inserção de arcos o usuário deve: apontar para o nó origem e dar um clique (o nó se torna acinzentado temporariamente para indicar esta operação) e apontar para o nó destino e dar um clique (um arco é inserido entre os dois nós com as características definidas a partir da opção *Preferences/Arcs* e o nó origem volta a sua cor normal). Para alterar qualquer uma destas características basta, estando fora do modo de edição, posicionar o apontador do *mouse* sobre o arco em questão e dar um clique duplo. Com isto, o sistema exibe o quadro de diálogo mostrado na Figura D9, no qual o usuário define as características desejadas.

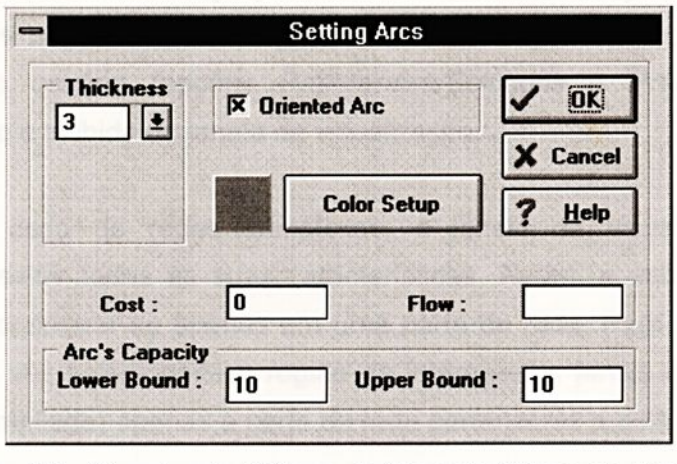


Fig. D9 - Quadro de Diálogo de Características de Arcos

A operação de movimentação é permitida apenas para os vértices da rede uma vez que, na movimentação de vértices os arcos são, automaticamente, realocados.

Window/Normal. Com a opção de ampliação (ver opção *Tools/Zoom*) pode-se selecionar e exibir apenas parte da rede na janela de visualização. A opção *Windows/Normal* permite exibir a rede na janela de visualização novamente em modo normal (não ampliado).

Window/Network. Uma vez obtida a solução de um problema (ver a opção *Model/Solve*), algumas características de nós ou arcos são alteradas, de modo a poder destacar, na janela de visualização, os nós e arcos que compõem a solução. Com a opção *Window/Network* as características de todos os nós e arcos são restabelecidas aos seus valores originais. O efeito disto é a apresentação apenas da rede (ou seja, não se exibe a solução obtida) na janela de visualização.

Window/Solution. Esta opção permite que a solução obtida volte a ser exibida na janela de visualização, através da alteração das características dos componentes da rede.

Model/Problem. Permite ao usuário especificar o tipo de problema a ser resolvido, correspondente à rede exibida na janela de visualização.

Model/Solve. Esta opção permite resolver o problema especificado (para a rede existente na janela de visualização). Uma vez obtida a solução do problema, as características dos nós e dos arcos que compõe a solução são estabelecidas como especificado pelo usuário (opções *Preferences/Solutions* e *Preferences/Solutions colors*) e a solução é exibida na janela de visualização.

Tools/Zoom. No caso de redes complexas, a janela de visualização pode ser insuficiente para exibir todas as suas características. Assim, o usuário poderia estar interessado em concentrar-se apenas em uma parte da rede. Para isto, com a opção *Tools/Zoom*, uma vez selecionada a região de interesse, a janela de visualização irá exibir (de forma ampliada) apenas a parte da rede pertencente a esta região.

Tools/Show grid. Com esta opção, é possível exibir uma grade na janela de visualização, na qual o usuário poderá ou não fixar os nós na rede (opcão *Tools/Snap to grid*). Esta é uma opção cujo estado pode ser ativo (exibir a grade) ou inativo (não exibir a grade). Um tique (símbolo ✓) irá aparecer à esquerda desta opção para indicar que seu estado atual é ativo.

unesp

Preferences/Solution colors. Esta opção é usada para estabelecer cores para faixas de valores de fluxo nos arcos, de modo a permitir a visualização (inclusive com animação) da solução de um determinado problema.

Help/Using. Esta opção exibe informação de ajuda sobre um tópico específico como, por exemplo, informações sobre o que constitui e quais são as informações necessárias para um problema tratado nesta interface.

unesp