MINIMIZAÇÃO DA AFECTAÇÃO DE RECURSOS NUM PROBLEMA DE CALENDARIZAÇÃO

por

Maria Fernanda Pereira Fonseca

MINIMIZAÇÃO DA AFECTAÇÃO DE RECURSOS NUM PROBLEMA DE CALENDARIZAÇÃO

por

Maria Fernanda Pereira Fonseca

Dissertação apresentada como requisito parcial para a obtenção de grau de Mestre em Métodos Matemáticos para Economia e Gestão de Empresas.

INSTITUTO SUPERIOR DE ECONOMIA

da

UNIVERSIDADE TÉCNICA DE LISBOA

LISBOA, 1989

AGRADECIMENTOS

Ao Professor Doutor Helder Coelho pela orientação estimulante e empenhada da presente dissertação.

Às minhas chefias hierárquicas dos TLP, S.A. pelas facilidades que me foram concedidas para a realização do Mestrado.

Ao Laboratório Nacional de Engenharia Civil pela utilização dos seus meios informáticos. Aos técnicos do Centro de Informática do LNEC pela disponibilidade demonstrada, contribuindo assim para a concretização deste projecto.

A todos os que directa ou indirectamente me apoiaram e incentivaram tornando possível a realização deste trabalho.

Todos os erros, lacunas e deficiências que esta dissertação possa apresentar são da exclusiva responsabilidade da sua autora.

ÍNDICE

RESUMO

A presente dissertação aborda a minimização da afectação de recursos num problema de calendarização ("scheduling").

O problema proposto consiste na minimização do custo global em salários de trabalhadores para satisfazer a realização de um determinado conjunto de actividades num dado período.

As actividades são constituídas por tarefas com relações de precedência entre si. Cada tarefa é efectuada por um tipo específico de trabalhador agrupado em equipas de um ou mais elementos. Cada uma destas equipas tem associado um tempo de ocupação para realização da tarefa. A cada tipo de trabalhador está associado um salário.

O sistema construído para a resolução do problema proposto "sistema de calendarização", tem uma arquitectura modular e está escrito na linguagem de programação PROLOG. É constituído por 8 módulos e tem uma ligação à biblioteca IMSL ("International Mathematical Standard Library"), para o acesso à rotina DDPLRS, que resolve o problema de programação linear associado à heurística.

O utilizador do "sistema de calendarização" pretende que todas as actividades por ele propostas sejam realizadas dentro do prazo máximo para ele especificado para a sua realização e que o custo global seja mínimo.

O "sistema de calendarização" para resolver a minimização do custo operacional global faz a análise temporal das actividades, e determina a flexibilidade temporal na realização de cada tarefa de todas as actividades. Aproveita a flexibilidade das tarefas de todas as actividades que a apresentam, e explora este grau de liberdade de modo a que conjugando-o com a modularidade das equipas de trabalhadores e o salário dos diferentes tipos de trabalhadores obtenha um custo global em salários mínimo.

O sistema proposto sendo modular é portável, no sentido de poder ser aplicado a vários tipos de calendarização, dada a generalidade como são definidas as actividades e o sentido lato que o custo pode ter.

A técnica de pesquisa que o "sistema de calendarização" utiliza para resolver este problema discreto de optimização é importada da área da Inteligência Artificial (o algoritmo de pesquisa A^*). Resolve com esta técnica um problema discreto de optimização do mundo real em termos de calendarização.

Os problemas de calendarização são um tema actual de investigação e de desenvolvimento em áreas como a Investigação Operacional (IO), a Inteligência Artificial (IA), (veja-se o exemplo da Gestão de Edifícios Inteligentes), Ciências da Decisão, Ciências de Computação, Automação de Fábricas, etc.

No sistema que agora relatamos recorremos a técnicas de IO e da IA, tendo elaborado uma nova heurística como contributo original para a resolução do problema escolhido.

PREFÁCIO

O problema que vai ser tratado consiste na minimização da afectação de recursos num problema de calendarização.

I - Descrição sumária do "Sistema de Calendarização"

O problema que se procura resolver pelo "Sistema de Calendarização" implementado é o de obter um escalonamento das actividades, sendo

Actividade - Conjunto de tarefas com relações de precedência entre si.

Tarefa - Tipo de trabalho a fazer, por um tipo específico de trabalhador agrupado em equipas alternativas para a sua realização.

Equipa - Agrupamento de um ou mais trabalhadores do mesmo tipo, que leva um determinado intervalo de tempo a concluir uma tarefa.

Salário - Custo assignado a um elemento de um determinado tipo de trabalhador.

Duração Máxima de uma Actividade - Duração máxima que o utilizador permite para a conclusão dessa actividade.

1 - Diálogo de utilizador com o "Sistema de Calendarização"

O utilizador do "Sistema de Calendarização" define o problema de calendarização que pretende resolver através de:

1.1 - Especificação de cada actividade que quer realizar indicando por que tarefas é constituída e as relações de precedência entre as tarefas.

1.2 - Especificação das modularidades das equipas de trabalhadores, ou seja como os diferentes tipos de trabalhadores podem estar agrupados nas equipas para realizar as tarefas.

1.3 - Especificação da ocupação temporal de cada equipa na realização de uma tarefa.

1.4 - Especificação do salário gasto com cada tipo de trabalhador.

1.5 - Especificação da duração máxima que permite para a conclusão de cada actividade.

1.6 - Especificação do dia de começo de cada actividade.

O conjunto destes 6 aspectos suportam o diálogo entre o utilizador e o "Sistema de Calendarização", onde se define o problema a resolver com esta ferramenta inteligente, como se apresenta esquematicamente na Fig. 1.

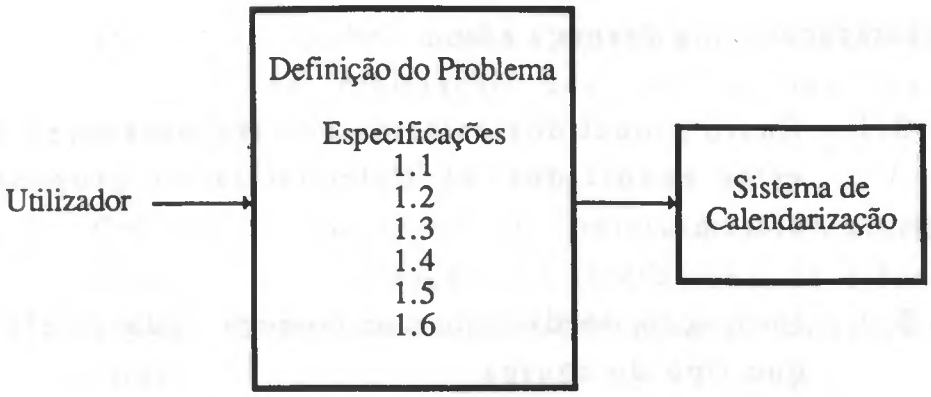


Fig. 1 - Interacção entre o utlizador e o sistema de calendarização

2 - Expectativas do utilizador em relação ao "Sistema de Calendarização"

O utilizador ao usar o "Sistema de Calendarização" para resolver o problema, pretende deste que:

2.1 - A calendarização proposta pelo "Sistema de Calendarização" respeite a duração máxima para cada actividade por ele especificada em 1.5.

2.2 - O custo operacional global seja mínimo. Pretende assim que com a calendarização proposta se gaste o mínimo possível com os salários dos trabalhadores que vão ser necessários para satisfazer a calendarização proposta.

3 - Diálogo do utilizador com o "Sistema de Calendarização"

As respostas que o utilizador pretende que o "Sistema de

Calendarização" lhe forneça são:

3.1 - Custo global dos salários dos trabalhadores que vão estar envolvidos na Calendarização proposta pelo sistema.

3.2 - Indicação do dia em que começa cada tarefa e com que tipo de equipa.

3.3 - Contingente necessário por cada tipo de trabalhador.

As respostas 3.1, 3.2 e 3.3 suportam o diálogo entre o sistema e o utilizador, como se representa esquematicamente na Fig. 2.

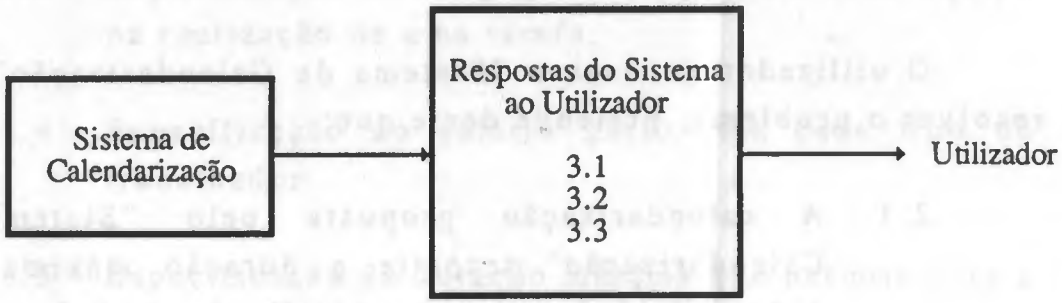


Fig. 2 - Interacção entre o sistema de calendarização e o utilizador

4 - Sistema de Calendarização

Internamente o sistema de calendarização é uma estrutura modular que:

4.1 - Analisa temporalmente as actividades, respeitando as relações de precedência entre as tarefas. Infere

assim o conhecimento relativo à flexibilidade temporal na realização das tarefas das diversas actividades.

4.2 - Conjuga o conhecimento relativo à flexibilidade temporal na realização das tarefas com as diferentes opções de equipas a escolher para a realização das tarefas.

A conjugação de 4.1 e 4.2 com o conhecimento relativo ao salário dos diferentes tipos de trabalhadores define o problema discreto de optimização que o sistema de calendarização se propõe resolver, recorrendo a:

4.3 - Algoritmo inteligente de pesquisa, o algoritmo A^*.

4.4 - Heurística implementada para orientar inteligentemente a busca do algoritmo A^*.

4.5 - Custos de transmissão entre etapas de calendarização, ou seja custo de transmissão entre estados definidos pela árvore de busca explicitada pelo algoritmo A^*.

A estrutura interna do "Sistema de Calendarização" é representada esquematicamente na Fig. 3.

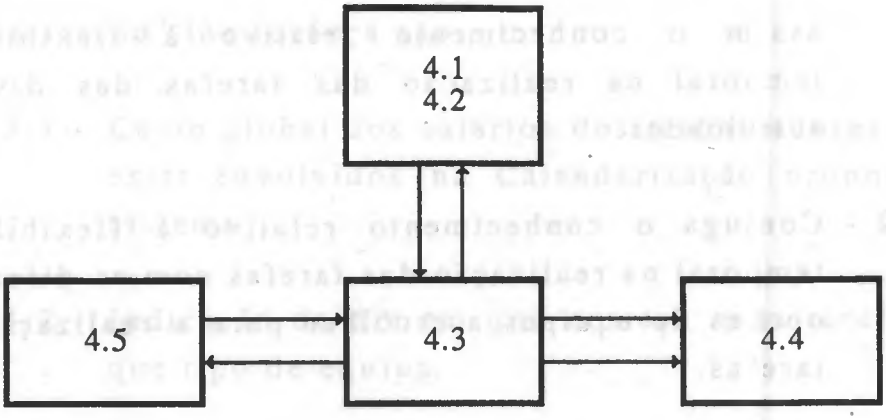


Fig. 3 - Estrutura interna do sistema de calendarização

II - Descrição da estrutura da dissertação

A dissertação está organizada em 2 PARTES, sendo a 1ª PARTE constituída por 11 capítulos, abaixo discriminados:

Capítulo 1 - Especificação do Problema

Este capítulo explicita o problema e a sua formalização.

Capítulo 2 - Metodologia de Resolução

Propõe-se uma discussão dos métodos possíveis da resolução do problema discreto de optimização que foi definido no capítulo 1, justificando o método de resolução escolhido que combina conhecimentos da Inteligência Artificial e da Investigação Operacional.

Capítulo 3 - Arquitectura do Sistema de Calendarização

Apresenta-se a arquitectura modular do sistema de calendarização, explicitando-se as ligações funcionais de cada módulo do sistema.

Capítulo 4 - Representação do Conhecimento Associado ao Problema de Calendarização a Resolver

Detalhadamente discute-se como se tranfere o fluxo de conhecimento que o utilizador tem do problema a resolver para o "Sistema de Calendarização", ou seja o diálogo do utilizador com o sistema.

Capítulo 5 - Representação do Conhecimento Associado à Resolução do Problema de Calendarização

Descreve-se o fluxo de conhecimento associado à resolução do problema, do sistema para o utilizador.

Capítulo 6 - Inferência Automática do Conhecimento

Aborda a inferência automática do conhecimento, que está armazenado na "Base de Conhecimento" do sistema.
Esta "Base de Conhecimento" é constituida por conhecimento associado às actividades, ao caminho crítico das actividades, ao relaxamento das repetições das actividades, e ainda por conhecimento geral para utilização na Heurística e para utilização no Problema de Programação Linear associado à Heurística.

Capítulo 7 - Descrição da Árvore de Estados

Descreve-se a Estrutura de Estado, as Regras de Produção e Estratégia de Controle que definem a árvore de estados explicitada pelo algoritmo de pesquisa A^*.

Capítulo 8 - Custo de Transição entre Estados

Descreve-se a contabilização do custo de transição entre os estados de árvore de estados explicitada pelo algoritmo de pesquisa A^*.

Capítulo 9 - Heurística Adoptada

Descreve a conceptualização da Heurística adoptada, e determinação da Heurística adoptada associada a cada estado. Descreve-se também o módulo de comunicação do "Sistema de Calendarização" com a biblioteca IMSL para resolução do Problema de Programação Linear associado à Heurística.

Capítulo 10 - Algoritmo de Pesquisa

Apresentam-se as propriedades formais do algoritmo de pesquisa A^*, e faz-se a prova da admissibilidade da heurística adoptada.

Capítulo 11 - Conclusões

Na 2ª PARTE apresentam-se os resultados computacionais e em ANEXO adiciona-se a listagem dos programas do "Sistema de Calendarização" escritos em PROLOG.

INTRODUÇÃO

O trabalho apresentado, "Minimização da Afectação de Recursos num Problema de Calendarização", apoia-se num "sistema (inteligente) de calendarização" de actividades de modo a que o custo global dispendido em salários para a realização de todas as actividads seja mínimo.

O utilizador do "sistema de calendarização" pretende obter um escalonamento das actividades que deseja realizar num determinado período, com a mínima afectação de recursos de pessoal, ou seja dispendendo o menos possível em salários. O problema do utilizador consiste assim na realização, num determinado período, de diversas actividades, que são constituídas por tarefas com relações de precedência entre si.

As tarefas são realizadas por tipos específicos de trabalhadores. E, para a realização de uma certa tarefa, o tipo de trabalhador para a realizar está agrupado em equipas de um ou mais elementos.

Cada equipa consome um determinado período de tempo para a realização de uma tarefa, sendo que equipas menores levam mais tempo a realizar uma tarefa, e as maiores menos tempo, podendo ter um ganho de factor de escala.

A este agrupamento de elementos em equipas para a realização das tarefas chamaremos modularidade das equipas.

A especificação do problema realizada pelo utilizador consta assim da descrição das actividades que pretende realizar, da atribuição da modularidade das equipas, da fixação do tempo máximo que permite para a realização de cada actividade, assim como da definição do custo associado a cada tipo de trabalhador. A especificação das actividades é feita indicando as tarefas que constituem cada actividade com as suas relações de precedência.

O utilizador deseja que o sistema lhe calendarize as tarefas das actividades, de modo a ter um custo global em salários mínimo.

Para a pesquisa da solução óptima, o "sistema de calendarização" vai fazer uma análise temporal exaustiva das actividades, para determinar a flexibilidade temporal na realização das tarefas de todas as actividades.

É a conjugação do:

- Grau de liberdade dado pela flexibilidade temporal da realização das tarefas, do
- Grau de liberdade dado pela modularidade das equipas, e dos
- Diferentes tipos de salários de trabalhadores,

que constitui o problema discreto de optimização que o "sistema de calendarização" resolve.

Este problema de calendarização ("scheduling") atrás exposto é comum a muitas áreas de acção. Uma vasta gama de empresas defrontam-se com o problema exposto, como por exemplo empresas de construção civil que pretendem calendarizar trabalhos de estaleiro, empresas vocacionadas para manutenção de parques industriais que pretendem o mínimo dispêndio em salários das equipas para realizar as actividades de manutenção, empresas de manutenção de imóveis, companhias de transportes que pretendam a calendarização das actividades de manutenção do parque móvel, etc., sendo que o objectivo comum em todos os casos é a minimização do custo global operacional para a realização das actividades, havendo uma forte motivação e um mercado crescente para sistemas de calendarização.

O mercado para sistemas de "scheduling" não tem parado de crescer nos últimos anos encontrando-se em fase ascendente de procura.

Sob a crescente e urgente procura pelo mercado de sistemas de calendarização, eles constituem uma área de investigação e desenvolvimento em disciplinas como Ciências de Decisão, Ciências de Computação, Investigação Operacional, Inteligência Artificial, sendo bem conhecidos por exemplo os problemas de clandarização associados à automação de fábricas, escalonamento em fábricas, e à gestão de edifícios inteligentes.

No projecto JNICT N. 87.454, "Sistemas Avançados para Edificíos Inteligentes", este problema surge associado à manutenção, face à quantidade e diversidade de equipamentos em presença. A sua resolução implicará ganhos apreciáveis e obviamente melhores serviços para os utentes dos edificíos.

A complexidade, inerente ao tratamento de sistemas de calendarização, impele a uma convergência das disciplinas acima referidas para um esforço conjunto de investigação e desenvovimento de sistemas de calendarização que satisfaçam a procura de mercado nas mais variadas áreas.

O sistema de calendarização proposto pelo presente trabalho propõe-se como contributo para esta área de investigação e desenvolvimento.

Para resolver o problema discreto de optimização que o sistema de calendarização constitui, várias alternativas se apresentavam: algoritmos de pesquisa, tradicionalmente conectados com a área de Investigação Operacional, e algoritmos de pesquisa da área de Inteligência Artificial.

De entre os algoritmos da primeira área, tinham-se como alternativas possíveis algoritmos de pesquisa tipo guloso, algoritmos de pesquisa em profundidade, algoritmos de pesquisa em largura, algoritmos tipo Partição e Avaliação, "Branch and Bound" ou Programação Dinâmica. Optou-se por um algoritmo de pesquisa da área de Inteligência Artificial, o algoritmo de pesquisa A^*. O "sistema de calendarização" é

assim um sistema que usa um algoritmo A^* para busca da solução do problema de calendarização. O algoritmo A^* é guiado por uma heurística que garante as condições de admissibilidade de A^*, pelo que o "sistema de calendarização" garante a solução óptima.

O respeito estrito pela admissibilidade da heurística implementada, garante assim uma solução óptima para a calendarização, justificando o título do trabalho: "Minimização da Afectação de Recursos num Problema de Calendarização".

O "sistema de calendarização" é um sistema modular, escrito em PROLOG, com 2430 linhas de programação. É constituído por 8 módulos básicos:

- Módulo de comunicação utilizador → "sistema de calendarização"
- Módulo de comunicação "sistema de calendarização" → → utilizador
- Base de Conhecimento (inferência automática do conhecimento)
- Algoritmo de pesquisa A^*
- Regras de Produção e Estratégia de Controlo
- Custo de Transição entre Estados
- Heurística
- Módulo de comunicação do "sistema de calendarização" com a biblioteca IMSL

A heurística implementada, tem associado um problema de programação linear que é resolvido numa biblioteca exterior,

IMSL ("International Mathematical Standard Library"), especificamente pela rotina DDPLRS escrita em FORTRAN, donde a necessidade do módulo de comunicação entre o "sistema de calendarização" e a biblioteca.

Dada a modularidade do "sistema de calendarização" ele é portável no sentido de se poder fazer uma alteração num dos módulos sem se alterar a estrutura interna do sistema. Por exemplo, se se quisesse fazer uma alteração na heurística adoptada, apenas se alterava o módulo da heurística sem alterar os outros módulos da estrutura interna do "sistema de calendarização".

O "sistema de calendarização" é também portável no sentido de poder ser aplicado a uma vasta gama de problemas. Dada a generalidade como são definidas as actividades e o sentido lato associado ao salário dos trabalhadores, que pode representar qualquer custo operacional, o sistema é aplicável a uma grande variedade de problemas, e portanto portável nesse sentido.

O trabalho "Minimização da Afectação de Recursos num Problema de Calendarização" implementado num sistema inteligente a que se chamou "sistema de calendarização" é apresentado como requisito parcial para obtenção do grau de Mestre em Métodos Matemáticos para Economia e Gestão de Empresas, e dado o objectivo que se propõe resolver, perfeitamente enquadrado no âmbito e abrangência do referido Mestrado.

PARTE I

1. - ESPECIFICAÇÃO DO PROBLEMA

1. 1 - INTRODUÇÃO

Neste primeiro capítulo vai-se fazer a descrição detalhada do problema de calendarização e sua representação, do seu objectivo que é a minimização da afectação de recursos, e a sua formalização.

1. 2 - DESCRIÇÃO E REPRESENTAÇÃO DO PROBLEMA

A minimização da afectação de recursos num problema de calendarização consiste na optimização do custo global gasto em salários dos trabalhadores necessários para realizar as actividades que se querem realizar.

Portanto a motivação do problema consiste em:

- Executar as actividades que são especificadas com o mínimo de custo.

Entende-se por:

Actividade - conjunto de tarefas com relações de precedências entre si

Tarefa - tipo de trabalho a fazer, por um tipo específico de trabalhador. Os trabalhadores para executarem uma tarefa estão agrupados em equipas de um ou mais elementos

A representação das actividades que se querem realizar é feita pelos grafos orientados associados às actividades. Nos grafos orientados associados às actividades, cada arco corresponde a uma tarefa a ser executada. Em 1. 5 apresenta-se um resumo dos conceitos e da notação da teoria dos grafos que vai ser usada neste e nos capítulos seguintes.

Vejamos um exemplo de uma actividade constituida por 5 tarefas com a seguinte tabela de precedências entre as tarefas:

Tarefa	Precedência
1	-
2	1
3	2
4	1
1	3,4

Esta actividade seria representada pelo grafo apresentado na Fig. 1.1

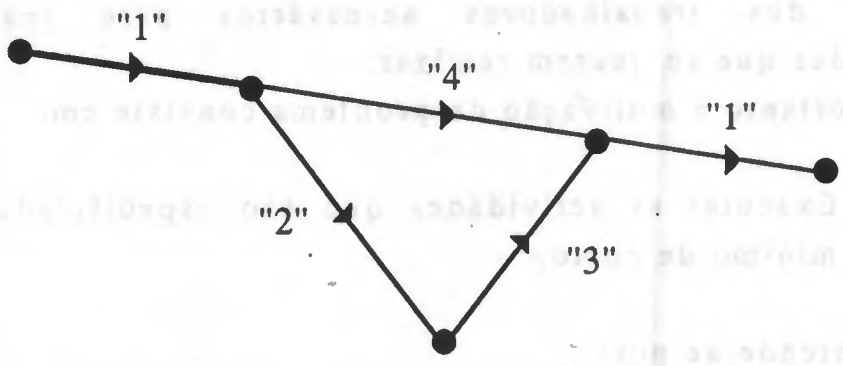


Fig. 1.1 - Grafo associado à actividade.

Portanto todas as actividades que se pretendem realizar vão ser representadas por grafos orientados associados.

Pode-se pretender que uma actividade seja realizada mais de uma vez durante o período em estudo, pelo que se individualizam as repetições das actividades.

No âmbito da descrição e representação do problema está também a especificação da:

Duração Máxima de uma Actividade - duração máxima que o utilizador permite para a conclusão de uma actividade.

Salário - custo assignado a cada tipo de trabalhador.

Equipas - descrição das alternativas de agrupamento de trabalhadores de um determinado tipo para realização de uma tarefa. A cada alternativa de agrupamento está associada uma duração para a realização da tarefa.

Por exemplo, a tarefa 2 é realizada por trabalhadores do tipo 2, com as seguintes alternativas de equipas:

Descrição da equipa		Ocupação da equipa para realizar a tarefa
2 trabalhadores tipo 2	→	2 dias
3 trabalhadores tipo 2	→	1 dia

Com a descrição e representação das actividades que se querem realizar, do dia do seu começo e das suas durações máximas permitidas, assim como das equipas alternativas para a realização das tarefas, temos descrito e representado o problema que o utilizador quer resolver, gastando o mínimo possível em salários. No capítulo 4, especifica-se detalhadamente como se

estabelece o diálogo do utilizador com o "sistema de calendarização".

1. 3 - OBJECTIVO DO PROBLEMA

O próprio título da dissertação, "Minimização da afectação de recursos num problema de calendarização" sugere que o objectivo consiste em:

- **Calendarizar** a realização das actividades, ou seja calendarizar as tarefas que constituem todas as actividades que vão ser executadas no período em estudo.

- **Minimizar** o custo de salários gastos na realização das actividades.

Portanto, o objectivo consiste em minimizar o custo global gasto em salários, sendo esta minimização possível dado o grau de flexibilidade temporal que alguns ou todos os arcos dos grafos das actividades têm.

O "sistema de calendarização" faz uma análise temporal de todas as actividades, determinando quais os arcos que apresentam flexibilidade temporal.

Um arco apresenta flexibilidade temporal se apresenta um grau de liberdade quanto ao dia de começo da sua realização.

Vejamos um exemplo. Considere-se que tinhamos a seguinte actividade:

Tarefa	Precedência
1	-
2	1
3	2
4	1
1	3,4

cujo grafo associado é indicado na Fig. 1.2

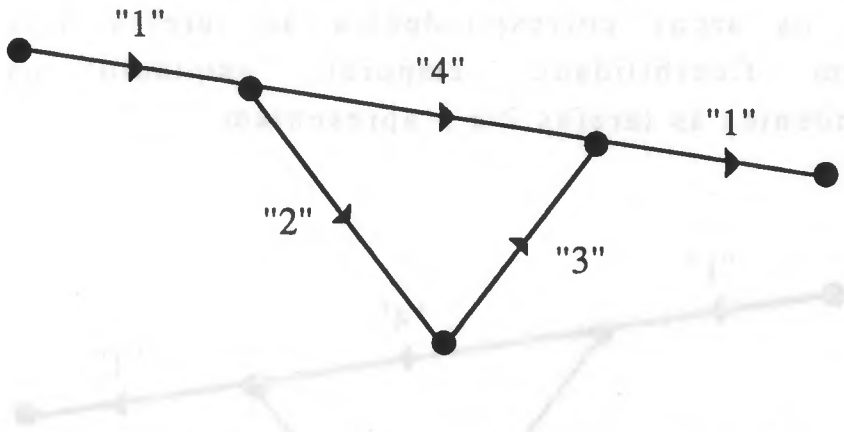


Fig. 1.2 - Grafo associado à actividade

e que os trabalhadores para realizar as tarefas eram

Tarefa	Tipo de trabalhador
1	tipo 1
2	tipo 2
3	tipo 3
4	tipo 4
1	tipo 1

Os trabalhadores apresentavam uma modularidade de equipas especificada. A equipa descreve-se por:

Número de trabalhadores, Número de dias para realizar a tarefa

Tipo de Trabalhador	Descrição de Equipas
tipo 1	1,1
tipo 2	2,2; 3,1
tipo 3	2,3; 3,1
tipo 4	1,4

Se se quisesse realizar a actividade no mínimo tempo possível, os arcos correspondentes às tarefas 1 e 4 não apresentam flexibilidade temporal, enquanto os arcos correspondentes às tarefas 2 e 3 apresentam

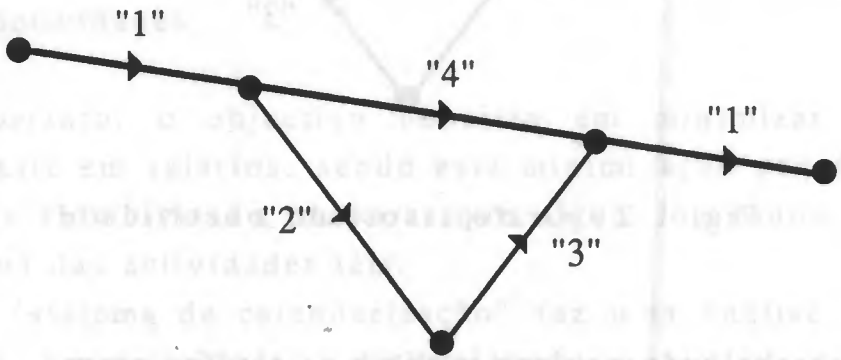


Fig. 1.3 - Grafos com indicação dos arcos com flexibilidade temporal

Da análise do grafo e da modularidades das equipas se vê

que se gastava um tempo mínimo para realização da actividade de 6 dias,

Tarefa 1 → 1 dia
Tarefa 4 → 4 dias
Tarefa 1 → 1 dia

e que as tarefas 2 e 3 podem gastar conjuntamente 4 dias, apresentando flexibilidade temporal.

Neste caso a minimização do custo global de salários entrava apenas em jogo com as flexibilidades temporárias das tarefas 2 e 3, e modularidades das equipas de trabalhadores para as realizar. Portanto a minimização vai jogar com isso e com os salários dos trabalhadores envolvidos.

Tabela de Salários:

Tipo de Trabalhadores	Salário
tipo 1	1
tipo 2	2
tipo 3	5
tipo 4	1

Da análise temporal da actividade, sabe-se que as tarefas 2 e 3 podem demorar conjuntamente no máximo 4 dias. Portanto as hipóteses de realização são:

Alternativa 1:

	Nº de dias	Nº de trabalhadores	Custos
tarefa 2	2	2	4
tarefa 3	1	3	15
total de custos			19

Alternativa 2:

	Nº de dias	Nº de trabalhadores	Custos
tarefa 2	1	3	6
tarefa 3	3	2	10
total de custos			16

Donde a alternativa a escolher seria a alternativa 2, para realizar as tarefas 2 e 3.

Se se quisessem realizar 2 actividades A e B iguais à referida num período de 9 dias, teriamos que realizar conjuntamente as tarefas 2 e 3 das actividades A, B chamemos-lhes 2A, 3A, 2B e 3B, em 7dias, e de entre todas as alternativas possíveis, a mais barata para realizar as tarefas 2A, 3A, 2B e 3B seria:

	Nº de dias	Nº de trabalhadores	Custos
tarefa 2A	1	3	6
tarefa 2B	1	3	-
tarefa 3A	3	2	10
tarefa 3B	3	2	-
total de custos			16

em que a calendarização das tarefas 2A, 3A, 2B e 3B seria

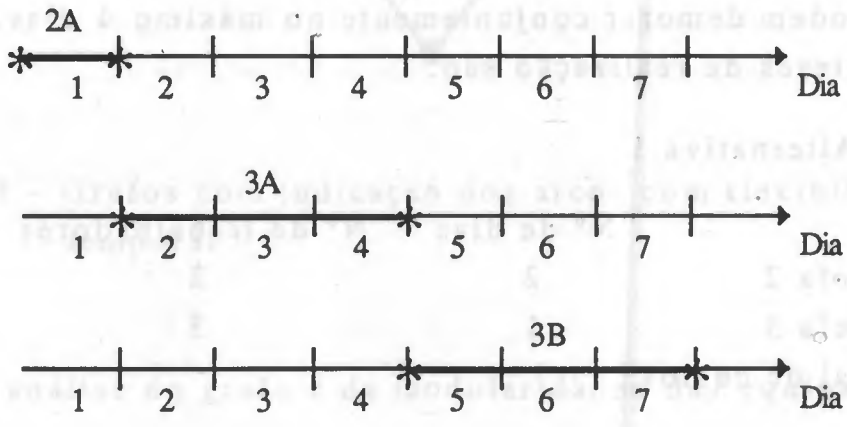


Fig. 1.4 - Calendarização das tarefas 2A, 3A e 3B

tendo a tarefa 2B o grau de liberdade de ser efectuada no dia 2, 3 ou 4.

Portanto a minimização obteve-se conjugando os factores:

- Flexibilidade temporal dos arcos correspondentes às tarefas 2 e 3.

- Modularidades das equipas de trabalhadores necessários para as tarefas com flexibilidade temporal.

- Salários dos mesmos trabalhadores.

De entre todas as combinações possíveis resultantes da conjugação dos factores escolhe-se a de menor custo.

Para este exemplo em que se tem de realizar as 2 actividades A e B num período de 9 dias, se as tarefas 1 e 4 tivessem mais alternativas de modularidade de equipas para as realizar, ficavam também com flexibilidade temporal, e o problema complicava-se obviamente, aumentando o número de combinações possíveis para calendarização de todas as tarefas, das quais se pretende a mais económica.

Serviram estes exemplos para uma compreensão mais concreta do problema discreto de optimização que é a "minimização da afectação de recursos num problema de calendarização".

A implementação do "sistema de calendarização" é a ferramenta inteligente proposta para resolver o problema discreto de optimização exposto.

1.4 - FORMALIZAÇÃO DO PROBLEMA

O problema discreto de optimização descrito anterior-

mente vai ser formalizado como um Processo de Decisão Composto ("Composite Decision Process" ou simplesmente "CDP"), segundo Kumar e Kanal, (Kumar, V. e Kanal, L., 1988).

1.4.1 - Descrição da Estrutura do Processo de Decisão Composto

Um Processo de Desisão Composto tem associados 3 elementos

1 - Gramática - G
2 - Custo de Produção - t
3 - Função de Custo Recursiva - c

1 - Gramática - G

A gramática do Processo de Desisão Composto é um tuplo 4-àrio

G (V, N, S, P)

onde S - estado inicial para a gramática
N - conjunto de estados não terminais
V - conjunto de estados terminais
P - conjunto das produções

sendo

Estado - símbolo que corresponde a uma situação, uma configuração do processo, associado a uma Estrutura de Estado.

Estrutura do Estado - estrutura de representação que possa representar todos os símbolos possíveis do processo. Um símbolo associado a uma Estrutura de Estado chama-se Estado.

Regras de Produção - regras capazes de transformar o código da estrutura de um estado na estrutura de outro estado seu sucessor, sujeitando-se a uma estratégia de controlo.

Estratégia de Controlo - processo sistemático de utilização das regras de produção que garanta a geração de todos os estados sucessores possíveis de um estado, e que não gere um estado mais de uma vez para evitar repetições.

Cada produção p em P é da forma

$$p: n \rightarrow w$$

onde n é um estado não terminal, $n \subset N$ e w é um caminho em $(N \cup V)$.

Isto traduz que um caminho w em $(N \cup V)$ é derivado de um estado não terminal em N se w pode ser obtido por uma aplicação sucessiva de regras de produção. Ou seja uma produção em P é uma aplicação sucessiva de regras de produção.

A gramática do Processo de Decisão Composto gera precisamente estes caminhos w de estados não terminais para estados terminais pela aplicação de elementos de P, incluíndo-se nos estados não terminais o estado inicial S.

Um elemento não terminal B de N, ao derivar um conjunto x de elementos de V, descreve uma árvore de derivação T. Assim, B é a raiz da árvore de derivação T com estados

terminais que constituem o conjunto $x \subset V$, sendo T a árvore de derivação de x a partir de B.

2 - Custo de Produção - t

Cada produção p em P tem associada uma função custo de produção t_p.

Seja a produção

$$p: n \rightarrow w$$

onde n é um estado em N e w um caminho em $(N \cup V)$, $w = n, n_1 \cdots n_k$. Então t_p é uma função custo de k elementos, associada à produção p.

3 - Função de Custo Recursiva - c

Seja $c: V \rightarrow R$ uma função real definida sobre o conjunto dos estados terminais.

A função c_T pode ser definida sucessivamente nos nodos de uma árvore T como se segue:

i) Se n é um estado terminal de G, tem-se

$$c_T(n) = c(n)$$

ii) Se n é um estado não terminal e $n \cdots n_k$ são descendentes de n em T, isto é se $p: n \rightarrow n_1, \cdots n_k$ é uma produção em G, tem-se

$$c_T(n) = t_p(c_T(n_1), \cdots c_T(n_k))$$

Se n é a raiz de uma árvore de derivação T, uma função de custo f é definida

$$f(T) = c_T(n) \qquad (1.1)$$

onde $c_T(n)$ é o custo da árvore T com raiz em n.

Para uma produção p: $n \rightarrow n_1, \cdots n_k$, $t_p(a_1, \cdots a_k)$ é o custo da árvore de derivação T com raiz em n, se $a_1, \cdots a_k$ forem os custos das sub-árvores de T com raizes em $n_1, \cdots n_k$.

Isto significa que o custo da árvore de derivação é definido recursivamente à custa dos custos das suas sub-árvores.

1.4.2 - Processo de Decisão Composto

Um Processo de Decisão Composto é um tuplo 3-ário,

$$PDC = (G(V,N,S,P),\ t,\ c)$$

onde G é uma gramática, t um custo de produção e c uma função de custo recursiva definidos em 1.4.1.

Uma função de custo f(T) definida em (1.1) está associada a cada árvore de derivação T de G.

O problema de minimização para o Processo de Decisão Composto, PDC é então encontrar a árvore de derivação T^*, tal que

$$f(T^*) = \min \{f(T) | T \text{ é uma árvore de derivação com raiz em } S\}$$

1.4.3 - Formalização do PDC Associado ao Problema de Calendarização

No nosso problema de calendarização, o estado inicial está associado ao dia zero em que nada está realizado, e os estados sucessores associados aos dias da calendarização, pelo que as transições de estado correspondem a transições diárias.

As transições diárias estão sujeitas a restrições temporais e de modularidade das equipas para realizar as tarefas:

Restrições Temporais
- Dia de começo de cada actividade
- Relações de precedência entre as tarefas das actividades

Restrições de Modularidade das Equipas

Estas restrições temporais conjuntamente com as restrições das modularidades das equipas definem as regras de produção associadas ao problema.

O problema de calendarização vai definir uma árvore de estados, cujos níveis de profundidade correspondem aos dias da calendarização, e cujos graus de abertura estão relacionados com as regras de produção. O grau de abertura de um estado depende da conjugação das restrições temporais e das restrições impostas pela modularidade das equipas.

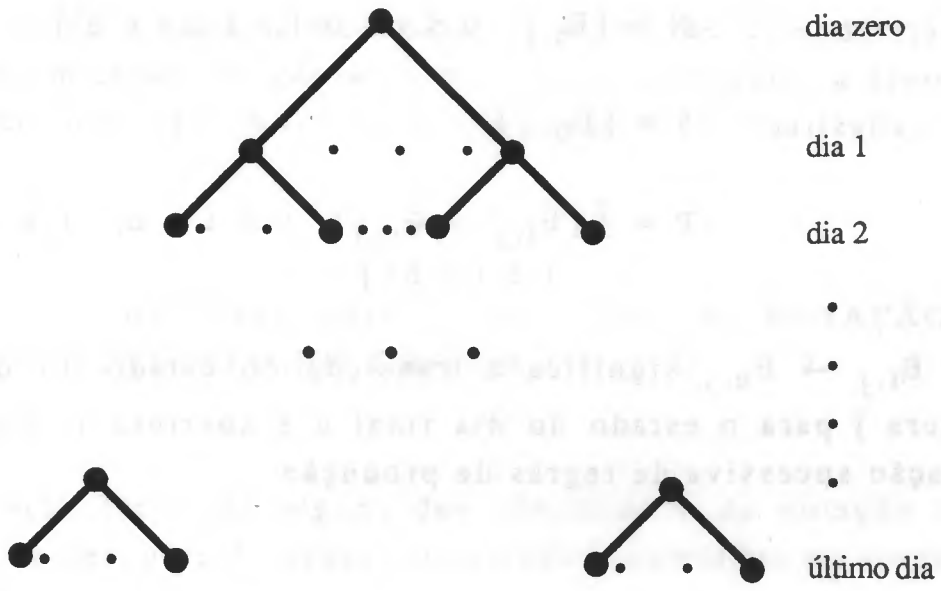


Fig. 1.5 - Árvore de estados do problema de calendarização

Seja

u - último dia de calendarização

b - grau de abertura máxima de um estado da árvore de estados

$E_{i,j}$ - estado associado ao dia i e abertura j

$c_{i,\ i+1}$ - custo da transição permitida por uma regra de produção de um estado associado ao dia i para um estado associado ao dia i+1

Vai-se formalizar o problema de calendarização como um Processo de Decisão Composto:

$$PDC = (G(V,N,S,P),t,c)$$

$$G(V,N,S,P):$$

$$V = \{E_{u,k},\ 1 \leq k \leq b^u\}$$

$$N = \{E_{i,j},\ 0 \le i < u-1,\ 1 \le j \le b^i\}$$

$$S = \{E_{0,1}\}$$

$$P = \{\{E_{i,j} \rightarrow E_{u,l}\},\ 0 \le i < u,\ 1 \le j \le b^i,\ 1 \le l \le b^u\}$$

onde $E_{i,j} \rightarrow E_{u,l}$ significa a transição do estado do dia i e abertura j para o estado do dia final u e abertura l, sujeita à aplicação sucessiva de regras de produção

t:

Para a produção p: $E_{i,j} \rightarrow E_{u,l}$

tem-se:

$$t_p(E_{i,j}) = \sum_{k=i}^{u-1} c_{k,k+1}$$

c:

$$c(E_{i,j}) = \min \{t_{p \in P}(E_{i,j})\}$$

O problema de minimização do Processo de Decisão Composto

$$PDC = (G(V,N,S,P),t,c)$$

é encontrar a árvore de derivação T^* tal que

$$f(T^*) = \min \{f(T) | T \text{ é uma árvore de derivação com raiz em } S\}$$

Como a solução do nosso problema de calendarização é apenas um estado do conjunto de estados terminais, a árvore T^* degenera num caminho, cuja minimização foi formalizada.

1. 5 - RESUMO DOS CONCEITOS E NOTAÇÃO DA TEORIA DOS GRAFOS

Abordar-se-ão alguns dos conceitos e da notação básica usada na pesquisa do espaço de estados, extraídos da teoria dos grafos.

Grafo não orientado:

Um grafo consiste num conjunto de nodos ou vértices

$$V = \{v_i \mid i=1, \ldots N\}$$

associado a um conjunto de arestas.

Os pares de nodos estão ligados, gerando pares não ordenados de elementos de V, chamados arestas

$$A = \{[v_i, v_j] \mid v_i \in V,\ v_j \in V\}$$

$$A \subset V^2$$

O grafo não orientado é então definido por

$$G = (V, E), \qquad E \subseteq A$$

V - conjunto dos nodos
E - conjunto das arestas

Exemplo:

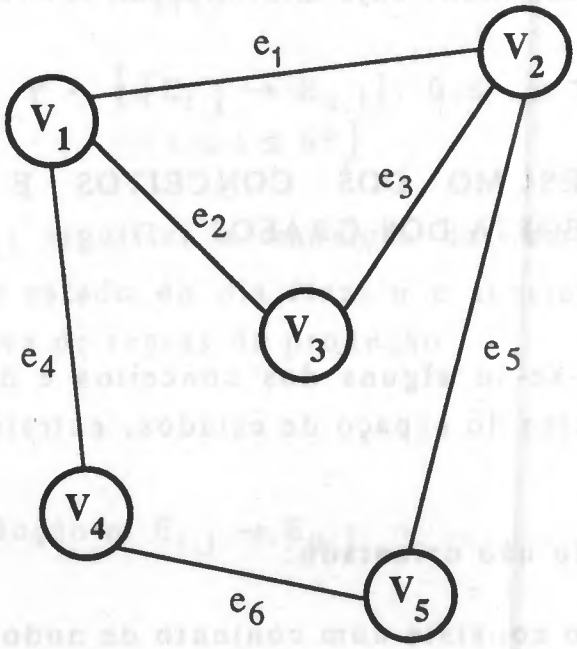


Fig. 1.6 - Grafo não orientado

$$V = \{v_1, v_2, v_3, v_4, v_5\}$$
$$E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$$

onde

$$e_1 = [v_1, v_2] \equiv [v_2, v_1]$$
$$e_2 = [v_1, v_3] \equiv [v_3, v_1]$$
$$e_3 = [v_2, v_3] \equiv [v_3, v_2]$$
$$e_4 = [v_1, v_4] \equiv [v_4, v_1]$$
$$e_5 = [v_2, v_5] \equiv [v_5, v_2]$$
$$e_6 = [v_4, v_5] \equiv [v_5, v_4]$$

Grafo orientado:

Num grafo orientado, temos o conjunto de nodos V, associado a um conjunto de arcos. Pares de nodos estão ligados gerando pares ordenados de V, chamados arcos

$$Q = \{[v_i, v_j] \mid v_i \in V,\ v_j \in V\}$$

$$Q \subset V^2$$

Define-se assim um grafo orientado

$$G = (V, U), \qquad U \subseteq Q$$

V - conjunto dos nodos
U - conjunto dos arcos

Exemplo:

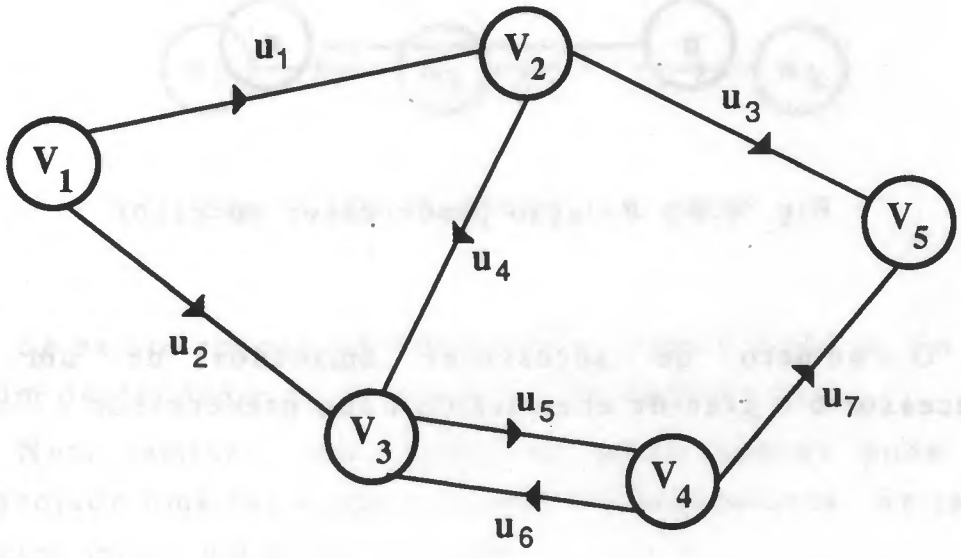


Fig. 1.7 - Grafo orientado

$$V = \{v_1, v_2, v_3, v_4, v_5\}$$
$$U = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}$$

onde

$u_1 = [v_1, v_2]$
$u_2 = [v_1, v_3]$
$u_3 = [v_2, v_5]$
$u_4 = [v_2, v_3]$
$u_5 = [v_3, v_4]$
$u_6 = [v_4, v_3]$
$u_7 = [v_4, v_5]$

Os grafos que vamos considerar têm um único nodo inicial.

Se um arco é dirigido de um vértice n para um vértice n', n' é um sucessor de n, sendo n o predecessor de n'.

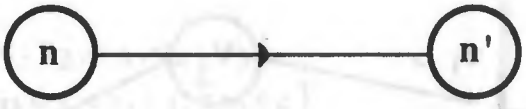


Fig. 1.8 - Relação predecessor-sucessor

O número de sucessores emanados de um nodo predecessor é o grau de abertura do nodo predecessor

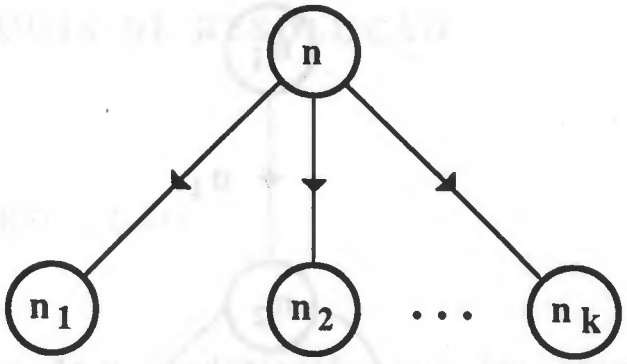


Fig. 1.9 - Grau de abertura de um nodo

o grau de abertura do nodo n é k.

O grau de abertura de um nodo pode ser finito ou infinito. Os grafos em que qualquer nodo tem grau de abertura finito definem uma classe de grafos localmente finitos.

Uma sequência de nodos n_1, n_2, ... n_k onde cada n_i é sucessor de n_{i-1} é um caminho de comprimento k do nodo n_1 para o nodo n_k.

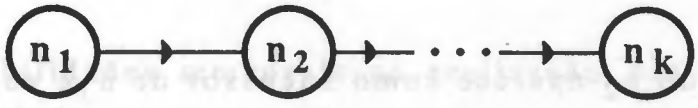


Fig. 1.10 - Caminho num grafo

Se existe um caminho do nodo n_1 para o nodo n_k, o nodo n_k é um descendente de n_1 sendo n_1 ascendente de n_k.

Num caminho, um vértice do grafo apenas pode estar representado uma vez como sucessor ou predecessor. Se tal não acontecer temos um ciclo definido.

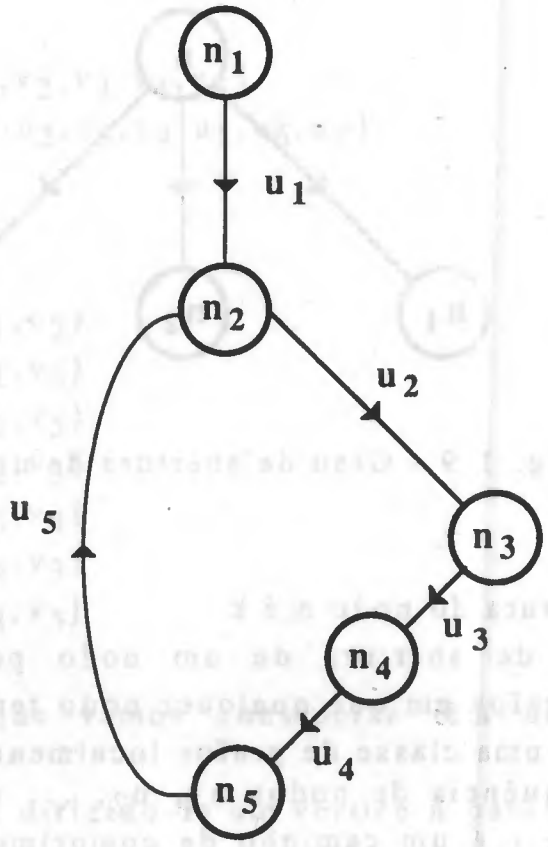


Fig. 1.11 - Ciclo num grafo

O nodo n_2 aparece como sucessor de n_1 e sucessor de n_5.

Os grafos orientados podem ter associados aos arcos um custo, podendo neste caso os grafos terem o nome particular de redes. No nosso caso, aos grafos orientados com um custo associado aos arcos, continuaremos a chamar grafos.

Árvore é um grafo em que cada nodo tem apenas um predecessor (é um grafo sem ciclos). Um nodo de uma árvore que não tem sucessores é um nodo terminal.

2. - METODOLOGIA DE RESOLUÇÃO

2. 1 - INTRODUÇÃO

O problema de calendarização que foi especificado no Capítulo 1 é um problema discreto de optimização. O problema gera uma árvore de estados, em que os estados representam os passos intermédios da calendarização. O estado inicial corresponde ao dia inicial em que nada está feito e os estados finais são os estados associados ao último dia em que todas as actividades estão realizadas.

A solução é um caminho na árvore de estados, do estado inicial até um estado final, que conduza a um custo global, dispendido em salários para realizar todas as actividades, que seja mínimo.

Um problema deste tipo gera uma explosão combinatória, dada pela conjugação das:

- Flexibilidades temporais na realização das tarefas das actividades

- Modularidades das equipas de realização das tarefas

A procura de uma solução para este problema discreto de optimização no âmbito da Investigação Operacional, terá como alternativas possíveis de pesquisa:

- Algoritmos gulosos

- Algoritmos de busca em profundidade

- Algoritmos de busca em largura

- Algoritmos de Partição e Avaliação, "Branch and Bound"

- Algoritmos de programação dinâmica

Uma outra abordagem para obter uma solução para o problema consiste em importar técnicas da área da Inteligência Artificial, através de algoritmos de pesquisa orientados por heurísticas, nomeadamente:

- Algoritmos "Best-First"

e as suas particularizações

- Algoritmo Z^*

- Algoritmo A^*

Nestes algoritmos a estratégia ou política para determinar a ordem pela qual os nodos são gerados é orientada por heurísticas, chamando-se por vezes a estes algoritmos, algoritmos informados. Este nome de algoritmos informados advem do facto da busca no espaço de estados ser orientada, ou informada por heurísticas, ou seja têm uma estratégia informada ou orientada.

As heurísticas são simplificações do modelo de um problema. A primeira exigência que se faz a uma heurística é que seja mais simples de resolver que o problema original. Uma das maneiras mais usuais de obter heurísticas é pela eliminação de restrições do problema original dando origem a modelos relaxados.

A metodologia escolhida para resolução do problema de calendarização foi um algoritmo de pesquisa informado, um algoritmo de pesquisa A^*, com uma heurística obtida pela técnica de relaxação. Como ficará demonstrado no Capítulo 10,

a heurística adoptada está nas condições de admissibilidade, pelo que a solução encontrada pelo algoritmo A^* é garantidamente a solução óptima.

Com esta metodologia há uma poupança na busca de estados, pois só os mais concorrenciais são investigados. O algoritmo informado de pesquisa A^* é uma programação concorrencial, em que a cada estado investigado está associado o somatório dos custos de transição do estado inicial até ao referido estado, e a heurística do custo de transição do referido estado até ao estado final. A pesquisa é sempre continuada através do estado mais concorrencial, havendo concorrência entre estados. A poupança que se consegue está relacionada com a qualidade de heurística adoptada.

Este capítulo vai ter a seguinte estrutura:

2. 2 - ALGORITMOS DE PESQUISA ORIENTADOS POR HEURÍSTICAS

2. 3 - ALGORITMOS DE PESQUISA NÃO ORIENTADOS POR HEURÍSTICAS

2. 4 - DISCUSSÃO DA ESCOLHA DA METODOLOGIA ADOPTADA

Em 2.2 vão-se referir os algoritmos orientados por heurísticas referenciados na área de Inteligência Artificial, para a resolução de problemas discretos de optimização. No ponto 2.2.2 faz-se uma rápida abordagem dos métodos usuais de obtenção de heurísticas, e em 2.2.3 uma sensibilização ao uso de heurísticas na resolução de problemas.

Em 2.3 vão-se referir os algoritmos tradicionalmente referenciados na área de Investigação Operacional para a resolução do problema discreto de optimização que constitui o problema de calendarização especificado no Capítulo 1.

Em 2.4 faz-se uma discussão acerca da escolha da metodologia adoptada.

2. 2 - ALGORITMOS DE PESQUISA ORIENTADOS POR HEURÍSTICAS

2.2.1 - Introdução

A pesquisa da solução de um problema discreto de optimização gera um espaço de estados candidatos no qual a solução vai ser procurada.

Um espaço de estados é um grafo orientado em que os nodos são estados do problema e os arcos representam os movimentos possíveis entre os estados.

Um algoritmo de pesquisa é uma estratégia para determinar a ordem pela qual os nodos são gerados, sendo que num algoritmo orientado por heurísticas se tem também informação acerca do tipo do problema e a natureza do objectivo a atingir, para ajudar a guiar a pesquisa nas direcções mais promissoras.

Como o problema é procurar um estado com determinadas características num espaço de estados, temos de dispor para a sua resolução por computador da:

- **Estrutura de estado** - que possa representar todos os estados candidatos do espaço de estados

- **Regras de produção** - regras capazes de transformar o código da estrutura de um estado na estrutura de outro estado

- **Estratégia de controlo** - um processo sistemático de utilização das regras de produção que garanta a geração de todos os estados possíveis e que não gere o estado mais de uma vez de modo a não ter repetição de computações. A primeira condição garante que não se perde a oportunidade de gerar um estado que seja solução se ele existir.

O espaço de estados é um grafo orientado, em que os nodos correspondem aos estados representados pela estrutura do estado, e os arcos estão associados a regras de produção.

As regras de produção e estratégia de controlo produzem a expansão dos nodos. A expansão de um nodo é a geração de todos os nodos sucessores do nodo que se chama antecessor, dizendo-se o nodo antecessor expandido.

Com uma estrutura de estado, regras de produção e estratégia de controlo, o que se consegue é evitar a memorização da representação explícita de todo o espaço de estados associado ao problema a resolver. Com estas ferramentas consegue-se ir gerando incrementalmente a partir do estado inicial, uma porção do espaço de estados deixando grande parte dele não explorado. O objectivo é explicitar o menos possível do espaço de estados total implícito. O conjunto dos nodos de um grafo que está a ser pesquisado é constituido por nodos que já foram expandidos que são chamados "fechados" e nodos que foram gerados e estão a aguardar expansão, que são chamados "abertos".

Nos algoritmos de pesquisa orientados por heurísticas, o conhecimento traduzido pelas heurísticas vai ser usado para auxiliar a busca. A informação das heurísticas vai ser usada na decisão de qual dos nodos vai ser expandido, que será o melhor de entre todos os nodos encontrados na análise desenvolvida até esse ponto.

Os métodos usuais de obtenção de heurísticas, vão ser referidos em 2.2.2. Em 2.2.3 faz-se uma sensibilização ao uso

de heurísticas através de exemplos simples de aplicação de heurísticas na orientação de algoritmos de pesquisa. Em 2.2.4 apresenta-se o algoritmo de pesquisa orientado por heurísticas "Best-First", e em 2.2.5 e 2.2.6 as particularizações do algoritmo "Best-First", os algoritmos Z^* e A^*.

2.2.2 - Métodos Usuais de Obtenção de Heurísticas

As heurísticas são critérios, métodos para decidir, entre muitas alternativas, a que permite ser a melhor para nos conduzir à solução de um problema. O compromisso que existe sempre na escolha de uma heurística é entre o seu grau de complexidade e a sua capacidade para discriminar correctamente entre boas e más alternativas.

No comportamento humano do dia a dia, a maioria das acções são guiadas pela chamada intuicão que cabe no conceito de heurística. É com base nessa intuição que em cada momento podemos fazer a melhor escolha, ou pelo menos na grande maioria das vezes uma boa escolha, que conduzimos carros, escrevemos programas, gerimos o comportamento social, e jogamos jogos em que a cada etapa temos de decidir qual a melhor decisão para obtermos um determinado resultado.

Esta intuição não garante sempre a melhor escolha, mas dá bons resultados a grande maioria das vezes. O método de obtenção destas heurísticas, associadas ao dia a dia, é muitas vezes difuso, pouco claro, sendo as pessoas incapazes, na maior parte dos casos, de os descrever com precisão.

Normalmente o processo mental que leva à implementação de heurísticas é a simplificação do modelo do problema original. É evidente que a heurística associada a um problema tem de ser mais simples de resolver que o problema original. Se assim não fosse nada se ganharia com a heurística que em vez de ajudar na pesquisa da solução do problema original, seria ela própria mais difícil de resolver que o problema original.

Assim, as heurísticas são obtidas por simplificação do modelo original do problema para o qual se procura uma solução.

Um método usual de obter heurísticas por simplificação do modelo original é removendo restrições do problema original obtendo modelos relaxados. Mas, pode haver situações em que a heurística seja originada pelo processo contrário, ou seja, impondo restrições adicionais, carregando o modelo original com mais restrições de modo a obter-se um modelo simplificado.

Uma outra maneira usual de obter modelos simplificados é através de considerações probabilísticas. Em certos casos pode-se possuir suficiente conhecimento acerca do problema que permita usar modelos probabilísticos que ajudem a guiar a pesquisa funcionando com heurística.

Resumindo, os métodos mais usuais de obter simplificações do modelo original do problema para funcionarem como heurísticas, que ajudarão a discriminar correctamente entre boas e más direcções de pesquisa são:

- Remover restrições associadas ao modelo original originando modelos relaxados.

- Impôr restrições adicionais ao problema original, gerando modelos simplificados do mesmo.

- Usar conhecimento probabilístico acerca do modelo original do problema originando modelos probabilísticos que funcionem como heurísticas.

Mas qualquer que seja o método utilizado para a implementação da heurística, em todas elas o compromisso a exigir é que o grau de simplificação do modelo original seja grande e que ao mesmo tempo reproduzam o melhor possível o modelo original no espaço de estados.

2.2.3 - Exemplos de Aplicação de Heurísticas na Orientação de Algoritmos de Pesquisa

Vai-se fazer a sensibilização para a necessidade do uso de heurísticas na orientação da pesquisa da solução de um problema, através de dois exemplos simples, e em que as heurísticas descobertas são intuitivas.

A maior parte dos problemas complexos requer a avaliação de um enorme número de possibilidades para determinar a solução óptima. O tempo requerido para encontrar essa solução é muitas das vezes proibitivo, se não de todo impossível. As heurísticas têm um papel importante neste tipo de problemas, indicando o caminho de modo a reduzir o número de avaliações e obter soluções dentro de tempos razoáveis.

Vamo-nos debruçar sobre 2 exemplos elucidativos do que atrás se disse.

Exemplo 1 - 8-puzzle

O objectivo do 8-puzzle é rearranjar uma configuração inicial de um tabuleiro 3x3, numa configuração final.

A configuração final é:

1	2	3
8	■	4
7	6	5

Fig. 2.1 - Configuração final do problema 8-puzzle

O rearranjo é feito podendo-se mudar uma só peça ortogonalmente para o quadrado vazio.

Se a configuração inicial for,

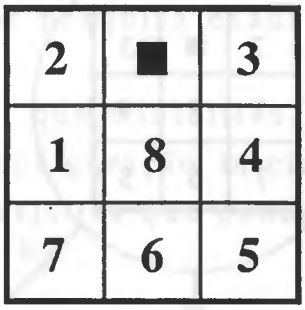


Fig. 2.2 - Configuração inicial do problema 8-puzzle

as alternativas que temos para a configuração seguinte são:

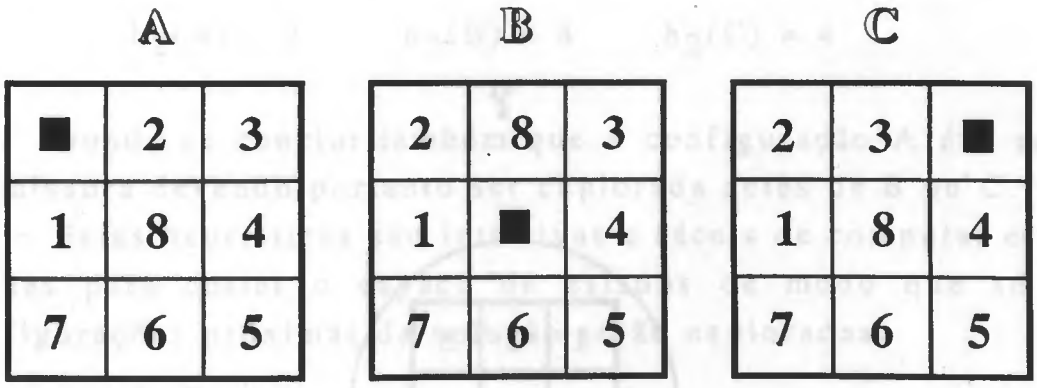


Fig. 2.3 - Configurações geradas pela configuração inicial do problema 8-puzzle

Qual das 3 alternativas A, B ou C é a mais promissora? A resposta pode ser obtida por uma pesquisa exaustiva das

movimentações seguintes do puzzle para descobrir qual dos 3 estados tem um caminho mais curto para a configuração final. Assim, tendo-se o estado inicial, que tem como sucessores os estados A, B e C, pretende-se chegar ao estado final.

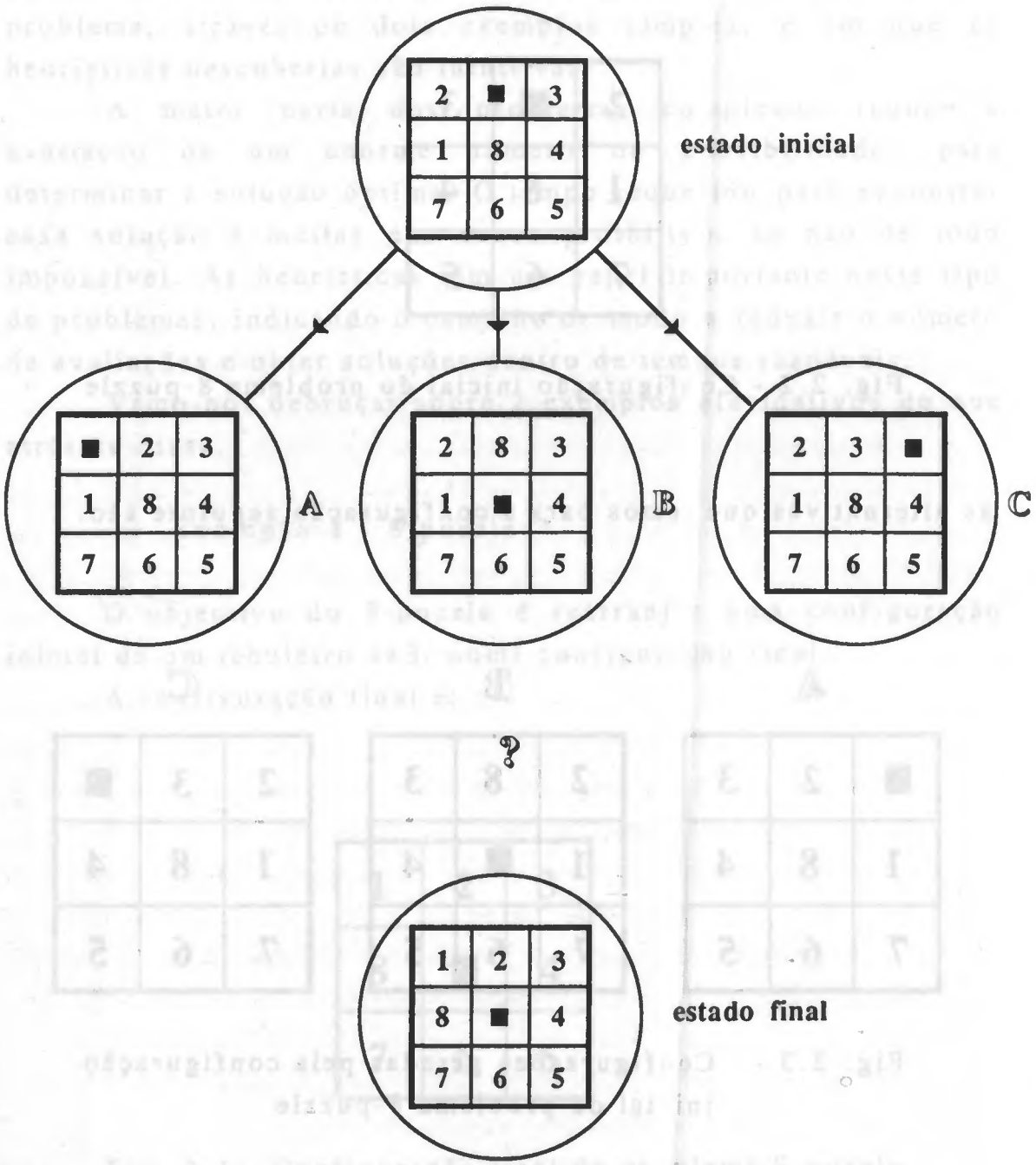


Fig. 2.4 - Problema 8-puzzle

Esta pesquisa exaustiva dá uma explosão combinatória, tanto maior quanto o estado inicial estiver afastado da configuração final e para tabuleiros maiores (por exemplo 4x4) começa a ser impraticável porque o número de estados a ser examinado começa a ser incomportável.

A decisão para seleccionar a alternativa mais promissora não pode ser feita por pesquisa exaustiva, mas sim baseada em heurísticas.

Uma heurística, que é intuitiva, consiste na estimativa da "distância" entre a configuração inicial e a configuração final. Por exemplo, uma heurística que poderíamos adoptar é o número de posições trocadas, h_1:

$$h_1(A) = 2 \qquad h_1(B) = 3 \qquad h_1(C) = 4$$

Donde se conclui que a configuração A é a mais promissora.

Outra heurística possível seria a soma das distâncias vertical e horizontal das posições trocadas, a chamada distância Manhattan, h_2

$$h_2(A) = 2 \qquad h_2(B) = 4 \qquad h_2(C) = 4$$

Donde se conclui também que a configuração A é a mais promissora devendo portanto ser explorada antes de B ou C.

Estas heurísticas são intuitivas e fáceis de computar e são usadas para cortar o espaço de estados de modo que só as configurações próximas da solução serão exploradas.

Exemplo 2 - Mapa de estradas

Consideremos o mapa de estradas

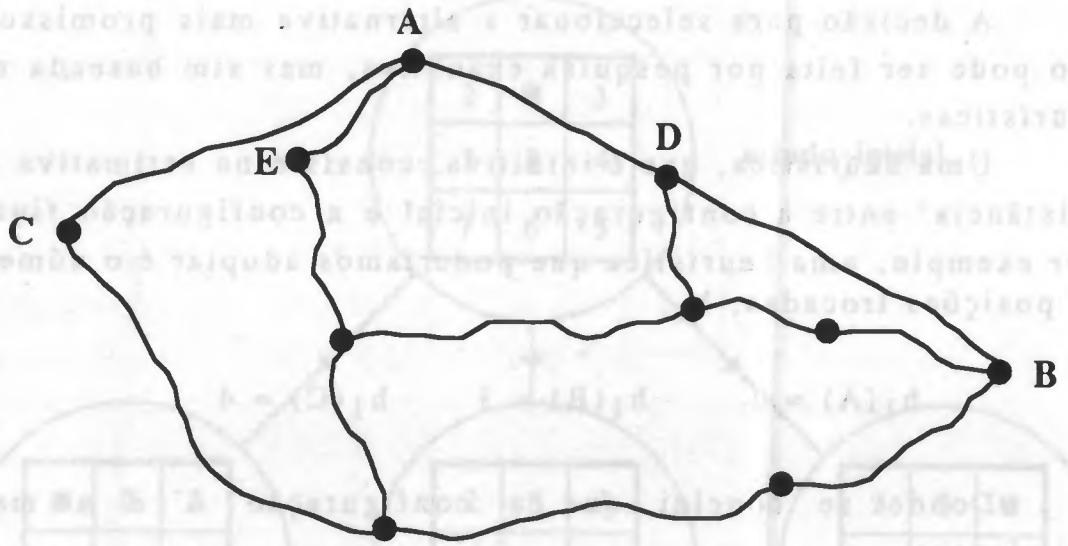


Fig. 2.5 - Mapa de estradas

e que pretendemos o caminho mais curto para ir da cidade A para a cidade B. O mapa indica o comprimento de cada troço de estrada.

Olhando para o mapa de imediato eliminamos as alternativas que nos afastem da direcção da A para B. Assim, a cidade D parece-nos mais promissora que as cidades E e C pois estas não se encontram na direcção geral de A para B.

Assim, os caminhos que passem por D serão explorados primeiro que os caminhos passando por E e C, isto apesar da distância entre A e E ser menor que a distância entre A e D. A informação extra que se está a utilizar é uma estimativa de distância euclidiana. É esta informação extra que constitui uma heurística possível para este caso, sendo a pesquisa baseada na distância real dada pelo mapa das estradas e pela heurística

referida. Assim as alternativas

A - D
A - E
e A - C

seriam pesadas por

d (A, D) + h (D)
d (A, E) + h (E)
d (A, C) + h (C)

sendo d (I, J) a distância real dada pelo mapa de estradas entre a cidade I e a cidade J e h (J) a estimativa da distância directa entre J e B.

A cidade que tivesse menor soma seria a escolhida para ser explorada antes das outras.

Nos dois exemplos, as heurísticas servem para indicar quais são, dos possíveis caminhos de exploração, os melhores para chegar a uma solução. As heurísticas são simples de calcular relativamente à complexidade de encontrar a solução e embora não dêem necessariamente a direcção correcta, dão-na a maior parte das vezes. E mesmo que eventualmente indiquem uma direcção de pesquisa que não é a melhor, pode-se sempre recomeçar a pesquisa de outro ponto. O cerne da questão é que as heurísticas poupam tempo de pesquisa, evitando muitas pesquisas fúteis.

Qual foi o processo mental pelo qual as heurísticas foram descobertas? Tanto num caso como no outro, as heurísticas foram descobertas através da simplificação do modelo dos problemas.

Nos dois exemplos foram geradas, removendo restrições que proibiam certos movimentos nos problemas originais. Foram obtidas assim através de modelos relaxados.

No caso do 8-puzzle, a heurística foi implementada transgredindo certos movimentos proibidos no problema original. A heurística adoptada para o problema de calendarização vai usar um processo análogo ao permitir que algumas tarefas do problema transgridam as relações de precedência. No caso do mapa de estradas à heurística associa-se um conceito matemático, a distância euclidiana, assim como no problema de calendarização se vai associar à heurística um conceito matemático, a programação linear associada à heurística. Esta similitude aparecerá evidente no Capítulo 9 em que se explica detalhadamente a conceptualização da heurística adoptada para o problema de calendarização.

Estes 2 exemplos simples demonstram claramente o papel reservado às heurísticas na orientação da direcção de pesquisa no espaço de estados associados a um problema de optimização.

2.2.4 - Algoritmo "Best-First"

Dentro dos algoritmos de pesquisa orientada por heurística vai-se apresentar o algoritmo "Best-First", e em 2.3.5 e 2.3.6 as suas particularizações, os algoritmos Z^* e A^*.

Neste algoritmo, a quantificação de quanto um nodo n é promissor para ser expandido, é estimada numericamente por uma função de avaliação heurística f(n), que pode depender da descrição de n, da descrição do objectivo, da informação já conseguida pela busca até esse ponto e por qualquer outra informação extra acerca do domínio do problema.

O tipo desta função de avaliação heurística pode diferenciar o algoritmo de busca "Best-First" dando origem a algoritmos particularizados.

Apresenta-se seguidamente um esqueleto possível do algoritmo de busca "Best-First":

1- Pôr o nodo origem s em "ABERTOS".

2- Se "ABERTOS" está vazia, sair sem sucesso, não existe solução.

3- Remover de "ABERTOS" o nodo n que tenha uma função de avaliação heurística f mínima, e ponha-o em "FECHADOS".

4- Expandir o nodo n, gerando todos os sucessores de n (usando um sistema de pointers para n).

5- Se algum dos sucessores de n é o objectivo, a solução, sair com a solução obtida e indicar o "trace" de solução, o caminho do objectivo até s.

6- Para cada sucessor n' de n

   a) Calcular f(n')
   b) Assignar f(n') ao nodo n' e acrescentá-lo a "ABERTOS"

7- Voltar a 2)

"ABERTOS" - lista dos nodos gerados mas não expandidos.
"FECHADOS" - lista dos nodos gerados e já expandidos.

O processo de busca do algoritmo "Best-First" gera uma árvore explícita T, a árvore de pesquisa que é uma sub-árvore do espaço de estados implícito G.

Vamo-nos agora debruçar sobre a maneira como se chega à função de avaliação f, donde vem a informação necessária para se ter a melhor escolha e como ela se propaga através do grafo.

Funções de peso recursivas:

Para um dado grafo G, designa-se o seu peso por W_G, onde W_G é a medida de optimização escolhida, representando ou um mérito Q ou um custo C.

Se removermos de G todos os nodos excepto os descendentes de um nodo n, a porção remanescente de G é um grafo solução para n, e o seu peso é representado por $W_G(n)$. Em geral o peso de qualquer grafo solução pode ser uma função complexa dos pesos dos nodos, peso dos arcos e peso dos nodos terminais.

Vamos definir uma classe de funções de peso recursivas,

$$W_G(n) = F[E(n); W_G(n_1), W_G(n_2), \ldots W_G(n_b)]$$

onde $n_1, n_2, \ldots n_b$ são os sucessores imediatos de n

E(n) - função de propriedades caracterizando o nodo n

F - é uma função arbitrária combinando todas as variáveis

Vamos agora ver como a função de peso recursiva facilita a ordenação dos grafos solução candidatos. O peso significa ou um mérito ou um custo, sendo que se pretende maximizar o mérito e minimizar o custo, donde se pode trabalhar com o custo como negativo do mérito.

Se todo o espaço de estados tivesse sido explorado, um grafo solução óptimo podia ser construido e o seu mérito $Q^*(s)$ podia ser computado tomando o máximo de $Q_G(s)$ de todos os

grafos solução com raiz em s. O procedimento recursivo de maximização é o seguinte:

1- Se n é um nodo terminal, $Q^*(n) = v(n)$ onde v(n) é o peso do nó terminal.

2- Se n é um nodo não solução e sem sucessores, representa uma situação insolúvel e $Q^*(n) = -\infty$

3- Se n é um nodo com sucessores $n_1, n_2, \ldots n_b$ então

$$Q^*(n) = \max_i F[E(n); Q^*(n_i)]$$

De acordo com esta definição, $Q^*(n)$ é finito se e só o problema associado ao nodo n fôr solúvel. Para cada nodo solúvel n, $Q^*(n)$ dá o mérito do grafo solução óptimo com raiz em n. Para o nodo raiz s, $Q^*(s)$ é o mérito da solução óptima do problema. Associado a $Q^*(s)$ temos o grafo solução óptimo G^* que é definido por

1- O nodo raiz s está contido em G^*

2- se um nodo n está em G^*, um dos seus sucessores n' está em G^*, tal que

$$F[E(n); Q^*(n')] = \max_i F[E(n); Q^*(n_i)]$$

Mas o problema que temos é que se o espaço de estados está apenas explorado parcialmente, não temos maneira de computar a solução óptima, nem identificar qual dos nodos da fronteira de pesquisa vai dar uma solução óptima. Mas, se a cada nodo n na fronteira de pesquisa se assignar uma estimativa h(n) de $Q^*(n)$, pode-se computar o que parece ser a potencial

solução mais promissora. Esta solução é computada em 2 passos:

1- Assignar a cada nodo na fronteira de pesquisa uma função h(n) que é uma estimativa do mérito da melhor solução com raiz em n.

2- Usar a função F e o procedimento recursivo de maximização como se a estimativa h(n) fosse o verdadeiro peso do nó terminal.

Define-se uma função de avaliação e(n) que dá uma estimativa de $Q^*(n)$ para todos os nodos do grafo explorado.

$$e(n) = \begin{cases} h(n) & \text{se n está aberto} \\ \max_i F[E(n); e(n_i)] & \text{se n está fechado} \end{cases}$$

O algoritmo "Best-First" vai usar esta função de avaliação para computar a função de selecção dos nodos f(n). Cada nodo n em "ABERTOS", define na árvore de pesquisa T um caminho P(n) de s a n. Para decidir no passo 3 qual dos nodos merece expansão, a função de avaliação $e_{P(n)}(s)$ deve ser assignada a cada nodo em "ABERTOS", onde $e_{P(n)}(s)$ é computada ao longo do caminho de n para s. Então a função f(n) computada no passo 6 é apenas

$$f(n) = -e_{P(n)}(s)$$

(f representa custos e portanto é para ser minimizada, enquanto e representa mérito e é maximizada).

2.2.5 - Algoritmo Z^*

Se o algoritmo de pesquisa "Best-First" tem uma função de selecção f(n) recursiva do tipo referido atrás

$$f(n) = -e_{P(n)}(s)$$

com

$$e(n) = \begin{cases} h(n) & \text{se n está aberto} \\ \max_i F[E(n); e(n_i)] & \text{se n está fechado} \end{cases}$$

o algoritmo "Best-First" chama-se Z^*.

O algoritmo Z^* é assim um caso particular do algoritmo "Best-First".

2.2.6 - Algoritmo A^*

O algoritmo A^* é uma especialização de Z^*.

No algoritmo A^* não é preciso percorrer todo o caminho de n para s para se computar $e_{P(n)}(s)$. Um parâmetro armazenado no predecessor de n permite calcular $e_{P(n)}(s)$ localmente e transmitir $e_{P(n)}(s)$ de predecessor para sucessor em cada expansão. Sendo o custo aditivo o parâmetro a transmitir é g(n) que é o custo total do caminho de s para n, sendo a heurística de A^*

$$f(n) = -e_{P(n)}(s) = g(n)+h(n)$$

para cada nodo aberto.

Apresenta-se seguidamente um esqueleto possível do algoritmo A^*:

1- Pôr o nodo origem s em "ABERTOS".

2- Se "ABERTOS" está vazia sair sem sucesso.

3- Remover de "ABERTOS" o nodo n com f mínima e pô-lo em "FECHADOS".

4- Se n é o objectivo, a solução, sair com a solução obtida e indicar o "trace" da solução, o caminho do objectivo até s.

5- Expandir n, gerando todos os seus sucessores (usando um sistema de pointers para n).

6- Para cada sucessor n' de n estimar h(n') (uma estimativa do custo do melhor caminho de n' para uma solução) e calcular

$$f(n') = g(n')+h(n')$$

onde $g(n') = g(n)+c(n,n')$

com $g(s) = 0$ e onde

$c(n,n')$ é o custo de transição de n para n'

e pô-los em "ABERTOS".

7- Voltar a 2.

O algoritmo Z^* tem essencialmente um esqueleto do tipo do referido para A^*, diferindo apenas no passo 6 em que pode usar uma função de avaliação recursiva arbitrária.

2.3 - ALGORITMOS DE PESQUISA NÃO ORIENTADOS POR HEURÍSTICAS

2.3.1 - Introdução

Vão-se neste item apresentar os algoritmos de pesquisa que poderiam ser usados numa tentativa de resolver o problema discreto de optimização, que é o problema de calendarização, no âmbito da Investigação Operacional.

Os algoritmos que vão ser referidos estão tradicionalmente ligados à área de Investigação Operacional. Vão-se referir os algoritmos tipo guloso, de pesquisa em profundidade e em largura, de avaliação e partição, e programação dinâmica. Esta catalogação dos algoritmos é algo discutível, pois por exemplo, não é perfeitamente definitivo que os algoritmos de avaliação e partição não sejam algoritmos orientados, em que as indicações de avaliação funcionam como heurísticas. Com a apresentação sumária dos algoritmos referidos neste item fica completa a gama de algoritmos que se apresentavam para escolha do algoritmo a utilizar na busca da solução para o problema de calendarização. Em 2.4 faz-se a discussão dessa escolha.

2.3.2 - Algoritmos Gulosos

Estes algoritmos têm a sua estratégia baseada em optimizações locais, feitas através de passos incrementais numa direcção que com base na informação local disponível parece ser a mais promissora.

Em termos do nosso modelo de espaço de estados, os métodos gulosos consistem em expandir um nodo, inspeccionar os seus sucessores e escolher e expandir o melhor de todos eles, sem reter informação acerca do predecessor nem dos outros nodos gerados.

Com estes métodos corre-se o risco de não se fazer uma busca sistemática, ou seja, não se garante que se vá gerar um nodo que é a solução óptima. Assim que um óptimo local é encontrado não é possível fazer mais melhoramentos e o processo termina sem se chegar à solução óptima. Outro risco que se corre é o processo entrar na exploração de caminhos infinitos sem encontrar nenhuma solução.

Em termos computacionais estes métodos são atractivos, porque não requerem a memorização do caminho que levou a uma determinada situação nem a memorização de outras alternativas.

2.3.3 - Algoritmos de Busca em Profundidade

Os dois algoritmos que vamos tratar, em seguida o algoritmo de busca em profundidade e o algoritmo de busca em largura, diferem dos algoritmos gulosos, na medida em que memorizam a informação das outras alternativas que não foram expandidas. Se um caminho de pesquisa é escolhido para exploração, os outros candidatos alternativos não são descarregados, mas mantidos em reserva, para o caso do caminho escolhido não produzir uma solução.

Os algoritmos de busca em profundidade e em largura são algoritmos não orientados, pois a progressão da procura não depende da natureza da solução que se procura, ou seja, a localização do ou dos nodos solução não influencia a ordem de expansão dos nodos.

Na procura em profundidade a prioridade é dada aos nodos do nível mais profundo do grafo de busca, e cada nodo escolhido para expansão tem todos os seus sucessores gerados antes de qualquer outro nodo ser explorado.

Depois da expansão, um dos novos nodos gerados é de novo selecionado e expandido e a busca continua até se encontrar uma solução ou até a progressão ficar bloqueada. Se ocorre um bloqueamento, o processo retoma a partir do nodo mais profundo deixado não expandido.

Este algoritmo pode ser perigoso quando implementado em grandes grafos, especialmente se forem de profundidade infinita. Pode cair num caminho sem solução e ir pesquisando cada vez mais fundo sem possibilidade de recuperar da situação sem solução. Uma maneira de obviar a questão apontada é impôr uma profundidade de pesquisa como medida de paragem. Quando essa profundidade é atingida esse caminho de exploração é abandonado.

Exemplo: vamos considerar o seguinte espaço de estados, com estado inicial a, e a seguinte sequência de passos na pesquisa em profundidade

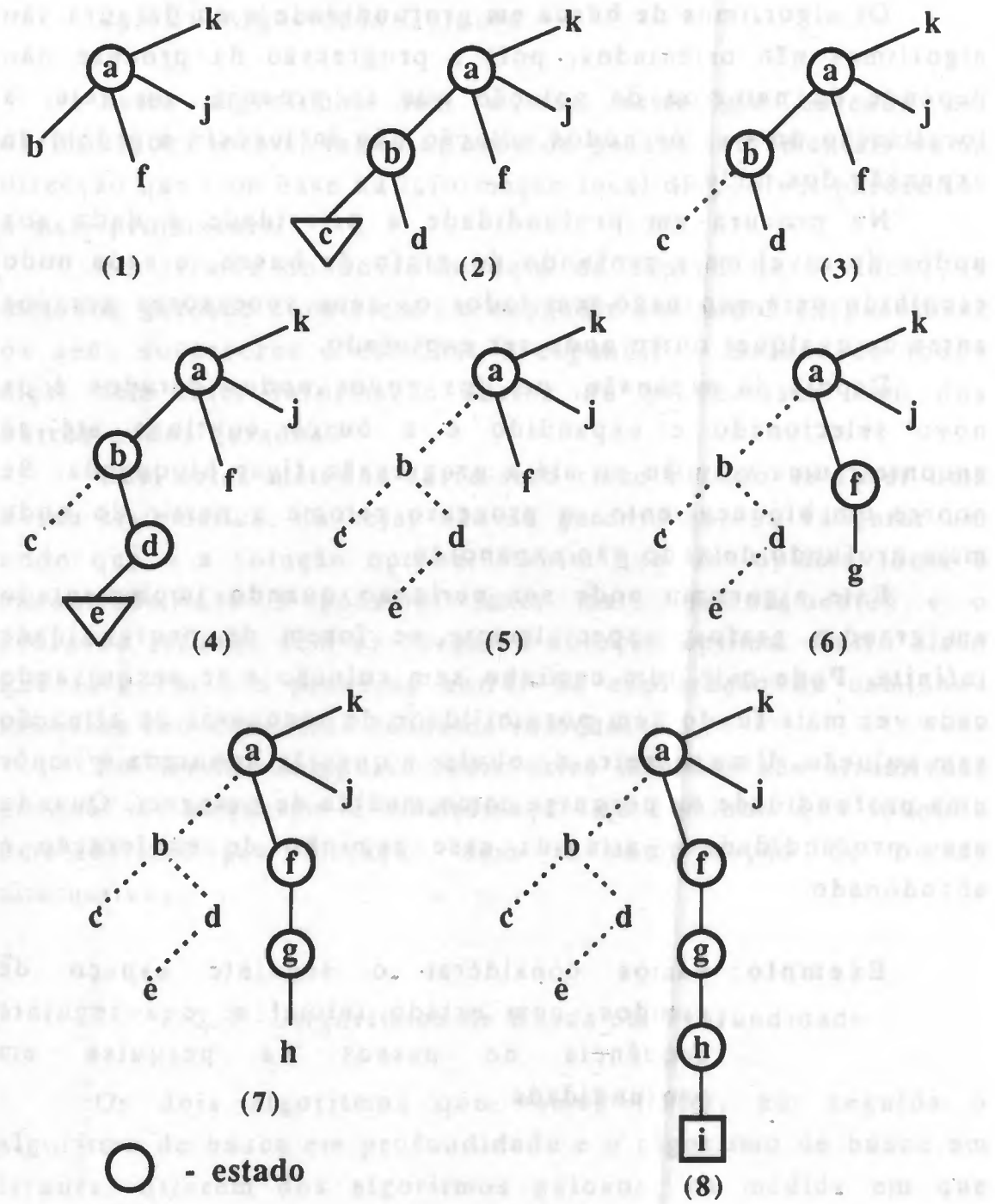


Fig. 2.6 - Pesquisa em profundidade

O traço a cheio indica o que está memorizado sendo o tracejado indicação do que foi apagada da memória.

O caminho de pesquisa no exemplo apresentado está esquematicamente representado na Fig. 2.7

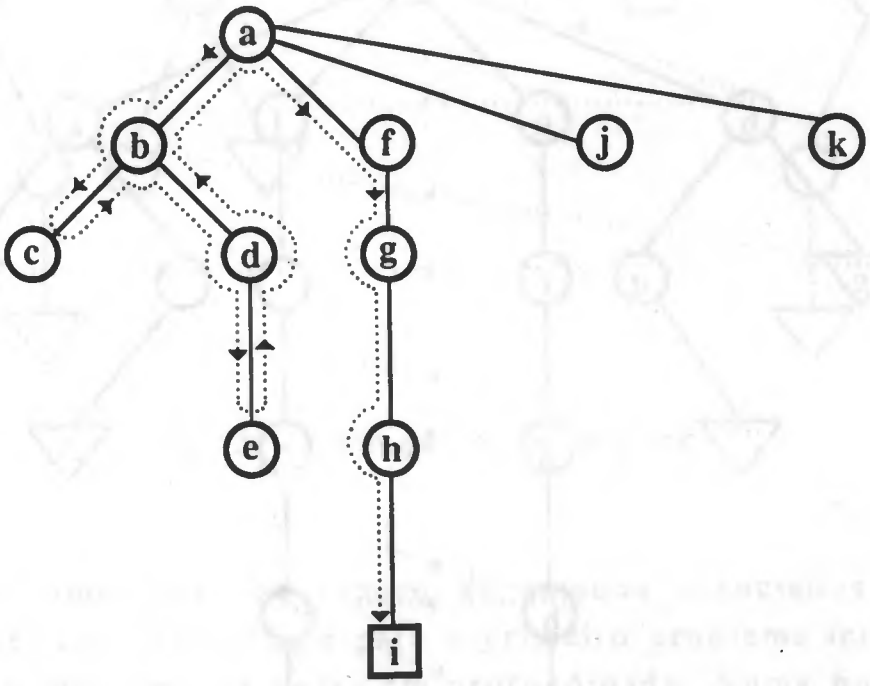


Fig. 2.7 - Sequência de pesquisa em profundidade

2.3.4 - Algoritmo de Busca em Largura

As estratégias de busca em largura, em oposição à busca em profundidade, dão prioridade aos nodos das camadas mais superficiais, explorando progressivamente secções de igual profundidade do grafo, como está ilustrado nas Fig. 2.8 e Fig. 2.9.

As Fig. 2.8 e Fig. 2.9 indicam a sequência de pesquisa em largura em 2 grafos representando o espaço de estados associados a 2 problemas cuja solução se procura.

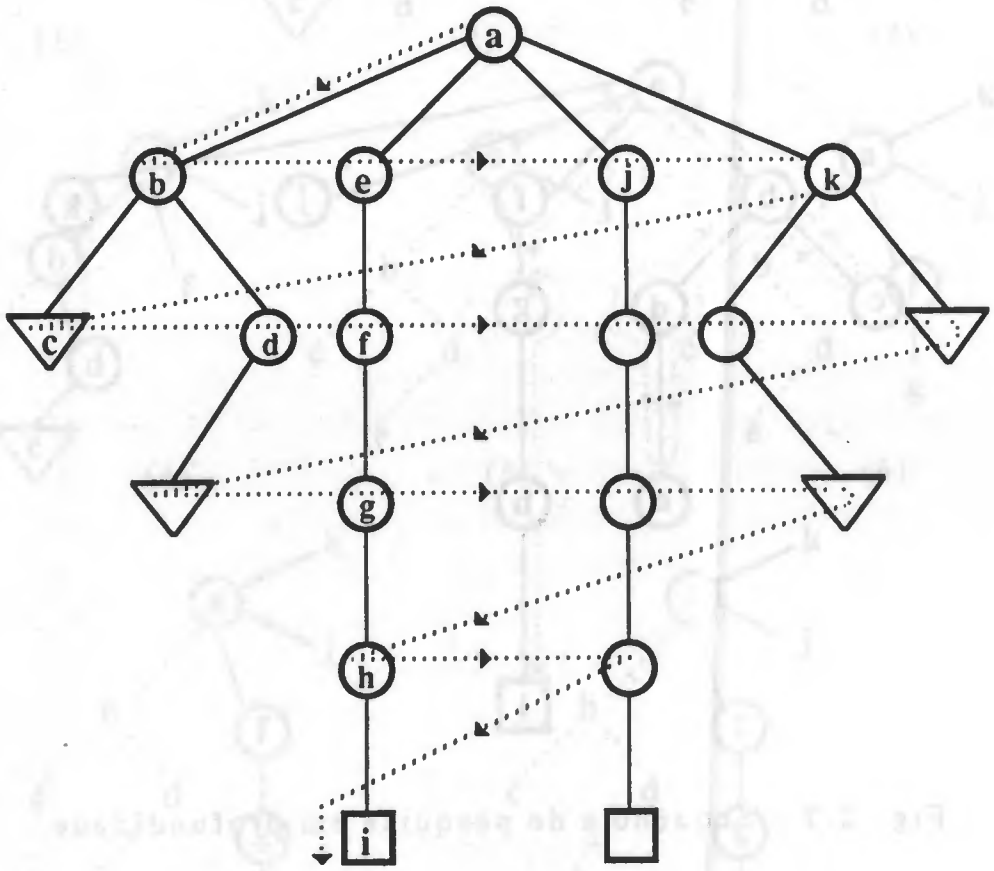


Fig. 2.8 - Pesquisa em largura

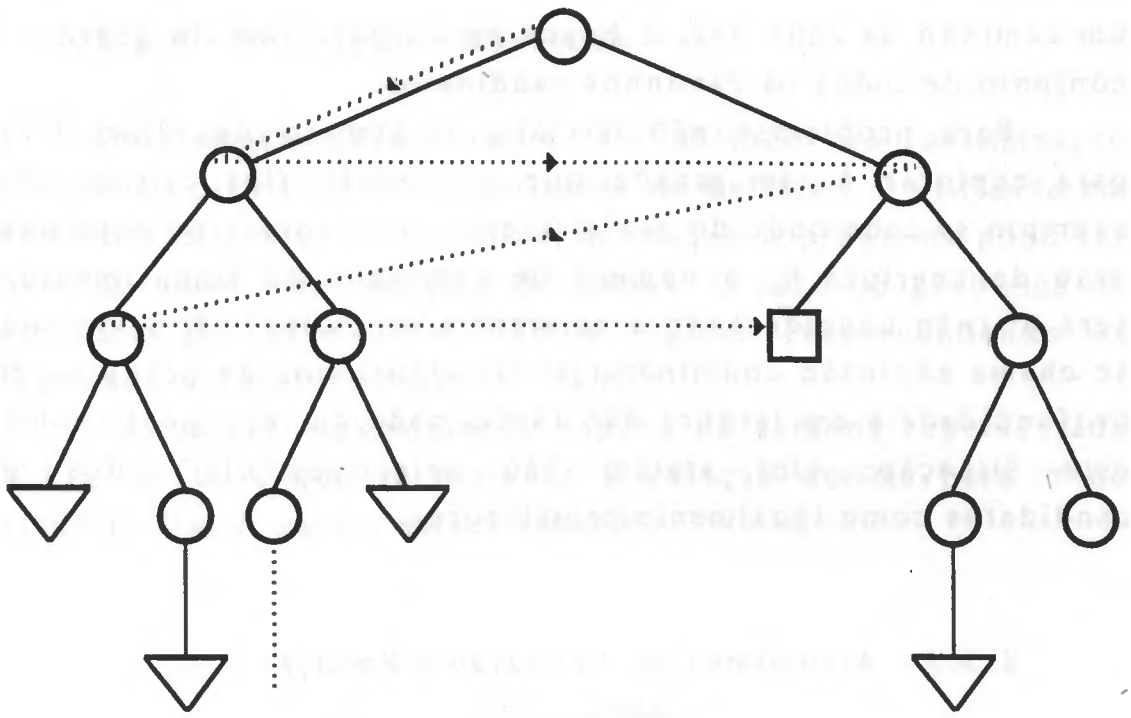


Fig. 2.9 - Pesquisa em largura

Da comparação do espaço de estados associados aos 2 problemas, concluiu-se que para o primeiro problema teria sido mais vantajoso uma pesquisa em profundidade. Numa busca em profundidade teriam sido investigados 9 estados até se encontrar uma solução, enquanto que na busca em largura se investigaram 17 estados até encontrar a solução.

O contrário se passa com o segundo problema em que uma pesquisa em profundidade investigava 11 estados até ter a solução, enquanto que a busca em largura seria mais vantajosa pois encontraria a solução do problema depois de investigar 5 estados.

Comparando ainda a busca em profundidade com a busca em largura, esta tem a vantagem de garantir a solução, se ela existir, num grafo localmente finito. Mas em termos de espaço de memória, enquanto a busca em profundidade guarda apenas

um caminho de cada vez, a busca em largura tem de guardar o conjunto de todos os caminhos candidatos.

Para problemas não triviais, o número de alternativas para explorar é tão grande que a questão fica crítica. Por exemplo se cada nodo do grafo tiver b sucessores, ou seja, tiver grau de abertura b, o número de caminhos de comprimento l será b^l (não considerando a existência de ciclos). É a isto que se chama explosão combinatória. Os algoritmos de pesquisa em profundidade e em largura não fazem nada de inteligente contra esta situação: eles tratam não selectivamente todos os candidatos como igualmente promissores.

2.3.5 - Algoritmos de Avaliação e Partição

Um problema discreto de optimização tem associado um espaço de estados. Nesse espaço de estados tem-se um conjunto discreto de soluções. A esse conjunto chamemos conjunto de soluções original.

Os algoritmos de avaliação e partição ("Branch and Bound"), decompõem, particionam, o conjunto de soluções original em conjuntos de soluções de tamanho decrescente. A decomposição de cada subconjunto de soluções S é continuada até que S tenha apenas um elemento (de que se pode calcular directamente o custo associado), ou até que exista um elemento melhor que não pertença a S. A avaliação de uma solução ou de um outro subconjunto de soluções que é melhor do que o conjunto S, leva a que todo o conjunto S seja eliminado, não se contando mais com S para futuras partições e avaliações.

Ou seja, os algoritmos de avaliação e partição usam uma relação de dominância para eliminar subconjuntos de soluções. Continuando o processo de "Branch and Bound" um elemento solução óptimo é eventualmente encontrado, explicitando-se normalmente apenas uma parte do conjunto original de soluções.

2.3.6 - Programação Dinâmica

Programação dinâmica é um método de optimização baseado no princípio de optimilidade de Bellman. É utilizado em problemas discretos de optimização em que o problema pode ser encarado como uma sucessão de etapas. O caso do problema de calendarização cabe dentro desta lógica, correspondendo as etapas aos sucessivos dias do calendário.

Vamos ver o exemplo do espaço de estados representado na Fig. 2.10, e que se pretende a solução óptima para ir do estado inicial A para o estado final H.

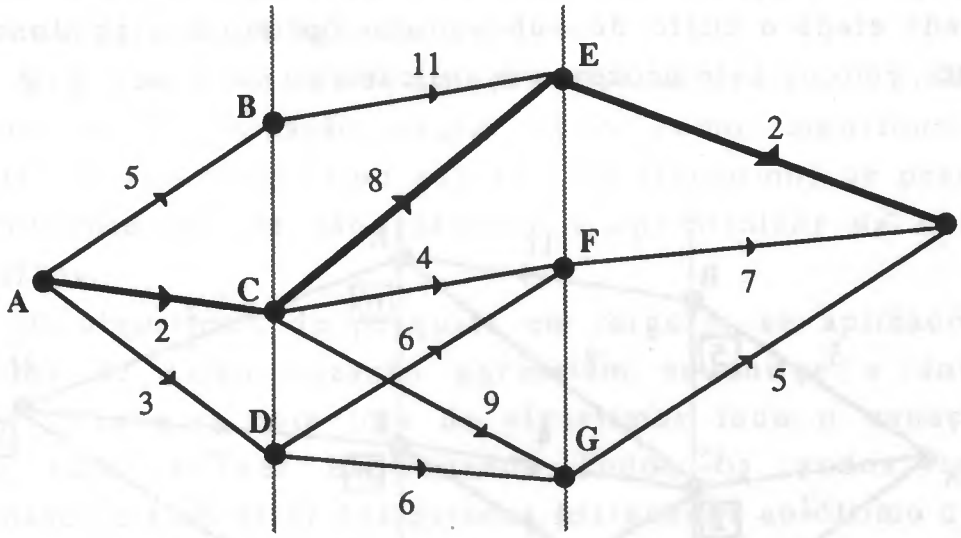


Fig. 2.10 - Problema a resolver por programação dinâmica

Por enumeração de todas as soluções possíveis chega-se à conclusão que a solução óptima é o caminho ACEH. Se analizarmos todos os subcaminhos de ACEH vê-se que todos eles são óptimos desde o ponto em que o subcaminho começa até ao ponto em que acaba. Por exemplo para ir de A a E o melhor caminho é ACE, ou seja o caminho óptimo contém apenas subcaminhos óptimos.

Isto constitui o princípio de optimilidade de Bellman: qualquer caminho óptimo tem de conter forçosamente apenas subcaminhos óptimos.

Com base neste princípio de optimilidade são construidos os algoritmos de programação dinâmica. Estes algoritmos utilizam a técnica de construir soluções óptimas para problemas pequenos e ir sucessivamente construindo soluções para problemas maiores.

Os algoritmos vão assignando aos estados de cada etapa o custo do subcaminho óptimo do estado inicial até aos estados da etapa. Vão aplicando sucessivamente uma relação de comparação para resolver problemas maiores partindo de problemas mais pequenos. Na Fig. 2.11 ilustra-se um processo em que se indica em cada etapa o custo do subcaminho óptimo assignado a cada estado, em que este processo é aplicado ao caso da Fig. 2.10.

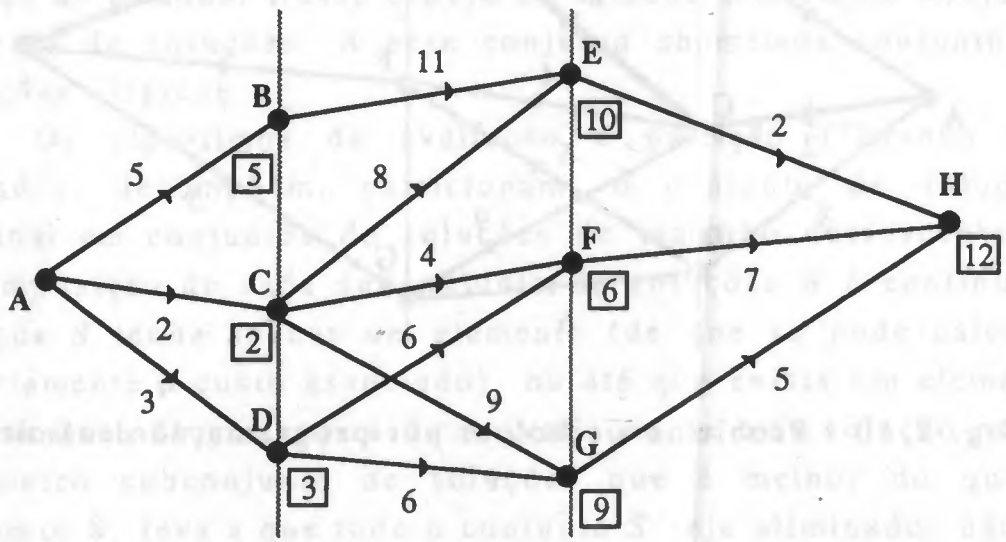


Fig. 2.11 - Aplicação da programação dinâmica ao grafo da Fig. 2.10

2.4 - DISCUSSÃO DA ESCOLHA DA METODOLOGIA ADOPTADA

Nos items anteriores descreveram-se diferentes tipos de algoritmos, tendo-se apresentado em 2.2 algoritmos orientados por heurísticas e usualmente referenciados na área de Inteligência Artificial, e em 2.3 algoritmos não orientados por heurísticas e tradicionalmente atribuídos à área de Investigação Operacional. Todos os tipos de algoritmos referidos se apresentavam como candidatos para resolução do problema discreto de optimização que constitui o sistema de calendarização especificado no Capítulo 1, para o qual se pretende uma solução óptima.

A procura de uma solução óptima para o problema discreto de optimização exclui logo como algoritmos de pesquisa os algoritmos tipo guloso e os algoritmos de pesquisa em profundidade que não garantem a optimilidade da solução encontrada.

Os algoritmos de pesquisa em largura, se aplicados ao problema de calendarização permitiam encontrar a solução óptima. Mas para este tipo de algoritmos todo o espaço de estado teria de ser explicitado. Todos os nodos seriam expandidos até ao nível dos estados assignados ao último dia do calendário, e todos os caminhos investigados.

Os algoritmos de programação dinâmica iam analisar o problema de calendarização por etapas, etapas que correspondem aos diferentes dias do calendário, e assim, iam resolvendo o problema sob o princípio de optimilidade de Bellman. A solução óptima seria encontrada, mas todos os nodos do espaço de estados seriam expandidos, sem no entanto se analisarem todos os caminhos que se poderiam explicitar do espaço de estados.

A escolha da metodologia para resolução do problema discreto de calendarização recaiu nos algoritmos informados por

heurísticas, o algoritmo A^*. Ao contrário por exemplo dos algoritmos de programação dinâmica, o algoritmo A^* contabiliza o que se passou do estado inicial até um determinado estado, mas também o que se infere que irá acontecer desse estado até ao estado final. O algoritmo A^* entra em consideração para determinar quanto um estado é concorrencial, com o passado do estado e também com uma previsão de futuro do estado dada pela heurística associada ao estado. Assim uma subsolução candidata é apreciada pelo custo de transição do estado inicial até ao último estado da subsolução, custo que é conhecido, e pelo custo desconhecido do último estado da subsolução até ao estado final. Esse custo é desconhecido, mas se houver uma indicação da sua avaliação, essa indicação vai ser levada em linha de conta. É esse o papel das heurísticas dos algoritmos de pesquisa por elas orientados. Assim ao contrário dos algoritmos de programação dinâmica, nos algoritmos de pesquisa orientados por heurísticas nem todos os nodos são expandidos. Apenas uma pequena parte dos nodos é expandida. A cada passo de pesquisa é expandido apenas o último nodo da subsolução mais concorrencial, sendo que as subsoluções menos concorrenciais nunca são expandidas.

Este mecanismo dos algoritmos de pesquisa informados por heurísticas engloba também as vantagens dos algoritmos de avaliação e partição. A concorrência que se establece nos algoritmos tipo "Best-First" faz com que algumas subsoluções nunca sejam consideradas, porque nunca se apresentam como concorrenciais para expansão. Isto engloba o conceito de eliminação de conjuntos de soluções dos algoritmos de avaliação e partição. Mas nos algoritmos tipo "Best-First" essa eliminação é feita de um modo sistemático, em que se entra com a avaliação de um custo conhecido, a transição do estado inicial até ao estado que se considere para expansão de uma subsolução, e o custo desconhecido do estado analisado para expansão até ao estado final, custo desconhecido mas do qual se tem uma aproximação dada pela heurística associada ao estado.

Além disso, no algoritmo escolhido como metodologia de resolução, o algoritmo A^*, se se garantir que a heurística usada é admissível, o algoritmo garante a solução óptima.

No Capítulo 9 descreve-se pormenorizadamente a conceptualização da heurística adoptada para o "sistema de calendarização" e no Capítulo 10 prova-se que a heurística adoptada é admissível garantindo-se assim que a solução obtida é óptima.

Pelas razões expostas, e conjugando-as com o facto do problema de calendarização gerar uma explosão combinatória no espaço de estados, é-se naturalmente encaminhado para uma solução de pesquisa orientada por heurísticas. Assim, fica justificada a escolha do algoritmo A^* como algoritmo de pesquisa da solução óptima do problema de calendarização que nos propusemos resolver.

3. - ARQUITECTURA DO SISTEMA DE CALENDARIZAÇÃO

3. 1 - INTRODUÇÃO

O sistema de calendarização é uma estrutura modular com 2 módulos de Comunicação com o Utilizador, e uma estrutura interna constituída por uma Base de Conhecimento, um algoritmo de pesquisa A^*, um módulo de Regras de Produção e Estratégia de Controlo, um módulo do Custo de Transição entre Estados e um módulo da Heurística adoptada, que tem uma ligação a uma biblioteca para resolução do problema de programação linear associado à resolução da heurística, através de um módulo de comunicação entre o sistema interno de calendarização e a biblioteca.

O sistema está desenvolvido em PROLOG e tem 2430 linhas de programação. A rotina da biblioteca IMSL ("International Mathematical Standard Library") é a rotina DDLPRS implementada em FORTRAN, pelo que se foi obrigado a construir um módulo de comunicação entre o sistema implementado em PROLOG e a linguagem FORTRAN da rotina de biblioteca.

3. 2 - MÓDULOS DE COMUNICAÇÃO COM O UTILIZADOR

O utilizador interactua com o sistema através de 2 módulos:

3.2.1 - Representação de Conhecimento Associado ao Problema de Calendarização

3.2.2 - Representação de Conhecimento Associado à Resolução do Problema de Calendarização

3.2.1 - Representação de Conhecimento Associado ao Problema de Calendarização

Através deste módulo o utilizador interactua com o sistema. O conhecimento que o utilizador tem do problema de calendarização que pretende resolver é o fluxo de conhecimento que o sistema vai receber do exterior. Esse fluxo de conhecimento é constituído por:

- As actividades constituídas por tarefas com relações de precedência que se querem calendarizar. O utilizador vai introduzir os arcos dos grafos associados às actividades.

- O dia de começo de cada repetição das actividades.

- O tempo máximo que o utilizador permite que cada repetição das actividades pode levar.

- As modularidades alternativas para a realização das tarefas.

- O salário associado a cada tipo de trabalhador que vai estar envolvido no problema de calendarização.

- O último dia do período que o utilizador pretende estudar.

- O número máximo de equipas envolvidas no problema.

A exemplificação do uso deste módulo de comunicação do utilizador com o sistema é feita no Capítulo 4, sendo aí detalhadamente especificado o fluxo de informação de utilizador para o sistema.

3.2.2 - Representação de Conhecimento Associado à Resolução do Problema de Calendarização

Este módulo estabelece o diálogo do sistema com o utilizador, sendo a direcção do fluxo de conhecimento do sistema para o utilizador.

O conhecimento que o utilizador obtem é:

- Custo global de resolução do problema de calendarização.

- Descrição dos recursos necessários por tipo de trabalhador envolvido no problema.

- Calendarização das actividades.

A exemplificação do uso deste módulo de comunicação do sistema com o utilizador é feita no Capítulo 5.

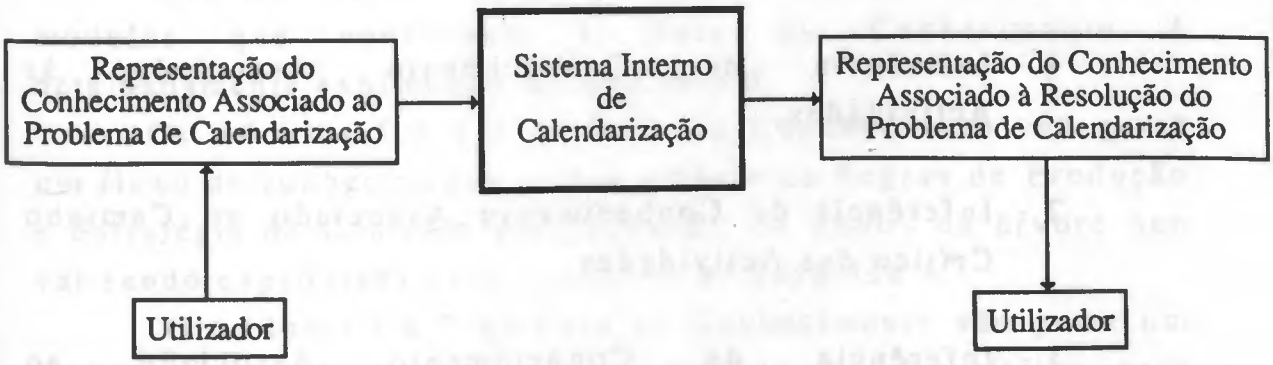


Fig. 3.1 - Fluxo do conhecimento do utilizador com o sistema

3. 3 - ESTRUTURA INTERNA DO SISTEMA DE CALENDARIZAÇÃO

A estrutura interna modular do sistema de calendarização é constituída por 6 módulos básicos:

3.3.1 - Base de Conhecimento

3.3.2 - Algoritmo de Pesquisa A*

3.3.3 - Regras de Produção e Estratégia de Controlo

3.3.4 - Custo de Transição entre Estados

3.3.5 - Heurística

3.3.6 - Comunicação do Sistema de Calendarização com a Biblioteca IMSL

3.3.1 - Base de Conhecimento

A Base de Conhecimento é gerada por 5 módulos de inferência automática de conhecimento:

1 - Inferência de Conhecimento Associado às Actividades.

2 - Inferência de Conhecimento Associado ao Caminho Crítico das Actividades.

3 - Inferência de Conhecimento Associado ao Relaxamento das Repetições das Actividades.

4 - Inferência de Conhecimento Geral para Utilização na Heurística.

5 - Inferência de Conhecimento para Utilização no Problema da Programação Linear Associado à Heurística.

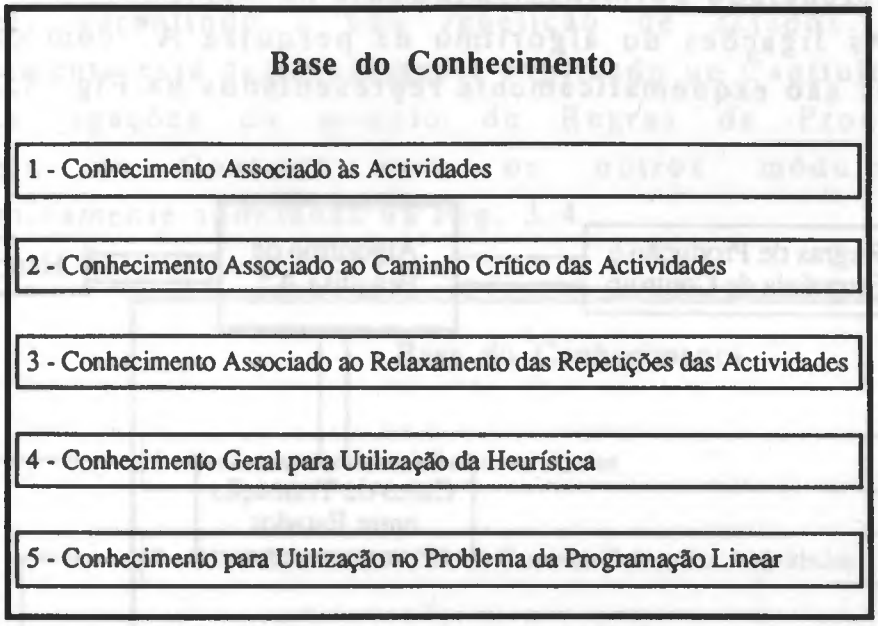


Fig. 3.2 - Estrutura da Base do Conhecimento

A inferência automática do conhecimento associado aos 5 módulos que constituem a Base de Conhecimento é detalhadamente explicitada no Capítulo 6.

Os módulos 1,2 e 3 da Base de Conhecimento vão gerar um fluxo de conhecimento para o módulo de Regras de Produção e Estratégia de Controlo para expandir os nodos da árvore que vai sendo explicitada pelo algoritmo de pesquisa A^*.

Os módulos 4 e 5 da Base de Conhecimento vão gerar um fluxo de conhecimento para o módulo da Heurística para

resolução da heurística associada a cada nodo expandido na árvore de estados explicitada pelo algoritmo de pesquisa A*.

3.3.2 - Algoritmo de Pesquisa A*

Este módulo é a implementação do algoritmo de pesquisa A*, referenciado pormenorizadamente no Capítulo 10.

As ligações do algoritmo de pesquisa A* com os outros módulos são esquematicamente representados na Fig. 3.3.

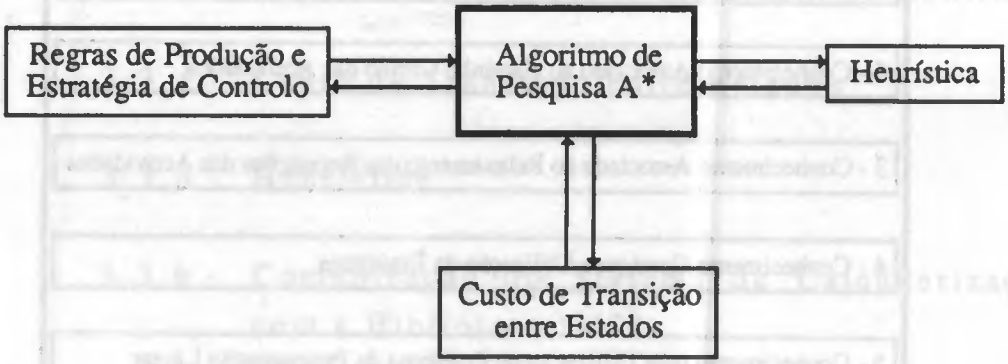


Fig. 3.3 - Fluxo de conhecimento do algoritmo de pesquisa A* com outros módulos

O fluxo de conhecimento entre o módulo A* e o módulo de Regras de Produção e Estratégia de Controlo é necessário para se fazer a expansão de um estado que foi seleccionado para expansão pelo algoritmo A*. A cada novo estado gerado por expansão é associada uma heurística que justifica o fluxo de conhecimento entre o módulo A* e o módulo da Heurística. E também por cada novo estado gerado é necessário saber qual o custo de transição associado à transição do estado predecessor para o novo estado gerado. Estabelece-se para isso o fluxo de

conhecimento entre o módulo A* e o módulo do Custo de Transição entre Estados.

3.3.3 - Regras de Produção e Estratégia de Controlo

Este módulo gera todos os estados sucessores de um estado representado por uma estrutura de estado, e só os gera uma vez, garantindo a não repetição de estados. O seu funcionamento está detalhadamente explicado no Capítulo 7.

As ligações do módulo de Regras de Produção e Estratégia de Controlo com os outros módulos são esquematicamente apontadas na Fig. 3.4

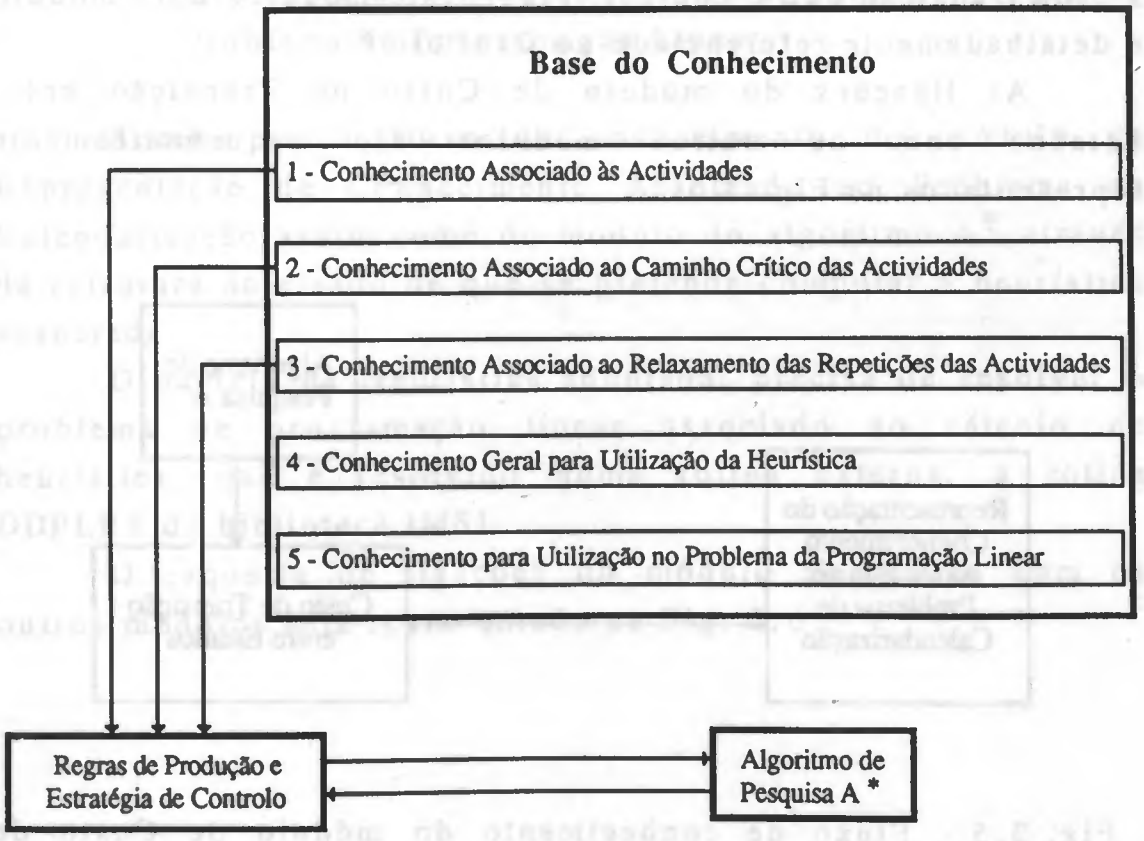


Fig. 3.4 - Fluxo de conhecimento do módulo de Regras de Produção e Estratégia de Controlo com os outros módulos

O módulo de regras de Produção e Estratégia de Controlo está intimamente ligado ao algoritmo de pesquisa A* para expandir os nodos e recebe fluxo de conhecimento dos módulos 1, 2 e 3 da Base de Conhecimento. Este módulo é detalhadamente referenciado no Capítulo 7.

3.3.4 - Custo de Transição entre Estados

Este módulo contabiliza o custo de transição entre estados. Quando o algoritmo A* através das Regras de Produção e Estratégia de Controlo gera os sucessores de um estado, essas transições do estado para os sucessores têm um custo associado a cada transição que é contabilizado neste módulo. Este módulo é detalhadamente referenciado no Capítulo 8.

As ligações do módulo de Custo de Transição entre Estados com os outros módulos são esquematicamente representados na Fig. 3.5.

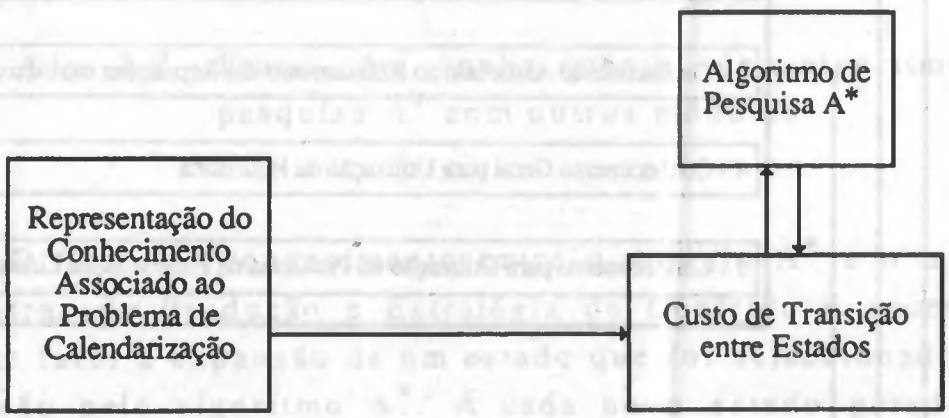


Fig. 3.5 - Fluxo de conhecimento do módulo de Custo de Transição entre Estados com os outros módulos

3.3.5 - Heurística

Este módulo contabiliza a heurística associada a cada estado gerado por A^* através das Regras de Produção e Estratégia de Controlo. A cada estado gerado é associada a heurística adoptada, cuja computação está explicitada detalhadamente no Capítulo 9. O módulo de heurística recebe fluxo de conhecimento da Base de Conhecimento através dos módulos

4 - Inferência do Conhecimento Geral para Utilização na Heurística

e

5 - Inferência do Conhecimento Geral para Utilização no Problema de Programação Linear

Recebe ainda fluxo de conhecimento do módulo de Representação de Conhecimento Associado ao Problema de Calendarização assim como do módulo do algorítmo A^* através da estrutura do estado de que se pretende computar a heurística associada.

O módulo da Heurística adoptada, precisa de resolver o problema de programação linear associado ao cálculo de heurística, que é resolvido numa rotina externa, a rotina DDPLRS da biblioteca IMSL.

O esquema de ligações do módulo Heurística com os outros módulos está representado na Fig. 3.6

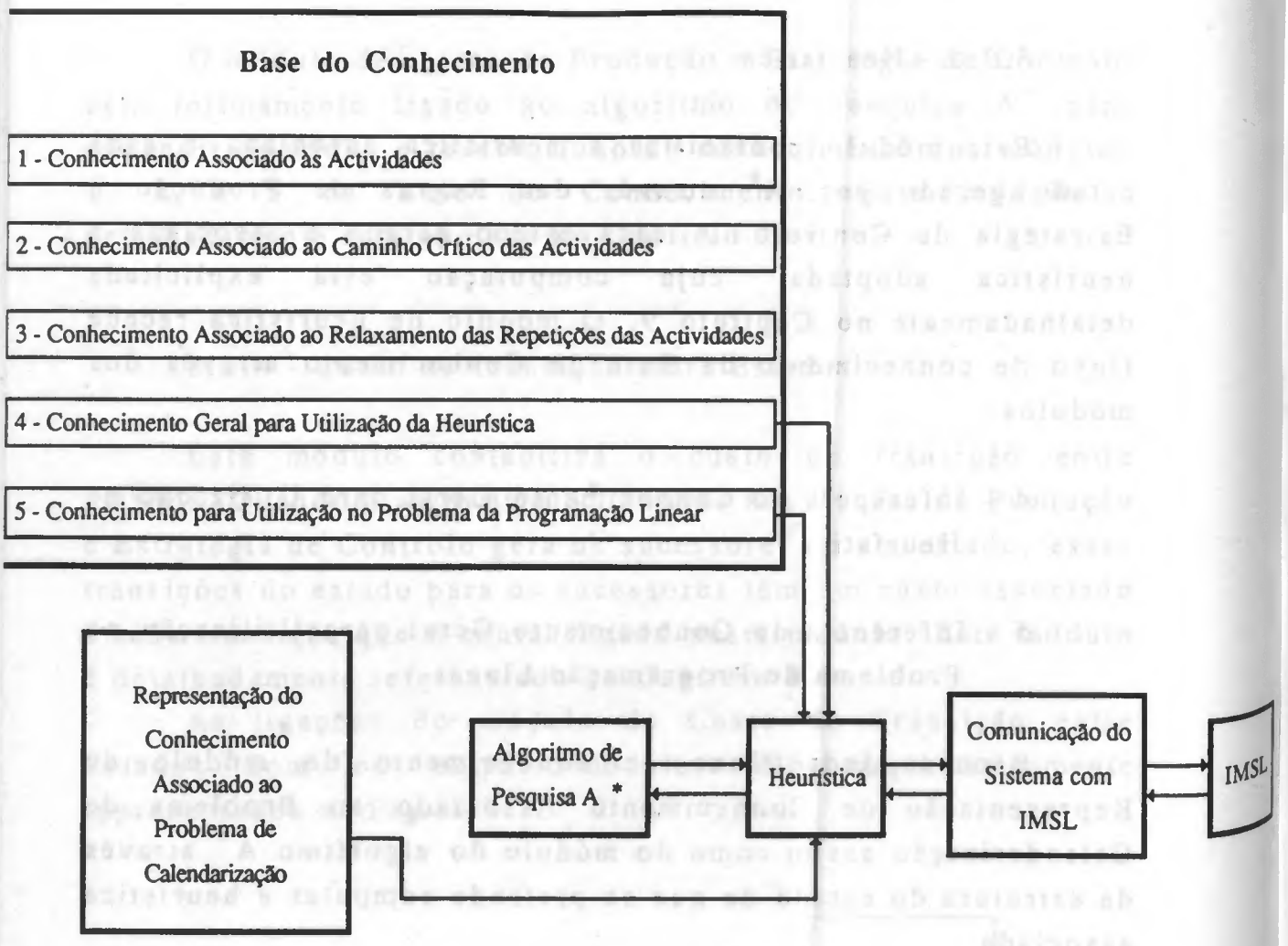


Fig. 3.6 - Fluxo de conhecimento do módulo Heurística com os outros módulos

3. 3. 6 - Comunicação do Sistema de Calendarização com a Biblioteca IMSL

O módulo da Heurística implementado em PROLOG comunica através deste módulo com a biblioteca IMSL ("International Mathematical Standard Library") para acesso à rotina DDLPRS implementado em FORTRAN.

Este módulo prepara os dados que estão em PROLOG em estrutura de lista para dados em formato acessível para FORTRAN de acordo com a especificação da rotina DDPLRS para resolução do problema de programação linear. Além disso, este módulo faz uma alocação dinâmica de memória para utilização da rotina DDPLRS, alocação necessária pois o comprimento das estruturtas de lista do PROLOG variam de acordo com o problema de programação linear associado aos diferentes tipos de trabalhadores.

O esquema de ligações do módulo de Comunicação do Sistema de Calendarização com a biblioteca IMSL está descrito na Fig. 3.7

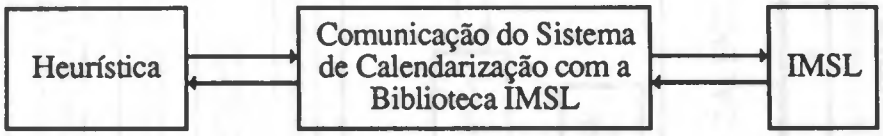


Fig. 3.7 - Esquema de ligação do Sistema de Calendarização com a biblioteca IMSL

3.4 - ESTRUTURA GLOBAL DO SISTEMA DE CALENDARIZAÇÃO

A estrutura modular do sistema de calendarização, com os fluxos de conhecimento entre os módulos já referidos nos items anteriores está esquematicamente descrita na Fig. 3.8

A estrutura modular do sistema de calendarização permite a sua fácil modificação e adaptação a qualquer alteração que se queira introduzir, por exemplo, se se quisesse alterar a heurística adoptada, o único módulo que se teria de actualizar seria o módulo da Heurística. Os módulos de comunicação do

utilizador com o sistema poderiam também ser facilmente alterados para satisfazer outros requisitos do utilizador em termos de comunicação sem alterar o resto da estrutura do sistema de calendarização.

A descrição detalhada de cada um dos módulos será feita nos capítulos que se seguem, tendo-se colocado em Anexo o programa em PROLOG correspondente a cada um dos módulos.

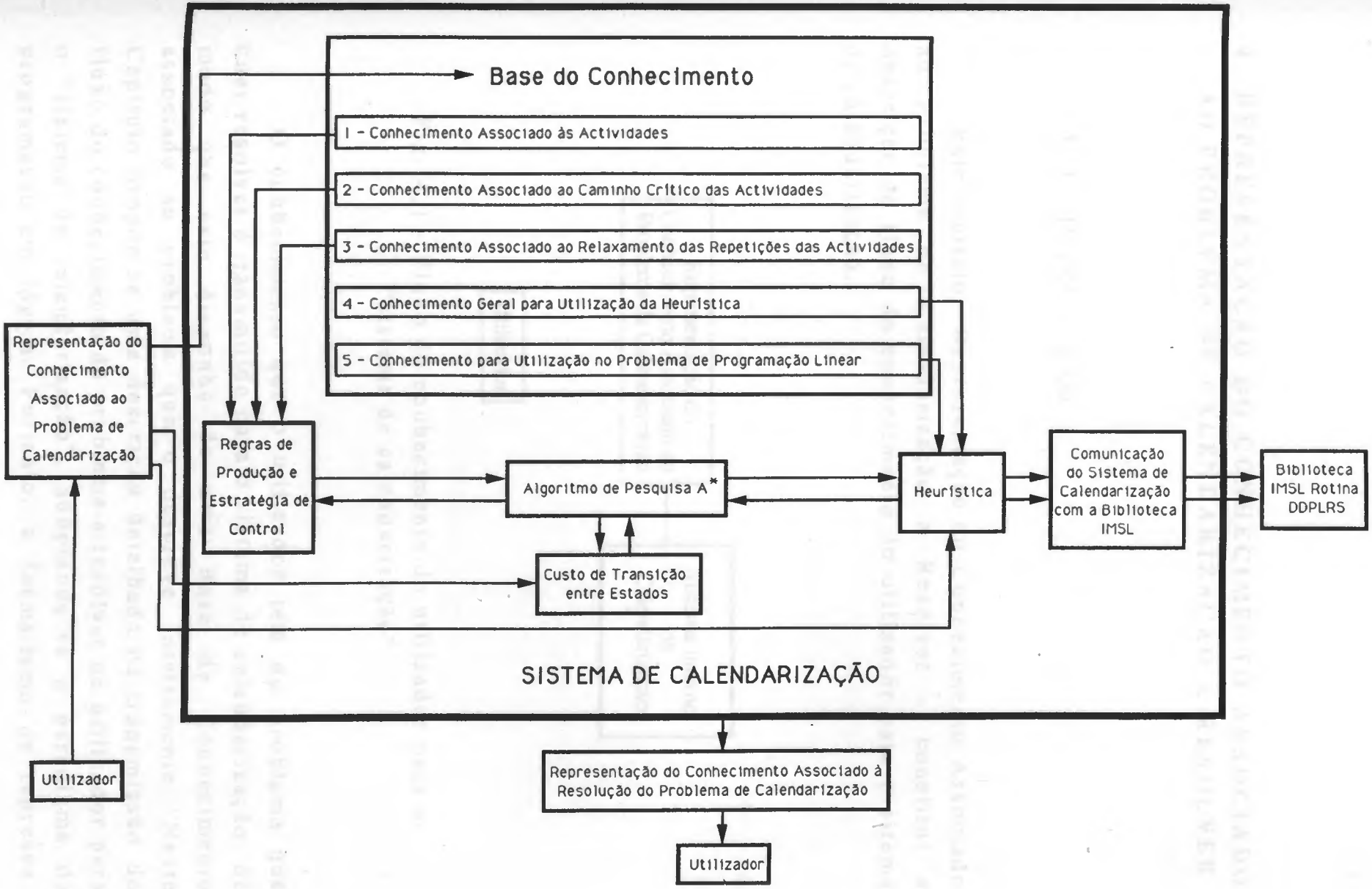


Fig. 3.8 - Estrutura global do "sistema de calendarização"

SISTEMA DE CONHECIMENTO
Base de Conhecimento

4. REPRESENTAÇÃO DO CONHECIMENTO ASSOCIADO AO PROBLEMA DE CALENDARIZAÇÃO A RESOLVER

4. 1 - INTRODUÇÃO

Este capítulo - Representação do Conhecimento Associado ao Problema de Calendarização a Resolver - constitui a descrição do fluxo do conhecimento do utilizador para o sitema de calendarização.

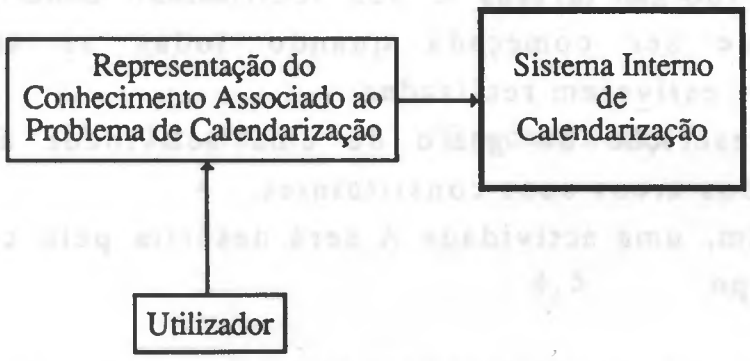


Fig. 4.1 - Fluxo do conhecimento do utilizador para o "sistema de calendarização"

O conhecimento que o utilizador tem do problema que quer resolver é transmitido para o sistema de calendarização, de modo que este disponha de uma Base de Conhecimento associado ao problema que o descreve inteiramente. Neste Capítulo propõe-se uma descrição detalhada da transmissão do fluxo do conhecimento do problema a resolver do utilizador para o "sistema de calendarização", adoptando-se o paradigma da programação em lógica. Portanto, o formalismo de represen-

tação subjacente é a lógica de primeira ordem ou cálculo dos predicados.

4. 2 - GRAFOS DE TAREFAS ASSOCIADOS A CADA ACTIVIDADE

As actividades que vão ser realizadas no calendário proposto, são constituídas por tarefas, com relações de precedência entre si. Assim a modelização das actividades é feita por grafos orientados.

Para cada actividade, os arcos do grafo orientado associado são as tarefas a ser realizadas. Uma tarefa só é possível de ser começada quando todas as tarefas suas precedentes estiverem realizadas.

A descrição do grafo de uma actividade é feita pela descrição dos arcos seus constituintes.

Assim, uma actividade A será descrita pelo conjunto dos arcos do tipo

arco (A,B,C,D)

em que

A - Identificação da actividade a que o arco pertence
B - Nó inicial do arco
C - Nó terminal do arco
D - Tarefa associada ao arco

A,B,C e D são codificadas numericamente. Para o conjunto das actividades consideradas, sendo

- m o número máximo de actividades

- **n** o número máximo de nodos de uma determinada actividade
- **p** o número máximo de possíveis tarefas a realizar no conjunto das actividades,

A varia de 1 a **m**,
B e C variam de 1 a **n**, e
D varia de 0 a **p**.

D varia de 0 a **p**, pois é necessário por vezes introduzir uma tarefa fictícia para poder modelizar o grafo associado a uma actividade. Essa tarefa fictícia é a tarefa codificada com o número zero.

Veja-se por exemplo uma actividade constituída por 3 tarefas, a tarefa número 4, 5 e 3 com a seguinte tabela de precedências:

Tarefa	Precedências
4	-
5	-
3	4,5

Um grafo para modelizar esta actividade seria o seguinte:

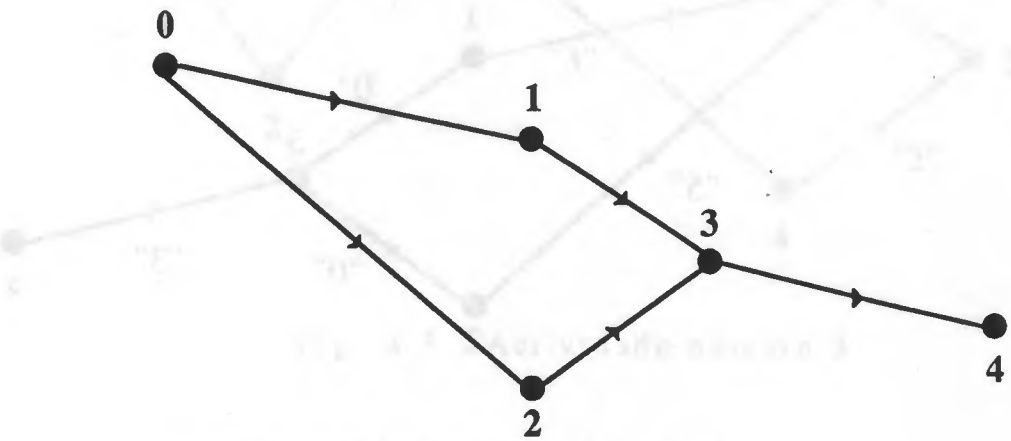


Fig. 4.2 - Grafo de actividade

em que as tarefas realizadas,

- do nó 0 para o nó 1 é a tarefa 4
- do nó 0 para o nó 2 é a tarefa 5
- do nó 1 para o nó 3 é a tarefa fictícia 0
- do nó 2 para o nó 3 é a tarefa fictícia 0
- do nó 3 para o nó 4 é a tarefa 3

Se esta actividade, representada na Fig. 2.1 fosse a actividade número 1, a descrição desta actividade seria dada pelos seguintes termos Prolog:

```
arco (1, 0, 1, 4).
arco (1, 0, 2, 5).
arco (1, 1, 3, 0).
arco (1, 1, 2, 0).
arco (1, 3, 4, 3).
```

Vamos usar como notação gráfica do grafo associado a uma actividade, o grafo com a indicação da numeração dos nodos e a tarefa associada aos arcos. Assim a actividade referida será a indicada na Fig. 4.3

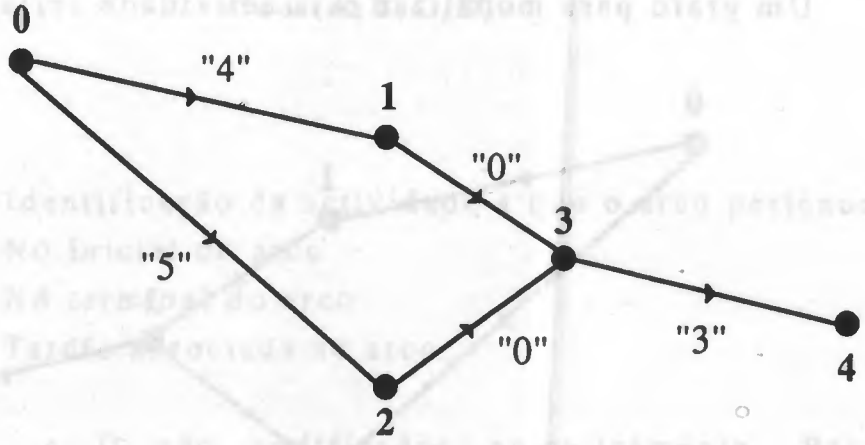


Fig. 4.3 - Actividade número 1

Considere-se que as actividades a serem realizadas são a referida actividade número 1, a actividade número 2 representada na Fig. 4.4 e a actividade número 3 indicada na Fig. 4.5.

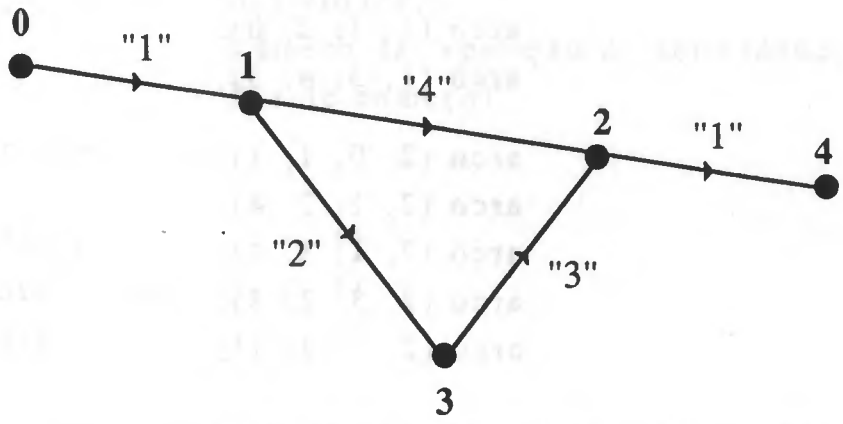


Fig. 4.4 - Actividade número 2

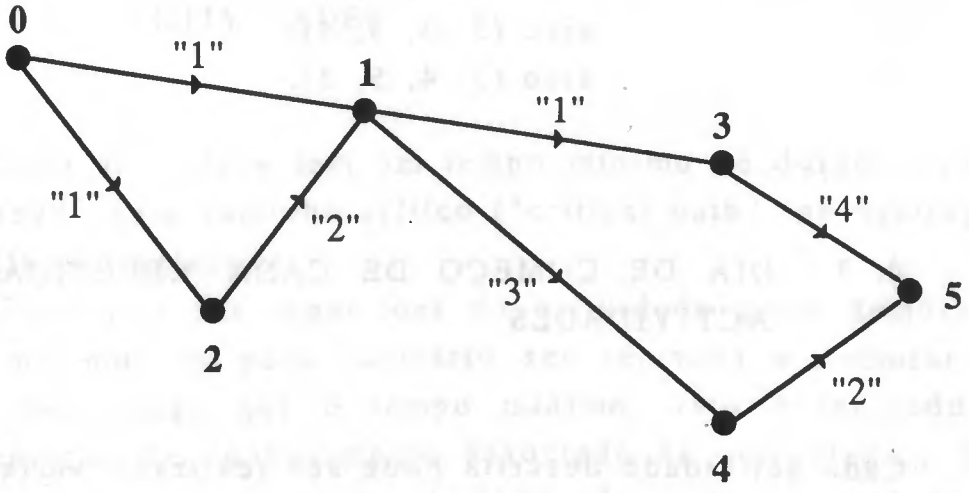


Fig. 4.5 - Actividade número 3

A representação de conhecimento em lógica referente àquelas actividades é dada pelos termos Prolog seguintes:

```
arco (1, 0, 1, 4).
arco (1, 0, 2, 5).
arco (1, 1, 3, 0).
arco (1, 1, 2, 0).
arco (1, 3, 4, 3).

arco (2, 0, 1, 1).
arco (2, 1, 2, 4).
arco (2, 1, 3, 2).
arco (2, 3, 2, 3).
arco (2, 2, 4, 1).

arco (3, 0, 1, 1).
arco (3, 0, 2, 1).
arco (3, 2, 1, 2).
arco (3, 1, 3, 1).
arco (3, 1, 4, 3).
arco (3, 3, 5, 4).
arco (3, 4, 5, 2).
```

4. 3 - DIA DE COMEÇO DE CADA REPETIÇÃO DAS ACTIVIDADES

Cada actividade descrita pode ser realizada várias vezes durante o período em estudo.

Assim, cada actividade gera um conjunto de actividades desse tipo que são individualizadas pelo dia em que começam.

Se a actividade 1 tiver uma ocorrência 3, a primeira começando no dia 1, a segunda ocorrência começando também

no dia 1, e a terceira ocorrência no dia 4, isso será especificado na representação de conhecimento associado às actividades pelo número da actividade, pelo número de repetição da actividade e pelo dia de começo dessa repetição da actividade,

```
dia_começo (Actividade,
            Número da repetição da actividade,
            Dia de começo).
```

Neste caso seria

```
dia_começo (1, 1, 1).
dia_começo (1, 2, 1).
dia_começo (1, 3, 4).
```

E, o mesmo se repetiria para as outras actividades.

4.4 - TEMPO MÁXIMO DE CADA REPETIÇÃO DAS ACTIVIDADES

Cada actividade tem um tempo mínimo de duração que é determinado pelo caminho crítico ("critical path", na bibliografia de língua inglesa).

Cada uma das repetições da actividade pode demorar o tempo mínimo, ou pelo contrário ser relaxada e demorar um tempo mais longo que o tempo mínimo. Isto é indicado na representação de conhecimento associado às actividades, pelo tempo máximo que uma determinada repetição de uma

determinada actividade pode levar:

tempo_máx (Actividade,
Nº de repetição da actividade,
Tempo máximo permitido para a
duração).

No exemplo anterior, para a actividade número 2, o tempo mínimo tomado pelo caminho crítico é de 5 dias. Supondo que a repetição número 1 tem de ser realizada no tempo mínimo, assim como a repetição número 2, e que a repetição número 3 pode levar até 10 dias, para a actividade número 2, na representação de conhecimento associado introduziam-se os seguintes termos Prolog:

```
tempo_máx (2, 1, 5).
tempo_máx (2, 2, 5).
tempo_máx (2, 3, 10).
```

e o mesmo se repete para as outras actividades.

4. 5 - MODULARIDADES ALTERNATIVAS PARA A REALIZAÇÃO DAS TAREFAS

Cada tarefa é descrita pela sua codificação numérica e pela lista das equipas alternativas para a realização dessa tarefa. A descrição das equipas alternativas para a realização da tarefa traduz a modularidade associada à realização da tarefa.

Cada equipa é descrita pelo número de trabalhadores que a constituem, pela especificação do tipo de trabalhador que a constitui, e pelo número de dias que a equipa leva a concluir a tarefa.

Assim, na representação de conhecimento associado às tarefas, a descrição das modularidades alternativas para a realização de uma tarefa é expressa pelo seguinte termo Prolog:

```
tarefa (Número da tarefa,
        [Nº de trabalhadores * Tipo de trabalhadores *
         Nº de dias para realizar a tarefa |
         Resto de lista]).
```

Exemplo:

O termo Prolog,

```
tarefa (3, [1*trab4*2,2*trab4*1]).
```

tem a seguinte leitura: "A tarefa 3 pode ser feita por 2 equipas alternativas. Pode ser feita com uma equipa de 1 trabalhador de tipo 4 e leva 2 dias a ser feita, ou por uma equipa de 2 trabalhadores de tipo 4 e leva 1 dia a ser feita". É evidente que para uma determinada tarefa, todas as equipas são constituídas pelo mesmo tipo de trabalhador. Assim, no exemplo representado, as 2 equipas alternativas são ambas constituídas pelo mesmo tipo de trabalhador, trab 4, neste caso.

Tinha-se referido em 4.2 que se introduz, por necessidade de modelização dos grafos associados às actividades, uma tarefa fictícia com a codificação numérica 0, que será descrita na representação de conhecimento associado às tarefas por

```
tarefa (0, [0*trab1*0]).
```

A tarefa fictícia toma ficticiamente zero trabalhadores de qualquer tipo, neste caso particularizado para tipo trab1, e leva zero dias a fazer.

E, repete-se o mesmo para todas as tarefas que constituem as actividades que vão ser realizadas no período em estudo.

4. 6 - SALÁRIO DOS TRABALHADORES

Como o objectivo último da optimização de recursos é a obtenção do custo mínimo global para a manutenção, é estritamente necessário que se introduzam os salários associados a cada tipo de trabalhador, através do termo Prolog:

salário (Tipo de trabalhador,
Salário do tipo de trabalhador).

Exemplo:

O termo Prolog,

salário (trab3, 3).

tem a seguinte leitura: "O salário do trabalhador tipo 3 é de 3 unidades".

4. 7 - ÚLTIMO DIA DO PERÍODO EM ESTUDO

Indica-se o último dia do período em estudo, o fim do calendário pelo termo Prolog:

dia_final (Último dia do período em estudo).

Se o período em estudo for anual, por exemplo, ter-se-á:

dia_final (365).

4. 8 - NÚMERO MÁXIMO DE EQUIPAS

Para posterior utilização no acesso à rotina de programação linear, introduz-se o número máximo de equipas, que vai definir um limite superior ("upperbound") das variáveis do problema de programação linear.

n_máximo_equipas (Máximo).

5. REPRESENTAÇÃO DO CONHECIMENTO ASSOCIADO À RESOLUÇÃO DO PROBLEMA DE CALENDARIZAÇÃO

5. 1 - INTRODUÇÃO

Este capítulo Representação do Conhecimento Associado à Resolução do Problema de Calendarização constitui a descrição do fluxo de conhecimento do sistema de calendarização para o utilizador.

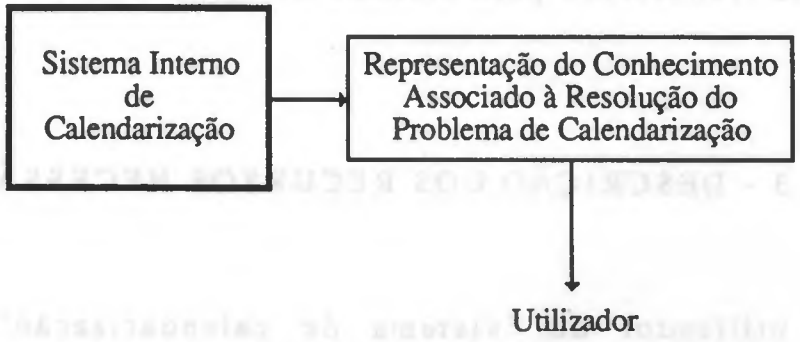


Fig. 5.1 - Fluxo do conhecimento do sistema de calendarização com o utilizador

O conhecimento que o utilizador pretende obter acerca da resolução do problema de calendarização, é-lhe fornecido pelo sistema do modo como se descreve neste capítulo. O fluxo de conhecimento do sistema de calendarização que o utilizador pretende receber é acerca do custo global da solução, dos contingentes de trabalhadores necessários, e a calendarização das tarefas das actividades que conduziu ao custo global mínimo.

5. 2 - CUSTO GLOBAL DA SOLUÇÃO

O fim último do utilizador do "sistema de calendarização" é realizar as actividades por ele especificadas com um custo operacional mínimo. O custo global da solução proposta pelo "sistema de calendarização" é assim um conhecimento fundamental a ser transferido para o utilizador. A despesa feita com um tipo de trabalhador, é o salário desse tipo de trabalhador multiplicado pelo número total de trabalhadores desse tipo que são necessários. O custo global da calendarização é o somatório das despesas feitas com todos os tipos de trabalhadores. Este custo global minimizado pelo "sistema de calendarização" é assim um conhecimento importante transferido pelo sistema ao utilizador.

5. 3 - DESCRIÇÃO DOS RECURSOS NECESSÁRIOS

O utilizador do "sistema de calendarização" necessita conhecer o contingente dos diferentes tipos de trabalhadores necessários para a realização das diferentes repetições das actividades. Por contingente de um tipo de trabalhador entende-se o total de trabalhadores necessários para concretizar a calendarização proposta pelo sistema.

Exemplo: Suponha-se que para realizar uma calendarização que envolve 3 tipos de trabalhadores, trab1, trab2 e trab3, necessita-se de 5 trabalhadores trab1, 3 trabalhadores trab2 e 7 trabalhadores trab3. Este conhecimento seria

transferido do sistema para o utilizador por:

TOTAL DE TRABALHADORES trab1: 5
TOTAL DE TRABALHADORES trab2: 3
TOTAL DE TRABALHADORES trab3: 7

5. 4 - CALENDARIZAÇÃO DAS ACTIVIDADES

Um conhecimento que se tem de possuir é o da calendarização das tarefas das diferentes repetições das diversas actividades a fazer. Tem de se conhecer em que dia começa cada tarefa, em que dia acaba e qual o tipo de equipa assignada à realização da tarefa. Este conhecimento é transferido do "sistema de calendarização" para o utilizador pela informação dia a dia das tarefas que se começam a fazer nesse dia indicando com quantos trabalhadores e em que dia estão concluídas.

Exemplo: Considere-se a informação fornecida pelo "sistema de calendarização":

DIA: 1
COMEÇAR A FAZER AS TAREFAS

arc(1,2,2,3,4,7,3) com 3 trabalhadores trab5 e termina no dia 7
arc(2,3,1,2,5,3,2) com 2 trabalhadores trab3 e termina no dia 3

DIA: 2
COMEÇAR A FAZER AS TAREFAS

arc(3,1,0,1,3,3,2) com 2 trabalhadores trab1 e termina no dia 3

Assim, para o dia 1 começam-se a fazer:

- A tarefa que corresponde ao arco do nodo 2 para o nodo 3 que tem assignada a tarefa número 4, da actividade número 1 e sua repetição número 2, com 3 trabalhadores tipo trab5 estando a tarefa concluída no dia 7.
- A tarefa que corresponde ao arco do nodo 1 para o nodo 2 que tem assignada a tarefa número 5, da actividade número 2 e sua repetição número 3, com 2 trabalhadores tipo trab3 estando a tarefa concluída no dia 3.

De igual modo se terá a informação da calendarização para o dia 2 e para os restantes dias.

6. INFERÊNCIA AUTOMÁTICA DE CONHECIMENTO

6. 1 - INTRODUÇÃO

Este capítulo descreve detalhadamente a inferência automática do conhecimento que o sistema de calendarização produz de modo a constituir uma Base de Conhecimento. Esta inferência automática é feita a partir do conhecimento contido na "Representação de Conhecimento Associado ao Problema de Calendarização" fornecido pelo utilizador do sistema.

Assim o sistema constitui por inferência automática do conhecimento uma Base de Conhecimento:

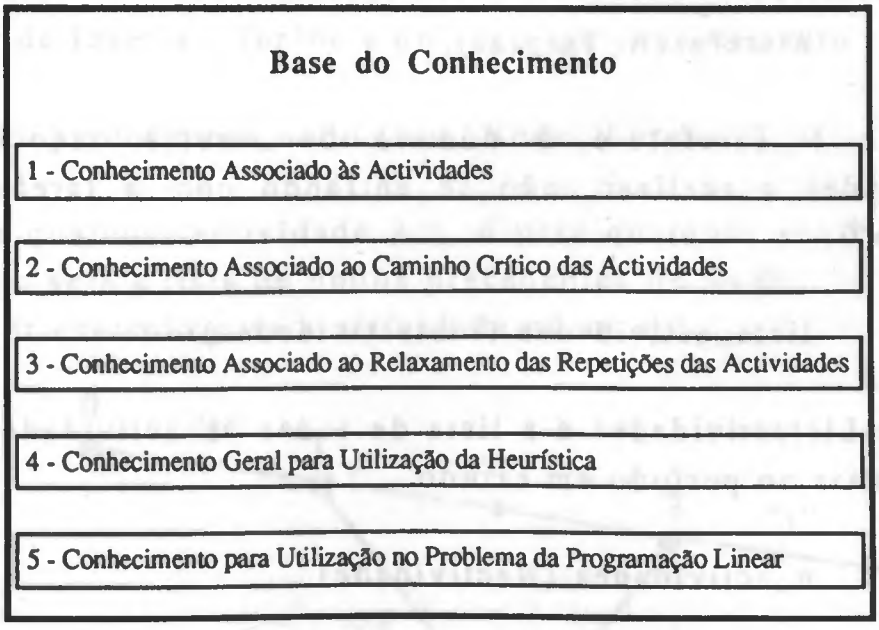


Fig. 6.1 - Base de Conhecimento

Esta "Base de Conhecimento" vai fornecer fluxo de conhecimento ao módulo de "Regras de Produção e Estratégia de Controlo" e de "Heurística" que interactuam com o algoritmo de pesquisa A*.

Este capítulo ilustra detalhadamente a inferência automática de conhecimento, associada à Base de Conhecimento.

6. 2 - INFERÊNCIA DE CONHECIMENTO ASSOCIADO ÀS ACTIVIDADES

Da "Representação de Conhecimento Associado ao Problema de Calendarização a Resolver", derivam-se por inferência, os conhecimentos gerais associados às actividades,

ntarefas(N_Tarefas),

em que N_Tarefas é o número de tarefas associadas às actividades a realizar, não se entrando com a tarefa fictícia número 0,

lista_actividades (Listactividades).

em que Listactividades é a lista de todas as actividades que se vão realizr no período em estudo,

n_actividades (Nactividade).

sendo Nactividades o número total das actividades referidas,

lista_arcos (Act, Listarcos).

para qualquer actividade Act, a Listarcos dá a lista dos arcos

seus constituintes,

n_arcos_actividade (Act, Narcos).

para qualquer actividade Act, Narcos dá o número de arcos seus constituintes,

lista_nodos (Act, Lnodos).

para qualquer actividade Act, Lnodos dá a lista de nodos seus constituintes,

n_nodos_actividade (Act, Nnodos).

para qualquer actividade Act, Nnodos dá o número de nodos seus constituintes.

Para a análise temporal das actividades a realizar, vai-se precisar de fazer a inferência do seguinte conhecimento

precedentes_nodo (Act, Nodo, Lprec).

que para qualquer actividade Act, e para qualquer seu Nodo, dá Lprec, ou seja a lista de nodos precedentes de Nodo.

Por exemplo para a actividade número 2,

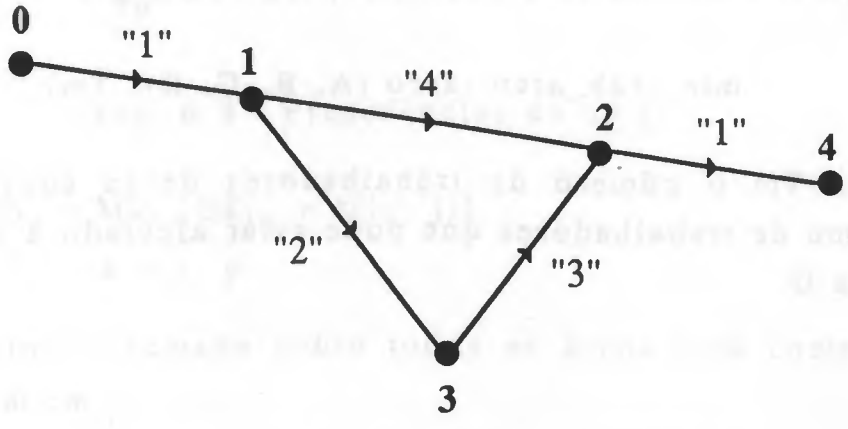


Fig. 6.2 - Actividade número 2

precedentes_nodo (2, 2, Lprec).

temos Lprec=[1,3].

Outro tipo de conhecimento que vai ser preciso, são os tempos mínimos e máximos que um determinado arco, ou seja tarefa, pode levar, tendo em conta a modularidade das equipas. Assim temos,

dura_min_arco (arco (A, B, C, D), Dm).

que dá o tempo mínimo Dm, que o arco arco (A, B, C, D) pode levar a fazer, que corresponde à sua realização com a maior equipa das alternativas possíveis,

max_trab_arco (arco (A, B, C, D), TM).

sendo TM o número de trabalhadores dessa equipa. TM é o máximo de trabalhadores que pode estar afectado à realização da tarefa D.

dura_max_arco (arco (A, B, C, D), DM).

dá o tempo máximo DM, que o arco arco (A, B, C, D) pode levar a fazer, que corresponde à sua realização com a menor equipa das alternativas possíveis, traduzida por

min_trab_arco (arco (A, B, C, D), Tm).

sendo Tm o número de trabalhadores dessa equipa. Tm é o mínimo de trabalhadores que pode estar afectado à realização da tarefa D.

6. 3 - INFERÊNCIA DE CONHECIMENTO ASSOCIADO AO CAMINHO CRÍTICO DAS ACTIVIDADES

Para a análise temporal de uma actividade, um conhecimento de que vamos precisar, é o Início Mais Cedo ("Earliest Start Time", ou simplesmente "ES" na bibliografia da língua inglesa) de todos os nós dessa actividade.

Consideremos o nó j que tem como precedentes os nós i_1, i_2 , ... i_p. Sendo $d(i_k, j)$ a duração mínima que a tarefa assignada ao arco de i_k a j pode levar,

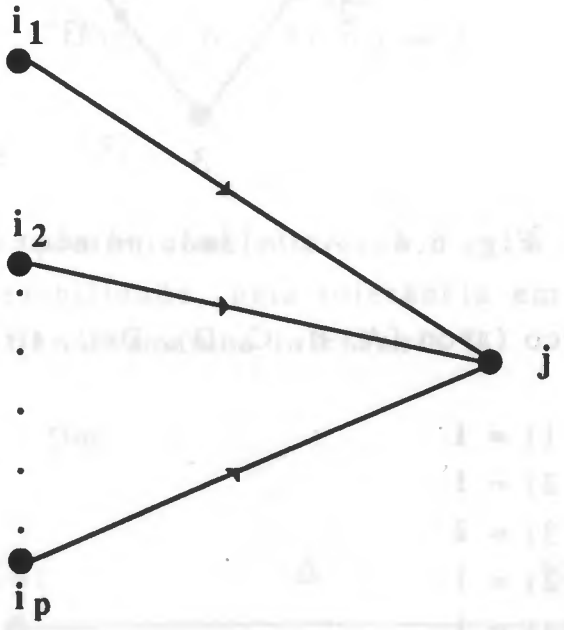


Fig. 6.3 - Precedências do nó j

$$ES_j = \underset{k = 1, p}{\text{Max}} \left[ES_{i_k} + d(i_k, j)\right]$$

onde o máximo é tomado sobre todos os arcos com começo i_k que terminem em j.

Portanto todos os arcos cujo nó inicial seja o nó j, têm como dia mais cedo possível de início, ESj, que é o dia mais cedo em que todos os arcos que terminam em j já estão realizados.

Para o exemplo da actividade número 2 referido

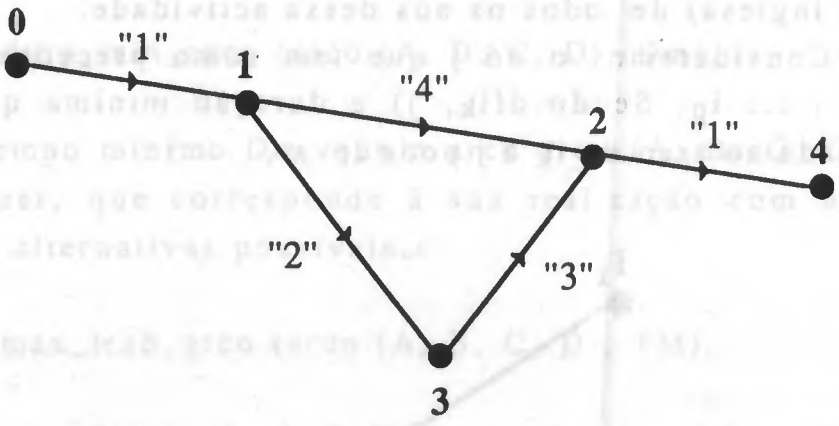


Fig. 6.4 - Actividade número 2

de dura_min_arco (arco (A, B, C, D), Dm), tiram-se

d(0, 1) = 1
d(0, 2) = 1
d(0, 3) = 2
d(3, 2) = 1
d(2, 4) = 1

e infere-se es(Act, Nodo, Es), em que

Act - codificação da actividade,
Nodo - nó da actividade
ES - valor de ES para esse nodo

$ES_0 = 0$
$ES_1 = 1$

$ES_2 = 4$

$ES_3 = 3$

$ES_4 = 5$

Uma medida de flexibilidade de um arco é feita comparando a diferença de ES do nó terminal e o ES do nó inicial, Δ, com a duração mínima que o arco pode tomar.

Fig. 6.6 - Arco i → j

$$\Delta = ESj - ESi$$

sendo Dm a duração mínima do arco, se Δ > Dm, o arco apresenta uma flexibilidade, uma tolerância em relação ao seu dia de começo, a que chamamos tolerância T,

$$T = \Delta - Dm$$

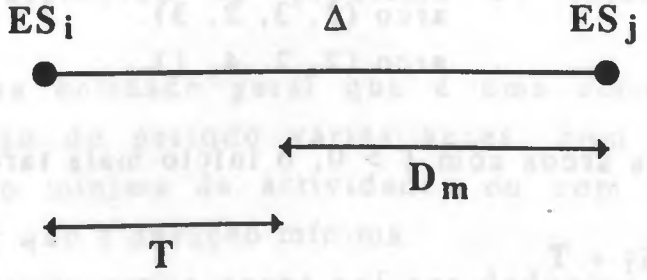


Fig. 6.7 - Tolerância T do arco de i para j

Se T > 0, o começo do arco pode-se atrasar de T sem que a actividade como um todo seja atrasada.

Se T = 0, o arco pertence ao caminho crítico, ou seja pertence ao conjunto dos arcos da actividade que têm de começar a ser feitos no ES dos seus nós iniciais, pois se tal não acontecer, toda a actividade fica atrasada.

Para o exemplo considerado, o caminho crítico é apontado na Fig. 6.8:

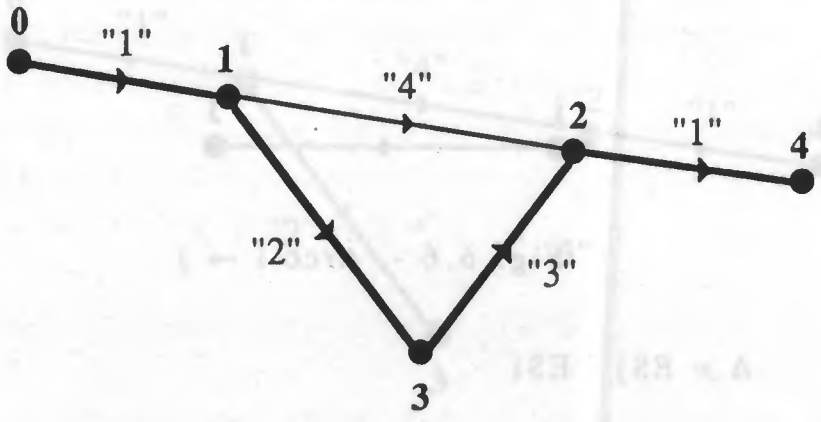


Fig. 6.8 - Caminho crítico de actividade número 2

e formado pelo conjunto dos arcos:

arco (2, 0, 1, 1).
arco (2, 1, 3, 2).
arco (2, 3, 2, 3).
arco (2, 2, 4, 1).

Para os arcos com T > 0, o início mais tarde possível, é

$ES_i + T$

É evidente que para os arcos do caminho crítico o início mais tarde possível coincide com ES_i.

Estes conceitos são referenciados nos termos Prolog:

tolerância (arco (A, B, C, D), T).

e

início_mais_tarde_arco (arco (A, B, C, D), Imt).

A duração mínima que uma actividade pode levar é o ES do nó terminal da actividade, que pode ser inquirido por

dura_min_activ (Act, Mindura).

em que a variável Mindura é a duração mínima da actividade Act.

6. 4 - INFERÊNCIA DE CONHECIMENTO ASSOCIADO AO RELAXAMENTO DAS REPETIÇÕES DAS ACTIVIDADES

O conhecimento até agora deduzido é geral para as actividades definidas pelos arcos da representação de conhecimento associado ao problema de calendarização a resolver.

Mas, essa entidade geral que é uma actividade, pode ocorrer ao longo do período várias vezes, com uma duração igual à duração mínima da actividade, ou com uma duração permitida maior que a duração mínima.

O conhecimento que agora vai ser deduzido é para essas entidades menos abstractas que são as repetições das actividades.

Cada actividade Act, tem associada uma lista das suas repetições Lrep,

repetições_actividade (Act, Lrep).

Para as repetições das actividades, os arcos que as constituem têm que indicar a repetição de que se trata, pelo que temos agora um novo termo:

arc (A, B, C, D, E, F, G)

em que

A - Identificação da actividade a que o arco pertence
B - Identificação da repetição da actividade
C - Nó inicial do arco
D - Nó terminal do arco
E - Tarefa associada ao arco
F - É uma marca do dia em que o arc está concluído
G - É uma indicação do número de trabalhadores afectos à realização da tarefa E.

Cada uma das repetições tem uma indicação da rapidez, ou seja se vai ser realizado na duração mínima da actividade, ou não.

Se o tempo máximo indicado para a realização da repetição de uma actividade for igual à duração mínima da actividade, a rapidez é indicada com uma marca 1

rapidez (Act, Nº da repetição, 1).

Para as repetições das actividades nestas circunstâncias, o início mais tarde dos "arcs" seus constituintes é o indicado para os arcos correspondentes das suas actividades geradoras.

Se o tempo máximo indicado para a realização da repetição de uma actividade for maior que a duração mínima da

actividade, a rapidez dessa actividade é indicada com uma marca 0

rapidez (Act, Nº da repetição, 0).

Para as repetições das actividades nestas circunstâncias, os arcos seus constituintes estão todos relaxados, inclusive os termos "arcs" correspondentes aos arcos do caminho crítico das actividades geradoras. Todos os termos "arcs" têm um início mais tarde_relaxado,

início_+tarde_relaxado (arc(A,B,C,D,E,F,G), Mtard)

que para o arc(A,B,C,D,E,F,G), da actividade geradora A e sua repetição B, que é relaxada, Mtard dá o início mais tarde possível para o arc(A,B,C,D,E,F,G).

Se D1 for a duração permitida para a repetição da actividade A, se D2 for a duração mínima da actividade A,

$\Delta 1 = D1 - D2$,

o início mais tarde relaxado dos arcos é o início mais tarde acrescido de $\Delta 1$.

6.5 - INFERÊNCIA DE CONHECIMENTO GERAL PARA UTILIZAÇÃO NA HEURÍSTICA

O conhecimento que vamos descrever, vai ser necessário para o cálculo das heurísticas associadas aos estados do espaço de estados, que no processo de programação concorrencial vão ser analisados.

A caracterização de um estado é feita antes de mais, pelo dia da calendarização que ele representa. A heurística associada a esse estado vai fazer uma estimativa do custo que vai ser necessário para fazer a transição desse dia até ao dia final do calendário. Assim, interessa-nos saber o que há ainda para fazer até ao dia final.

Para todas as repetições de todas as actividades que tenham uma duração mínima não relaxada, vão-se analisar os arcos que pertencem aos caminhos críticos, e contabilizar as suas sobreposições.

Por exemplo, vamos considerar que 2 actividades número 2 vão ser realizadas, a 1ª repetição a começar no dia 1, e a 2ª repetição a começar no dia 2.

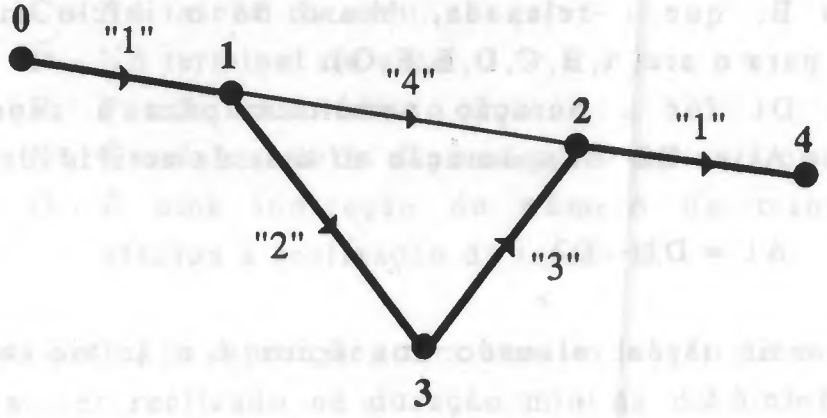


Fig. 6.9 - Actividade número 2 com indicação do seu caminho crítico

Tinha-se inferido o valor de ES para os nodos desta actividade:

$ES_0 = 0$
$ES_1 = 1$
$ES_2 = 4$
$ES_3 = 3$
$ES_4 = 5$

Assim teremos que realizar no dia 1 uma tarefa 1, no dia 2 uma tarefa 2 e uma tarefa 1, no dia 3 duas tarefas 2, no dia 4 uma tarefa 3 e uma tarefa 2, no dia 5 uma tarefa 1 e uma tarefa 3, e no dia 6 uma tarefa 1.

Todas estas tarefas, dado que os arcos pertencem a caminhos críticos de repetições da actividade não relaxadas, têm de ser realizadas pelas equipas mais rápidas, ou seja pelas equipas maiores das alternativas possíveis, que denominaremos por equipas críticas.

O conhecimento referido está indicado no termo:

calendário (Dia, Lista das sobreposições de equipas críticas desse Dia).

No caso anterior, teriamos, por exemplo para o dia 2:

calendário (2, [equipa (0,0), equipa (1,1), equipa (2,1), equipa (3,0), equipa (4,0)]).

Ou seja, para o dia 2, temos uma sobreposição crítica para a tarefa 1, e para a tarefa 2.

O dia 3, terá

calendário (3, [equipa (0,0), equipa (1,0), equipa (2,2), equipa (3,0), equipa (4,0)]).

2 sobreposições críticas para a tarefa 2.

Este calendário é gerado para todos os dias do período em estudo.

Interessa-nos saber para o cálculo da heurística, de um determinado estado, qual o máximo de sobreposições de equipas

críticas que ainda vão ser necessárias, desde o dia assignado ao estado até ao fim do período. Isto é indicado no termo:

sobreposições (Dia, Lista do máximo de sobreposições de equipas críticas que vão ocorrer até ao final do período em estudo).

No mesmo caso,

sobreposições (1, [sobreposição (0,0), sobreposição (1,1), sobreposição (2,2), sobreposição (3,1), sobreposição (4,0)]).

Ou seja, do dia 1 até ao fim do período, ainda vai ocorrer a sobreposição num determinado dia pelo menos, de 1 tarefa 1, 2 tarefas 2 e 1 tarefa 3, realizadas com as equipas mais rápidas.

Esta informação das sobreposições das equipas críticas é gerada para todos os dias do período em estudo.

Outra informação que vai ser necessária para o cálculo da heurística associada a um estado, é a contabilização das tarefas assignadas aos arcos dos caminhos críticos das repetições das actividades não relaxadas que ainda não começaram a ser feitas.

Essa informação está contida no termo:

críticas (Dia, Lista de contabilização global das equipas críticas das repetições das actividades que ao dia Dia ainda não começaram).

Exemplo:

críticas (6, [crítica (0,0), crítica (1,5), crítica (2,0), crítica (3,1), crítica (4,0)]).

Ou seja, das actividades e suas repetições que começam depois do dia 6, vão ainda ser feitas por todo, mesmo sem sobreposições, 5 arcos de caminhos críticos assignados à tarefa 1, e 1 assignado à tarefa 3.

Esta informação é gerada para todos os dias do período em estudo.

Outra informação necessária é a contabilização das tarefas atribuídas aos arcos que não pertencem ao caminho crítico das repetições das actividades não relaxadas que ainda não começaram a ser feitas, e aos arcos das repetições das actividades relaxadas que ainda não começaram a ser feitas.

relaxadas (Dia, Lista de contabilização global das equipas não críticas das repetições das actividades que ao dia Dia ainda não começaram).

Exemplo:

relaxadas (6, [relaxada (0,0), relaxada (1,3), relaxada (2,0), relaxada (3,0), relaxada (4,1)]).

Ou seja, das actividades e suas repetições que começam depois do dia 6, vão ainda ser feitas por todo, 3 tarefas 1 e 1 tarefa 4, não necessariamente com equipas críticas, ou seja com qualquer tipo de equipa.

Esta informação é gerada para todos os dias do período em estudo.

6.6 - INFERÊNCIA DE CONHECIMENTO PARA UTILIZAÇÃO NO PROBLEMA DE PROGRAMAÇÃO LINEAR ASSOCIADO

O conhecimento deduzido é necessário para o cálculo das heurísticas associadas aos estados do espaço de estados. Este conhecimento é utilizado para a resolução do problema de programação linear associado ao cálculo das heurísticas. Por exemplo, para a particularização do cálculo da heurística de um determinado estado, é necessário o conhecimento que iremos descrever, o conhecimento deduzido em 6.5 e ainda conhecimento particular do estado, como se descreve no Capítulo 9 que trata do problema da heurística adoptada.

Para a dedução do conhecimento posterior, precisa-se desde logo do termo:

```
lista_trabalhadores (Ltrab).
```

em que Ltrab é a lista dos tipos de tabalhadores que vamos ter no nosso caso, e de

```
trabalhador_tarefas (Trab, Ltarefas).
```

que nos dá para o tipo de trabalhadores Trab, a lista de tarefas em que intervem.

Por exemplo se tivermos:

```
tarefa (0, [0*trab0*0]).
tarefa (1, [2*trab1*3, 3*trab1*1]).
tarefa (2, [2*trab2*2]).
tarefa (3, [1*trab3*2, 2*trab3*3*1]).
tarefa (4, [1*trab4*2, 2*trab4*1]).
tarefa (5, [3*trab1*4, 4*trab1*2, 5*trab1*1]).
```

teremos como

Ltrab = [trab1, trab2, trab3, trab4].

pois o trab0 é o tipo fictício de trabalhador associado a uma tarefa fictícia introduzida para facilitar a modelização dos grafos associados às actividades.

E teremos por exemplo:

trabalhador_tarefas (tab1, [1, 5]).

pois o tipo de trabalhador tab1 intervem nas tarefas 1 e 5.

Outro conhecimento que vamos precisar é do número de equipas associado a uma tarefa, ou seja a modularidade associada à realização de uma tarefa.

n_equipas (Tarefa, Cont).

por exemplo

n_equipas (5, 3).

pois a tarefa 5 pode ser realizada alternativamente por 3 equipas.

Inicia-se agora propriamente a dedução do conhecimento para resolução do problema de programação linear associado ao cálculo da heurística de um estado.

Este conhecimento é associado a cada tipo de trabalhador. O conhecimento deduzido é baseado nas tarefas que cada tipo de trabalhador realiza.

Deduz-se assim, associado a cada tipo de trabalhador, um vector de custos, que é o número de trabalhadores de cada equipa de todas as tarefas que o tipo de trabalhador realiza,

custos (Trab, C).

que para o tipo de trabalhador Trab, dá o vector de custos C.

Por exemplo para trab1, tem-se

custos (trab1, [2, 3, 3, 4, 5]).

em que 2, 3 representam o número de trabalhadores das equipas da tarefa 1, e 3, 4 e 5 o número de trabalhadores das equipas da tarefa 5.

Para o caso de trab2, teriamos

custos (trab2, [2]).

pois trab2 intervem apenas na tarefa 2, apenas com 1 equipa de 2 trabalhadores tipo trab2.

Deduz-se o número de variáveis associadas a cada tipo de trabalhador, que é o total de equipas de todas as tarefas associadas a esse tipo de trabalhador,

n_variáveis (Trab, Nvar).

No nosso caso exemplificativo,

n_variáveis (trab1, 5).

pois se tem 2 equipas de uma tarefa associada a trab1 e 3 equipas da outra tarefa associada a trab1.

Para a resolução da programação linear, define-se um limite superior para o número de equipas, um vector de limite superior

upperbound (Trab, Upp).

que associa ao tipo de trabalhador Trab o limite superior de cada uma das variáveis a ele associadas, que foram expressas na

"Representação do Conhecimento Associado ao Problema de Calendarização" por

n_máximo_equipas (Nmax).

Supondo que se define

n_máximo_equipas (1000).

tem-se

upperbound (trab1, [1000,1000,1000,1000,1000]).

Outro conhecimento necessário, é o número de equações de restrição associado a cada tipo de trabalhador. Este número é o número de tarefas em que o tipo de trabalhador intervem, mais uma, como se verá no Capítulo 9. Assim,

n_equações_restrição (Trab, Restr).

dá para o tipo de trabalhador Trab, o número de equações de restrição Restr.

Por exemplo,

n_equações_restrição (trab1, 3).

Para cada equação de restrição está definido um símbolo para a desigualdade a ela associada, que é um sinal ≥, como se verá no Capítulo 9. A codificação da desigualdade ≥, é 2, e assim tem-se

irtype (Trab, Irty).

que associa a cada equação de restrição a desigualdade 2 que

codifica ≥. Para o mesmo caso,

irtype (trab1, [2, 2, 2]).

Finalmente, refere-se a construção da matriz A associada a cada tipo de trabalhador.

A matriz A é constituida pelo vector de custos e por uma linha por cada tarefa que utiliza o tipo de trabalhador, em que a posição referente às equipas dessa tarefa são preenchidas pelo número de dias que a equipa leva a prefazer a tarefa, sendo as outras posições preenchidas a "0",

matrix_a (Trab, Matriz_a).

Associa-se ao tipo de trabalhador Trab, a matriz A correspondente, Matriz_a.

Para o tipo de trabalhador trab1, teriamos por matriz A

$$\begin{bmatrix} 2 & 3 & 3 & 4 & 5 \\ 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 & 1 \end{bmatrix}$$

pelo que

matrix_a (trab1, [2, 3, 3, 4, 5, 3, 1,0, 0, 0, 0, 0, 4, 2, 1,]).

7. DESCRIÇÃO DA ÁRVORE DE ESTADOS

7. 1 - INTRODUÇÃO

Este capítulo está intimamente ligado à árvore de estados explicitada pelo algoritmo de pesquisa A*. Assim, é neste Capítulo que se descreve a Estrutura de Estado adoptada para a representação das situações ou configurações do problema, e é neste Capítulo que se explicam as Regras de Produção e Estratégia de Controle para geração dos estados sucessores de um estado seleccionado para expansão pelo algoritmo A*.

O módulo de "Regras de Produção e Estragégia de Controlo" está assim intimamente ligado ao módulo do algoritmo A*, e recebe fluxo de conhecimento da Base de Conhecimento, como esquematicamente se indica na Fig. 7.1.

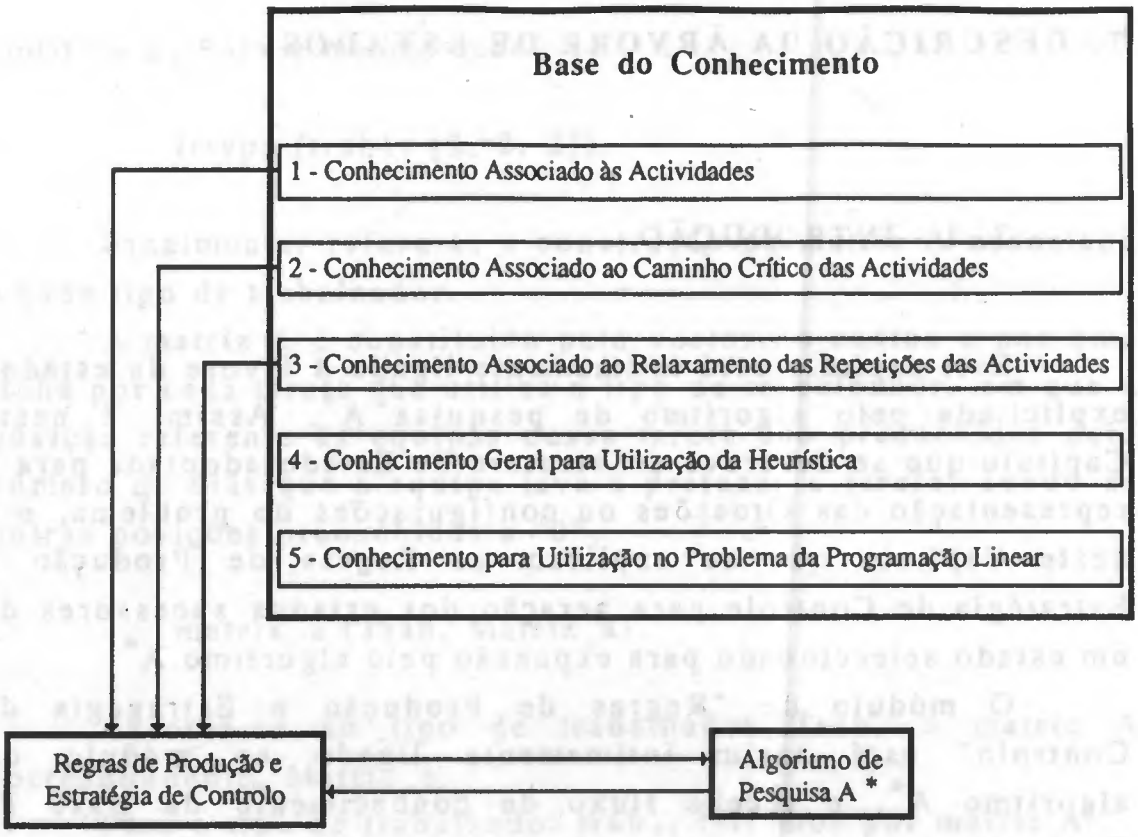


Fig. 7.1 - Fluxo de conhecimento do módulo de Regras de Produção e Estratégia de Controlo com outros módulos

No fim deste Capítulo apresenta-se um exemplo de geração dos estados sucessores de um estado especificado.

7. 2 - ESTRUTURA DE ESTADO

Um estado da árvore de estados fica completamente

caracterizado pelas seguintes variáveis:

- Dia - dia do calendário que caracteriza o estado.

- L1 - lista das tarefas já concluídas. De todas as repetições das actividades, os seus "arcs" que no dia Dia, já foram concluídas estão memorizadas na lista L1. Esta lista é constituída por elementos de tipo

  arc (A, B, C, D, E, F, G)

  em que F ≤ Dia.

- L2 - lista das tarefas ainda a fazer. Esta lista é constituída por elementos do tipo

  arc (A, B, C, D, E, F, G)

  em que F ≥ Dia.

- L3 - lista das tarefas ainda não começadas a fazer, das repetições das actividades que já começaram a ser feitas. Esta lista é constituída por elementos do tipo

  arc (A, B, C, D, E, F, G)

  em que F e G ainda estão indefinidos.

- L4 - lista do total de trabalhadores, por tipo de trabalhador. Esta lista representa por tipo de trabalhador, o máximo que já se precisou simultaneamente.

Esta lista é do tipo

[total (trab1, N1), total (trab2, N2) |
Resto da lista]

Para o tipo de trabalhador trab1, tem-se um total de N1, para o tipo de trabalhador trab2, tem-se um total de N2 e assim sucessivamente.

- L5 - lista de trabalhadores livres por tipo de trabalhador. Esta lista representa os trabalhadores que estão disponíveis no dia Dia.
A lista é do tipo

[livres (trab1, L1), livres (trab2, L2)|
Resto da lista]

Para o tipo de trabalhador trab1, tem-se L1 deles livres, para o tipo de trabalhador trab2, tem-se livres L2 e assim sucessivamente.

Um estado é caracterizado por:

Dia ^ L1 ^ L2 ^ L3 ^ L4 ^ L5

Com esta informação um estado fica definido sem ambiguidades.

É a associação do Dia, L1, L2, L3, L4 e L5 com os significados que foram explicados que constitue a Estrutura de Estado adoptada.

7. 3 - REGRAS DE PRODUÇÃO E ESTRATÉGIA DE CONTROLE

Um estado no dia Dia tem como sucessores todos os possíveis estados do dia Dia + 1.

Para fazer a geração de todos os estados possíveis que se podem gerar do estado no dia Dia, refere-se a seguintes ordem de operações:

1º Vai-se verificar quais as repetições das actividades que começam no dia Dia, transformam-se os seus arcos associados em arc (A,B,C,D,E,F,G), com F e G ainda indefinidos, e carregam-se na lista das tarefas ainda não começadas, L3.

2º Vai-se descarregar da lista de tarefas a serem executadas L2, as tarefas que acabam nesse dia,

arc (A,B,C,D,E,F,G)

em que F=Dia, e carregam-se na lista das tarefas já feitas L1.
Libertam-se os trabalhadores associados a essas tarefas e carregam-se na lista dos trabalhadores livres L5.

3º Procurar na lista dos termos "arcs" ainda não começados, L3, os que pertencem às repetições das actividades com

rapidez (Act, Nact, Rap)

Rap=1, ou seja os que pertencem a repetições de actividades não relaxadas, e não tenham um ES deslocado do nó inicial que seja inferior ou igual a Dia, e formar uma lista, L31 com estes termos "arcs".

Os termos "arcs" da lista L31 que pertençam ao caminho crítico são descarregados em L31 e L3 e carregados em L2, com as marcas F e G já determinadas.

F - Dia em que o "arc" está concluído.

G - Número de trabalhadores assignados ao "arc", que neste caso é o número de trabalhdores da equipa mais rápida.

Estes "arcs" são começados a fazer obrigatoriamente porque pertencem ao caminho crítico de repetições não relaxadas de actividades, e se não se começassem a fazer, toda a repetição da actividade a que pertencem ficaria atrasada.

Faz-se também a actualização das listas do total de trabalhadores L4 e dos trabalhadores livres L5. Esta actualização consiste em procurar preencher as necessidades com os trabalhadores livres que existem, e caso não cheguem, aumentar o número total de trabalhadores de modo a satisfazer as necessidades.

Da nova lista L31, os termos arc(A,B,C,D,E,F,G) cujos arcos associados têm início_mais_tarde_arco no dia Dia, são obrigatoriamente passados para a lista dos "arcs" a fazer L2 com as marcas F e G já determinadas, são descarregados nas listas L31 e L3 e faz-se a actualização das listas do total de trabalhadores L4 e da lista dos trabalhadores livres L5.

A lista L31 é agora constituída pelos termos "arcs" que podem ser ou não começados a fazer no dia Dia, não havendo obrigatoriedade em relação ao seu começo, embora estando em condições de poderem começar a ser feitos.

4º Procurar na lista dos termos "arcs" ainda não começados,

L3, os que pertencem às repetições das actividades com

rapidez (Act, Nact, Rap)

Rap=0, ou seja repetições relaxadas das actividades, e que tiverem um Es deslocado que seja inferior ou igual a Dia, e constituir com eles uma lista L32.

Desta lista, os termos "arcs" que tiverem início_+tarde_relaxado igual a Dia, são carregados na lista das tarefas a fazer L2 com as marcas F e G já determinadas, e descarregados nas listas L32 e L3 e faz-se a actualização das listas do total de trabalhadores L4 e da lista dos trabalhadores livres L5. Estes "arcs" são começados a fazer obrigatoriamente porque apesar de pertencerem a repetições relaxadas de actividades, têm o seu início_+tarde_relaxado igual a Dia, ou seja, se não se começarem a fazer neste Dia, as repetições relaxadas das actividades a que pertencem não ficarão terminadas dentro do tempo máximo permitido para a sua conclusão.
Dos restantes termos "arcs" de L32, procuram-se os termos "arcs" que tenham os seus precedentes já feitos, e juntam-se à lista L31.
Fez-se assim a triagem das tarefas que têm de ser obrigatoriamente começadas a fazer, das tarefas que podem ou não ser começadas nesse dia, que estão contidas em L31. São estas tarefas não obrigatórias que dão a explosão de ramificações do estado.

NOTA: o Es deslocado de um nó, de uma repetição de uma actividade, é o es, início mais cedo possível, "earliest start time", desse nó da actividade geradora, mais a deslocação temporal para o dia do começo da repetição da actividade em questão. Considere-se por exemplo a repetição número 2 da actividade número 1 e pretende-se saber o Es deslocado do

nodo 3. Da Representação do Conhecimento Associado ao Problema de Calendarização tem-se

dia_começo (1,2,Com)

e na Base do Conhecimento tem-se do conhecimento associado às activiades

es (1,3,Es)

donde o Es deslocado do nodo 3 ser:

Es deslocado = Es + Com

5º Neste ponto já se descarregaram as tarefas já executadas de L2 e se passaram para L1, e já se carregou a lista L2 com todas as tarefas que obrigatoriamente têm de começar a ser feitas no dia Dia.

Vai-se agora tratar das tarefas que estando em condições de poderem começar a ser feitas, apresentam flexibilidade com dois graus de liberdade. Os graus de liberdade que apresentam são o dia de começo, e o tipo de equipa escolhida para as realizar. É a configuração destes 2 graus de liberdade de todas as tarefas da actual lista L31 que dão a ramificação do estado nos seus estados sucessores. O número de estados sucessores que se vai ter é o número de combinações que as tarefas de L31 podem gerar.

Para melhor exemplificar, vamos supor que a lista L31 era dada pela lista:

L31 = [arc (A,B,C,D,E,_,_),arc (M,N,O,P,Q,_,_)]

As combinações possíveis são,

1 - [ ]

2 - [arc (A,B,C,D,E,_,_)]

3 - [arc (M,N,O,P,Q,_,_)]

4 - [arc (A,B,C,D,E,_,_),arc (M,N,O,P,Q,_,_)]

5 - [arc (M,N,O,P,Q,_,_),arc (A,B,C,D,E_, _)]

As combinações 4 e 5 são redundantes, pelo que ficamos apenas com as 4 primeiras opções para a realização das tarefas contidas na lista L31:

1 - Não se começa a fazer nenhuma das tarefas sem obrigatoriedade de começo no dia Dia, que constituem a lista L31.

2 - Começa-se a fazer o arc(A,B,C,D,E,_,_)

3 - Começa-se a fazer o arc(M,N,O,P,Q,_,_)

4 - Começam-se a fazer os 2 termos "arcs" que se podiam começar a fazer embora sem obrigatoriedade

arc (A,B,C,D,E,_,_) e arc (M,N,O,P,Q,_,_)

Vai-se agora entrar com o grau de flexibilidade das modularidades das equipas.

Vejamos então quais as possibilidades de realização, por exemplo para o

arc (A,B,C,D,E,_,_)

Tome-se o valor de

es (A,D,Es)

deslocado de

dia_começo (A,B,Com)

Se a repetição B da actividade A não for relaxada, tem-se como data limite para a conclusão da tarefa

Limite = Es + Com

Se a repetição B da actividade A for relaxada, tem-se

tempo_max (A,B,Temax)

e

dura_min_activ (A,Temin)

Delta = Temax - Temin

e tem-se como data limite para concluir a tarefa

Limite = Delta + Es + Com

Vai-se ver quais são as equipas que podem realizar a tarefa, de modo a que a começar no dia Dia, ela esteja concluída o mais tardar no dia Limite. Todas as equipas que satisfizerem esta restrição temporal são alternativas válidas.

Vamos supor que haviam 2 equipas que satisfaziam a restrição anterior, terminando a

1ª equipa no dia F1 ≤ Limite, com G1 trabalhadores

2ª equipa no dia F2 ≤ Limite, com G2 trabalhadores

então

arc (A,B,C,D,E,_,_) ⟨ arc (A,B,C,D,E,F1,G1) / arc (A,B,C,D,E,F2,G2)

Tem-se agora as novas combinações possíveis:

1 - [ ]

2A - [arc (A,B,C,D,E,F1,G1)]

2B - [arc (A,B,C,D,E,F2,G2)]

3 - [arc (M,N,O,P,Q,_,_)]

4A - [arc (A,B,C,D,E,F1,G1),arc (M,N,O,P,Q,_,_)]

4B - [arc (A,B,C,D,E,F2,G2),arc (M,N,O,P,Q,_,_)]

Supondo que se tinham também 2 possibilidades de

realização para o

arc (M,N,O,P,Q,_,_)

arc (M,N,O,P,Q,_,_) ⟨ arc (M,N,O,P,Q,R1,S1)
arc (M,N,O,P,Q,R2,S2)

Teriamos como total de possibilidades

1 - [ ]

2A - [arc (A,B,C,D,E,F1,G1)]

2B - [arc (A,B,C,D,E,F2,G2)]

3A - [arc (M,N,O,P,Q,R1,S1)]

3B - [arc (M,N,O,P,Q,R2,S2)]

4A -[arc(A,B,C,D,E,F1,G1),arc(M,N,O,P,Q,R1,S1)]

4B -[arc(A,B,C,D,E,F1,G1),arc(M,N,O,P,Q,R2,S2)]

4C-[arc(A,B,C,D,E,F2,G2),arc (M,N,O,P,Q,R1,S1)]

4D-[arc(A,B,C,D,E,F2,G2),arc (M,N,O,P,Q,R2,S2)]

Cada uma destas opções vai gerar um estado sucessor. É evidente que se tem de fazer para cada uma destas opções o descarregamento da opção na lista L3 e o carregamento na

lista L2, e a actualização das listas do total de trabalhadores e de trabalhadores livres.
Voltando a tratar do problema de redundâncias, já tinhamos visto atrás que as 2 opções

[arc (A,B,C,D,E,_,_),arc (M,N,O,P,Q,_,_)]

e

[arc (M,N,O,P,Q,_,_),arc (A,B,C,D,E,_,_)]

são redundantes e como tal a redundância era eliminada, considerando-se apenas uma das opções:

[arc (A,B,C,D,E,_,_),arc (M,N,O,P,Q,_,_)]

Há outra situação de redundância que vai ser tratada que é por exemplo a situação das opções:

[arc (A,B,C,D,E,F1,G1),arc (M,N,O,P,Q,R1,S1),
arc (A,B1,C,D,E,F2,G2)]

[arc (A,B,C,D,E,F2,G2),arc (M,N,O,P,Q,R1,S1),
arc (A,B1,C,D,E,F1,G1)]

com F1 ≠ F2 e consequentemente G1 ≠ G2.
Vê-se facilmente que do ponto de vista de recursos empregues e sua ocupação, as 2 opções são iguais, pelo que são redundantes. A redundância é eliminada, considerando-se apenas uma opção:

[arc (A,B,C,D,E,F1,G1),arc (M,N,O,P,Q,R1,S1),
arc (A,B1,C,D,E,F2,G2)]

EXEMPLO

Considere-se a actividade número 1, cujo grafo está representado na Fig. 7.2, com 2 repetições não relaxadas ambas começadas no dia 0,

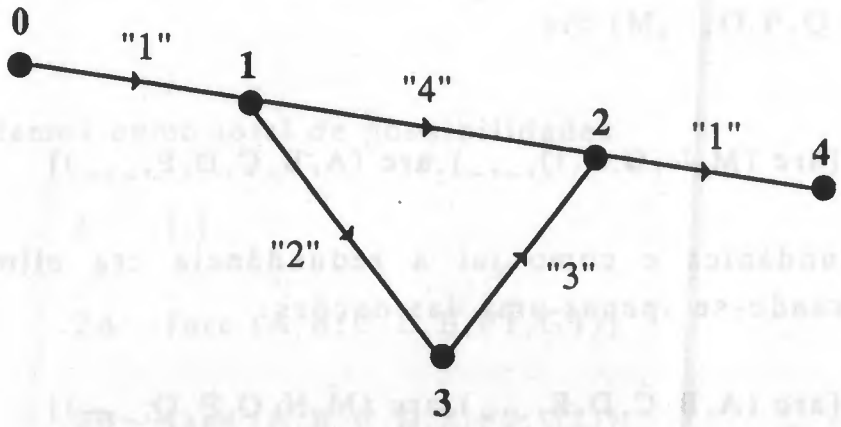


Fig. 7.2 - Grafo de actividade número 1

e uma modularidade de equipas:

```
tarefa (1, [2*trab1*3,3*trab1*1]).
tarefa (2, [2*trab2*2]).
tarefa (3, [1*trab3*2,2*trab3*1]).
tarefa (4, [1*trab4*2,2*trab4*1]).
```

Na Base de Conhecimento estão os termos referentes ao conhecimento do início mais cedo "earliest start time", "es", dos nodos de actividade:

```
es (1,0,0)
es (1,1,1)
es (1,2,4)
es (1,3,3)
es (1,4,5)
```

assim como se tem também conhecimento relativo ao caminho crítico associado à actividade.

OBJECTIVO

Pretendem-se os estados sucessores de

```
ESTADO - 1^[ ]^[arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

         ^[arc (1,1,1,2,4,_,_),arc (1,2,1,2,4,_,_),
           arc (1,1,1,3,2,_,_),arc (1,2,1,3,2,_,_),
           arc (1,1,3,2,3,_,_),arc (1,2,3,2,3,_,_),
           arc (1,1,2,4,1,_,_),arc (1,2,2,4,1,_,_)]

         ^[total (trab1,6),total (trab2,0), (trab3,0),
           total (trab4,0)]
         ^[livres (trab1,0),livres (trab2,0),
           livres (trab3,0),livres (trab4,0)]
```

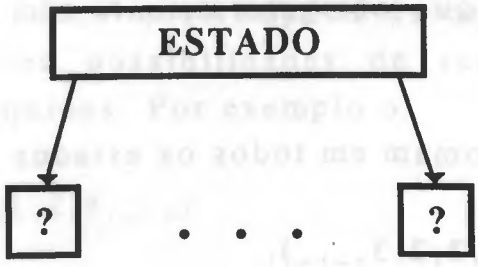


Fig. 7.3 - Geração dos sucessores do nodo ESTADO

Geração dos estados sucessores de Estado

Os estados sucessores vão todos ter em comum

Dia = 2

L1 = [arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

Os estados sucessores de um estado no dia 1 serão como é evidente estados associados ao dia 2, e os "arcs" que terminam no dia 1 passam para a lista das tarefas já feitas L1, em todos os estados sucessores.

L2 tem em comum em todos os estados sucessores os termos:

arc (1,1,1,3,2,3,1),
arc (1,2,1,3,2,3,1)

que são "arcs" que obrigatoriamente têm de começar a ser feitos pois pertencem ao caminho crítico. Em todos os estados sucessores, a lista das tarefas que estão a ser feitas L2, têm em comum os "arcs" que obrigatoriamente têm de começar a ser feitos no dia 2.

L3 tem em comum em todos os estados sucessores:

arc (1,1,3,2,3,_,_),
arc (1,2,3,2,3,_,_),
arc (1,1,2,4,1,_,_),
arc (1,2,2,4,1,_,_),

que são "arcs" que ainda não estão em condições de começarem a ser feitos.

A diferenciação entre os estados vai estar nas combinações possíveis de L31, constituído pelos "arcs" que podem começar a ser feitos embora sem obrigatoriedade.

L31 = [arc (1,1,1,2,4,_,_),arc (1,2,1,2,4,_,_)],

Os "arcs" que pertencem a L31 não pertencem ao caminho crítico nem têm o seu início mais tarde possível igual ao dia 2, pelo que podem ou não começar a ser feitos no dia 2. As possibilidades que se oferecem de diversificação de opções a tomar são dadas pelas combinações possíveis que se podem formar com as tarefas da lista L31:

1 - [ ]

2 - [arc (1,1,1,2,4,_,_)]

3 - [arc (1,2,1,2,4,_,_)]

4 - [arc (1,1,1,2,4,_,_),arc (1,1,1,2,4,_,_)]

Cada um dos "arcs" que compõem as combinações 1,2,3 e 4 vão ter diferentes possibilidades de realização devido à modularidade das equipas. Por exemplo o

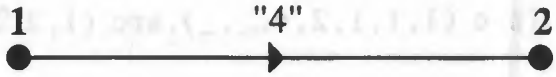
arc (1,1,1,2,4,_,_)

pode ter a sua tarefa assignada número 4 realizada por:

1*trab4*2
2*trab4*1

Se for realizada por 1 trabalhador trab4 leva 2 dias a ser feita e se for realizada por 2 trabalhadores trab4 leva 1 dia a ser feita. Verifiquemos se as duas possibilidades são

compatíveis com as restrições temporais de repetição número 1 da actividade número 1. O arco que se está a considerar é



e de es(1,1,Es1) e de es(1,2,Es2) têm-se que Es1 = 1 e Es2 = 4. Donde se tem 4 - 1 = 3 dias para realizar o arco do nodo 1 para o nodo 2. Então ambas as equipas são compatíveis para a realização da tarefa 4, vindo o arc (1,1,1,2,4,_,_) desdobrar-se em

arc (1,1,1,2,4,3,1) e
arc (1,1,1,2,4,2,2)

O desdobramento de todos os "arcs" dá o total de possibilidades para a geração dos estados sucessores:

1 - [ ]

2A - [arc (1,1,1,2,4,3,1)]

2B - [arc (1,1,1,2,4,2,2)]

3A - [arc (1,2,1,2,4,3,1)]

3B - [arc (1,2,1,2,4,2,2)]

4A - [arc (1,1,1,2,4,3,1),arc (1,2,1,2,4,3,1)]

4B - [arc (1,1,1,2,4,3,1),arc (1,2,1,2,4,2,2)]

4C - [arc (1,1,1,2,4,2,2),arc (1,2,1,2,4,3,1)]

4D - [arc (1,1,1,2,4,2,2),arc (1,2,1,2,4,2,2)]

Os estados 2A e 3A são redundantes do ponto de vista de ocupação de recursos, assim como os estados 2B e 3B e os estados 4B e 4C.

Portanto os estados sucessores são

```
S1→ 2^ [arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

     ^ [arc (1,1,1,3,2,3,2),arc (1,2,1,3,2,3,2)]

     ^ [arc (1,1,1,2,4,_,_),arc (1,2,1,2,4,_,_),
        arc (1,1,3,2,3,_,_),arc (1,2,3,2,3,_,_),
        arc (1,1,2,4,1,_,_),arc (1,2,2,4,1,_,_)]
     ^ [total (trab1,6),total (trab2,4),
        total (trab3,0),total (trab4,0)]

     ^ [livres (trab1,6),livres (trab2,0),
        livres(trab3,0),livres(trab4,0)]
```

O estado S1 é o estado correspondente à opção 1. Tem o dia 2, a lista L1 já determinada, a lista L2 apenas com os "arcs" obrigatoriamente começados a fazer, e na lista L3 todos os outros "arcs", e as respectivas listas L4 e L5 actualizadas.

```
S2→ 2^ [arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

     ^ [arc (1,1,1,3,2,3,2),arc (1,2,1,3,2,3,2),
        arc (1,1,1,2,4,3,1)]

     ^ [arc (1,2,1,2,4,_,_),arc (1,1,3,2,3,_,_),
        arc (1,2,3,2,3,_,_),arc (1,1,2,4,1,_,_),
        arc (1,2,2,4,1,_,_)]

     ^ [total (trab1,6),total (trab2,4),
        total (trab3,0),total (trab4,1)]
```

```
        ^ [livres (trab1,6), livres (trab2,0),
           livres (trab3,0),livres (trab4,0)]
```

O estado S2 é o estado correspondente à opção 2A.

```
S3→ 2^ [arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

        ^ [arc (1,1,1,3,2,3,2),arc (1,2,1,3,2,3,2),
           arc (1,1,1,2,4,2,2)]

        ^ [arc (1,2,1,2,4,_,_),arc (1,1,3,2,3,_,_),
           arc (1,2,3,2,3,_,_),arc (1,1,2,4,1,_,_),
           arc (1,2,2,4,1,_,_)]

        ^ [total (trab1,6),total (trab2,4),
           total (trab3,0),total (trab4,2)]

        ^ [livres (trab1,6),livres (trab2,0),
           livres (trab3,0),livres (trab4,0)]
```

O estado S3 corresponde à opção 2B.

```
S4→ 2^ [arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

        ^ [arc (1,1,1,3,2,3,2),arc (1,2,1,3,2,3,2),
           arc (1,1,1,2,4,3,1),arc (1,2,1,2,4,3,1)]

        ^ [arc (1,1,3,2,3,_,_),arc (1,2,3,2,3,_,_),
           arc (1,1,2,4,1,_,_),arc (1,2,2,4,1,_,_)]

        ^ [total (trab1,6),total (trab2,4),
           total (trab3,0),total (trab4,0)]

        ^ [livres (trab1,6),livres (trab2,0),
           livres (trab3,0),livres (trab4,0)]
```

O estado S4 corresponde à opção 4A.

```
S5→ 2^ [arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

     ^ [arc (1,1,1,3,2,3,2),arc (1,2,1,3,2,3,2),
        arc (1,1,1,2,4,3,1),arc (1,2,1,2,4,2,2)]

     ^ [arc (1,1,3,2,3,_,_),arc (1,2,3,2,3,_,_),
        arc (1,1,2,4,1,_,_),arc (1,2,2,4,1,_,_)]

     ^ [total (trab1,6),total (trab2,4),
        total (trab3,0),total (trab4,3)]

     ^ [livres (trab1,6),livres (trab2,0),
        livres (trab3,0),livres (trab4,0)]
```

O estado S5 é o estado correspondente à opção 4B.

```
S6→ 2^ [arc (1,1,0,1,1,1,3),arc (1,2,0,1,1,1,3)]

     ^ [arc (1,1,1,3,2,3,2),arc (1,2,1,3,2,3,2),
        arc (1,1,1,2,4,2,2),arc (1,2,1,2,4,2,2)]

     ^ [arc (1,1,3,2,3,_,_),arc (1,2,3,2,3,_,_),
        arc (1,1,2,4,1,_,_),arc (1,2,2,4,1,_,_)]

     ^ [total (trab1,6),total (trab2,4),
        total (trab3,0),total (trab4,4)]

     ^ [livres (trab1,6),livres (trab2,0),
        livres (trab3,0),livres (trab4,0)]
```

O estado S6 é o estado correspondente à opção 4D.

Portanto tem-se

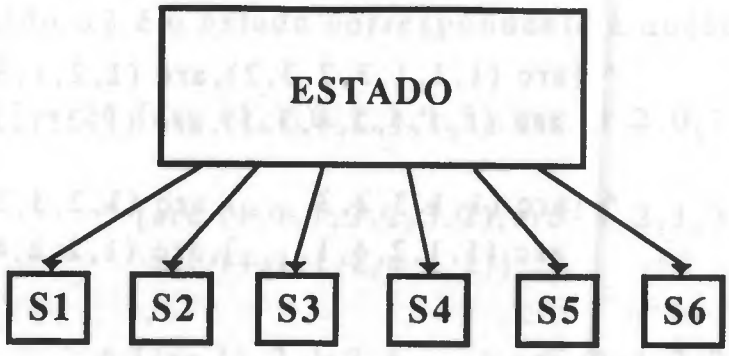


Fig. 7.4 - Estados sucessores do nodo ESTADO

8. CUSTO DE TRANSIÇÃO ENTRE ESTADOS

8. 1 - INTRODUÇÃO

Este Capítulo descreve como é calculado o custo de transição entre 2 estados do problema de calendarização. O módulo do Custo de Transição entre Estados está ligado com o algoritmo de pesquisa A*. Este ao seleccionar um estado para expansão, e ao expandi-lo tem de ter acesso ao custo de transição do estado seleccionado para cada um dos seus sucessores. É este custo de transição do estado seleccionado para um seu sucessor que este Capítulo descreve.

As ligações do módulo de Custo de Transição entre Estados com os outros módulos do "sistema de calendarização" são esquematicamente representadas na Fig. 8.1.

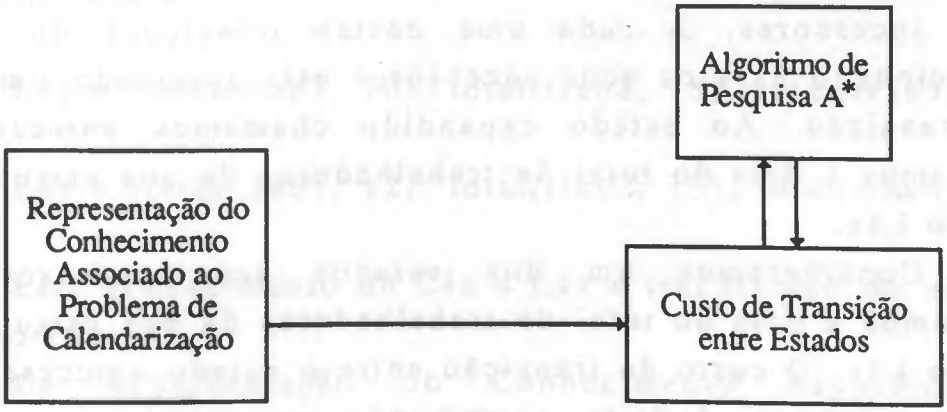


Fig. 8.1 - Ligações do módulo de custo de transição entre estados com os outros módulos

A ligação do módulo de Custo de Transição entre Estados ao módulo de Representação de Conhecimento Associado ao Problema de Calendarização advem da necessidade de se conhecerem os salários dos diferentes tipos de trabalhadores para o cálculo do custo de transição entre estados. O módulo do algoritmo A^* transfere o conhecimento das listas L4 da estrutura de estado, dos 2 estados cujo custo de transição se quer calcular. O módulo de Custo de Transição entre Estados, transfere para o módulo do algoritmo A^* o custo de transição entre os referidos estados.

8. 2 - CUSTO DE TRANSIÇÃO ENTRE ESTADOS

O custo de transição entre 2 estados C, é calculado por comparação das listas L4 dos dois estados.

Suponha-se que um estado é seleccionado para expansão pelo algoritmo A^*. Ao ser expandido o estado gera todos os seus sucessores. A cada uma destas transições do estado seleccionado para os seus sucessores está associado um custo de transição. Ao estado expandido chamamos antecessor e chamamos à lista do total de trabalhadores da sua estrutura de estado L4a.

Consideremos um dos estados seus sucessores, e chamemos à lista do total de trabalhadores da sua estrutura de estado L4s. O custo de transição entre o estado antecessor e o estado sucessor é feito comparando os custos em salários assignados a L4a e L4s. Este custo C, é feito comparando o total de trabalhadores, por cada tipo de trabalhador, e multiplicando o diferencial pelo custo associado ao tipo de

trabalhador. O somatório destes diferenciais para todos os tipos de trabalhadores dá o custo de transição entre os 2 estados.

L4a - [total (trab1, N1a), total (trab2, N2a),...,
total (trabn, Nna)]

L4s - [total (trab1, N1s), total (trab2, N2s),...,
total (trabn, Nns)]

salário (trabi, Si)

$$C = \sum_{K=1}^{n} (Nks - Nka) * Sk$$

Exemplo:

Considerem-se 2 estados, um antecessor e um sucessor cujas listas L4 da estrutura de estado, L4a e L4s são respectivamente:

L4a = [total(trab1, 10), total(trab2, 15), total(trab3, 2)]

L4s = [total(trab1, 12), total(trab2, 15), total(trab3, 3)]

Este conhecimento de L4a e L4s é transferido do módulo do algoritmo A*.

Da Representação do Conhecimento Associado ao Problema de Calendarização tem-se:

salário(trab1, 10).
salário(trab2, 20).
salário(trab3, 30).

O custo de transição entre estados é assim,

$$C = \sum_{K=1}^{3} (Nks - Nka) * Sk$$

ou seja

$$C = (12-10) * 10 + (15-15) * 20 + (3-2) * 30$$

$$C = 50$$

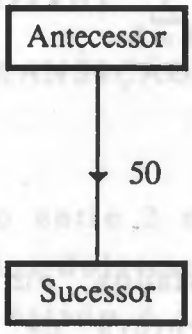


Fig. 8.2 - Custo de transição entre 2 estados, o antecessor e o seu sucessor

Este custo de transição é o retorno do módulo de Custo de Transição entre Estados para o módulo do algoritmo A*.

9. HEURÍSTICA ADOPTADA

9. 1 - INTRODUÇÃO

Neste Capítulo faz-se a descrição da heurística adoptada para pesquisa pelo algoritmo A^* no espaço de estados definido pelo problema de calendarização.

Em 9.2 faz-se a descrição da conceptualização da heurística, cuja originalidade é uma das contribuições deste trabalho para o estudo do sistema de calendarização.

Em 9.3 faz-se a descrição do fluxo de conheciemnto da Base de Conhecimento para o módulo da heurística.

Em 9.4 faz-se a descrição da inferência de conhecimento pontual também necessário para o cálculo da heurística associada a um estado.

Em 9.5 descreve-se a determinação da heurística adoptada associada a um estado, referindo-se a ligação do módulo da heurística à biblioteca IMSL para determinação do problema de programação linear associado à heurística.

O esquema de ligações do módulo Heurística com os outros módulos do "sistema de calendarização" é representado na Fig. 9.1.

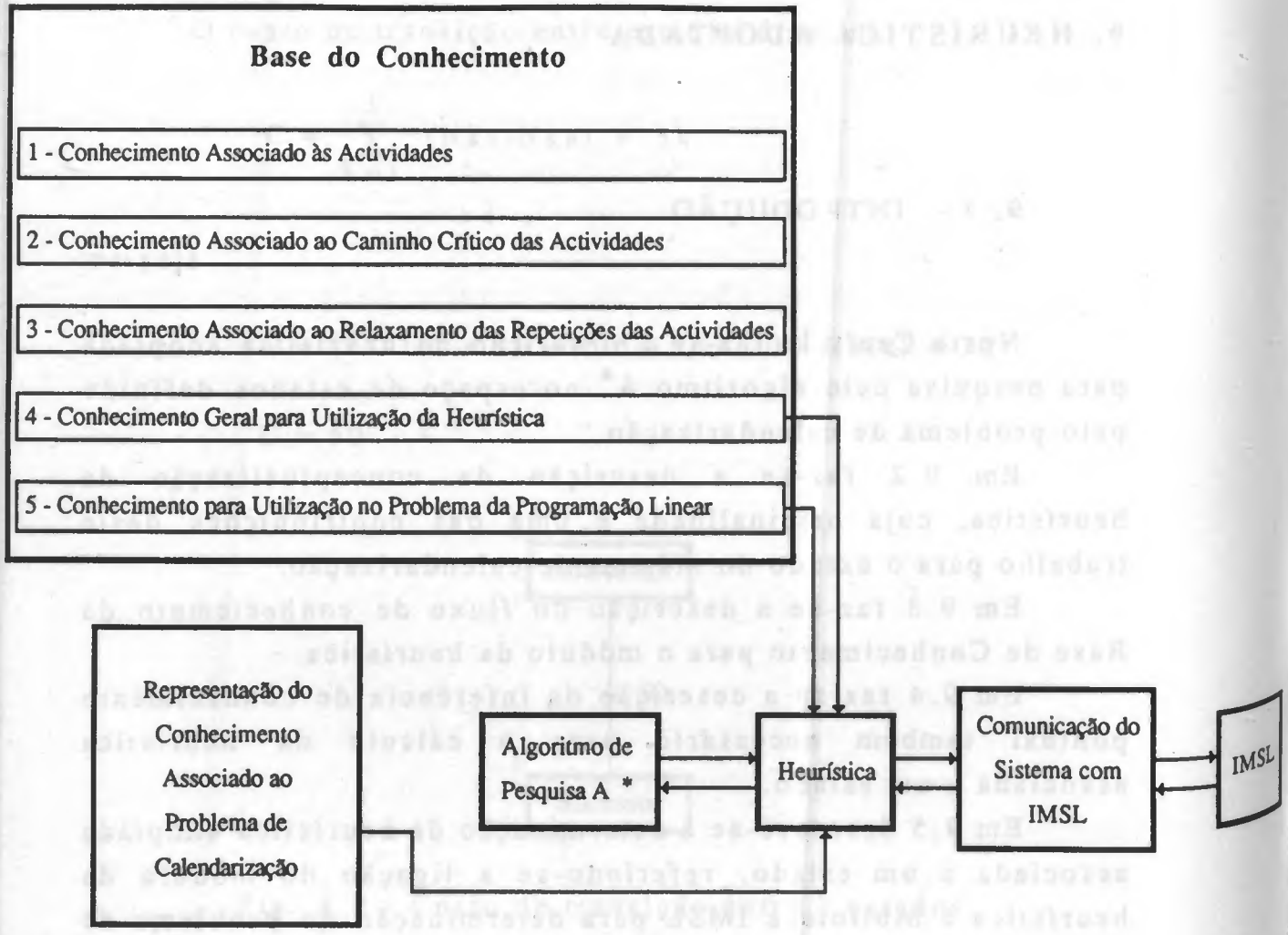


Fig. 9.1 - Fluxo de conhecimento do módulo Heurística com os outros módulos

9. 2 - CONCEPTUALIZAÇÃO DA HEURÍSTICA

A heurística é uma estimativa do custo que se vai gastar do estado em que nos encontramos no dia D, até ao dia final Dfinal, em que todas as actividades estão concluídas. É a

estimativa do incremento que se tem de fazer no contingente de todos os tipos de trabalhadores.

Vamos considerar por exemplo o incremento no contingente do tipo de trabalhadores trab1, que intervem nas tarefas 1 e 5:

```
tarefa (1, [2*trab1*3,3*trab1*1])
tarefa (5, [3*trab1*4,4*trab1*2,5*trab1*1])
```

O que se pretende é o número mínimo de trabalhadores tipo trab1, que estão distribuidos pelas equipas

x_1	→	equipa	2*trab1*3	da	tarefa 1
x_2	→	equipa	3*trab1*1	da	tarefa 1
y_1	→	equipa	3*trab1*4	da	tarefa 5
y_2	→	equipa	4*trab1*2	da	tarefa 5
y_3	→	equipa	5*trab1*1	da	tarefa 5

A função objectivo é então

$$\min 2x_1+3x_2+3y_1+4y_2+5y_3$$

s.a

$$x_1 \leq Nmax$$
$$x_2 \leq Nmax$$
$$y_1 \leq Nmax$$
$$y_2 \leq Nmax$$
$$y_3 \leq Nmax$$

sendo n_máximo_equipas (Nmax) definido na Representação do Conhecimento Associado ao Problema de Calendarização.

Associado ao estado que estamos a considerar, está o número total de trabalhadores trab1 que já se tem no

contingente, total(trab1,N1), pelo que temos mais uma equação de restrição

$$2x_1+3x_2+3y_1+4y_2+5y_3 \geq N1$$

Para as outras equações de restrição, vamos definir

T - tempo útil
Im (N) - Intensidade média de tarefas N
P (x) - Produtividade da equipa x

T = Dfinal - D

$$Im\ (1) = \frac{\text{Total de tarefas 1}}{T}$$

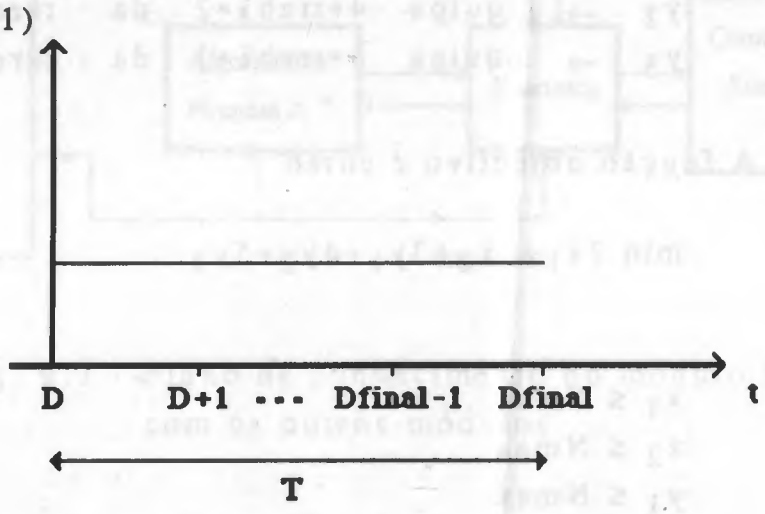


Fig. 9.2 - Intensidade média de tarefas número 1

$$Im(5) = \frac{\text{Total de tarefas } 5}{T}$$

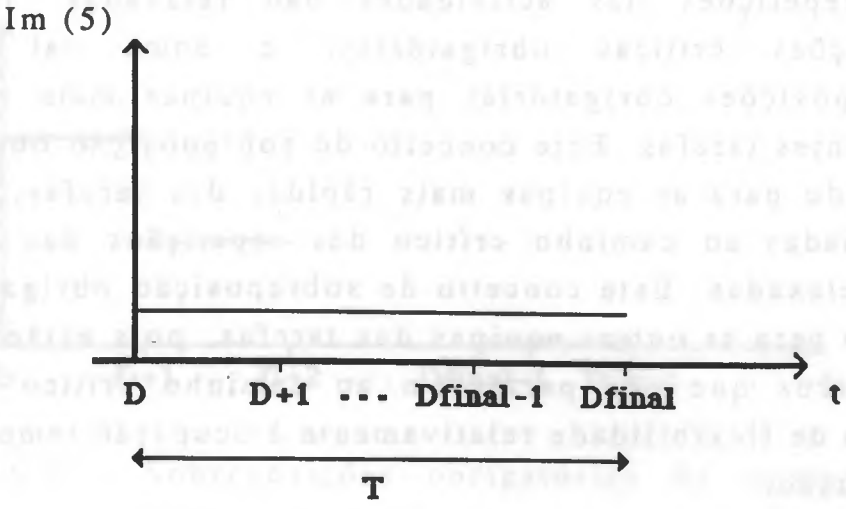


Fig. 9.3 - Intensidade média de tarefas número 5

$$P(x) = \frac{1}{Ndias}$$

com $x \rightarrow Ntrab*trab1*Ndias$

ou seja P(x) é a média de tarefas realizadas pela equipa x num dia.

As outras equações de restrição que se têm são

$$P(x_1)x_1+P(x_2)x_2+ \quad 0y_1+ \quad 0y_2+ \quad 0y_3 \geq Im(1)$$

$$0x_1+ \quad 0x_2+P(y_1)y_1+P(y_2)y_2+P(y_3)y_3 \geq Im(5)$$

Ao considerarmos um estado para o cálculo da sua heurística, consideramos o dia a ele associado D, e cada dia do calendário e portanto D, têm associadas as sobreposições que

ainda vão ser precisas para as diferentes tarefas, das suas equipas mais rápidas. Do exame prévio dos caminhos críticos das repetições das actividades não relaxadas, tiram-se as ocupações críticas obrigatórias, e como tal temos as sobreposições obrigatórias para as equipas mais rápidas das diferentes tarefas. Este conceito de sobreposição obrigatória só é válido para as equipas mais rápidas das tarefas, pois estão associadas ao caminho crítico das repetiçãos das actividades não relaxadas. Este conceito de sobreposição obrigatória não é válido para as outras equipas das tarefas, pois estão associadas aos arcos que não pertencem ao caminho crítico e portanto gozam de flexibilidade relativamente à ocupação temporal da sua realização.

Assim, voltando ao nosso exemplo, suponhamos que o diagrama de ocupação temporal da equipa mais rápida da tarefa 1, é descrito na Fig. 9.4.

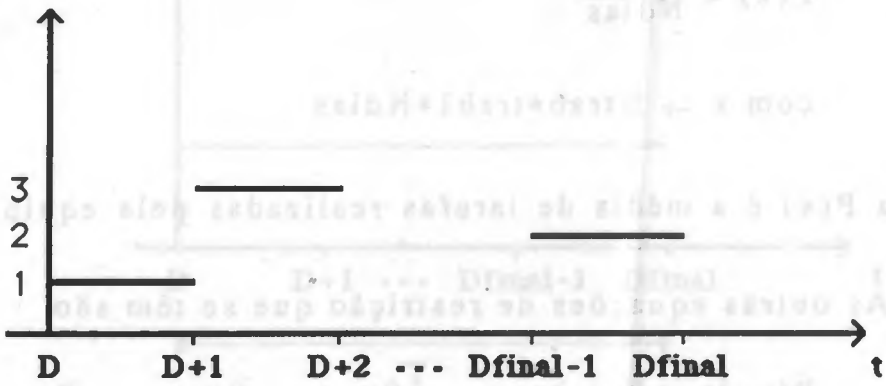


Fig. 9.4 - Sobreposições obrigatórias da equipa mais rápida da tarefa 1

Donde o máximo de sobreposições da equipa x_2 é de 3 unidades.

Para a tarefa 5 temos, por exemplo,

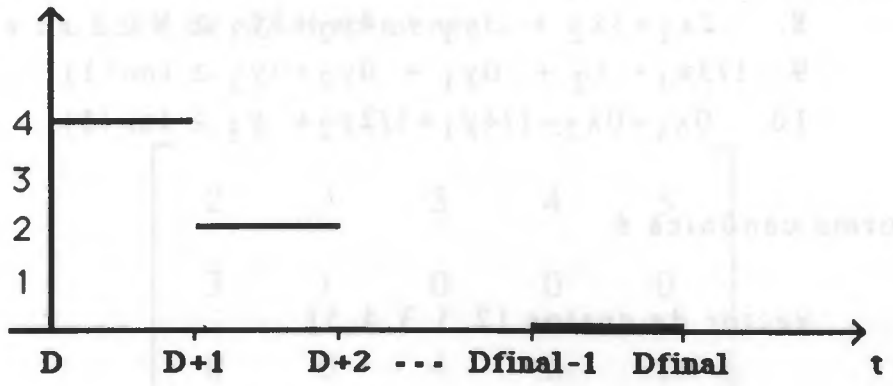


Fig. 9.5 - Sobreposições obrigatórias da equipa mais rápida da tarefa 5

Donde o máximo de sobreposições da equipa y_3 é de 4 unidades.

Estas sobreposições dão então um limite inferior "lowerbound" para este tipo de equipas. Teriamos então mais as condições

$$x_2 \geq 3$$
$$y_3 \geq 4$$

Neste ponto temos o problema definido para o nosso exemplo

$$\min 2x_1+3x_2+3y_1+4y_2+5y_3$$

s.a

1. $x_1 \leq Nmax$
2. $x_2 \leq Nmax$
3. $y_1 \leq Nmax$
4. $y_2 \leq Nmax$

5. $y_3 \leq Nmax$

6. $x_2 \geq 3$

7. $y_3 \geq 4$

8. $2x_1+3x_2 + 3y_1 + 4y_2+5y_3 \geq N1$

9. $1/3x_1+ x_2 + 0y_1 + 0y_2+0y_3 \geq Im\ (1)$

10. $0x_1+0x_2+1/4y_1+1/2y_2+ y_3 \geq Im\ (5)$

cuja forma canónica é

vector de custos [2,3,3,4,5]

vector de upperbound [Nmax,Nmax,Nmax,Nmax, Nmax] (1,2,3,4,5)

vector de lowerbound [0 3 0 0 4] (6,7)

Matriz A **Vector B**

$$\begin{bmatrix} 2 & 3 & 3 & 4 & 5 \\ 1/3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/4 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \geq \begin{bmatrix} N1 \\ Im(1) \\ Im(5) \end{bmatrix} \begin{matrix} \\ (8) \\ (9) \\ (10) \end{matrix}$$

As equações 1-5 impõem um limite superior para as equipas, as equações 6-7 traduzem o máximo de sobreposições obrigatórias das equipas mais rápidas das tarefas envolvidas com o tipo de trabalhador considerado, a equação 8 traduz o contingente de que já se dispõe e as equações 9 e 10 relacionam a produtividade das equipas de cada tarefa com o número médio de tarefas a realizar durante o tempo útil. As linhas da matriz A correspondentes às equações 9 e 10 são as inversas das linhas correspondentes da Matrix_a de matrix_a (trab1, Matrix_a) da

Base de Conhecimento, pois a Matrix_a está expressa em dias que as equipas levam a realizar uma tarefa, e as equações 9 e 10 estão expressas na produtividade das equipas. A correspondente Matrix_a na Base de Conhecimento é:

$$\begin{bmatrix} 2 & 3 & 3 & 4 & 5 \\ 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 & 1 \end{bmatrix}$$

9. 3 - DESCRIÇÃO DE CONHECIMENTO NA "BASE DE CONHECIMENTO" NECESSÁRIO PARA A HEURÍSTICA

No Capítulo 6 nos pontos 6.3 e 6.4 já descrevemos a base de conhecimento que vai ser utilizada no cálculo das heurísticas.

Assim em 6.3 já tinhamos descrito a integração na base de conhecimento de

sobreposição (D, Lsobre)

com Lsobre sendo a lista de sobreposições obrigatórias das equipas mais rápidas das tarefas,

críticas (D,Lcrit)

com Lcrit a lista do total das tarefas críticas das actividades

que ainda não começaram no dia D.

relaxadas (D, Lrelax)

com Lrelax a lista do total das tarefas não críticas das actividades ainda não começadas no dia D.

Em 6.4 já se descreveu a integração na Base de Conhecimento por tipo de trabalhador do número de variáveis, dos vectores de custos e de "upperbound", da matriz A e do vector de irtype que é o vector de codificação das desigualdades da matriz A com o vector B.

9. 4 - DESCRICÃO DE CONHECIMENTO PONTUAL NECESSÁRIO PARA A HEURÍSTICA

Associado ao estado temos a lista do total de trabalhadores

L4 = [total(trab1,N1),total(trab2,N2)...]

Este conhecimento é preciso para se compor o vector B para cada tipo de trabalhador. O 1º elemento do vector B é o número total de trabalhadores, que se vai buscar à lista L4 do estado.

D ^ L1 ^ L2 ^ L3 ^ L4 ^ L5

Para compor o resto do vector B, tem de se associar a lista L3 dos arcs ainda por fazer com a lista Lcrit de

críticas (D,Lcrit)

e a lista Lrelax de

relaxadas (D,Lrelax)

e contabilizar para cada tipo de trabalhador, e por tarefa em que ele intervem, os arcs de L3 assignados a essa tarefa com o que vem de Lcrit para essa tarefa, e o que vem de Lrelax para essa tarefa. O resultado dividido pelo tempo útil T dá um elemento do vector B. Assim se constroi o vector B para cada tipo de trabalhador.

A composição do vector de limite inferior para cada tipo de trabalhador faz-se utilizando a lista

sobreposições (D,Lsobre)

e tirando o valor de sobreposições críticas máximas de Lsobre por tarefa em que intervem o tipo de trbalhador considerado.

9. 5 - DETERMINAÇÃO DA HEURÍSTICA ASSOCIADA A UM ESTADO

O cálculo da heurística é feito, considerando a contribuição parcelar de cada tipo de trabalhador, sendo o somatório destas contribuições parcelares o valor global da heurística.

Para cada tipo de trabalhador já se têm todos os dados para a resolução do problema de programação linear associado, como se explicou nos items anteriores.

O conhecimento descrito em 9.3 está determinado de uma vez por todas e portanto integrado na base de conhecimento e o conhecimento descrito em 9.4 é determinado todas as vezes que se determina a heurística associada a um estado.

O problema de programação linear é resolvido pela rotina DDLPRS da IMSL ("International Mathematical Standard Library"). A rotina DDLPRS é baseada na rotina LPMGB de Richard Hanson - Hanson, Richard J. e J. A. Wisniewski (1979), A revised simplex code for LP problems using orthogonal decomposition - A User's Guide, Sandia Labs, Technical Report SAND78 - 2322.

O acesso do "sistema de calendarização" à rotina DDPLRS da biblioteca põe problemas que o módulo de comunicação do "sistema de calendarização" com a biblioteca resolve. Os problemas advêm do facto do "sistema de calendarização" estar escrito em PROLOG e a rotina DDPLRS estar escrita em FORTRAN. Assim, os dados que se têm de transferir têm de ser compatibilizados, de uma estrutura em lista em PROLOG, para dados em formato acessível para FORTRAN de acordo com as especificações da rotina DDPLRS.

Além disso, tem de se fazer uma atribuição dinâmica de memória para utilização da rotina DDPLRS, atribuição necessária pois o comprimento das estruturas da lista do PROLOG variam de acordo com o tipo de trabalhador cujo problema de programação linear associado se quer resolver.

Para o cálculo da heurística, a rotina DDLPRS é chamada tantas vezes quantos os tipos de trabalhadores que se têm.

Para cada tipo de trabalhador, o resultado da rotina DDLPRS, a função objectivo dá o total de trabalhadores desse tipo que ainda se vai precisar. A comparação da função objectivo com o total de trabalhadores desse tipo que se tem, dá-nos a contribuição parcelar desse tipo de trabalhador para a heurística, depois de pesado pelo salário do tipo de trabalhador em questão.

Portanto, no cálculo da heurística faz-se a suposição que a partir do dia associado ao estado todo o contingente de trabalhadores fica livre e disponível para satisfazer as exigências futuras. Assim a heurística é um minorante do real

valor do custo de transição do estado actual para o dia final do calendário em que todas as actividades estão realizadas.

10. ALGORITMO DE PESQUISA

10. 1 - INTRODUÇÃO

Neste Capítulo vão-se considerar as propriedades formais do algoritmo de pesquisa A^*.

Em 10.2 prova-se que o algoritmo A^* é completo e admissível, desde que as heurísticas utilizadas pelo algoritmo sejam admissíveis. Para a demonstração destas propriedades formais do algoritmo A^* seguiu-se Nilsson, 1971, Nilsson, 1980 e Pearl, 1984.

Em 10.3 prova-se que a heurística adoptada pelo "sistema de calendarização" e cuja conceptualização se descreve no capítulo 9 é admissível.

Sendo assim, a solução para a "Minimização da Afectação de Recursos num Problema de Calendarização" que o utilizador obtem ao usar o "sistema de calendarização" é garantidamente uma solução óptima.

10. 2 - PROPRIEDADES FORMAIS DE A^*

Vamos mostrar neste Capítulo que se algumas proriedades das heurísticas foram garantidas, o algoritmo A* garante que se encontra uma solução se ela existir e que essa solução é óptima.

Portanto se tivermos para cada nodo do grafo uma heurística computável h(n), se esta heurística tiver à priori algumas propriedades garantidas, encontra-se uma caminho de s para um nodo solução, e esse caminho é um óptimo.

Vamos usar alguma notação para simplificar a escrita dos raciocínios que vão ser explicados.

$SCS(n_i)$ - é o conjunto dos sucessores de n_i, que é finito porque estamos a considerar grafos localmente finitos. SCS quando aplicado a n_i gera todos os sucessores de n_i, $n_j \in SCS(n)$ significa portanto que n_j é um sucessor de n_i .

Γ - conjunto de todos os nodos terminais solução, $\gamma \in \Gamma$.

$c(n_i, n_j)$ - custo positivo do arco $n_i \rightarrow n_j$, sendo o custo dum caminho a soma do custo de todos os seus arcos.

$P_{n_i-n_j}$ - conjunto de todos os caminhos indo de n_i para n_j , sendo n_j acessível do nodo n_i .

$P_{n-\Gamma}$ - conjunto dos caminhos de n para Γ.

$P_{n-\gamma}$ - qualquer caminho de n para γ,
ou seja $P_{n-\gamma} \in P_{n-\Gamma}$.

$P^*_{n_i-n_j}$ - conjunto dos caminhos mais baratos de n_i para n_j.

$k(n_i, n_j)$ - custo do caminho mais barato de n_i para n_j.

$g^*(n)$ - o custo mais barato dos caminhos de s para um nodo n,

$$g^*(n) = k(s,n)$$

$h^*(n)$ - o custo mais barato dos caminhos de n para Γ,

$$h^*(n) = \min_{\gamma \in \Gamma} k(s,\gamma)$$

C^* - o custo mais barato dos caminhos de s para Γ,

$$C^* = h^*(s)$$

Γ^* - conjunto das soluções óptimas, ou seja o subconjunto dos nodos solução acessíveis de s por caminhos cujo custo é C^*.

O algoritmo A* explora o espaço dos estados G usando uma árvore T. A união de todos os ramos das árvores construídas durante a pesquisa constituem o subgrupo explicado G_e do grafo implícito G.

$PP_{n_1-n_2}$ - é um caminho de n_1 para n_2, sendo $PP_{n_1-n_2} \in T$.

Vamos ver em que condições o algoritmo A* é

COMPLETO - Um algoritmo é completo se termina com uma solução se ela existir.

ADMISSÍVEL - Um algoritmo é admissível se garante uma solução óptima quando uma solução existe.

Já tinhamos visto na "Metodologia de Resolução", em 2.2.6, um esqueleto do algoritmo A*.

1- Pôr o nodo origem s em "ABERTOS".

2- Se "ABERTOS" está vazia sair sem sucesso.

3- Remover de "ABERTOS" o nodo n com f mínima, e ponha-o em "FECHADOS".

4- Se n é o objectivo, a solução, sair com a solução obtida e indicar o "trace" da solução, o caminho do objectivo até s.

5- Expandir n, gerando SCS(n).

6- Para cada n' $\in$ SCS(n) estimar h(n') (uma estimativa de custo do melhor caminho de n' para uma solução) e calcular

$$f(n') = g(n') + h(n')$$

onde $g(n') = g(n) + c(n,n')$

com $g(s) = 0$.

Pôr SCS(n) em "ABERTOS" com os valores f assignados.

7- Voltar a 2)

A função f de seleção dos nodos usada em A* é

$$f(n) = g(n) + h(n)$$

com

g(n) - o custo do caminho PP_{s-n},

com g(s) = 0

$h(n)$ - uma estimativa de $h^*(n)$ com

$$h(\gamma) = 0$$

Se $g_p(n)$ fôr o custo de P_{s-n} e $h_p(n)$ o custo de $P_{n-\gamma}$ por um caminho particular P, temos para todos os caminhos P

$$g_p(n) \geq g^*(n) \text{ e } h_p(n) \geq h^*(n) \qquad (1)$$

Quando g e h coincidem com o seu valor óptimo,

$$f^*(n) = g^*(n) + h^*(n) \qquad (2)$$

e $f^*(n)$ é o custo óptimo de todos os caminhos solução com a imposição de passarem por n.

Em particular

$$f^*(s) = h^*(s) = g^*(\gamma) = f^*(\gamma) = C^* \quad \text{para todos } \gamma \in \Gamma^* \qquad (3)$$

e ainda para qualquer nodo n^* num caminho óptimo de s para γ, tem-se

$$f^*(n^*) = C^* \quad , \quad n^* \in P^*_{s-\Gamma} \qquad (4)$$

Prova: se $n^* \in P^*_{s-\Gamma}$ é porque existe um caminho solução P que passa por n e custa C^*, ou seja ao longo deste caminho P,

$$g_p(n) + h_p(n) = C^*$$

Usando a propriedade (1) de óptimo de g^* e h^*, tem-se

$$g^*(n) + h^*(n) \leq C^*$$

mas esta inequação não pode ser estrita, porque senão existia um outro caminho P' ao longo do qual

$$g_{p'}(n) + h_{p'}(n) \leq C^*,$$

contrariando a condição de óptimo de C^*.

Mas f^* não é computável, especialmente não é computável a sua parte h^*. Temos a sua aproximação f que é computável, que é o que A^* utiliza, com $f = g + h$.

10.2.1 - A^* é COMPLETO

A^* é COMPLETO em grafos finitos:

Demonstração:

Tem-se um insucesso, ou seja o algoritmo não produz uma solução quando "ABERTOS" estiver vazia, e se existir um caminho solução $P_{s-\gamma}$, "ABERTOS" nunca poderá estar vazia antes de $P_{s-\gamma}$ estar descoberto. Se isso acontecesse, um último nodo $n' \in P_{s-\gamma}$ estaria em "ABERTOS" e ao ser expandido não produziria sucessores. Isto contradiz a condição de n' estar num caminho solução. Todos os nodos do caminho solução $P_{s-\gamma}$ (à excepção do nodo terminal solução) têm pelo menos um sucessor também em $P_{s-\gamma}$.

A^* é COMPLETO mesmo em grafos infinitos:

Demonstração:

O conjunto de nodos do caminho solução $P_{s-\gamma}$ têm assignados valores de f finitos e está sempre pelo menos um deles em "ABERTOS". A^* não pode voltar com um insucesso pela mesma razão porque não podia voltar com um insucesso em

grafos finitos. Portanto se A^* em grafos infinitos não produz uma solução é porque não termina pura e simplesmente. Se A^* não termina é porque está a seguir um caminho infinito, mas isto contraria a política de selecção de nodos de A^*.

Como o custo dos arcos é positivo, todos os caminhos infinitos têm um custo a tender para infinito, e como se viu está sempre em "ABERTOS" pelo menos um nodo com um custo limitado, um f finito, e A^* pára a pesquisa pelo caminho infinito e selecciona esse nodo de "ABERTOS" para expansão.

10.2.2 - Admissibilidade de A^*

A^* é ADMISSÍVEL:

Vamos ver que sempre que h(n) fôr uma estimativa optimista de $h^*(n)$, A^* fornece uma solução óptima. As heurísticas optimistas chamam-se heurísticas admissíveis.

Uma função heurística h é admissível, se

$$h(n) \leq h^*(n) \qquad \forall\, n \qquad (5)$$

Portanto qualquer heurística que satisfaça a condição anterior é chamada de heurística admissível.

Lema: (Nilsson, 1980, Resultado 2) antes que A^* termine, existe sempre em "ABERTOS" um nodo n' de $P^*_{s-\gamma}$ com $f(n') \leq C^*$, se a heurística utilizada fôr admissível.

Demonstração:

Considere-se qualquer caminho óptimo

$$P^*_{s-\gamma} \in P^*_{s-\Gamma},$$

$$P^*_{s-\gamma} = s, n1, n2, \cdots, n', \cdots, \gamma$$

e seja n' o último nodo de $P^*_{s-\gamma}$ em "ABERTOS" (há pelo menos um nodo de $P^*_{s-\gamma}$ em "ABERTOS" porque γ não está em "FECHADOS" enquanto A^* não terminar).

Estando n' em "ABERTOS" todos os seus predecessores estão em "FECHADOS" e como o caminho s, n1, n2, $\cdots$ n' é óptimo, $g(n') = g^*(n')$.

Usando a admissibilidade de h, tem-se

$$f(n') = g^*(n') + h(n') \leq g^*(n') + h^*(n') = f^*(n')$$

e como $n' \in P^*_{s-\gamma}$, e por (4) $f^*(n') = C^*$, tem-se

$$f(n') \leq C^* \qquad (6)$$

Teorema: (Nilsson, 1980, Resultado 4) A^* usando uma função heurística admissível, é admissível.

Demonstração: Suponha-se que A^* termina com uma solução $t \in \Gamma$ para a qual $f(t) = g(t) > C^*$. O algoritmo A^* inspecciona se um nodo é solução, só depois do nodo ter sido escolhido para expansão, como se pode ver da descrição dos passos computacionais de A^* já indicada. Então quando t foi escolhido para expansão, tem-se

$$f(t) \leq f(n) \quad \forall n \in \text{"ABERTOS"}$$

Isto quer dizer que todos os nodos em "ABERTOS" tinham $f(n) > C^*$. Mas isto contradiz o Lema que garante a existência de pelo menos um nodo n' com $f(n') \leq C^*$ em "ABERTOS". Então o nodo terminal t tem de ter $g(t) = C^*$ o que quer dizer que A^* dá como solução o caminho óptimo.

Portanto se garantirmos à priori que a heurística que se utiliza é uma heurística admissível, pode-se usar o algoritmo A^* com a garantia de que ele é completo e admissível, ou seja ele dá uma solução óptima se existir solução para o problema a tratar.

Evidentemente que além de admissível, a heurística tem de ser também computável, e quanto mais facilmente computável melhor, pois de nada adiantaria ter uma heurística admissível mas não computável ou com um grau de dificuldade semelhante ao problema original.

10.3 - PROVA DE ADMISSIBILIDADE DA HEURÍSTICA USADA EM A^*

Uma heurística h é admissível se

$$h(n) \leq h^*(n) \quad \forall n$$

Uma verificação exaustiva se uma heurística é admissível é impraticável, tanto mais que $h^*(n)$ pela sua própria natureza é desconhecida para quase todos os nodos do grafo. É precisamente por não se conhecer $h^*(n)$ que se está a usar uma sua estimativa h(n). Então como é que se pode estar certo da condição $h(n) \leq h^*(n)$, $\forall n$? O processo que se emprega para verificar esta proposição é similar às demonstrações simbólicas em matemática, onde a verdade de afirmações universais é estabelecida pela sequência de regras de inferência aplicadas a um número finito de axiomas sem enumeração exaustiva.

Por exemplo, voltando ao caso do exemplo do mapa de estradas, a admissibilidade da heurística usada, a distância directa (distância aérea) é facilmente explicada. Aplicando as propriedades de espaços euclidianos, sabe-se que uma linha

recta entre quaisquer 2 pontos é mais curta que qualquer outra ligação alternativa entre esses 2 pontos.

Para o caso de 8-puzzle, por exemplo, a heurística h_2 usando a distância Manhattan dá-nos o número de passos que se teria de fazer se cada posição fosse independente das outras. Mas como realmente as posições não são independentes umas das outras, mas sim interferem umas com as outras, a situação só pode agravar-se. Tem-se assim a garantia que a heurística h_2 é óptimista em relação à realidade, ou seja h_2 é admissível. Portanto a heurística admissível descreve um procedimento para resolver um problema auxiliar cujas regras de jogo foram relaxadas.

No caso do mapa de estradas tem-se uma função matemática h(n) como aproximação de $h^*(n)$, e no caso do 8-puzzle contam-se o número de passos para resolver o problema relaxado e é esse número que se usa como estimativa de $h^*(n)$.

Vamos agora ver para o problema de calendarização que estamos a considerar, que a heurística utilizada é admissível.

A heurística que usamos é a resolução do problema relaxado do problema original sendo as relaxações envolvidas:

1- Faz-se a suposição que os trabalhadores assignados ao estado ficam livres e disponíveis para fazer outras tarefas sem primeiro terem que acabar as tarefas que estão a fazer.

2- Para a determinação do número de equipas, utiliza-se a intensidade média das tarefas a realizar, ou seja, considera-se que as tarefas são uniformemente distribuidas ao longo do tempo que ainda se dispõe. Isto é uma relaxação optimista da realidade. Esta relaxação é limitada no que concerne às sobreposições

de tarefas críticas que são contabilizadas, como se viu no Cap. 9.

3- Utiliza-se programação linear para a determinação do número de equipas como relaxação da programação inteira que traduz o problema associado às equipas (unidades inteiras).

Portanto as relaxações 1), 2) e 3) garantem que a heurística é uma estimativa optimista da realidade, para qualquer dos estados e portanto é admissível.

11. CONCLUSÕES

Num edifício inteligente é necessário fornecer um conjunto de serviços de informação, os quais baseiam-se em sistemas baseados no conhecimento. Entre eles destaca-se a tarefa de decidir qual das operações de um certo conjunto deve ser realizada, e quando, sobre um conjunto dado de máquinas. A atribuição sobre o tempo de operações a máquinas chama-se calendarização ("scheduling"), e no caso concreto dos edifícios (inteligentes) diz respeito, por exemplo, às operações de manutenção dos equipamentos instalados, tais como os para o aquecimento, ventilação e ar condicionado (AVAC).

Em geral, existem relações entre aquelas operações que limitam a liberdade de atribuição das operações, isto é um equipamento pode processar apenas uma operação de cada vez, ou uma operação particular só pode ser inicializada após todas as que a antecedem terem sido completadas. Portanto, existe uma função que mede a qualidade da calendarização que deve ser calculada.

Os computadores e a calendarização estiveram sempre fortemente relacionados, e por várias razões. Em muitas situações, tais como nas fábricas, o processo de escalonamento envolve uma computação substancial, e daí ser natural realizar este procedimento num computador. No caso dos edifícios inteligentes, pretende-se também integrar vários sistemas (AVAC, iluminação, segurança, comunicação de voz e dados) afim de gerir efectivamente os recursos de modo coordenado para maximizar o desempenho do ocupante, o investimento e os custos de operação, e ainda a flexibilidade.

O trabalho agora apresentado "Minimização da Afectação de Recursos num Problema de Calendarização", e detalhadamente descrito nos Capítulos anteriores, aborda um tipo de problemas que têm sido tema de investigação e

desenvolvimento nas disciplinas da Investigação Operacional (IO), da Inteligência Artificial (IA), das Ciências de Decisão, das Ciências de Computação, etc.. O "sistema de calendarização" que projectámos e desenvolvemos é um contributo para esta temática, pois resolve um problema discreto de optimização que foi formalizado como um processo de decisão composto, segundo Kumar e Kanal.

O algoritmo de pesquisa, um caso particular dos algoritmos "best-first", o algoritmo A^*, utilizado para resolver este problema discreto de optimização foi importado da área da Inteligência Artificial. Embora vários algoritmos de pesquisa tradicionalmente referenciados na área de Investigação Operacional se apresentassem como alternativas possíveis para a pesquisa da solução do problema de calendarização, a escolha recaíu num algoritmo concorrencial guiado por heurísticas, o algoritmo A^*. No Capítulo referente à Metodologia de Resolução discute-se e justifica-se esta escolha.

A heurística que foi implementada tem associado um problema de programação linear, e traduz um contributo original para a resolução do problema escolhido, conseguindo misturar as técnicas da IA e IO.

A heurística adoptada respeita as condições de admissibilidade, como se infere da sua descrição e como se justifica no Capítulo 10. Usando esta heurística admissível a solução proposta pelo algoritmo A^* pode ser considerada como óptima para o problema de calendarização que se está a tratar. Portanto o sistema proposto está balizado pela condição da solução óptima estrita.

O "sistema de calendarização" está implementado adoptando o paradigma da programação em lógica, recorrendo ao formalismo de representação baseado na lógica de primeira ordem ou cálculo dos predicados. Está escrito na linguagem de programação PROLOG (interpretador C-Prolog) com uma estrutura constituída por oito módulos. Esta arquitectura permite alterações e adaptações de um determinado módulo sem

se mudar a estrutura interna do sistema. No caso de ser necessário alterar o módulo da heurística, as mudanças não alteram os outros módulos do sistema. Neste sentido o "sistema de calendarização" é portável para uma vasta gama de problemas, dada a generalidade como são definidas as actividades e o sentido lato associado ao salário dos trabalhadores que pode representar qualquer custo operacional. Esta ferramenta inteligente permite além de optimizar um problema de calendarização, ser também usada para ensaiar vários cenários possíveis de uma determinada conjuntura, funcionando como uma ferramenta inteligente de simulação.

Este "sistema de calendarização" está circunscrito à optimização estrita de problemas de calendarização. No entanto para se resolverem problemas de grandes dimensões, e por causa das limitações de memória dos meios computacionais disponíveis, será necessário prosseguir a investigação através do:

- Melhoramento da heurística: aferir a heurística ao problema que se pretende resolver, pagando para isso uma complexidade crescente na resolução da heurística. O objectivo é investigar um ponto de equilíbrio entre a qualidade da heurística e a complexidade inerente à sua resolução.

- Relaxamento da condição de optimização: este relaxamento pode ser introduzido via relaxamento da condição de admissibilidade da heurística, ou por manipulação no algoritmo A^*. Esta manipulação consiste por exemplo na condensação de soluções candidatas que não difiram substancialmente entre si, funcionando esta função de condensação como uma meta-heurística. Outra manipulação possível consistirá na truncagem, sob critério estabelecido, da lista das árvores das soluções candidatas.

A complexidade inerente a estes problemas de calendarização, impõe cada vez mais um esforço conjunto das áreas nas quais elas são objectivo de investigação e desenvolvimento, tais como a Investigação Operacional, a Inteligência Artificial, as Ciências de Decisão e as Ciências de Computação, entre outras. Novas possibilidades de resolução para este tipo de problemas se abrem com o advento da 5ª geração de computadores com arquitecturas paralelas. Começam a aparecer trabalhos interessantes em implementações com execução em paralelo dos algoritmos de pesquisa, e os desempenhos alcançados são animadores. O potencial do processamento em paralelo para algoritmos de pesquisa da área da IA é tão prometedor que se abrem enormes expectativas para a resolução dos problemas de calendarização de um modo particular e de todos os problemas discretos de optimização de um modo geral.

PARTE II

1. - Introdução

Nesta segunda parte apresenta-se a aplicação do "sistema de calendarização" a dois caso estudados. A aplicação A é uma calendarização das actividades de manutenção de 4 tipos diferentes de equipamento e a aplicação B consiste na calendarização das actividades de um gabinete de estudos de marketing. No item 2 descreve-se a utilização do sistema na aplicação A, apresenta-se o fluxo de conhecimento do utilizador para o sistema e a solução proposta por este ao utilizador. No item 3 é apresentada a interacção do "sistema de calendarização" com o utilizador na aplicação B.

As aplicações foram executadas no computador VAX 8700 do LNEC, sistema operativo VMS, usando a versão 1.5 do PROLOG de Edinburgh.

2. - Aplicação A

Esta aplicação consiste na calendarização das actividades de manutenção de 4 tipos diferentes de equipamento. A manutenção dos equipamentos tem associado uma periocidade de 2 semanas, 10 dias de trabalho útil ao longo do ano. A actividade de manutenção do equipamento tipo 1 está representada na Fig. A.1, a actividade de manutenção do equipamento de tipo 2 está indicada na Fig. A.2, a manutenção do equipamento tipo 3 está descrita na Fig. A.3 e a manutenção do equipamento tipo 4 representa-se na Fig. A.4.

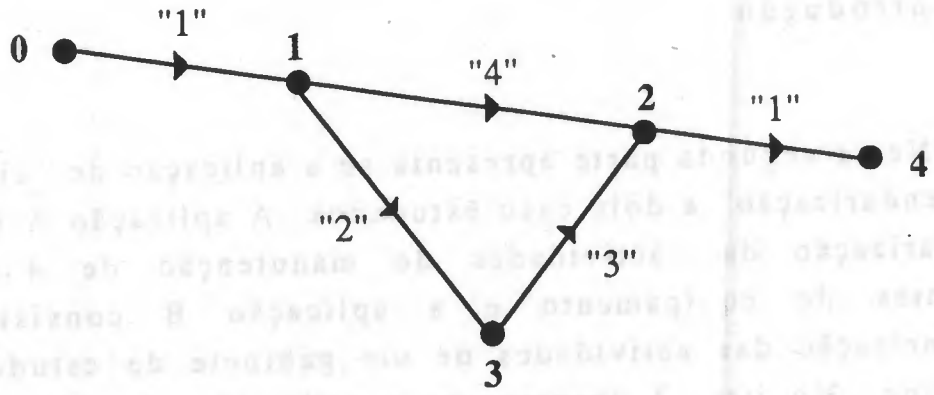


Fig. A.1 - Actividade nº 1

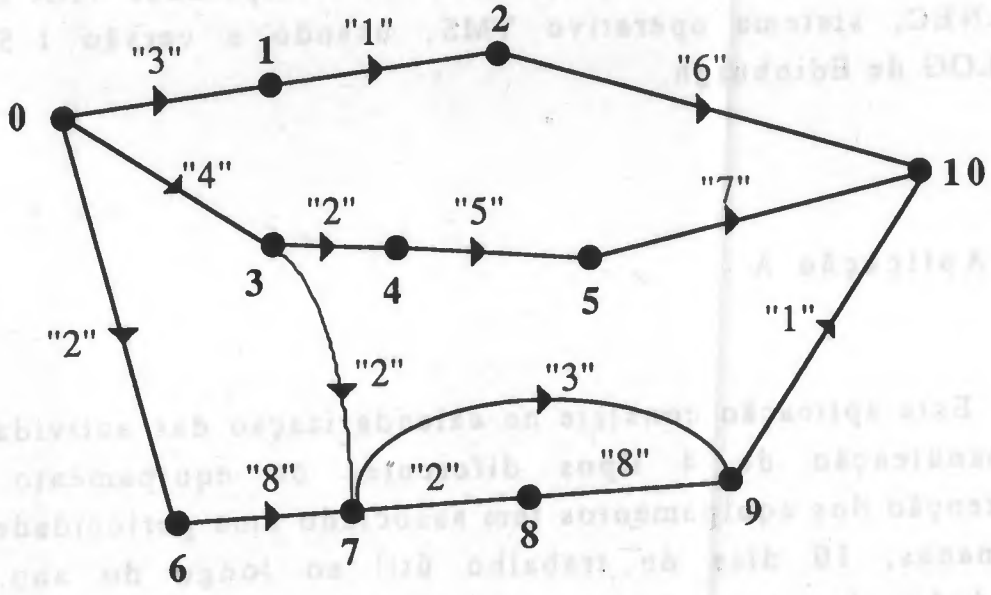


Fig. A.2 - Actividade nº 2

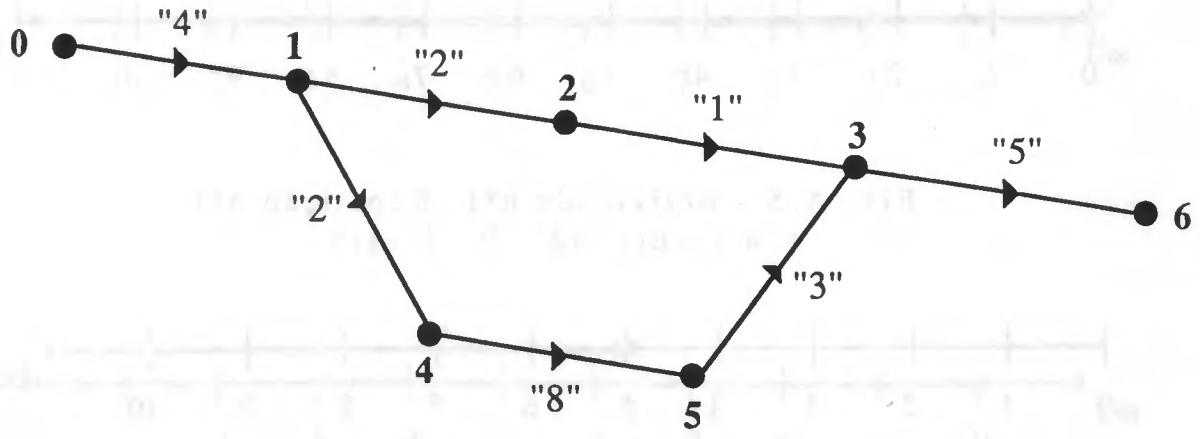


Fig. A.3 - Actividade nº 3

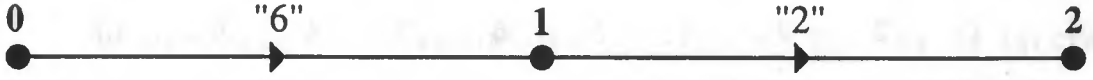


Fig. A.4 - Actividade nº 4

A manutenção do equipamento tipo 1 leva 5 dias a ser executada, a manutenção do equipamento tipo 2 toma 10 dias, o equipamento tipo 3 leva 7 dias, demorando a manutenção do equipamento tipo 4, 10 dias a ser realizada.

No período de 10 dias correspondentes a duas semanas de trabalho, tem de se realizar a manutenção de 4 equipamentos tipo 1, 1 equipamento tipo 2, 1 equipamento tipo 3 e um equipamento tipo 4.

A ocupação no calendário de 2 semanas das manutenções dos 4 equipamentos tipo 1 estão representadas nas Fig. A.5, Fig. A.6, Fig. A.7 e Fig. A.8, do equipamento tipo 2 na Fig. A.9, do equipamento tipo 3 na Fig. A.10 e do equipamento tipo 4 está descrito na Fig. A.11.

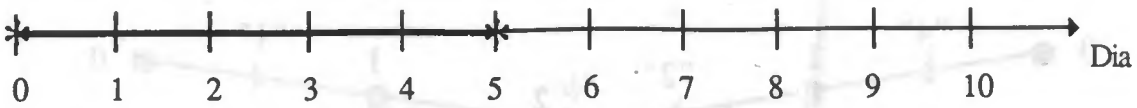


Fig. A.5 - Actividade nº1, Repetição nº1

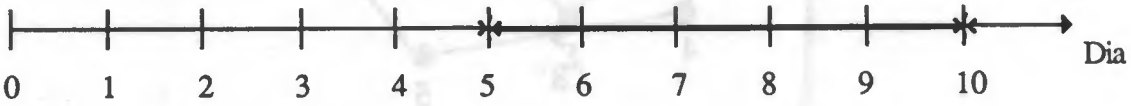


Fig. A.6 - Actividade nº1, Repetição nº2

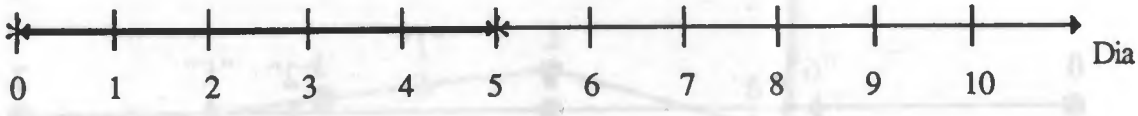


Fig. A.7 - Actividade nº1, Repetição nº3

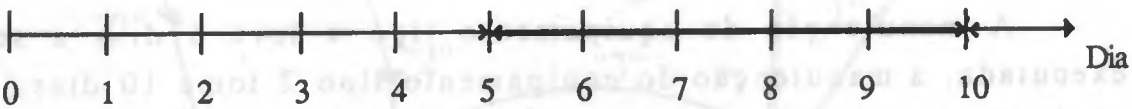


Fig. A.8 - Actividade nº1, Repetição nº4

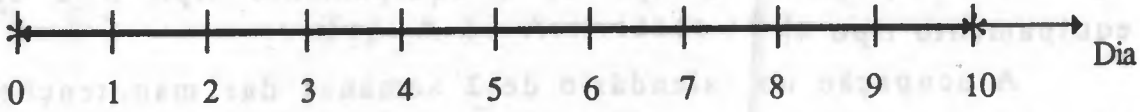


Fig. A.9 - Actividade nº2

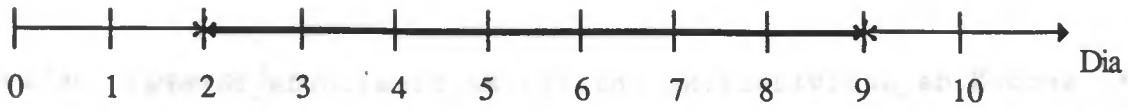


Fig. A.10 - Actividade nº3

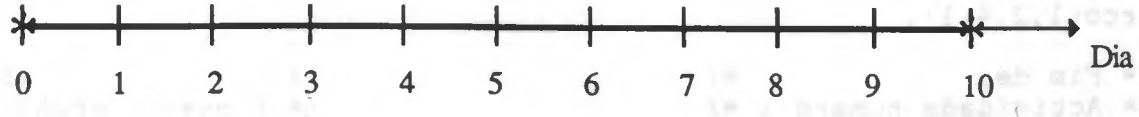


Fig. A.11 - Actividade nº4

As operações de manutenção constam de 8 tipos de tarefas diferentes que envolvem 7 tipos de trabalhadores, sendo que a tarefa número 1 e a tarefa número 5 são realizadas pelo mesmo tipo de trabalhadores.

Os salários dos diferentes tipos de trabalhadores estão expressos em unidades de milhar de contos por anuidade.

Todo o Conhecimento Associado ao Problema de Calendarização que se pretende resolver está descrito tal como o utilizador o introduz no "sistema de calendarização" nas páginas 190 a 193. Nas páginas 190, 191 e 192 descrevem-se as actividades de manutenção e a sua ocupação no calendário de 2 semanas. Na página 193 está descrito o conhecimento associado aos salários dos trabalhadores e à modularidade das equipas.

```
/* ################## */


/*  arco(N.da_actividade,No_inicial,No_final,N.da_tarefa)      */


/* Actividade numero 1 */

arco(1,0,1,1).
arco(1,1,2,4).
arco(1,1,3,2).
arco(1,3,2,3).
arco(1,2,4,1).

/* Fim de:              */
/* Actividade numero 1 */



/* Actividade numero 2 */


arco(2,0,1,3).
arco(2,1,2,1).
arco(2,2,10,6).
arco(2,0,3,4).
arco(2,3,4,2).
arco(2,4,5,5).
arco(2,5,10,7).
arco(2,0,6,2).
arco(2,6,7,8).
arco(2,3,7,2).
arco(2,7,8,2).
arco(2,8,9,8).
arco(2,7,9,3).
arco(2,9,10,1).

/* Fim de:              */
/* Actividade numero 2 */
```

```
/* Actividade numero 3 */

arco(3,0,1,4).
arco(3,1,2,2).
arco(3,2,3,1).
arco(3,1,4,2).
arco(3,4,5,8).
arco(3,5,3,3).
arco(3,3,6,5).

/* Fim de:              */
/* Actividade numero 3 */

/* Actividade numero 4 */

arco(4,0,1,6).
arco(4,1,2,2).

/* Fim de:              */
/* Actividade numero 4 */

/* ################## */
```

```
/* ################## */


dia_comeco(1,1,0).
dia_comeco(1,2,0).
dia_comeco(1,3,5).
dia_comeco(1,4,5).

dia_comeco(2,1,0).

dia_comeco(3,1,2).

dia_comeco(4,1,0).


/* ################## */

tempo_max(1,1,5).
tempo_max(1,2,5).
tempo_max(1,3,5).
tempo_max(1,4,5).

tempo_max(2,1,10).

tempo_max(3,1,7).

tempo_max(4,1,10).

/* ################## */

dia_final(10).

/* ################## */


n_maximo_equipas(1000).

/* ################## */


/* ============================================================*/
```

```
   ============================================================*/

/*    salario (Tipo_de_trabalhador,Salario).    */

salario( trab1,1.5).

salario( trab2,2.5).

salario( trab3,3).

salario( trab4,2).

salario( trab5,2.5).

salario( trab6,1.8).

salario( trab7,2.2).

/* ################### */

/*  tarefa(N._da_tarefa,,[Lista das opcoes de realizacao]).        */
/*                        [N._de_trabalhadores*Tipo_de_trabalhador*   */
/*                         N._de_dias_para_realizar_a_tarefa|_]       */


 tarefa( 1, [ 2*trab1*3 , 3*trab1*1 ] ).

 tarefa( 2, [ 2*trab2*2 ] ).

 tarefa( 3, [ 1*trab3*2 , 3*trab3*1 ] ).

 tarefa( 4, [ 1*trab4*2 , 3*trab4*1 ] ).

 tarefa( 5,  [ 3*trab1*2 , 4*trab1*1 ] ).

 tarefa( 6,  [ 3*trab5*8 ] ).

 tarefa( 7,  [ 3*trab6*6 ] ).

 tarefa( 8,  [ 3*trab7*2 ] ).

/* ################### */
```

Nas páginas 195 a 198, tem-se uma cópia da interacção do utilizador com o "sistema de calendarização", em que se tem como resposta do sistema o custo global da solução por ele proposta, que é o custo mínimo operacional expresso em unidades de milhar de contos. De seguida o sistema indica qual o contingente por tipo de trabalhador necessário, e a calendarização das tarefas dia a dia. Diariamente o sistema indica quais as tarefas a começar, com que tipo de equipa, ou seja com que número de trabalhadores e em que dia acabam as tarefas a começar no dia referido.

Nas páginas 199 até 203 está a resposta do "sistema de calendarização" ao problema proposto, que é a sua solução óptima. Descreve o caminho na árvore de estados que conduziu à solução do custo global mínimo. As informações anteriores são o resumo desta informação completa dado pelo sistema.

Na página 204 está contida a informação estatística computacional referente à aplicação do "sistema de calendarização" ao caso estudado.

```
$ cprolog -g 3500 -l 3500 -h 800 -t 500
-Prolog version 1.5

yes
| ?-[sistema_de_calendarizacao].

?

?
sistema_de_calendarizacao consulted 48144 bytes 0.28 sec.

yes
| ?-consult(dados1).

dados1 consulted 4192 bytes 2.700010e-02 sec.

yes
| ?-executar.

-----------------------
CUSTO GLOBAL DA SOLUCAO: 92.0999
-----------------------

 TOTAL DE TRABALHADORES trab1 : 7

 TOTAL DE TRABALHADORES trab2 : 8

 TOTAL DE TRABALHADORES trab3 : 6

 TOTAL DE TRABALHADORES trab4 : 5

 TOTAL DE TRABALHADORES trab5 : 6

 TOTAL DE TRABALHADORES trab6 : 3

 TOTAL DE TRABALHADORES trab7 : 6
```

```
--------------------------
CALENDARIZACAO DAS TAREFAS
--------------------------


 DIA : 0
 ----

COMECAR A FAZER AS TAREFAS :

arc(4,1,0,1,6,8,3)   com 3 trabalhadores trab5 e termina no dia 8 .
arc(2,1,0,6,2,2,2)   com 2 trabalhadores trab2 e termina no dia 2 .
arc(2,1,0,3,4,1,3)   com 3 trabalhadores trab4 e termina no dia 1 .
arc(2,1,0,1,3,1,3)   com 3 trabalhadores trab3 e termina no dia 1 .
arc(1,2,0,1,1,1,3)   com 3 trabalhadores trab1 e termina no dia 1 .
arc(1,1,0,1,1,1,3)   com 3 trabalhadores trab1 e termina no dia 1 .


 DIA : 1
 ----

COMECAR A FAZER AS TAREFAS :

arc(2,1,3,4,2,3,2)   com 2 trabalhadores trab2 e termina no dia 3 .
arc(2,1,1,2,1,2,3)   com 3 trabalhadores trab1 e termina no dia 2 .
arc(1,2,1,3,2,3,2)   com 2 trabalhadores trab2 e termina no dia 3 .
arc(1,1,1,3,2,3,2)   com 2 trabalhadores trab2 e termina no dia 3 .


 DIA : 2
 ----

COMECAR A FAZER AS TAREFAS :

arc(1,2,1,2,4,4,1)   com 1 trabalhadores trab4 e termina no dia 4 .
arc(1,1,1,2,4,4,1)   com 1 trabalhadores trab4 e termina no dia 4 .
arc(3,1,0,1,4,3,3)   com 3 trabalhadores trab4 e termina no dia 3 .
arc(2,1,6,7,8,4,3)   com 3 trabalhadores trab7 e termina no dia 4 .
arc(2,1,2,10,6,10,3) com 3 trabalhadores trab5 e termina no dia 10 .
arc(2,1,3,7,2,4,2)   com 2 trabalhadores trab2 e termina no dia 4 .
```

```
DIA : 3
----

COMECAR A FAZER AS TAREFAS :

arc(3,1,1,4,2,5,2)    com 2 trabalhadores trab2 e termina no dia 5 .
arc(3,1,1,2,2,5,2)    com 2 trabalhadores trab2 e termina no dia 5 .
arc(2,1,4,5,5,4,4)    com 4 trabalhadores trab1 e termina no dia 4 .
arc(1,2,3,2,3,4,3)    com 3 trabalhadores trab3 e termina no dia 4 .
arc(1,1,3,2,3,4,3)    com 3 trabalhadores trab3 e termina no dia 4 .


DIA : 4
----

COMECAR A FAZER AS TAREFAS :

arc(2,1,7,8,2,6,2)    com 2 trabalhadores trab2 e termina no dia 6 .
arc(2,1,5,10,7,10,3) com 3 trabalhadores trab6 e termina no dia 10 .
arc(1,2,2,4,1,5,3)    com 3 trabalhadores trab1 e termina no dia 5 .
arc(1,1,2,4,1,5,3)    com 3 trabalhadores trab1 e termina no dia 5 .


DIA : 5
----

COMECAR A FAZER AS TAREFAS :

arc(3,1,4,5,8,7,3)    com 3 trabalhadores trab7 e termina no dia 7 .
arc(1,4,0,1,1,6,3)    com 3 trabalhadores trab1 e termina no dia 6 .
arc(1,3,0,1,1,6,3)    com 3 trabalhadores trab1 e termina no dia 6 .


DIA : 6
----

COMECAR A FAZER AS TAREFAS :

arc(3,1,2,3,1,7,3)    com 3 trabalhadores trab1 e termina no dia 7 .
arc(2,1,7,9,3,7,3)    com 3 trabalhadores trab3 e termina no dia 7 .
arc(1,4,1,2,4,7,3)    com 3 trabalhadores trab4 e termina no dia 7 .
arc(1,3,1,2,4,8,1)    com 1 trabalhadores trab4 e termina no dia 8 .
arc(2,1,8,9,8,8,3)    com 3 trabalhadores trab7 e termina no dia 8 .
arc(1,4,1,3,2,8,2)    com 2 trabalhadores trab2 e termina no dia 8 .
arc(1,3,1,3,2,8,2)    com 2 trabalhadores trab2 e termina no dia 8 .
```

```
 DIA : 7
 ----

COMECAR A FAZER AS TAREFAS :

arc(3,1,5,3,3,8,3)    com 3 trabalhadores trab3 e termina no dia 8 .


 DIA : 8
 ----

COMECAR A FAZER AS TAREFAS :

arc(2,1,9,10,1,9,3)   com 3 trabalhadores trab1 e termina no dia 9 .
arc(4,1,1,2,2,10,2)   com 2 trabalhadores trab2 e termina no dia 10 .
arc(3,1,3,6,5,9,4)    com 4 trabalhadores trab1 e termina no dia 9 .
arc(1,4,3,2,3,9,3)    com 3 trabalhadores trab3 e termina no dia 9 .
arc(1,3,3,2,3,9,3)    com 3 trabalhadores trab3 e termina no dia 9 .


 DIA : 9
 ----

COMECAR A FAZER AS TAREFAS :

arc(1,4,2,4,1,10,3)    com 3 trabalhadores trab1 e termina no dia 10 .
arc(1,3,2,4,1,10,3)    com 3 trabalhadores trab1 e termina no dia 10 .
```

```
-------------------------------------------------
Solucao proprosta pelo "sistema de calendarizacao" :
-------------------------------------------------

0^[]^[]^[]^[total(trab1,0),total(trab2,0),total(trab3,0),total(trab4,0),
           total(trab5,0),total(trab6,0),total(trab7,0)]^
          [livres(trab1,0),livres(trab2,0),livres(trab3,0),livres(trab4,0),
           livres(trab5,0),livres(trab6,0),livres(trab7,0)]

1^[]^[arc(4,1,0,1,6,8,3),arc(2,1,0,6,2,2,2),arc(2,1,0,3,4,1,3),
      arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),arc(1,1,0,1,1,1,3)]^
     [arc(1,1,1,2,4,_126482,_126483),arc(1,2,1,2,4,_126484,_126485),
      arc(1,1,1,3,2,_126486,_126487),arc(1,2,1,3,2,_126488,_126489),
      arc(1,1,3,2,3,_126490,_126491),arc(1,2,3,2,3,_126492,_126493),
      arc(1,1,2,4,1,_126494,_126495),arc(1,2,2,4,1,_126496,_126497),
      arc(2,1,1,2,1,_126498,_126499),arc(2,1,2,10,6,_126500,_126501),
      arc(2,1,3,4,2,_126502,_126503),arc(2,1,4,5,5,_126504,_126505),
      arc(2,1,5,10,7,_126506,_126507),arc(2,1,6,7,8,_126508,_126509),
      arc(2,1,3,7,2,_126510,_126511),arc(2,1,7,8,2,_126512,_126513),
      arc(2,1,8,9,8,_126514,_126515),arc(2,1,7,9,3,_126516,_126517),
      arc(2,1,9,10,1,_126518,_126519),arc(4,1,1,2,2,_126520,_126521)]^
     [total(trab1,6),total(trab2,2),total(trab3,3),total(trab4,3),
      total(trab5,3),total(trab6,0),total(trab7,0)]^
     [livres(trab1,0),livres(trab2,0),livres(trab3,0),livres(trab4,0),
      livres(trab5,0),livres(trab6,0),livres(trab7,0)]

2^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3)]^
  [arc(4,1,0,1,6,8,3),arc(2,1,0,6,2,2,2),arc(2,1,3,4,2,3,2),
   arc(2,1,1,2,1,2,3),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2)]^
  [arc(1,1,1,2,4,_158721,_158722),arc(1,2,1,2,4,_158723,_158724),
   arc(1,1,3,2,3,_158725,_158726),arc(1,2,3,2,3,_158727,_158728),
   arc(1,1,2,4,1,_158729,_158730),arc(1,2,2,4,1,_158731,_158732),
   arc(2,1,2,10,6,_158733,_158734),arc(2,1,4,5,5,_158735,_158736),
   arc(2,1,5,10,7,_158737,_158738),arc(2,1,6,7,8,_158739,_158740),
   arc(2,1,3,7,2,_158741,_158742),arc(2,1,7,8,2,_158743,_158744),
   arc(2,1,8,9,8,_158745,_158746),arc(2,1,7,9,3,_158747,_158748),
   arc(2,1,9,10,1,_158749,_158750),arc(4,1,1,2,2,_158751,_158752)]^
  [total(trab1,6),total(trab2,8),total(trab3,3),total(trab4,3),total(trab5,3
   total(trab6,0),total(trab7,0)]^
  [livres(trab1,3),livres(trab2,0),livres(trab3,3),livres(trab4,3),
   livres(trab5,0),livres(trab6,0),livres(trab7,0)]
```

```
3^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3)]^
  [arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),arc(4,1,0,1,6,8,3),
   arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
   arc(3,1,0,1,4,3,3),arc(2,1,6,7,8,4,3),arc(2,1,2,10,6,10,3),
   arc(2,1,3,7,2,4,2)]^
  [arc(1,1,3,2,3,_256964,_256965),arc(1,2,3,2,3,_256966,_256967),
   arc(1,1,2,4,1,_256968,_256969),arc(1,2,2,4,1,_256970,_256971),
   arc(2,1,4,5,5,_256972,_256973),arc(2,1,5,10,7,_256974,_256975),
   arc(2,1,7,8,2,_256976,_256977),arc(2,1,8,9,8,_256978,_256979),
   arc(2,1,7,9,3,_256980,_256981),arc(2,1,9,10,1,_256982,_256983),
   arc(4,1,1,2,2,_256984,_256985),arc(3,1,1,2,2,_256986,_256987),
   arc(3,1,2,3,1,_256988,_256989),arc(3,1,1,4,2,_256990,_256991),
   arc(3,1,4,5,8,_256992,_256993),arc(3,1,5,3,3,_256994,_256995),
   arc(3,1,3,6,5,_256996,_256997)]^
  [total(trab1,6),total(trab2,8),total(trab3,3),total(trab4,5),
   total(trab5,6),total(trab6,0),total(trab7,3)]^
  [livres(trab1,6),livres(trab2,0),livres(trab3,3),livres(trab4,0),
   livres(trab5,0),livres(trab6,0),livres(trab7,0)]

4^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3),
   arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
   arc(3,1,0,1,4,3,3)]^[arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),
   arc(4,1,0,1,6,8,3),arc(2,1,6,7,8,4,3),arc(2,1,2,10,6,10,3),
   arc(2,1,3,7,2,4,2),arc(3,1,1,4,2,5,2),arc(3,1,1,2,2,5,2),
   arc(2,1,4,5,5,4,4),arc(1,2,3,2,3,4,3),arc(1,1,3,2,3,4,3)]^
  [arc(1,1,2,4,1,_323261,_323262),arc(1,2,2,4,1,_323263,_323264),
   arc(2,1,5,10,7,_323265,_323266),arc(2,1,7,8,2,_323267,_323268),
   arc(2,1,8,9,8,_323269,_323270),arc(2,1,7,9,3,_323271,_323272),
   arc(2,1,9,10,1,_323273,_323274),arc(4,1,1,2,2,_323275,_323276),
   arc(3,1,2,3,1,_323277,_323278),arc(3,1,4,5,8,_323279,_323280),
   arc(3,1,5,3,3,_323281,_323282),arc(3,1,3,6,5,_323283,_323284)]^
  [total(trab1,6),total(trab2,8),total(trab3,6),total(trab4,5),
   total(trab5,6),total(trab6,0),total(trab7,3)]^
  [livres(trab1,2),livres(trab2,2),livres(trab3,0),livres(trab4,3),
   livres(trab5,0),livres(trab6,0),livres(trab7,0)]
```

```
5^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3),
   arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
   arc(3,1,0,1,4,3,3),arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),
   arc(2,1,6,7,8,4,3),arc(2,1,3,7,2,4,2),arc(2,1,4,5,5,4,4),
   arc(1,2,3,2,3,4,3),arc(1,1,3,2,3,4,3)]^
  [arc(4,1,0,1,6,8,3),arc(2,1,2,10,6,10,3),arc(3,1,1,4,2,5,2),
   arc(3,1,1,2,2,5,2),arc(2,1,7,8,2,6,2),arc(2,1,5,10,7,10,3),
   arc(1,2,2,4,1,5,3),arc(1,1,2,4,1,5,3)]^
  [arc(2,1,8,9,8,_344520,_344521),arc(2,1,7,9,3,_344522,_344523),
   arc(2,1,9,10,1,_344524,_344525),arc(4,1,1,2,2,_344526,_344527),
   arc(3,1,2,3,1,_344528,_344529),arc(3,1,4,5,8,_344530,_344531),
   arc(3,1,5,3,3,_344532,_344533),arc(3,1,3,6,5,_344534,_344535)]^
  [total(trab1,6),total(trab2,8),total(trab3,6),total(trab4,5),
   total(trab5,6),total(trab6,3),total(trab7,3)]^
  [livres(trab1,0),livres(trab2,2),livres(trab3,6),livres(trab4,5),
   livres(trab5,0),livres(trab6,0),livres(trab7,3)]

6^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3),
   arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
   arc(3,1,0,1,4,3,3),arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),
   arc(2,1,6,7,8,4,3),arc(2,1,3,7,2,4,2),arc(2,1,4,5,5,4,4),
   arc(1,2,3,2,3,4,3),arc(1,1,3,2,3,4,3),arc(3,1,1,4,2,5,2),
   arc(3,1,1,2,2,5,2),arc(1,2,2,4,1,5,3),arc(1,1,2,4,1,5,3)]^
  [arc(4,1,0,1,6,8,3),arc(2,1,2,10,6,10,3),arc(2,1,7,8,2,6,2),
   arc(2,1,5,10,7,10,3),arc(3,1,4,5,8,7,3),arc(1,4,0,1,1,6,3),
   arc(1,3,0,1,1,6,3)]^[arc(2,1,8,9,8,_372508,_372509),
   arc(2,1,7,9,3,_372510,_372511),arc(2,1,9,10,1,_372512,_372513),
   arc(4,1,1,2,2,_372514,_372515),arc(3,1,2,3,1,_372516,_372517),
   arc(3,1,5,3,3,_372518,_372519),arc(3,1,3,6,5,_372520,_372521),
   arc(1,3,1,2,4,_372522,_372523),arc(1,4,1,2,4,_372524,_372525),
   arc(1,3,1,3,2,_372526,_372527),arc(1,4,1,3,2,_372528,_372529),
   arc(1,3,3,2,3,_372530,_372531),arc(1,4,3,2,3,_372532,_372533),
   arc(1,3,2,4,1,_372534,_372535),arc(1,4,2,4,1,_372536,_372537)]^
  [total(trab1,6),total(trab2,8),total(trab3,6),total(trab4,5),
   total(trab5,6),total(trab6,3),total(trab7,3)]^
  [livres(trab1,0),livres(trab2,6),livres(trab3,6),livres(trab4,5),
   livres(trab5,0),livres(trab6,0),livres(trab7,0)]
```

```
7^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3),
   arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
   arc(3,1,0,1,4,3,3),arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),
   arc(2,1,6,7,8,4,3),arc(2,1,3,7,2,4,2),arc(2,1,4,5,5,4,4),
   arc(1,2,3,2,3,4,3),arc(1,1,3,2,3,4,3),arc(3,1,1,4,2,5,2),
   arc(3,1,1,2,2,5,2),arc(1,2,2,4,1,5,3),arc(1,1,2,4,1,5,3),
   arc(2,1,7,8,2,6,2),arc(1,4,0,1,1,6,3),arc(1,3,0,1,1,6,3)]^
  [arc(3,1,2,3,1,7,3),arc(2,1,7,9,3,7,3),arc(1,4,1,2,4,7,3),
   arc(1,3,1,2,4,8,1),arc(4,1,0,1,6,8,3),arc(2,1,2,10,6,10,3),
   arc(2,1,5,10,7,10,3),arc(3,1,4,5,8,7,3),arc(2,1,8,9,8,8,3),
   arc(1,4,1,3,2,8,2),arc(1,3,1,3,2,8,2)]^
  [arc(2,1,9,10,1,_503630,_503631),arc(4,1,1,2,2,_503632,_503633),
   arc(3,1,5,3,3,_503634,_503635),arc(3,1,3,6,5,_503636,_503637),
   arc(1,3,3,2,3,_503638,_503639),arc(1,4,3,2,3,_503640,_503641),
   arc(1,3,2,4,1,_503642,_503643),arc(1,4,2,4,1,_503644,_503645)]^
  [total(trab1,6),total(trab2,8),total(trab3,6),total(trab4,5),
   total(trab5,6),total(trab6,3),total(trab7,6)]^
  [livres(trab1,3),livres(trab2,4),livres(trab3,3),livres(trab4,1),
   livres(trab5,0),livres(trab6,0),livres(trab7,0)]

8^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3),
   arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
   arc(3,1,0,1,4,3,3),arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),
   arc(2,1,6,7,8,4,3),arc(2,1,3,7,2,4,2),arc(2,1,4,5,5,4,4),
   arc(1,2,3,2,3,4,3),arc(1,1,3,2,3,4,3),arc(3,1,1,4,2,5,2),
   arc(3,1,1,2,2,5,2),arc(1,2,2,4,1,5,3),arc(1,1,2,4,1,5,3),
   arc(2,1,7,8,2,6,2),arc(1,4,0,1,1,6,3),arc(1,3,0,1,1,6,3),
   arc(3,1,2,3,1,7,3),arc(2,1,7,9,3,7,3),arc(1,4,1,2,4,7,3),
   arc(3,1,4,5,8,7,3)]^[arc(1,3,1,2,4,8,1),arc(4,1,0,1,6,8,3),
   arc(2,1,2,10,6,10,3),arc(2,1,5,10,7,10,3),arc(2,1,8,9,8,8,3),
   arc(1,4,1,3,2,8,2),arc(1,3,1,3,2,8,2),arc(3,1,5,3,3,8,3)]^
  [arc(2,1,9,10,1,_748277,_748278),arc(4,1,1,2,2,_748279,_748280),
   arc(3,1,3,6,5,_748281,_748282),arc(1,3,3,2,3,_748283,_748284),
   arc(1,4,3,2,3,_748285,_748286),arc(1,3,2,4,1,_748287,_748288),
   arc(1,4,2,4,1,_748289,_748290)]^
  [total(trab1,6),total(trab2,8),total(trab3,6),total(trab4,5),
   total(trab5,6),total(trab6,3),total(trab7,6)]^
  [livres(trab1,6),livres(trab2,4),livres(trab3,3),livres(trab4,4),
   livres(trab5,0),livres(trab6,0),livres(trab7,3)]
```

```
9^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
   arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3),
   arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
   arc(3,1,0,1,4,3,3),arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),
   arc(2,1,6,7,8,4,3),arc(2,1,3,7,2,4,2),arc(2,1,4,5,5,4,4),
   arc(1,2,3,2,3,4,3),arc(1,1,3,2,3,4,3),arc(3,1,1,4,2,5,2),
   arc(3,1,1,2,2,5,2),arc(1,2,2,4,1,5,3),arc(1,1,2,4,1,5,3),
   arc(2,1,7,8,2,6,2),arc(1,4,0,1,1,6,3),arc(1,3,0,1,1,6,3),
   arc(3,1,2,3,1,7,3),arc(2,1,7,9,3,7,3),arc(1,4,1,2,4,7,3),
   arc(3,1,4,5,8,7,3),arc(1,3,1,2,4,8,1),arc(4,1,0,1,6,8,3),
   arc(2,1,8,9,8,8,3),arc(1,4,1,3,2,8,2),arc(1,3,1,3,2,8,2),
   arc(3,1,5,3,3,8,3)]^[arc(2,1,9,10,1,9,3),arc(2,1,2,10,6,10,3),
   arc(2,1,5,10,7,10,3),arc(4,1,1,2,2,10,2),arc(3,1,3,6,5,9,4),
   arc(1,4,3,2,3,9,3),arc(1,3,3,2,3,9,3)]^
  [arc(1,3,2,4,1,_758451,_758452),arc(1,4,2,4,1,_758453,_758454)]^
  [total(trab1,7),total(trab2,8),total(trab3,6),total(trab4,5),
   total(trab5,6),total(trab6,3),total(trab7,6)]^
  [livres(trab1,0),livres(trab2,6),livres(trab3,0),livres(trab4,5),
   livres(trab5,3),livres(trab6,0),livres(trab7,6)]

10^[arc(2,1,0,3,4,1,3),arc(2,1,0,1,3,1,3),arc(1,2,0,1,1,1,3),
    arc(1,1,0,1,1,1,3),arc(2,1,0,6,2,2,2),arc(2,1,1,2,1,2,3),
    arc(2,1,3,4,2,3,2),arc(1,2,1,3,2,3,2),arc(1,1,1,3,2,3,2),
    arc(3,1,0,1,4,3,3),arc(1,2,1,2,4,4,1),arc(1,1,1,2,4,4,1),
    arc(2,1,6,7,8,4,3),arc(2,1,3,7,2,4,2),arc(2,1,4,5,5,4,4),
    arc(1,2,3,2,3,4,3),arc(1,1,3,2,3,4,3),arc(3,1,1,4,2,5,2),
    arc(3,1,1,2,2,5,2),arc(1,2,2,4,1,5,3),arc(1,1,2,4,1,5,3),
    arc(2,1,7,8,2,6,2),arc(1,4,0,1,1,6,3),arc(1,3,0,1,1,6,3),
    arc(3,1,2,3,1,7,3),arc(2,1,7,9,3,7,3),arc(1,4,1,2,4,7,3),
    arc(3,1,4,5,8,7,3),arc(1,3,1,2,4,8,1),arc(4,1,0,1,6,8,3),
    arc(2,1,8,9,8,8,3),arc(1,4,1,3,2,8,2),arc(1,3,1,3,2,8,2),
    arc(3,1,5,3,3,8,3),arc(2,1,9,10,1,9,3),arc(3,1,3,6,5,9,4),
    arc(1,4,3,2,3,9,3),arc(1,3,3,2,3,9,3)]^[arc(2,1,2,10,6,10,3),
    arc(2,1,5,10,7,10,3),arc(4,1,1,2,2,10,2),arc(1,4,2,4,1,10,3),
    arc(1,3,2,4,1,10,3)]^
   []^
   [total(trab1,7),total(trab2,8),total(trab3,6),total(trab4,5),
    total(trab5,6),total(trab6,3),total(trab7,6)]^
   [livres(trab1,1),livres(trab2,6),livres(trab3,6),livres(trab4,5),
    livres(trab5,3),livres(trab6,0),livres(trab7,6)]


yes
```

```
| ?-statistics.
atom space: 128K (in use: 29636, max. used: 29636)
aux. stack: 8K (in use: 0, max. used: 1692)
trail: 500K (in use: 48, max. used: 180932)
heap: 800K (in use: 119400, max. used: 607644)
global stack: 3500K (in use: 0, max. used: 3113124)
local stack: 3500K (in use: 300, max. used: 2472836)
Runtime:    23.00 sec.

yes
| ?-

[ Prolog execution halted ]
$
```

2. - Aplicação B

Esta aplicação consiste na calendarização das actividades de um gabinete de marketing. Em 10 dias de trabalho útil, pretende-se realizar uma campanha de maior dimensão, campanha tipo 1 e 4 campanhas menores do mesmo tipo, tipo 2. Destas 4 campanhas tipo 2, 3 delas são relaxadas, ou seja podem tomar mais tempo que o mínimo associado à actividade tipo 2.

A campanha tipo 1 tem associadas as tarefas

1 - prospecção de mercado
3 - análise da prospecção
5 - previsão
7 - conceptualização da campanha
9 - implementação gráfica
10 - avaliação

com a seguinte tabela de precedências:

Tarefa	Precedências
1	—
3	1
5	1
7	3 e 5
9	7
10	2

A modelização da campanha tipo 1, está representada na Fig. B.1

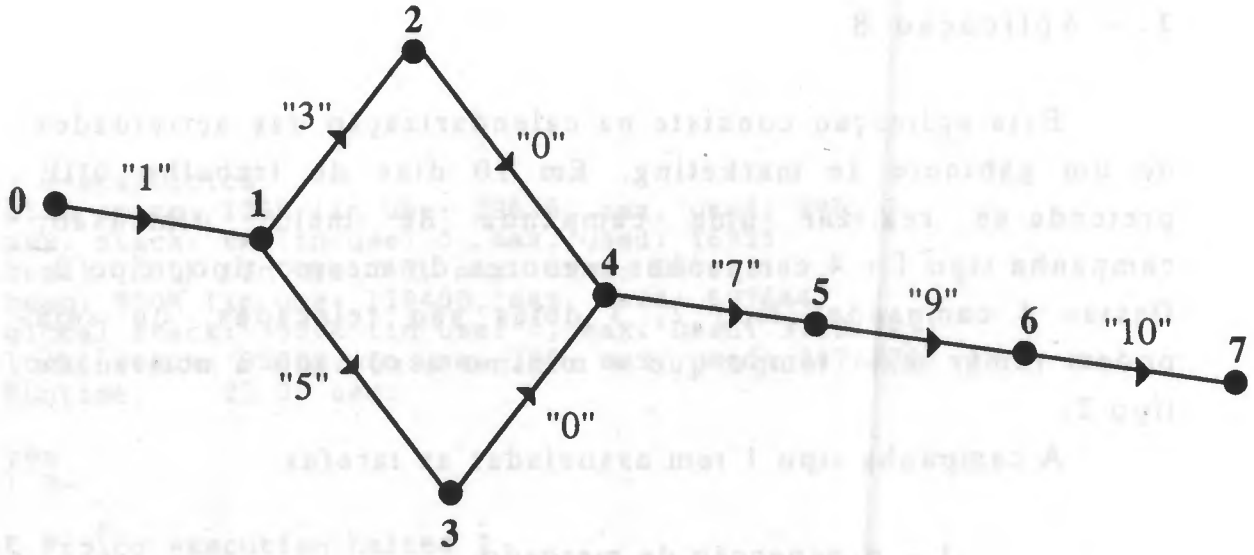


Fig. B.1 - Actividade Nº 1

A campanha tipo 2, tem associadas as tarefas

2 - prospecção de mercado
4 - análise da prospecção
6 - previsão
8 - conceptualização e implementação da campanha

com a seguinte tabela de precedências:

Tarefa	Precedências
2	—
4	2
6	2
8	4 e 6

A modelização da actividade nº 2, está descrita na Fig. B.2

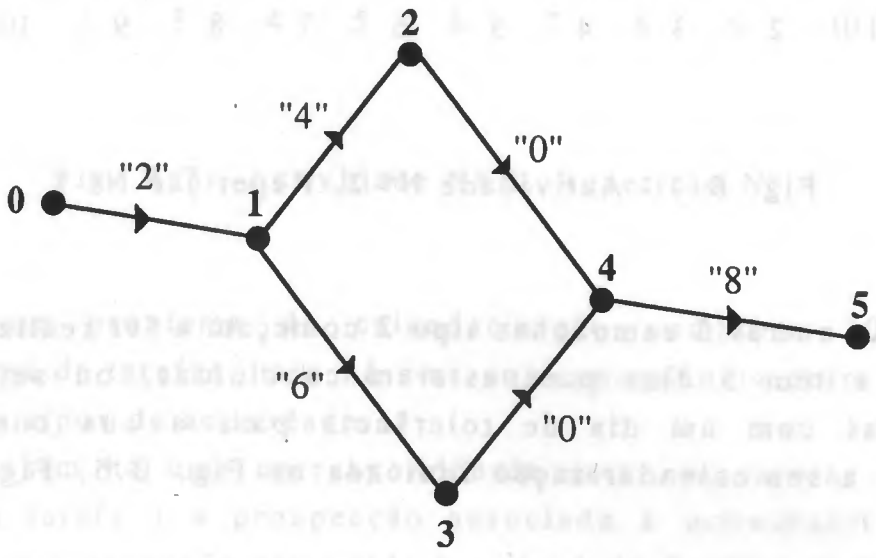


Fig. B.2 - Actividade Nº 2

A calendarização pretendida para a realização da campanha tipo 1 está representada na Fig. B.3

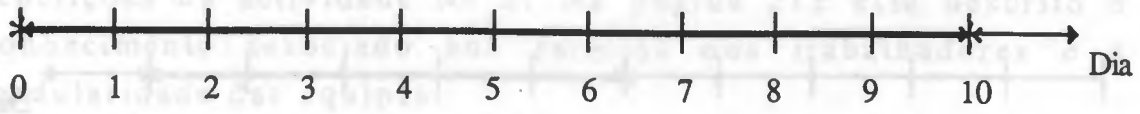


Fig. B.3 - Actividade Nº 1

Das 4 campanhas tipo 2, uma tem de ser realizada no tempo mínimo e com calendarização indicada na Fig. B.4

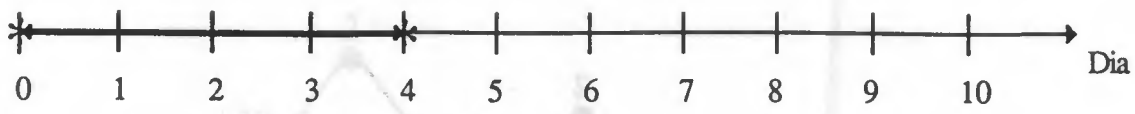


Fig. B.4 - Actividade Nº 2, Repetição Nº 1

As outras 3 campanhas tipo 2 começam a ser realizadas no dia 5, e têm 5 dias para estarem concluídas, ou seja estão relaxadas com um dia de tolerância para a sua conclusão, estando a sua calendarização indicada na Fig. B.5, Fig. B.6 e Fig. B.7

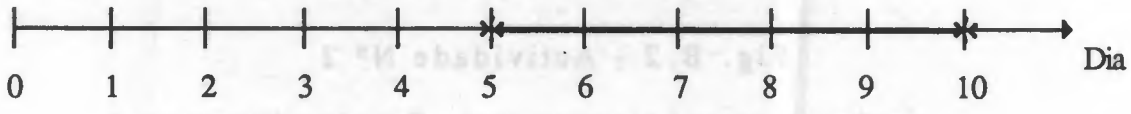


Fig. B.5 - Actividade Nº 2, Repetição Nº 2

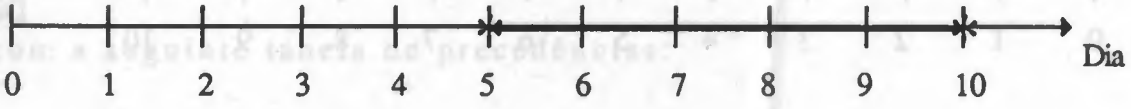


Fig. B.6 - Actividade Nº 2, Repetição Nº 3

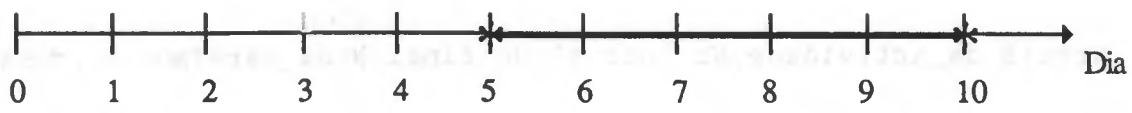


Fig. B.7 - Actividade Nº 2, Repetição Nº 4

Este problema de calendarização comporta 10 tipos diferentes de tarefas, mas há uma correspondência entre tarefas dos dois tipos de actividades.

Assim há uma correspondência entre as tarefas 1 e 2, sendo a tarefa 1 a prospecção associada à actividade 1, e a tarefa 2 a prospecção associada à actividade 2. As tarefas 1 e 2 são assim realizadas pelo mesmo tipo de trabalhador.

Do mesmo modo há uma correspondência entre as tarefas 3 e 4, as tarefas 5 e 6, e entre as tarefas 7 e 8, sendo as tarefas correspondentes realizadas pelo mesmo tipo de trabalhador.

Todo este conhecimento, que é introduzido pelo utilizador no "sistema de calendarização" está descrito nas páginas 210, 211 e 212. Na página 210 descrevem-se as 2 actividades, na página 211 a ocupação no calendário da actividade Nº 1 e das 4 repetições da actividade Nº 2. Na página 212 está descrito o conhecimento associado aos salários dos trabalhadores e à modularidade das equipas.

```
/-  ################## */


/*  arco(N.da_actividade,No_inicial,No_final,N.da_tarefa)      */


/* Actividade numero 1 */


arco(1,0,1,1).
arco(1,1,2,3).
arco(1,2,4,0).
arco(1,1,3,5).
arco(1,3,4,0).
arco(1,4,5,7).
arco(1,5,6,9).
arco(1,6,7,10).


/* Fim de:                */
/* Actividade numero 1 */


/* Actividade numero 2 */


arco(2,0,1,2).
arco(2,1,2,4).
arco(2,2,4,0).
arco(2,1,3,6).
arco(2,3,4,0).
arco(2,4,5,8).


/* Fim de:                */
/* Actividade numero 2 */


/* ################## */
```

```
/* ################### */


dia_comeco(1,1,0).


dia_comeco(2,1,0).

dia_comeco(2,2,5).
dia_comeco(2,3,5).
dia_comeco(2,4,5).



/* ################### */


tempo_max(1,1,10).

tempo_max(2,1,4).

tempo_max(2,2,5).
tempo_max(2,3,5).
tempo_max(2,4,5).


/* ################### */


dia_final(10).

/* ################### */


n_maximo_equipas(1000).


/* ################### */
```

```
   ============================================================*/

/*     salario (Tipo_de_trabalhador,Salario).    */

salario( trab1,1).

salario( trab2,2.6).

salario( trab3,2.8).

salario( trab4,3.8).

salario( trab5,3).

/* ################### */


/*  tarefa(N._da_tarefa,,[Lista das opcoes de realizacao]).          */
/*                        [N._de_trabalhadores*Tipo_de_trabalhador*  */
/*                         N._de_dias_para_realizar_a_tarefa|_]      */


 tarefa( 0, [ 0*trab1*0 ] ).

 tarefa( 1, [ 4*trab1*2 ] ).

 tarefa( 2, [ 1*trab1*2 , 2*trab1*1 ] ).

 tarefa( 3, [ 2*trab2*2 ] ).

 tarefa( 4, [ 1*trab2*2, 2*trab2*1 ] ).

 tarefa( 5, [ 2*trab3*2 , 3*trab3*1 ] ).

 tarefa( 6, [ 1*trab3*1 ] ).
 tarefa( 7,  [ 1*trab4*2 ] ).

 tarefa( 8,  [ 1*trab4*3, 2*trab4*2 ] ).

 tarefa( 9,  [ 2*trab5*3 ] ).
 tarefa( 10,  [ 4*trab1*1 ] ).


/* ################### */
```

Nas páginas 214 a 216 tem-se uma cópia da interacção do utilizador com o "sistema de calendarização", obtendo-se como resposta o custo global associado à realização das actividades e o contingente necessário por tipo de trabalhador. Obtem-se também a informação diária das tarefas a começar, com o número de trabalhadores envolvidos na sua realização e data de conclusão.

Nas páginas 217 a 221 descreve-se o caminho na árvore de estados que conduziu à solução proposta.

Na página 222 está indicada a informação estatística referente à execução desta calendarização.

```
$ cprolog -g 4000 -l 3350 -h 700 -t 300
C-Prolog version 1.5

yes
| ?-[sistema_de_calendarizacao].

?

?
sistema_de_calendarizacao consulted 47948 bytes 0.33 sec.

yes
| ?-consult(dados2).

dados2 consulted 3248 bytes 2.800000e-02 sec.

yes
| ?-executar.


-----------------------
CUSTO GLOBAL DA SOLUCAO: 46
-----------------------

 TOTAL DE TRABALHADORES trab1 : 6

 TOTAL DE TRABALHADORES trab2 : 4

 TOTAL DE TRABALHADORES trab3 : 3

 TOTAL DE TRABALHADORES trab4 : 4

 TOTAL DE TRABALHADORES trab5 : 2
```

```
-------------------------
ALENDARIZACAO DAS TAREFAS
-------------------------


 DIA : 0
 ----

COMECAR A FAZER AS TAREFAS :

arc(2,1,0,1,2,1,2)    com 2 trabalhadores trab1 e termina no dia 1 .
arc(1,1,0,1,1,2,4)    com 4 trabalhadores trab1 e termina no dia 2 .


 DIA : 1
 ----

COMECAR A FAZER AS TAREFAS :

arc(2,1,1,3,6,2,1)    com 1 trabalhadores trab3 e termina no dia 2 .
arc(2,1,1,2,4,2,2)    com 2 trabalhadores trab2 e termina no dia 2 .


 DIA : 2
 ----

COMECAR A FAZER AS TAREFAS :

arc(1,1,1,3,5,3,3)    com 3 trabalhadores trab3 e termina no dia 3 .
arc(2,1,4,5,8,4,2)    com 2 trabalhadores trab4 e termina no dia 4 .
arc(1,1,1,2,3,4,2)    com 2 trabalhadores trab2 e termina no dia 4 .


 DIA : 4
 ----

COMECAR A FAZER AS TAREFAS :

arc(1,1,4,5,7,6,1)    com 1 trabalhadores trab4 e termina no dia 6 .
```

```
DIA : 5
----

COMECAR A FAZER AS TAREFAS :

arc(2,2,0,1,2,7,1)   com 1 trabalhadores trab1 e termina no dia 7 .
arc(2,3,0,1,2,6,2)   com 2 trabalhadores trab1 e termina no dia 6 .
arc(2,4,0,1,2,6,2)   com 2 trabalhadores trab1 e termina no dia 6 .


DIA : 6
----

COMECAR A FAZER AS TAREFAS :

arc(2,3,1,3,6,7,1)   com 1 trabalhadores trab3 e termina no dia 7 .
arc(2,3,1,2,4,7,2)   com 2 trabalhadores trab2 e termina no dia 7 .
arc(2,4,1,3,6,7,1)   com 1 trabalhadores trab3 e termina no dia 7 .
arc(2,4,1,2,4,7,2)   com 2 trabalhadores trab2 e termina no dia 7 .
arc(1,1,5,6,9,9,2)   com 2 trabalhadores trab5 e termina no dia 9 .


DIA : 7
----

COMECAR A FAZER AS TAREFAS :

arc(2,3,4,5,8,10,1)   com 1 trabalhadores trab4 e termina no dia 10 .
arc(2,4,4,5,8,10,1)   com 1 trabalhadores trab4 e termina no dia 10 .
arc(2,2,1,3,6,8,1)    com 1 trabalhadores trab3 e termina no dia 8 .
arc(2,2,1,2,4,8,2)    com 2 trabalhadores trab2 e termina no dia 8 .


DIA : 8
----

COMECAR A FAZER AS TAREFAS :

arc(2,2,4,5,8,10,2)   com 2 trabalhadores trab4 e termina no dia 10 .


DIA : 9
----

COMECAR A FAZER AS TAREFAS :

arc(1,1,6,7,10,10,4)   com 4 trabalhadores trab1 e termina no dia 10 .
```

```
-----------------------------------------------
Solucao proprosta pelo "sistema de calendarizacao" :
-----------------------------------------------

0^[]^[]^[]^[total(trab1,0),total(trab2,0),total(trab3,0),total(trab4,0),
            total(trab5,0)]^
           [livres(trab1,0),livres(trab2,0),livres(trab3,0),livres(trab4,0),
            livres(trab5,0)]

1^[]^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4)]^[arc(1,1,1,2,3,_106533,_106534)
      arc(1,1,2,4,0,_106535,_106536),arc(1,1,1,3,5,_106537,_106538),
      arc(1,1,3,4,0,_106539,_106540),arc(1,1,4,5,7,_106541,_106542),
      arc(1,1,5,6,9,_106543,_106544),arc(1,1,6,7,10,_106545,_106546),
      arc(2,1,1,2,4,_106547,_106548),arc(2,1,2,4,0,_106549,_106550),
      arc(2,1,1,3,6,_106551,_106552),arc(2,1,3,4,0,_106553,_106554),
      arc(2,1,4,5,8,_106555,_106556)]^
     [total(trab1,6),total(trab2,0),total(trab3,0),total(trab4,0),
      total(trab5,0)]^
     [livres(trab1,0),livres(trab2,0),livres(trab3,0),livres(trab4,0),
      livres(trab5,0)]

2^[arc(2,1,0,1,2,1,2)]^
  [arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),arc(2,1,1,2,4,2,2)]^
  [arc(1,1,1,2,3,_114042,_114043),arc(1,1,2,4,0,_114044,_114045),
   arc(1,1,1,3,5,_114046,_114047),arc(1,1,3,4,0,_114048,_114049),
   arc(1,1,4,5,7,_114050,_114051),arc(1,1,5,6,9,_114052,_114053),
   arc(1,1,6,7,10,_114054,_114055),arc(2,1,2,4,0,_114056,_114057),
   arc(2,1,3,4,0,_114058,_114059),arc(2,1,4,5,8,_114060,_114061)]^
  [total(trab1,6),total(trab2,2),total(trab3,1),total(trab4,0),
   total(trab5,0)]^
  [livres(trab1,2),livres(trab2,0),livres(trab3,0),livres(trab4,0),
   livres(trab5,0)]

3^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
   arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0)]^
  [arc(1,1,1,3,5,3,3),arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2)]^
  [arc(1,1,2,4,0,_124252,_124253),arc(1,1,3,4,0,_124254,_124255),
   arc(1,1,4,5,7,_124256,_124257),arc(1,1,5,6,9,_124258,_124259),
   arc(1,1,6,7,10,_124260,_124261)]^
  [total(trab1,6),total(trab2,2),total(trab3,3),total(trab4,2),
   total(trab5,0)]^
  [livres(trab1,6),livres(trab2,0),livres(trab3,0),livres(trab4,0),
   livres(trab5,0)]
```

```
4^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
   arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0),
   arc(1,1,1,3,5,3,3)]^[arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2)]^
  [arc(1,1,2,4,0,_141987,_141988),arc(1,1,3,4,0,_141989,_141990),
   arc(1,1,4,5,7,_141991,_141992),arc(1,1,5,6,9,_141993,_141994),
   arc(1,1,6,7,10,_141995,_141996)]^
  [total(trab1,6),total(trab2,2),total(trab3,3),total(trab4,2),
   total(trab5,0)]^
  [livres(trab1,6),livres(trab2,0),livres(trab3,3),livres(trab4,0),
   livres(trab5,0)]

5^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
   arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0),
   arc(1,1,1,3,5,3,3),arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2),
   arc(1,1,3,4,0,4,0),arc(1,1,2,4,0,4,0)]^
  [arc(1,1,4,5,7,6,1)]^
  [arc(1,1,5,6,9,_149816,_149817),arc(1,1,6,7,10,_149818,_149819)]^
  [total(trab1,6),total(trab2,2),total(trab3,3),total(trab4,2),
   total(trab5,0)]^
  [livres(trab1,6),livres(trab2,2),livres(trab3,3),livres(trab4,1),
   livres(trab5,0)]
```

```
6^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
   arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0),
   arc(1,1,1,3,5,3,3),arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2),
   arc(1,1,3,4,0,4,0),arc(1,1,2,4,0,4,0)]^
  [arc(2,2,0,1,2,7,1),arc(2,3,0,1,2,6,2),arc(2,4,0,1,2,6,2),
   arc(1,1,4,5,7,6,1)]^
  [arc(1,1,5,6,9,_181884,_181885),arc(1,1,6,7,10,_181886,_181887),
   arc(2,2,1,2,4,_181888,_181889),arc(2,3,1,2,4,_181890,_181891),
   arc(2,4,1,2,4,_181892,_181893),arc(2,2,2,4,0,_181894,_181895),
   arc(2,3,2,4,0,_181896,_181897),arc(2,4,2,4,0,_181898,_181899),
   arc(2,2,1,3,6,_181900,_181901),arc(2,3,1,3,6,_181902,_181903),
   arc(2,4,1,3,6,_181904,_181905),arc(2,2,3,4,0,_181906,_181907),
   arc(2,3,3,4,0,_181908,_181909),arc(2,4,3,4,0,_181910,_181911),
   arc(2,2,4,5,8,_181912,_181913),arc(2,3,4,5,8,_181914,_181915),
   arc(2,4,4,5,8,_181916,_181917)]^
  [total(trab1,6),total(trab2,2),total(trab3,3),total(trab4,2),
   total(trab5,0)]^
  [livres(trab1,1),livres(trab2,2),livres(trab3,3),livres(trab4,1),
   livres(trab5,0)]

7^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
   arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0),
   arc(1,1,1,3,5,3,3),arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2),
   arc(1,1,3,4,0,4,0),arc(1,1,2,4,0,4,0),arc(2,3,0,1,2,6,2),
   arc(2,4,0,1,2,6,2),arc(1,1,4,5,7,6,1)]^
  [arc(2,3,1,3,6,7,1),arc(2,3,1,2,4,7,2),arc(2,4,1,3,6,7,1),
   arc(2,4,1,2,4,7,2),arc(2,2,0,1,2,7,1),arc(1,1,5,6,9,9,2)]^
  [arc(1,1,6,7,10,_349786,_349787),arc(2,2,1,2,4,_349788,_349789),
   arc(2,2,2,4,0,_349790,_349791),arc(2,3,2,4,0,_349792,_349793),
   arc(2,4,2,4,0,_349794,_349795),arc(2,2,1,3,6,_349796,_349797),
   arc(2,2,3,4,0,_349798,_349799),arc(2,3,3,4,0,_349800,_349801),
   arc(2,4,3,4,0,_349802,_349803),arc(2,2,4,5,8,_349804,_349805),
   arc(2,3,4,5,8,_349806,_349807),arc(2,4,4,5,8,_349808,_349809)]^
  [total(trab1,6),total(trab2,4),total(trab3,3),total(trab4,2),
   total(trab5,2)]^
  [livres(trab1,5),livres(trab2,0),livres(trab3,1),livres(trab4,2),
   livres(trab5,0)]
```

```
8^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
   arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0),
   arc(1,1,1,3,5,3,3),arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2),
   arc(1,1,3,4,0,4,0),arc(1,1,2,4,0,4,0),arc(2,3,0,1,2,6,2),
   arc(2,4,0,1,2,6,2),arc(1,1,4,5,7,6,1),arc(2,3,1,3,6,7,1),
   arc(2,3,1,2,4,7,2),arc(2,4,1,3,6,7,1),arc(2,4,1,2,4,7,2),
   arc(2,2,0,1,2,7,1),arc(2,3,2,4,0,7,0),arc(2,3,3,4,0,7,0),
   arc(2,4,2,4,0,7,0),arc(2,4,3,4,0,7,0)]^
  [arc(2,3,4,5,8,10,1),arc(2,4,4,5,8,10,1),arc(1,1,5,6,9,9,2),
   arc(2,2,1,3,6,8,1),arc(2,2,1,2,4,8,2)]^
  [arc(1,1,6,7,10,_584519,_584520),arc(2,2,2,4,0,_584521,_584522),
   arc(2,2,3,4,0,_584523,_584524),arc(2,2,4,5,8,_584525,_584526)]^
  [total(trab1,6),total(trab2,4),total(trab3,3),total(trab4,2),
   total(trab5,2)]^
  [livres(trab1,6),livres(trab2,2),livres(trab3,2),livres(trab4,0),
   livres(trab5,0)]

9^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
   arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0),
   arc(1,1,1,3,5,3,3),arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2),
   arc(1,1,3,4,0,4,0),arc(1,1,2,4,0,4,0),arc(2,3,0,1,2,6,2),
   arc(2,4,0,1,2,6,2),arc(1,1,4,5,7,6,1),arc(2,3,1,3,6,7,1),
   arc(2,3,1,2,4,7,2),arc(2,4,1,3,6,7,1),arc(2,4,1,2,4,7,2),
   arc(2,2,0,1,2,7,1),arc(2,3,2,4,0,7,0),arc(2,3,3,4,0,7,0),
   arc(2,4,2,4,0,7,0),arc(2,4,3,4,0,7,0),arc(2,2,1,3,6,8,1),
   arc(2,2,1,2,4,8,2),arc(2,2,3,4,0,8,0),arc(2,2,2,4,0,8,0)]^
  [arc(2,3,4,5,8,10,1),arc(2,4,4,5,8,10,1),arc(1,1,5,6,9,9,2),
   arc(2,2,4,5,8,10,2)]^[arc(1,1,6,7,10,_630610,_630611)]^
  [total(trab1,6),total(trab2,4),total(trab3,3),total(trab4,4),
   total(trab5,2)]^
  [livres(trab1,6),livres(trab2,4),livres(trab3,3),livres(trab4,0),
   livres(trab5,0)]
```

```
10^[arc(2,1,0,1,2,1,2),arc(1,1,0,1,1,2,4),arc(2,1,1,3,6,2,1),
    arc(2,1,1,2,4,2,2),arc(2,1,3,4,0,2,0),arc(2,1,2,4,0,2,0),
    arc(1,1,1,3,5,3,3),arc(2,1,4,5,8,4,2),arc(1,1,1,2,3,4,2),
    arc(1,1,3,4,0,4,0),arc(1,1,2,4,0,4,0),arc(2,3,0,1,2,6,2),
    arc(2,4,0,1,2,6,2),arc(1,1,4,5,7,6,1),arc(2,3,1,3,6,7,1),
    arc(2,3,1,2,4,7,2),arc(2,4,1,3,6,7,1),arc(2,4,1,2,4,7,2),
    arc(2,2,0,1,2,7,1),arc(2,3,2,4,0,7,0),arc(2,3,3,4,0,7,0),
    arc(2,4,2,4,0,7,0),arc(2,4,3,4,0,7,0),arc(2,2,1,3,6,8,1),
    arc(2,2,1,2,4,8,2),arc(2,2,3,4,0,8,0),arc(2,2,2,4,0,8,0),
    arc(1,1,5,6,9,9,2)]^
   [arc(2,3,4,5,8,10,1),arc(2,4,4,5,8,10,1),arc(2,2,4,5,8,10,2),
    arc(1,1,6,7,10,10,4)]^
   []^
   [total(trab1,6),total(trab2,4),total(trab3,3),total(trab4,4),
    total(trab5,2)]^
   [livres(trab1,2),livres(trab2,4),livres(trab3,3),livres(trab4,0),
    livres(trab5,2)]


yes
```

```
| ?-statistics.
atom space: 128K (in use: 29592, max. used: 29592)
aux. stack: 8K (in use: 0, max. used: 1692)
trail: 300K (in use: 48, max. used: 139320)
heap: 700K (in use: 114624, max. used: 480900)
global stack: 4000K (in use: 0, max. used: 2645100)
local stack: 3350K (in use: 300, max. used: 2143520)
Runtime:    18.01 sec.

yes
| ?-

[ Prolog execution halted ]
$
```

4. - Conclusões

Vimos nos items 2 e 3 a aplicação do "sistema de calendarização" aos casos estudados A e B. A aplicação A é constituída por 7 actividades: 4 repetições da actividade Nº1, uma actividade Nº2, uma actividade Nº3 e uma actividade Nº4, envolvendo 7 tipos de trabalhadores para a realização de 8 tarefas diferentes, num horizonte temporal de 10 dias úteis. A aplicação B é constituída por 5 actividades: uma actividade Nº1 e 4 actividades Nº2, sendo que 3 delas são repetições relaxadas, ou seja, todos os seus arcos têm flexibilidade temporal. A aplicação B no seu todo tem 10 tarefas diferentes, sendo uma delas a tarefa fictícia "0" necessária para a modelização das actividade. As 10 tarefas envolvem 5 tipos diferentes de trabalhadores. A ocupação temporal da aplicação é de 10 dias.

Nos dois casos estudados o "sistema de calendarização" propôs uma solução óptima num tempo de execução perfeitamente aceitável, respectivamente 23.00 segundos e 18.01 segundos. A ocupação de memória é uma questão a merecer reflexão como se pode constatar das estatísticas computacionais referentes à execução das 2 aplicações, reflexão que nos conduz às Conclusões explanadas na Parte I.

Tal como se referiu nas Conclusões da Parte I, para tratar de problemas de grandes dimensões, as vias possíveis a investigar seriam uma possível melhoria da heurística utilizada pagando por isso uma complexidade crescente na resolução da heurística, e a relaxação da condição da solução óptima. A relaxação da condição de optimização poderia ser encarada via perca de admissibilidade da heurística, ou via manipulação no algoritmo A^* tal como se referiu nas Conclusões da Parte I desta dissertação.

ANEXO

```
/*  OPERADORES                                                    */
/*  ____________________________________________________________  */
```

```
/*  ######################################################  */


/*  definicao dos operadores usados                         */


:-op(600,xfxfx,*).

:-op(650,yfyfyfyfyfy,^).


/*  Fim de:                                                 */
/*  definicao dos operadores usados                         */


/*  ######################################################  */
```

```
/*  FIM DE :                                                    */
/*  OPERADORES                                                  */
/*  ____________________________________________________________*/
```

/* INFERENCIA DE CONHECIMENTO ASSOCIADO AS ACTIVIDADES */
/* _____ */

```
/*   ######################################################   */


/*   ntarefas(N_de_tarefas)                                   */

ntarefa:- findall(Taref,
                  tarefa(Taref,_),
                  Ltaref),
          klength(Ltaref,Ktaref0),
          Ktaref is Ktaref0-1,
          asserta(ntarefas(Ktaref)).


/*   Fim de:                                                  */
/*   ntarefas(N_de_tarefas)                                   */



/*   ######################################################   */

/*   lista_actividades(Lista_das_actividades)                 */
/*   n__actividades(N_das_actividades)                        */

numeracao_actividades:-findall(Xac,arco(Xac,_,_,_),Lnum),
                       sort(Lnum,Lnum1),
                       asserta(lista_actividades(Lnum1)),
                       reverse(Lnum1,[Nnum|TNnum]),
                       asserta( n_actividades(Nnum)).


/*   lista_actividades(Lista_das_actividades)                 */
/*   n_actividades(N_das_actividades)                         */


/*   ######################################################   */
```

```
/*   ##################################################   */

/*   lista_arcos(Actividade,Lista_dos_seus_arcos)                    */

lista_arcos(Act,Listarcos):-findall( arco(Act,A,B,C),
                                     arco(Act,A,B,C),
                                     Listarcos ),
                              asserta(lista_arcos(Act,Listarcos)).

/*   Fim de:                                                         */
/*   lista_arcos(Actividade,Lista_dos_seus_arcos)                    */


/*   ##########################################################
*/


/*   n_arcos_actividade(Actividade,N_dos_seus_arcos)                 */
/*   lista_nodos(Actividade,Lista_dos_seus_nodos)                    */
/*   n_nodos_actividade(Actividade,N_dos_seus_nodos)                 */

activ_descricao:-lista_actividades(Lact),
                 aux(Lact).


aux([]).
aux([CLac|RLac]):-lista_arcos(CLac,WL),
                  klength(WL,Narcos),
                  asserta( n_arcos_actividade(CLac,Narcos)),
                  findall(N,arco(CLac,_,N,_),Wl),
                  sort(Wl,Wll),
                  asserta(lista_nodos(CLac,[0|Wll])),
                  reverse(Wll,[Nnodos|_]),
                  asserta(n_nodos_actividade(CLac,Nnodos)),
                  aux(RLac).


/*   Fim de :                                                        */
/*   n_arcos_actividade(Actividade,N_dos_seus_arcos)                 */
/*   lista_nodos(Actividade,Lista_dos_seus_nodos)                    */
/*   n_nodos_actividade(Actividade,N_dos_seus_nodos)                 */

/*   ##################################################   */
```

```
/*   ################################################   */


/*   precedentes_nodo(Actividade,Nodo,                  */
/*                    Lista_dos_nodos_precedentes)      */

prec_de_nodos:-lista_actividades(Lact),
               auxx(Lact).

auxx([]).
auxx([CLac|RLac]):-lista_nodos(CLac,Lnod),
                   prec(CLac,Lnod),
                   auxx(RLac).

prec(_,[]).
prec(CLac,[CLnod|RLnod]):-findall(Prec,arco(CLac,Prec,CLnod,_),Lprec),
                          asserta(precedentes_nodo(CLac,CLnod,Lprec)),
                          prec(CLac,RLnod).

/*   Fim de:                                            */
/*   precedentes_nodo(Actividade,Nodo,                  */
/*                    Lista_dos_nodos_precedentes)      */


/*   ################################################   */
```

```
/*  ##################################################  */


/*  dura_min_arco(Arco,Duracao_minima_do_arco)          */
/*  dura_max_arco(Arco,Duracao_maxima_do_arco)          */
/*  min_trab_arco(Arco,N_minimo_de_trabalhadores_arco)  */
/*  max_trab_arco(Arco,N_maximo_de_trabalhadores_arco)  */


duracao_arcos:-lista_actividades(Lact),
               dura(Lact).


dura([]).

dura([CLac|RLac]):-lista_arcos(CLac,Larcos),
                   durac(Larcos),
                   dura(RLac).


durac([]).
durac([arco(A,B,C,T)|Ta]):- tarefa(T,[Tm*_*DM|Rm]),
                            asserta(dura_max_arco(arco(A,B,C,T),DM)),
                            asserta(min_trab_arco(arco(A,B,C,T),Tm)),
                            tarefa(T,LT),
                            reverse(LT,[TM*_*Dm|RM]),
                            asserta(dura_min_arco(arco(A,B,C,T),Dm)),
                            asserta(max_trab_arco(arco(A,B,C,T),TM)),
                            durac(Ta).


/*  Fim de:                                             */
/*  dura_min_arco(Arco,Duracao_minima_do_arco)          */
/*  dura_max_arco(Arco,Duracao_maxima_do_arco)          */
/*  min_trab_arco(Arco,N_minimo_de_trabalhadores_arco)  */
/*  max_trab_arco(Arco,N_maximo_de_trabalhadores_arco)  */



/*  ##################################################  */
```

```
/*   FIM DE :                                                    */
/*   INFERENCIA DE CONHECIMENTO ASSOCIADO AS ACTIVIDADES         */
/*   ___________________________________________________________ */
```

```
/*   INFERENCIA DE CONHECIMENTO ASSOCIADO AO CAMINHO          */
/*   CRITICO DAS ACTIVIDADES                                  */
/*   ____________________________________________________     */
```

```
/*  ##################################################      */

/*      es(Actividade,Nodo,Inicio_mais_cedo_do_nodo)                */

criacao_es1:-n_actividades(Nnum),
             criacao(1,Nnum).

criacao(Cont,Nnum):-Cont>Nnum,!;
                    lista_nodos(Cont,Listnodos),
                    criac(Cont,Listnodos),
                    Cont1 is Cont+1,
                    criacao(Cont1,Nnum).

criac(_,[]):-!.
criac(Cont,[C|Ra]):-C=:=0,
                    criac(Cont,Ra);
                    asserta(es1(Cont,C,1)),
                    criac(Cont,Ra).


criacao_es:-n_actividades(Nnum),
            cries(1,Nnum).

cries(Cont,Nnum):-Cont>Nnum,!;
                  asserta(es(Cont,0,0)),
                  Cont1 is Cont+1,
                  cries(Cont1,Nnum).


proc_es:-lista_actividades(Lact),
         inicio_mais_cedo(Lact).


inicio_mais_cedo([]).

inicio_mais_cedo([CLact|RLact]):-lista_nodos(CLact,NCa),
                                 mais_cedo(CLact,NCa),
                                 inicio_mais_cedo(RLact).

mais_cedo(Mac,[]):-findall(Ccn,es1(Mac,Ccn,1),Lccn),
                   controle(Mac,Lccn).

mais_cedo(Mac,[CNCa|RNCa]):-es1(Mac,CNCa,Xes1),
                            Xes1=:=0,
                            mais_cedo(Mac,RNCa);
                            precedentes_nodo(Mac,CNCa,Lnopre),
                            trata_nodo(Mac,0,CNCa,Lnopre,Lnopre),
                            mais_cedo(Mac,RNCa).
```

```
controle(Mac,[]):-!.
controle(Mac,Lccn):-sort(Lccn,SLccn),
                    mais_cedo(Mac,SLccn).

trata_nodo(Macc,Ze,Cnod,[Capre|Rapre],[Cp|Rp]):-es1(Macc,Capre,Es1flag),
                                                Au is Ze+Es1flag,
                                                ( (Au=\=0),! ;
                                                trata_nodo(Macc,Au,Cnod
                                                ,Rapre,[Cp|Rp])).

trata_nodo(Macc,_,Cnod,[],[Cp|Rp]):- retract(es1(Macc,Cnod,1)),
                                     asserta(es1(Macc,Cnod,0)),
                                     trata1_nodo(Macc,Cnod,[Cp|Rp],[],V),
                                     asserta(es(Macc,Cnod,V)).

trata1_nodo(Macc,Cnod,[Cp|Rp],LTRR,V):-es(Macc,Cp,VV),
                                       dura_min_arco(arco(Macc,Cp,
                                       Cnod,_),DV),
                                       LCp is VV+DV,
                                       trata1_nodo(Macc,Cnod,Rp,
                                       [LCp|LTRR],V).

trata1_nodo(Macc,Cnod,[],LTR,V):-sort(LTR,LTR1),
                                 reverse(LTR1,[V|V1]).


proc_es_especial:-lista_actividades(Lact),
                  especial_es(Lact).


especial_es([]).
especial_es([CLact|RLact]):- findall(arco(CLact,B,C,D),
                             arco(CLact,B,C,0),
                             Ltarefa0),
                          trataespecial(Ltarefa0),
                          especial_es(RLact).

trataespecial([]):-!.
trataespecial([arco(A,B,C,0)|Resto]):-es(A,B,EsB),
                                      es(A,C,EsC),
                                      retract(es(A,B,EsB)),
                                      asserta(es(A,B,EsC)),
                                      trataespecial(Resto).

/*   Fim de:                                                    */
/*    es(Actividade,Nodo,Inicio_mais_cedo_do_nodo)              */


/*   ########################################################   */
```

```
/*  ######################################################  */


/*    dura_min_activ(Actividade,Duracao_minima_da_actividade)          */

proc_dura_min:-lista_actividades(Lact),
               min_dura(Lact).


min_dura([]).
min_dura([CLact|RLact]):-lista_nodos(CLact,Lnod),
                         reverse(Lnod,[Ultno|Rno]),
                         es(CLact,Ultno,Mindura),
                         asserta(dura_min_activ(CLact,Mindura)),
                         min_dura(RLact).


/*   Fim de:                                                          */
/*    dura_min_activ(Actividade,Duracao_minima_da_actividade)         */


/*  ######################################################  */
```

```
/*  ######################################################      */

/*   tolerancia(Arco,Tolerancia_do_arco)                        */
/*   inicio_mais_tarde_arco(arco(A,B,C,D),Inicio_mais_tarde_    */
/*                                          possivel_do_arco    */


proc_tol:-lista_actividades(Lact),
          tolera(Lact).


tolera([]).
tolera([CLact|RLact]):-lista_arcos(CLact,Larc),
                       tole(Larc),
                       tolera(RLact).


tole([]).
tole([arco(CLact,N1,N2,Tar)|RLarc]):-es(CLact,N2,Es2),
                                     es(CLact,N1,Es1),
                          dura_min_arco(arco(CLact,N1,N2,_),Dum),
                                  AA is Es1+Dum,
                                  Tol is Es2-AA,
                asserta(tolerancia(arco(CLact,N1,N2,Tar),Tol)),
                                  Vlst is Tol+Es1,
                asserta(inicio_mais_tarde_arco(arco(CLact,N1,N2,Tar),Vlst)),
                                  tole(RLarc).


/*   Fim de:                                                    */
/*   tolerancia(Arco,Tolerancia_do_arco)                        */
/*   inicio_mais_tarde_arco(arco(A,B,C,D),Inicio_mais_tarde_    */
/*                                          possivel_do_arco    */


/*  ######################################################      */
```

```
/*  FIM DE :                                                          */
/*  INFERENCIA DE CONHECIMENTO ASSOCIADO AO CAMINHO                   */
/*  CRITICO DAS ACTIVIDADES                                           */
/*  ____________________________________________________              */
```

```
/*    INFERENCIA DE CONHECIMENTO ASSOCIADO AO RELAXAMENTO  */
/*    DAS REPETICOES DAS ACTIVIDADES                       */
/*    ____________________________________________________ */
```

```
/*   ######################################################      */


proc_relax:-lista_actividades(Lac),
            repete(Lac),
            relaxamento(Lac).


/*   ######################################################      */


/*     repeticoes_actividade(Actividade,Lista_das_suas_repeticoes)      */


repete([]):-!.
repete([CLacti|RLacti]):-findall(B,(dia_comeco(CLacti,B,_)),LB),
                         sort(LB,LBS),
                         asserta( repeticoes_actividade(CLacti,LBS) ),
                         repete(RLacti).

/*   Fim de:                                                      */
/*     repeticoes_actividade(Actividade,Lista_das_suas_repeticoes)      */


/*   ######################################################      */
```

```
/*  ######################################################      */

/*    rapidez(Actividade,Marca_da_actividade_ser_relaxada_ou_nao)      */
/*    inicio_+tarde_relaxado(arc(A,B,C,D,E,FG),Inicio_mais_tarde_      */
/*                           possivel_do_arc_da_repeticao_B_de_A_      */
/*                           relaxada                                  */

relaxamento([]):-!.
relaxamento([CLacti|RLacti]):-repeticoes_actividade(CLacti,Repeti),
                              relax(CLacti,Repeti),
                              relaxamento(RLacti).

relax(_,[]):-!.
relax(CLacti,[CRepeti|RRepeti]):-tempo_max(CLacti,CRepeti,Temax),
                                 dura_min_activ(CLacti,Temin),
                                 Delta is Temax-Temin,
                                 Delta>0 ->

                                   asserta(rapidez(CLacti,CRepeti,0)),
                                   lista_arcos(CLacti,Listarcos),
                                   tratamento(CLacti,CRepeti,Delta,Listarcos),
                                   relax(CLacti,RRepeti);
                                   asserta(rapidez(CLacti,CRepeti,1)),
                                   relax(CLacti,RRepeti).

tratamento(_,_,_,[]):-!.
tratamento(Act,Nact,Del,[arco(Act,N1,N2,T)|RListarcos]):-
      es(Act,N2,Es2),
      dia_comeco(Act,Nact,Com),
      Del1 is Del+Com,
      Es21 is Es2+Del1,
      dua_min_arco(arco(Act,N1,N2,T),Duramin),
      Mais_tarde is Es21-Duramin,
      asserta(inicio_+tarde_relaxado(arc(Act,Nact,N1,N2,T,F,_),Mais_tarde)),
      tratamento(Act,Nact,Del,RListarcos).

/*   Fim de:                                                         */
/*    rapidez(Actividade,Marca_da_actividade_ser_relaxada_ou_nao)    */
/*    inicio_+tarde_relaxado(arc(A,B,C,D,E,FG),Inicio_mais_tarde_    */
/*                           possivel_do_arc_da_repeticao_B_de_A_    */
/*                           relaxada                                */

/*######################################################      */
```

```
/*  FIM DE:                                                          */
/*  INFERENCIA DE CONHECIMENTO ASSOCIADO AO RELAXAMENTO              */
/*  DAS REPETICOES DAS ACTIVIDADES                                   */
/*  ________________________________________________________________ */
```

```
/*   INFERENCIA DE CONHECIMENTO GERAL PARA UTILIZACAO NA      */
/*   HEURISTICA                                               */
/*   ____________________________________________________     */
```

```
/*   ######################################################      */


/*    equipas_inicio(Inicialacao_a_zero)                        */

equipas_inicial:-ntarefas(Ktaref1),
                 inicial(1,Ktaref1,[],Ini),
                 asserta(equipas_inicio(Ini)).


inicial(Cont,Ktaref,Ini,Ini):-Cont>Ktaref,!.
inicial(Cont,Ktaref,Inaux,Ini):-append(Inaux,[equipa(Cont,0)],Inaux1),
                                Cont1 is Cont+1,
                                inicial(Cont1,Ktaref,Inaux1,Ini).


/*   Fim de:                                                    */
/*    equipas_inicio(Inicializacao_a_zero)                      */

/*   ######################################################      */
```

```
/*  ##################################################  */

/*    calendario(Dia,Lista_das_sobreposicoes_de_equipas_                */
/*                  criticas_nesse_dia)                                 */


inicio_calendario:-dia_final(Df),
                   equipas_inicio(L),
                   inicio_calen(Df,0,L).


inicio_calen(Df,Cont,L):-Cont>Df,!;
                         asserta(calendario(Cont,L)),
                         Cont1 is Cont+1,
                         inicio_calen(Df,Cont1,L).


carregar_calendario:-n_actividades(Nnum),
                     carregar(1,Nnum).


carregar(Contact,Nnum):-Contact>Nnum,!;
                        repeticoes_actividade(Contact,LRep),
                        carrega(Contact,LRep),
                        Contact1 is Contact+1,
                        carregar(Contact1,Nnum).


carrega(_,[]):-!.
carrega(Contact,[H|R]):-rapidez(Contact,H,Rap),
                        Rap=:=0,
                        carrega(Contact,R);
                        findall(arco(Contact,N1,N2,T),
                              tolerancia(arco(Contact,N1,N2,T),0),
                              Lcritical),
                        calendariza(H,Lcritical),
                        carrega(Contact,R).


calendariza(_,[]):-!.
calendariza(H,[arco(Act,N1,N2,T)|Rarco]):-es(Act,N1,Ves),
                                          dia_comeco(Act,H,Com),
                                          dura_min_arco(arco(Act,N1,N2,T),
                                          Dum),
                                          Pdia is Ves+Com,
                                          Udia1 is Pdia+Dum,
                                          Udia is Udia1-1,
                                          calen(Pdia,Udia,T),
                                          calendariza(H,Rarco).
```

```
calen(Pdia,Udia,T):-Pdia>Udia,!;
                    calendario(Pdia,L),
                    actua_calen(T,L,[],Lact),
                    retract(calendario(Pdia,L)),
                    asserta(calendario(Pdia,Lact)),
                    Pdia1 is Pdia+1,
                    calen(Pdia1,Udia,T).


actua_calen(T,[equipa(Ta,Veq)|Requipa],Equipaux,Equipact):-
        T=:=Ta,
        Veqactual is Veq+1,
        append(Equipaux,[equipa(T,Veqactual)],Lequipaux1),
        append(Lequipaux1,Requipa,Equipact),!;
        append(Equipaux,[equipa(Ta,Veq)],Equipaux1),
        actua_calen(T,Requipa,Equipaux1,Equipact).

/*  Fim de:                                                    */
/*   calendario(Dia,Lista_das_sobreposicoes_de_equipas_        */
/*                    criticas_nesse_dia)                      */


/*  ########################################################   */
```

```
/*  ################################################  */


/*    sobreposicoes (Dia,Lista_do_maximo_de_sobreposicoes_de          */
/*                   equipas_criticas_que_vao_ocorrer_                */
/*                   ate_ao_final_do_periodo_em_estudo)               */

calen_critical:-dia_final(Dfinal),
                ntarefas(Nt),
                critical(0,Dfinal,Nt).


critical(Diac,Dfinal,Nt):-Diac>Dfinal,!;
                          critic(Diac,1,Nt,[],Lexigencias),
                          asserta(sobreposicoes(Diac,Lexigencias)),
                          Diac1 is Diac+1,
                          critical(Diac1,Dfinal,Nt).


critic(_,Contarefas,Nt,Lexigencias,Lexigencias):-
                            Contarefas>Nt,!.
critic(Diac,Contarefas,Nt,Lexigaux,Lexigencias):-
                            findall(Val,
                                    (calendario(Dia,Lista),
                                     Dia>=Diac,
                                     member(equipa(Contarefas,Val),Lista)),
                                     Lvalor),
                            sort(Lvalor,Lvalor1),
                            (Lvalor1,[Maior|Resto]),
                            append(Lexigaux,[sobreposicao(Contarefas,Maior)],
                                    Lexigaux1),
                            Contarefas1 is Contarefas+1,
                            critic(Diac,Contarefas1,Nt,Lexigaux1,Lexigencias).


/*    Fim de:                                                         */
/*     sobreposicoes (Dia,Lista_do_maximo_de_sobreposicoes_de         */
/*                    equipas_criticas_que_vao_ocorrer_               */
/*                    ate_ao_final_do_periodo_em_estudo)              */


/*  ################################################  */
```

```
/*  ##################################################  */

/*   relaxadas    (Dia,Lista_de_contabilizacao_global_das_          */
/*                  equipas_nao_criticas_das_repeticoes_            */
/*                 das_actividades_que_nesse_dia_ainda_             */
/*                  nao_comecaram)                                  */
/*   criticas     (Dia,Lista_de_contabilizacao_global_das_          */
/*                     equipas_criticas_das_repeticoes_das          */
/*                     actividades_que_nesse_dia_ainda_nao          */
/*                     comecaram)                                   */


inicio_computo:-dia_final(Dfinal),
                inicio_compu(Dfinal,0).


inicio_compu(Dfinal,Contdia):-Contdia>Dfinal,!;
        lista_actividades(Lact),
        carregar_computo(Contdia,Lact,[],Lrelaxa,[],Lcrit),
        ntarefas(Ntarefas),
        agrupamento(1,Ntarefas,Lrelaxa,[],Lrelaxadas,
                                    Lcrit,[],Lcriticas),
        asserta(relaxadas(Contdia,Lrelaxadas)),
        asserta(criticas(Contdia,Lcriticas)),
        Contdia1 is Contdia+1,
        inicio_compu(Dfinal,Contdia1).


carregar_computo(_,[],Lrelaxa,Lrelaxa,Lcriti,Lcriti):-!.
carregar_computo(Contdia,[CLact|RLact],Lrelaux,Lrelaxa,
                                        Lcritaux,Lcriti):-

                repeticoes_actividade(CLact,Repet),
                carregar_compu(Contdia,CLact,Repet,Lrelaux,Lrelaux1,
                                    Lcritaux,Lcritaux1),
                carregar_computo(Contdia,RLact,Lrelaux1,Lrelaxa,
                                                Lcritaux1,Lcriti).
```

```
carregar_compu(_,_,[],Lrelaux1,Lrelaux1,Lcritaux1,Lcritaux1):-!.
carregar_compu(Contdia,CLact,[CRepet|RRepet],Lrelaux,Lrelaux1,
                                                    Lcritaux,Lcritaux1):-

        findall(arco(CLact,N1,N2,T),
                (dia_comeco(CLact,CRepet,Com),
                Com>=Contdia,
                rapidez(CLact,CRepet,Rap),
                Rap=:=0,
                tolerancia(arco(CLact,N1,N2,T),Tol),
                Tol=:=0),
                Lrelaux22),


        findall(arco(CLact,N1,N2,T),
                (dia_comeco(CLact,CRepet,Com),
                Com>=Contdia,
                rapidez(CLact,CRepet,Rap),
                Rap=:=0,
                tolerancia(arco(CLact,N1,N2,T),Tol),
                Tol=\=0),
                Lrelaux222),

         append(Lrelaux22,Lrelaux222,Lrelaux2),

         findall(arco(CLact,N1,N2,T),
                (dia_comeco(CLact,CRepet,Com),
                Com>=Contdia,
                rapidez(CLact,CRepet,Rap),
                Rap=:=1,
                tolerancia(arco(CLact,N1,N2,T),Tol),
                Tol=\=0),
                Lrelaux3),



        append(Lrelaux2,Lrelaux3,Lrelaux4),

        append(Lrelaux,Lrelaux4,Lrelaux5),
```

```
        findall(arco(CLact,N1,N2,T),
                (dia_comeco(CLact,CRepet,Com),
                 Com>=Contdia,
                 rapidez(CLact,CRepet,Rap),
                 Rap=:=1,
                 tolerancia(arco(CLact,N1,N2,T),Tol),
                 Tol=:=0),
                Lcritaux2),

        append(Lcritaux,Lcritaux2,Lcritaux3),


        carregar_compu(Contdia,CLact,RRepet,Lrelaux5,Lrelaux1,
                       Lcritaux3,Lcritaux1).

agrupamento(Contarefas,Ntarefas,_,Lrelaxadas,Lrelaxadas,
            _,Lcriticas,Lcriticas):-Contarefas>Ntarefas,!.

agrupamento(Contarefas,Ntarefas,Lrelaxa,Aux,Lrelaxadas,
            Lcriti,Baux,Lcriticas):-

            findall(arco(Act,N1,N2,T),
                    (member(arco(Act,N1,N2,T),Lrelaxa),
                     T==Contarefas),
                    Larcost),
            klength(Larcost,KLarcost),
            append(Aux,[relaxada(Contarefas,KLarcost)],Novaux),

            findall(arco(Act,N1,N2,T),
                    (member(arco(Act,N1,N2,T),Lcriti),
                     T==Contarefas),
                    Larcosto),
            klength(Larcosto,KLarcosto),
            append(Baux,[critica(Contarefas,KLarcosto)],Novbaux),

        Contarefas1 is Contarefas+1,
         agrupamento(Contarefas1,Ntarefas,Lrelaxa,Novaux,Lrelaxadas,
            Lcriti,Novbaux,Lcriticas).

/*   Fim de:                                                          */
/*    relaxadas    (Dia,Lista_de_contabilizacao_global_das_          */
/*                      equipas_nao_criticas_das_repeticoes_         */
/*                      das_actividades_que_nesse_dia_ainda_         */
/*                      nao_comecaram)                               */
/*    criticas     (Dia,Lista_de_contabilizacao_global_das_          */
/*                      equipas_criticas_das_repeticoes_das          */
/*                      actividades_que_nesse_dia_ainda_nao          */
/*                      comecaram)                                   */


/*   ########################################################       */
```

```
/*  FIM DE :                                                          */
/*  INFERENCIA DE CONHECIMENTO GERAL PARA UTILIZACAO NA               */
/*  HEURISTICA                                                        */
/*  ____________________________________________________________     */
```

```
/*  INFERENCIA DE CONHECIMENTO ASSOCIADO AO PROBLEMA      */
/*  DE PROGRAMACAO LINEAR                                 */
/*  ____________________________________________________  */
```

```
/*   ######################################################    */


ordem2:-trabs,
       trabs_taref,
       taref_equipas,
       menu_p_linear.


/*   ######################################################    */


/*   lista_trabalhadores(Lista_de_trabalhadores)               */


trabs:-findall(Trab,
            salario(Trab,_),
            Ltrab),
       asserta(lista_trabalhadores(Ltrab)).


/*   Fim de:                                                   */
/*   lista_trabalhadores(Lista_de_trabalhadores)               */


/*   ######################################################    */
```

```
/*   ###################################################   */

/*    trabalhador_tarefas(Trabalhador,Lista_de_tarefas_a_                */
/*                        que_esta_assignado)                            */

trabs_taref:-lista_trabalhadores(Ltrab),
        trab_tar(Ltrab).

trab_tar([]):-!.
trab_tar([CLtrab|RLtrab]):-findall(Taref,
                        ( tarefa(Taref,[_*CLtrab*_|_]) ,
                          Taref=\=0),

                        Ltaref),
                    asserta(trabalhador_tarefas(CLtrab,Ltaref)),
                   trab_tar(RLtrab).

/*   Fim de:                                                            */
/*    trabalhador_tarefas(Trabalhador,Lista_de_tarefas_a_               */
/*                        que_esta_assignado)                           */

/*   ###################################################   */

/*    n_equipas(Tarefa,Modularidade_de_equipas_da_tarefa)               */

taref_equipas:-ntarefas(NT1),
          taref_equi(1,NT1).

taref_equi(Cont,NT1):-Cont>NT1,!;
               tarefa(Cont,L),
               klength(L,KL),
                asserta(n_equipas(Cont,KL)),
               Cont1 is Cont+1,
               taref_equi(Cont1,NT1).

/*   Fim de:                                                            */
/*    n_equipas(Tarefa,Modularidade_de_equipas_da_tarefa)               */

/*   ###################################################   */
```

```
/*   ##################################################    */


/*     n_equacoes_restricao(Trabalhador,N_equacoes_de_restricao)      */
/*     n_variaveis(Trabalhador,N_de_variaveis)                        */
/*     upperbound(Trabalhador,Vector_de_upperbound)                   */
/*     custos(Trabalhador,Vector_de_custos)                           */
/*     matrix_a(Trabalhador,Matriz_a)                                 */


menu_p_linear:-lista_trabalhadores(Ltrab),
          menu(Ltrab).

menu([]):-!.
menu([CLtrab|RLtrab]):-

        trabalhador_tarefas(CLtrab,Ltaref),
        klength(Ltaref,KLtaref),
        Restricoes is KLtaref+1,
        asserta(n_equacoes_restricao(CLtrab,Restricoes)),

        ir(1,Restricoes,[],Lir),
        asserta(irtype(CLtrab,Lir)),

        nvariaveis(0,Ltaref,Nvar),
        asserta(n_variaveis(CLtrab,Nvar)),

        upp(1,Nvar,[],Lupp),
        asserta(upperbound(CLtrab,Lupp)),

        cust(Ltaref,[],Lc),
        asserta(custos(CLtrab,Lc)),

        geracao_a(CLtrab,Lmatrixa),
        asserta(matrix_a(CLtrab,Lmatrixa)),

        menu(RLtrab).


/*   Fim de:                                                          */
/*     n_equacoes_restricao(Trabalhador,N_equacoes_de_restricao)      */
/*     n_variaveis(Trabalhador,N_de_variaveis)                        */
/*     upperbound(Trabalhador,Vector_de_upperbound)                   */
/*     custos(Trabalhador,Vector_de_custos)                           */
/*     matrix_a(Trabalhador,Matriz_a)                                 */


/*   ##################################################    */
```

```
/*  ######################################################  */


/*  geracao do numero de variaveis                          */


nvariaveis(Nvar,[],Nvar):-!.

nvariaveis(Nvar1,[CLtaref|RLtaref],Nvar):-n_equipas(CLtaref,Neq),
                                Nvar2 is Nvar1+Neq,
                                nvariaveis(Nvar2,RLtaref,Nvar).


/*  Fim de:                                                 */
/*  geracao do numero de variaveis                          */


/*  ######################################################  */


/*  geracao do vector de upperbound                         */


upp(Cont,Nvar,Lupp,Lupp):-Cont>Nvar,!.

upp(Cont,Nvar,Luppaux,Lupp):-n_maximo_equipas(Nmax),
                        append(Luppaux,[Nmax],Luppaux1),
                        Cont1 is Cont+1,
                        upp(Cont1,Nvar,Luppaux1,Lupp).


/*  Fim de:                                                 */
/*  geracao do vector de upperbound                         */


/*  ######################################################  */
```

```
/*  ######################################################  */


/*  geracao do vector de custos                             */

cust([],Lc,Lc):-!.
cust([CLtaref|RLtaref],LCaux,Lc):-tarefa(CLtaref,Lesp),
                    cus(LCaux,Lesp,LCaux1),
                    cust(RLtaref,LCaux1,Lc).

cus(LCaux1,[],LCaux1):-!.
cus(LCaux2,[Nt*_*_|RLesp],LCaux1):-append(LCaux2,[Nt],LCaux3),
                    cus(LCaux3,RLesp,LCaux1).


/*  Fim de:                                                 */
/*  geracao do vector de custos                             */


/*  ######################################################  */


/*  geracao da matriz a                                     */


geracao_a(CLtrab,Lmatrixa):-trabalhador_tarefas(CLtrab,Ltaref),
                    n_variaveis(CLtrab,Nvar),
                    matrixa(Ltaref,Nvar,[],[],Lmatrixa).


matrixa([],_,_,Lmatrixa,Lmatrixa):-!.
matrixa([CLtaref|RLtaref],Nvar,Lzeraux,Lmatrixau,Lmatrixa):-

        tarefa(CLtaref,Leq),
        n_equipas(CLtaref,Neq),
        appendar(Lzeraux,1,Neq,Lzeraux1),
        appendar1(Leq,Lzeraux,Lmatrixau1),
        klength(Lmatrixau1,KL),
        KL1 is KL+1,
        appendar(Lmatrixau1,KL1,Nvar,Lmatrixau2),
        append(Lmatrixau,Lmatrixau2,Lmatrixau3),
        matrixa(RLtaref,Nvar,Lzeraux1,Lmatrixau3,Lmatrixa).
```

```
appendar(Lzeraux1,Cont,Neq,Lzeraux1):-Cont>Neq,!.
appendar(Lzeraux,Cont,Neq,Lzeraux1):-append(Lzeraux,[0],Lzeraux2),
                    Cont1 is Cont+1,
                     appendar(Lzeraux2,Cont1,Neq,Lzeraux1).


appendar1([],Lmatrixau1,Lmatrixau1):-!.
appendar1([_*_*Nd|RLeq],Lmatrixau,Lmatrixau1):-
            append(Lmatrixau,[Nd],Lmatrixau2),
            appendar1(RLeq,Lmatrixau2,Lmatrixau1).


/*   Fim de:                                                    */
/*   geracao da matriz a                                        */


/*   ########################################################   */

/*   geracao do vector irtype                                   */

ir(Cont,Restricoes,Lir,Lir):-Cont>Restricoes,!.
ir(Cont,Restricoes,Liraux,Lir):-append(Liraux,[2],Liraux1),
                                Cont1 is Cont+1,
                                ir(Cont1,Restricoes,Liraux1,Lir).


/*   Fim de:                                                    */
/*   geracao do vector irtype                                   */


/*   ########################################################   */
```

```
/* FIM DE:                                                    */
/* INFERENCIA DE CONHECIMENTO ASSOCIADO AO PROBLEMA           */
/* DE PROGRAMACAO LINEAR                                      */
/*  ____________________________________________________      */
```

```
/*   REGRAS DE PRODUCAO E ESTRATEGIA DE CONTROLE                    */
/*   ___________________________________________                    */
```

```
/*   ##################################################      */


/*   Geracao de todos os estados sucessores dum estado      */


ante_sucessao(Dia^Ltjf^Ltaf^Ltp^Trabto^Trabli,Listsuc):-
                                    findall(Acti,dia_comeco(Acti,_,Dia),Lact),
                                    sort(Lact,Lacti),
                                    trata_Acti(Dia,Lacti,[],Ltpff),
                                    append(Ltp,Ltpff,Ltpf),
                  sucessao(Ltp,Dia^Ltjf^Ltaf^Ltpf^Trabto^Trabli,[],Listsuc0),
                  klength(Listsuc0,Kl),
                  eliminar(Listsuc0,Kl,Listsuc).


/*   Fim de:                                                */
/*   Geracao de todos os estados sucessores dum estado      */

/*   ##################################################      */


/*   Carregar todos os arcos das repeticoes das actividades */
/*   que comecam nesse dia na lista das tarefas por fazer L3*/


trata_Acti(_,[],Ltpff,Ltpff):-!.
trata_Acti(Dia,[CLacti|RLacti],Laux,Ltpff):-
                                    findall(arco(CLacti,N1,N2,T),
                                            arco(CLacti,N1,N2,T),
                                            Lar),
                                    findall(Nacti,
                                            dia_comeco(CLacti,Nacti,Dia),
                                            LNacti),
                                    trata1_Acti(Lar,LNacti,[],Lx),
                                    append(Laux,Lx,Ltpfx),
                                    trata_Acti(Dia,RLacti,Ltpfx,Ltpff).


/*   Fim de:                                                */
/*   Carregar todos os arcos das repeticoes das actividades */
/*   que comecam nesse dia na lista das tarefas por fazer L3*/


/*   ##################################################      */
```

```
/*   ######################################################   */

/*   Transformar os arcos carregados trata_Acti nos seus    */
/*   "arcs" correspondentes                                  */

trata1_Acti([],_,Lx,Lx):-!.
trata1_Acti([arco(A,N1,N2,T)|Rarco],[CNacti|RNacti],Lax,Lx):-
                    trata2_Acti(arco(A,N1,N2,T),[CNacti|RNacti],[],Laxx),
                    append(Lax,Laxx,Laxxx),
                    trata1_Acti(Rarco,[CNacti|RNacti],Laxxx,Lx).


trata2_Acti(_,[],Larc,Larc):-!.
trata2_Acti(arco(A,N1,N2,T),[CNacti|RNacti],Lar,Larc):-
                    append(Lar,[arc(A,CNacti,N1,N2,T,F,G)],Larr),
                    trata2_Acti(arco(A,N1,N2,T),RNacti,Larr,Larc).


/*   Fim de:                                                 */
/*   Transformar os arcos carregados trata_Acti nos seus     */
/*   "arcs" correspondentes                                  */

/*   ######################################################   */


/*   Eliminar dos estados sucessores os que tiverem uma      */
/*   lista de tarefas a fazer,L2, vazia .                    */

eliminar(Listsuc0,0,Listsuc0):-!.
eliminar(Listsuc0,1,Listsuc0):-!.
eliminar(Listsuc0,Kl,Listsuc):-
 Kl>1 ,
 findall( ( s(Dia^Ltjf^Ltaf^Ltp^Trabto^Trabli,
              _^_^[]^_^_^_,Cu) ),
    member( ( s(Dia^Ltjf^Ltaf^Ltp^Trabto^Trabli,_^_^[]^_^_^_,Cu) ),Listsuc0),
            Listsuc1),
  subtract(Listsuc0,Listsuc1,Listsuc).


/*   Fim de:                                                 */
/*   Eliminar dos estados sucessores os que tiverem uma      */
/*   lista de tarefas a fazer,L2, vazia .                    */


/*   ######################################################   */
```

```
/*   ################################################   */


/*   sucessao(Lista_das_novas_tarefas_por_fazer,Estado,   */
/*                Lista_dos_estados_sucessores)          */


sucessao(Ltp,Dia^Ltjf^Ltaf^Ltpf^Trabto^Trabli,Sucaux,Suc):-

Dia1 is Dia+1,



/*   Descarregar as tarefas que terminam nesse dia da lista   */
/*   das tarefas a fazer,L2, e carrega-las na lista das tarefas   */
/*   ja feitas,L1.                                            */

findall(arc(Act,NAct,N1,N2,T,Dia,Oc), (
                                member(arc(Act,NAct,N1,N2,T,Dia,Oc),Ltaf) ),
      Ltjfei),

findall(arc(Act,NAct,N1,N2,0,Di,0), (
                                member(arc(Act,NAct,N1,N2,0,Di,0),Ltaf),
                                Di=<Dia ),
      Ltjfei0),


desocupar(Ltjfei,Trabli,Trabli1),

append(Ltjfei,Ltjfei0,Ltjfei1),

append(Ltjf,Ltjfei1,Ltjf10),

subtract(Ltaf,Ltjfei1,Ltaf2),


/*   Fim de:                                                  */
/*   Descarregar as tarefas que terminam nesse dia da lista   */
/*   das tarefas a fazer,L2, e carrega-las na lista das tarefas   */
/*   ja feitas,L1.                                            */
```

```
/*   TRATAMENTO DOS "ARCS" DAS REPETICOES NAO RELAXADAS      */
/*   DAS ACTIVIDADES                                         */


/*    Geracao da lista,Larcan, de "arcs" passiveis de comecarem      */
/*   a ser feitos                                                    */


findall(arc(Act,NAct,Nod,Aa,Bb,Cc,_), (
                                    rapidez(Act,NAct,Rap),
                                    Rap=:=1,
                                    es(Act,Nod,Ves),
                                    dia_comeco(Act,NAct,Com),
                                    Ves_des is Ves+Com,
                                    Ves_des=<Dia,
                                    member( arc(Act,NAct,Nod,Aa,
                                    Bb,Cc,_),Ltpf) ),
     Larcan),


/*   Fim de:                                                         */
/*    Geracao da lista,Larcan, de "arcs" passiveis de comecarem      */
/*   a ser feitos                                                    */


/*    Geracao da lista ,Larafa,de "arcs" que obrigatoriamente        */
/*    teem de comecar a ser feitos por pertencerem ao caminho        */
/*   critico                                                         */

findall(arc(Act,NAct,Nod,Aa,Bb,Cc,_), ( tolerancia(arco(Act,Nod,Aa,Bb),0),
                                    member(arc(Act,NAct,Nod,Aa,
                                    Bb,Cc,_),Larcan) ) ,
      Larafa),


/*   Fim de:                                                         */
/*    Geracao da lista ,Larafa,de "arcs" que obrigatoriamente        */
/*    teem de comecar a ser feitos por pertencerem ao caminho        */
/*   critico                                                         */
```

```
/*    Descarregar Larafa da lista de tarefas por fazer,L3,          */
/*    e carrega-la na lista de tarefas a fazer,L2.                  */


trata(Larafa,Dia,Laraf1),
actualiza_tot_liv( Laraf1,Trabto,Trabto1,Trabli1,Trabli2 ),

append(Ltaf2,Laraf1,Ltaf3),

subtract(Ltpf,Larafa,Ltpf2),
subtract(Larcan,Larafa,Lrest),


/*    Fim de:                                                       */
/*    Descarregar Larafa da lista de tarefas por fazer,L3,          */
/*    e carrega-la na lista de tarefas a fazer,L2.                  */


/*    Geracao da lista ,Lobri,de "arcs" que obrigatoriamente        */
/*    teem de comecar a ser feitos apesar de nao pertencerem        */
/*    ao caminho,mas que teem o inicio mais tarde possivel          */
/*    nesse dia                                                     */


findall(arc(Act,NAct,Nod,Aa,Bb,Cc,_),
        ( inicio_mais_tarde_arco(arco(Act,Nod,Aa,Bb),Ini_tar),
        dia_comeco(Act,NAct,Com),
        In_dia is Ini_tar+Com,
        In_dia=:=Dia,
        member(arc(Act,NAct,Nod,Aa,Bb,Cc,_),Lrest) ) ,
      Lobri),


/*    Fim de:                                                       */
/*    Geracao da lista ,Lobri,de "arcs" que obrigatoriamente        */
/*    teem de comecar a ser feitos apesar de nao pertencerem        */
/*    ao caminho,mas que teem o inicio mais tarde possivel          */
/*    nesse dia                                                     */
```

```
/*    Descarregar Lobri da lista de tarefas por fazer,L3,                */
/*    e carrega-la na lista de tarefas a fazer,L2.                       */


trata(Lobri,Dia,Lobri1),

actualiza_tot_liv( Lobri1,Trabto1,Trabto2,Trabli2,Trabli3 ),

append(Ltaf3,Lobri1,Ltaf1),


/*    Fim de:                                                            */
/*    Descarregar Lobri da lista de tarefas por fazer,L3,                */
/*    e carrega-la na lista de tarefas a fazer,L2.                       */


/*    Geracao da lista ,Lrest1,de "arcs" que podem comecar a             */
/*    ser feitos sem obrigatoriedade                                     */


subtract(Lrest,Lobri,Lrest1),

subtract(Ltpf2,Lobri,Ltpf1),


/*    Fim de:                                                            */
/*    Geracao da lista ,Lrest1,de "arcs" que podem comecar a             */
/*    ser feitos sem obrigatoriedade                                     */


/*    FIM DE:                                                            */
/*    TRATAMENTO DOS "ARCS" DAS REPETICOES NAO  RELAXADAS                */
/*    DAS ACTIVIDADES                                                    */


/*    ###################################################              */
```

```
/*  TRATAMENTO DOS "ARCS" DAS REPETICOES  RELAXADAS DAS      */
/*   ACTIVIDADES                                             */


/*    Geracao da lista ,Larcanlento,dos "arcs" das repeticoes */
/*    relaxadas das actividades que teem um inicio mais cedo  */
/*    possivel inferior ou igual a este dia                   */

findall(arc(Act,NAct,Nod,Aa,Bb,Cc,_),
        ( rapidez(Act,NAct,Rap),
        Rap=:=0,
        es(Act,Nod,Ves),
        dia_comeco(Act,NAct,Com),
        Ves_des is Ves+Com,
        Ves_des=<Dia,
        member(arc(Act,NAct,Nod,Aa,Bb,Cc,_),Ltpf1) ),
        Larcanlento ),


/*    Fim de:                                                 */
/*    Geracao da lista ,Larcanlento,dos "arcs" das repeticoes */
/*    relaxadas das actividades que teem um inicio mais cedo  */
/*    possivel inferior ou igual a este dia                   */


/*    Geracao da lista ,Lobrilento,dos "arcs" das repeticoes  */
/*    relaxadas das actividades que teem obrigatoriamente de  */
/*    comecar a ser feitos por terem um inicio mais tarde     */
/*    relaxado igual a este dia                               */

findall(arc(Act,NAct,Nod,Aa,Bb,Cc,_),
        ( inicio_+tarde_relaxado( arc(Act,NAct,Nod,Aa,Bb,Cc,_), Dia),
        member(arc(Act,NAct,Nod,Aa,Bb,Cc,_),Larcanlento) ),
      Lobrilento),

/*    Fim de:                                                 */
/*    Geracao da lista ,Lobrilento,dos "arcs" das repeticoes  */
/*    relaxadas das actividades que teem obrigatoriamente de  */
/*    comecar a ser feitos por terem um inicio mais tarde     */
/*    relaxado igual a este dia                               */
```

```
/*    Descarregar Lobri da lista de tarefas por fazer,L3,                */
/*    e carrega-la na lista de tarefas a fazer,L2.                       */


trata(Lobrilento,Dia,Lobrilento1),

actualiza_tot_liv(Lobrilento1,Trabto2,Trabto3,Trabli3,Trabli4),

append(Ltaf1,Lobrilento1,Ltaf41),

subtract(Ltpf1,Lobrilento,Ltpf30),


/*    Fim de:                                                            */
/*    Descarregar Lobri da lista de tarefas por fazer,L3,                */
/*    e carrega-la na lista de tarefas a fazer,L2.                       */


/*    Geracao da lista ,Lrest3,dos "arcs" que podem comecar              */
/*    a ser feitos sem no entanto ser obrigatorio o seu comeco           */


subtract(Larcanlento,Lobrilento,Lrest2),


precedencias_realizadas(Lrest2,Ltjf10,[],Lrest30),


      findall(arc(Act,Nact,N1,N2,0,_,_),
            member( arc(Act,Nact,N1,N2,0,_,_), Lrest30 ),
            Lrest31 ),

trata(Lrest31,Dia,Lrest32),

subtract(Lrest30,Lrest31,Lrest3),


/*    Fim de:                                                            */
/*    Geracao da lista ,Lrest3,dos "arcs" que podem comecar              */
/*    a ser feitos sem no entanto ser obrigatorio o seu comeco           */
```

```
append(Ltaf41,Lrest32,Ltaf40),
subtract(Ltpf30,Lrest31,Ltpf3),


/*    Geracao da lista ,Lrest4,do total dos "arcs" que podem          */
/*    comecara ser feitos sem no entanto ser obrigatorio o            */
/*    seu comeco                                                      */

append(Lrest1,Lrest3,Lrest4),

/*    Fim de:                                                         */
/*    Geracao da lista ,Lrest4,do total dos "arcs" que podem          */
/*    comecara ser feitos sem no entanto ser obrigatorio o            */
/*    seu comeco                                                      */

      findall(arc(Act,Nact,N1,N2,0,F,G),
            member( arc(Act,Nact,N1,N2,0,F,G), Ltaf40 ),
            Lzero ),

subtract(Ltaf40,Lzero,Ltaf4),
append(Ltjf10,Lzero,Ltjf1),


/*  FIM DE:                                                           */
/*  TRATAMENTO DOS "ARCS" DAS REPETICOES  RELAXADAS DAS               */
/*  ACTIVIDADES                                                       */


/*    Geracao da lista ,Suc,lista dos estados sucessores              */
/*    dum estado                                                      */


combina(Dia,Ltjf,Ltaf,Ltp,Trabto,Trabli,
          Dia1,Ltjf1,Ltaf4,Ltpf3,Trabto3,Trabli4,Lrest4,Sucaux,Suc).


/*    Fim de:                                                         */
/*    Geracao da lista ,Suc,lista dos estados sucessores              */
/*    dum estado                                                      */


/*    ##################################################            */
```

```
/*  ##################################################  */


/*  Por os trabalhadores que estavam assignados nas tarefas  */
/*  que acabaram na lista dos trabalhadores livres           */


desocupar([],Livres2,Livres2):-!.
desocupar([arc(_,_,_,_,Ta,_,Ntra)|RLtjfei],Livres1,Livres2):-
        tarefa(Ta,[_*Tipo*_|_]),
        actualiza_livres(Tipo,Ntra,Livres1,[],Livresactuais),
        desocupar(RLtjfei,Livresactuais,Livres2).


actualiza_livres(_,_,[],Livresactuais,Livresactuais):-!.
actualiza_livres(Tipo,Ntra,[livres(T1,N1)|RLivres],Livresaux1,Livresactuais):-
      Tipo==T1,
      Nlivres1 is Ntra+N1,
      append(Livresaux1,[livres(T1,Nlivres1)],Livresaux2),
      append(Livresaux2,RLivres,Livresactuais),!;
      append(Livresaux1,[livres(T1,N1)],Livresaux2),
      actualiza_livres(Tipo,Ntra,RLivres,Livresaux2,Livresactuais).

/*  Fim de:                                                  */
/*   Por os trabalhadores que estavam assignados nas tarefas */
/*   que acabaram na lista dos trabalhadores livres          */


/*  ##################################################  */
```

```
/*  ################################################  */


/*    Por marcas F e G nos arc(A,B,C,D,E,F,G) ,nos arcs que      */
/*    vao ser realizados no menor tempo possivel ou seja com     */
/*    a maior equipa                                             */


trata([],_,[]):-!.
trata([arc(Act,NAct,N1,N2,Tar,F,_)|RLarafa],Dia,
      [arc(Act,NAct,N1,N2,Tar,Fa,Mtra)|Rlaraf1]):-
dura_min_arco(arco(Act,N1,N2,Tar),Ff),
max_trab_arco(arco(Act,N1,N2,Tar),Mtra),
Fa is Dia+Ff,
trata(RLarafa,Dia,Rlaraf1).


/*    Fim de:                                                    */
/*    Por marcas F e G nos arc(A,B,C,D,E,F,G) ,nos arcs que      */
/*    vao ser realizados no menor tempo possivel ou seja com     */
/*    a maior equipa                                             */

/*  ################################################  */
```

```
/*   ################################################   */


/*   Actualizacao da lista dos trabalhadores livres e da   */
/*   lista do total dos trabalhadores ,quando se comeca a   */
/*   realizar uma tarefa                                    */

actualiza_tot_liv([],Trabto1,Trabto1,Trabli1,Trabli1):-!.
actualiza_tot_liv([arc(_,_,_,_,Ta,_,Ntra)|Rlis],
        Trabto,Trabto1,Trabli,Trabli1):-

      tarefa(Ta,[_*Tipo*_|_]),
      tot_liv(Tipo,Ntra,Trabto,[],Trabto2,Trabli,[],Trabli2),
      actualiza_tot_liv(Rlis,Trabto2,Trabto1,Trabli2,Trabli1).


tot_liv(_,_,[],Trabto2,Trabto2,[],Trabli2,Trabli2):-!.
tot_liv(Tipo,Ntra,[total(T1,N1)|RTrabto],Totaux,Trabto2,
                [livres(T1,N2)|RTrabli],Livaux,Trabli2):-


Tipo==T1 ->

( Restan is N2-Ntra,
 Restan>=0 ->
 append(Totaux,[total(T1,N1)],Totaux1),
 append(Totaux1,RTrabto,Trabto2),
 append(Livaux,[livres(T1,Restan)],Livaux1),
 append(Livaux1,RTrabli,Trabli2),! ;
 Rest1 is Ntra-N2,
 N11 is Rest1+N1,
 append(Totaux,[total(T1,N11)],Totaux1),
 append(Totaux1,RTrabto,Trabto2),
 append(Livaux,[livres(T1,0)],Livaux1),
 append(Livaux1,RTrabli,Trabli2),! ) ;

 append(Totaux,[total(T1,N1)],Totaux1),
 append(Livaux,[livres(T1,N2)],Livaux1),
 tot_liv(Tipo,Ntra,RTrabto,Totaux1,Trabto2,RTrabli,Livaux1,Trabli2).


/*   fim de :                                               */
/*    Actualizacao da lista dos trabalhadores livres e da   */
/*   lista do total dos trabalhadores ,quando se comeca a   */
/*   realizar uma tarefa                                    */

/*   ################################################   */
```

```
/*   ################################################    */


/*   Seleccionar duma lista de arcos candidatos a comecar      */
/*   a serem realizados,os que estao em condicao de o serem    */
/*   por terem os seus "arcs" precedentes ja realizados        */


precedencias_realizadas([],_,Lpes,Lpes):-!.
precedencias_realizadas([arc(Act,NAct,N1,N2,T,F,G)|Rarc],Ltjf1,Lpesaux,
Lpes):-
        precedentes_nodo(Act,N1,Lprec),
        carregar_prec(Act,NAct,N1,Lprec,[],Arcospre),
        verificar_precedencias(Arcospre,Ltjf1,Sinal,1,0),
        Sinal=:=1 ->

            precedencias_realizadas(Rarc,Ltjf1,Lpesaux,Lpes);
            insert(arc(Act,NAct,N1,N2,T,F,G),Lpesaux,Lpesaux1),
            precedencias_realizadas(Rarc,Ltjf1,Lpesaux1,Lpes).

/*   fim de :                                                  */
/*   Seleccionar duma lista de arcos candidatos a comecar      */
/*   a serem realizados,os que estao em condicao de o serem    */
/*   por terem os seus "arcs" precedentes ja realizados        */


/*   ################################################    */
```

```
/*  ##################################################  */


/*  Carregar numa lista os "arc" precedentes a um nodo  */

carregar_prec(Act,NAct,N1,[],Arcospre,Arcospre).
carregar_prec(Act,NAct,N1,[N1P|RN1P],Arcospreaux,Arcospre):-
        arco(Act,N1P,N1,T),
        T=\=0,
            insert(arc(Act,NAct,N1P,N1,_,_,_),Arcospreaux,Arcospreaux1),
            carregar_prec(Act,NAct,N1,RN1P,Arcospreaux1,Arcospre) .

carregar_prec(Act,NAct,N1,[N1P|RN1P],Arcospreaux,Arcospre):-
        arco(Act,N1P,N1,T),
        T=:=0 ,
            precedentes_nodo(Act,N1P,L1),
            carregar_preca(Act,NAct,N1P,L1,[],Laux),
            append(Arcospreaux,Laux,Arcospreaux1),
            carregar_prec(Act,NAct,N1,RN1P,Arcospreaux1,Arcospre).


carregar_preca(_,_,_,[],Arcospre,Arcospre):-!.
carregar_preca(Act,NAct,N1,[N1P|RN1P],Arcospreaux,Arcospre):-
            insert(arc(Act,NAct,N1P,N1,_,_,_),Arcospreaux,Arcospreaux1),
            carregar_preca(Act,NAct,N1,RN1P,Arcospreaux1,Arcospre).


/*  fim de :                                            */
/*  Carregar numa lista os "arc" precedentes a um nodo  */


/*  ##################################################  */


/*  Verificar se todos os "arc" duma lista ja estao     */
/*  realizados                                          */


verificar_precedencias([],_,0,_,_):-!.
verificar_precedencias([arc(Act,NAct,N1P,N1,_,_,_)|Rarc],Ltjf1,Si,A,B):-
                member(arc(Act,NAct,N1P,N1,_,_,_),Ltjf1) ->
                verificar_precedencias(Rarc,Ltjf1,Si,A,B);
                Si is A+B,!.


/*  Fim de :                                            */
/*  Verificar se todos os "arc" duma lista ja estao     */
/*  realizados                                          */


/*  ##################################################  */
```

```
/*  ################################################  */


/*   Preparacao da geracao de todos os estados sucessores          */
/*   dum estado:                                                   */
/*   Com a lista das tarefas que estao em condicoes de             */
/*   comecarem a ser feitas embora sem obrigatoriedade do seu      */
/*   comeco, Lrestp ,realizar:                                     */
/*                                                                 */
/*    1.-Formar uma lista de listas, LLco ,em que cada elemento    */
/*       da lista e uma combinacao das tarefas que constituem      */
/*       a lista Lrestp.                                           */
/*    2.-Desdobrar a lista de listas LLco, de acordo com as        */
/*       possibilidades de realizacao de cada tarefa assignada     */
/*       aos "arcs" das listas de LLco, gerando a lista Suc.       */


combina(Da,Ltjfa,Ltafa,Ltpfa,Trabtoa,Trablia,
        Dp,Ltjfp,Ltafp,Ltpfp,Trabtop,Trablip,[],Sucaux,Suc):-
                custo(Trabtoa,Trabtop,0,C),
            append([ s(Da^Ltjfa^Ltafa^Ltpfa^Trabtoa^Trablia,
                       Dp^Ltjfp^Ltafp^Ltpfp^Trabtop^Trablip,C)],Sucaux,
                       Suc ),!.


combina(Da,Ltjfa,Ltafa,Ltpfa,Trabtoa,Trablia,
        Dp,Ltjfp,Ltafp,Ltpfp,Trabtop,Trablip,Lrestp,Sucaux,Suc):-
                  findall(Lco,combinations(Lrestp,Lco),LLco),
                  desdobramento(Da,LLco,[[]],LLco1),
                  append(LLco1,[[]],LLco2),

            combi(Da,Ltjfa,Ltafa,Ltpfa,Trabtoa,Trablia,
                  Dp,Ltjfp,Ltafp,Ltpfp,Trabtop,Trablip,LLco2,Sucaux,
                  Suc).


/*   Fim de:                                                       */
/*   Preparacao da geracao de todos os estados sucessores          */
/*   dum estado:                                                   */
/*   Com a lista das tarefas que estao em condicoes de             */
/*   comecarem a ser feitas embora sem obrigatoriedade do seu      */
/*   comeco, Lrestp ,realizar:                                     */
/*                                                                 */
/*    1.-Formar uma lista de listas, LLco ,em que cada elemento    */
/*       da lista e uma combinacao das tarefas que constituem      */
/*       a lista Lrestp.                                           */
/*    2.-Desdobrar a lista de listas LLco, de acordo com as        */
/*       possibilidades de realizacao de cada tarefa assignada     */
/*       aos "arcs" das listas de LLco, gerando a lista Suc.       */


/*  ################################################  */
```

```
/*   ######################################################    */


/*   Desdobrar uma lista de listas de "arcs". Cada lista de    */
/*   "arcs" vai ser desdobrada em conformidade com as          */
/*   alternativas possiveis para realizar cada um dos "arc"    */
/*   da lista. Desdobram-se todas as listas de "arcs" que      */
/*   compoem a lista de listas de "arcs".                      */


desdobramento(_,[_|[]],LLco1des,LLco1des):-!.

desdobramento(Daa,[[HHLLco1|RHLLco1]|RLLco1],LLco1aux,LLco1des):-
     desdobram(Daa,[HHLLco1|RHLLco1],[[HHLLco1|RHLLco1]],
     Ldesdobram2),
     append(Ldesdobram2,[[]],Ldesdobram1),
     geral(Ldesdobram1,Ldesdobram),
     append1(Ldesdobram,LLco1aux,LLco1aux1),
     desdobramento(Daa,RLLco1,LLco1aux1,LLco1des).


/*   Fim de:                                                   */
/*    Desdobrar uma lista de listas de "arcs". Cada lista de   */
/*    "arcs" vai ser desdobrada em conformidade com as         */
/*    alternativas possiveis para realizar cada um dos "arc"   */
/*    da lista. Desdobram-se todas as listas de "arcs" que     */
/*    compoem a lista de listas de "arcs".                     */


/*   ######################################################    */
```

```
/*   ################################################   */


/*   Desdobrar uma lista de "arcs". A lista de "arcs" vai   */
/*   ser desdobrada em conformidade com as alternativas   */
/*   possiveis para realizar as tarefas assignadas aos   */
/*   "arcs" que compoem a lista de "arcs". A lista de "arcs"   */
/*   vai gerar uma lista de listas de "arcs".   */


desdobram(_,[],Lam1,Lam1):-!.

desdobram(Diaa,[arc(Act,Nact,N1,N2,T,_,_)|RA],Lam,Lam1):-
          desdobra(Diaa,arc(Act,Nact,N1,N2,T,_,_),Ldesd),
          append(Lam,[[]],Lam0),
          desdobraux(Ldesd,Lam0,[[]],Lds),
          desdobram(Diaa,RA,Lds,Lam1).


/*   Fim de:   */
/*   Desdobrar uma lista de "arcs". A lista de "arcs" vai   */
/*   ser desdobrada em conformidade com as alternativas   */
/*   possiveis para realizar as tarefas assignadas aos   */
/*   "arcs" que compoem a lista de "arcs". A lista de "arcs"   */
/*   vai gerar uma lista de listas de "arcs".   */


/*   ################################################   */
```

```
/*   ##################################################   */


/*   Desdobrar um "arc" em todas os "arcs" possiveis de acordo   */
/*   com as diferentes possibilidades de realizacao da tarefa    */
/*   assignada ao "arc".                                         */


desdobra(Diab,arc(Act,NAct,N1,N2,T,_,_),Ldes):-tempo_max(Act,NAct,
Temax),
                                                dura_min_activ(Act,Temin),
                                                Delta is Temax-Temin,
                                                dia_comeco(Act,NAct,Com),
                                                Delta1 is Delta+Com,
                                                es(Act,N2,Es),
                                                Es2 is Delta1+Es,
                                                tarefa(T,Lta),
        desdobra1(Diab,Es2,arc(Act,NAct,N1,N2,T,_,_),Lta,[],Ldes).

/*   ##################################################   */


desdobra1(_,_,_,[],LLdes,LLdes):-!.

desdobra1(Diab,Es_des,arc(Act,NAct,N1,N2,T,_,_),
                        [NT*Tiptr*NDias|RLta],Ltaux,LLdes):-
Aux is NDias+Diab,
Aux>Es_des,
desdobra1(Diab,Es_des,arc(Act,NAct,N1,N2,T,_,_),RLta,Ltaux,LLdes);
Aux is NDias+Diab,
Aux=<Es_des,
insert(arc(Act,NAct,N1,N2,T,Aux,NT),Ltaux,Ltaux1),
desdobra1(Diab,Es_des,arc(Act,NAct,N1,N2,T,_,_),RLta,Ltaux1,LLdes).


/*   Fim de:                                                     */
/*   Desdobrar um "arc" em todas os "arcs" possiveis de acordo   */
/*   com as diferentes possibilidades de realizacao da tarefa    */
/*   assignada ao "arc".                                         */


/*   ##################################################   */
```

```
/*  ################################################  */


/*  Desdobrar uma lista de listas de "arcs" em conformidade  */
/*  com as alternativas de realizacao da tarefa assignada    */
/*  a um "arc" das listas da lista de "arcs".                */


desdobraux(_,[_|[]],Ldsa,Ldsa):-!.

desdobraux(Ldesda,[[Hla|THla]|Tla],Ldsaux,Ldsa):-
        desdobrar(Ldesda,[Hla|THla],[[]],Ldesdaux),
        append1(Ldesdaux,Ldsaux,Ldesdaux1),
        desdobraux(Ldesda,Tla,Ldesdaux1,Ldsa).


/*  Fim de:                                                  */
/*  Desdobrar uma lista de listas de "arcs" em conformidade  */
/*  com as alternativas de realizacao da tarefa assignada    */
/*  a um "arc" das listas da lista de "arcs".                */

/*  ################################################  */


/*  Desdobrar uma lista de "arcs",numa lista de listas de    */
/*  "arcs", que proveem do desdobramento dum "arc" da lista  */
/*  de "arcs" original.                                      */


desdobrar([],_,Ldesdobrar,Ldesdobrar):-!.

desdobrar([arc(Act,Nact,N1,N2,T,F,G)|RLar],H,Ldesdobra,Ldesdobrar):-
        subst(arc(Act,Nact,N1,N2,T,F,G),H,[],Lsu),
        append1([Lsu],Ldesdobra,Lsu1),
        desdobrar(RLar,H,Lsu1,Ldesdobrar).


/*  Fim de:                                                  */
/*  Desdobrar uma lista de "arcs",numa lista de listas de    */
/*  "arcs", que proveem do desdobramento dum "arc" da lista  */
/*  de "arcs" original.                                      */


/*  ################################################  */
```

```
/*   ################################################   */


/*   Substituir um arc(A,B,C,D,E,F,G),  com as marcas F e G                */
/*   ja determinadas numa cadeia de "arcs".                                */


subst(_,[],_,[]):-!.

subst(arc(Act,Nact,N1,N2,T,F,G),
      [arc(Act1,Nact1,N11,N21,T1,F1,G1)|RR],Lsubaux,Lsubst):-
Act=:=Act1,
Nact=:=Nact1,
N1=:=N11,
N2=:=N21,
T=:=T1,
append(Lsubaux,[arc(Act,Nact,N1,N2,T,F,G)],Lsubaux1),
append(Lsubaux1,RR,Lsubst),!;
append(Lsubaux,[arc(Act1,Nact1,N11,N21,T1,F1,G1)],Lsubaux1),
subst(arc(Act,Nact,N1,N2,T,F,G),RR,Lsubaux1,Lsubst).


/*   Fim de:                                                               */
/*   Substituir um arc(A,B,C,D,E,F,G),  com as marcas F e G                */
/*   ja determinadas numa cadeia de "arcs".                                */


/*   ################################################   */
```

```
/*   ##################################################    */

/*   Geracao de todos os estados sucessores dum estado                */

combi(Da,Ltjfa,Ltafa,Ltpfa,Trabtoa,Trablia,
        Dp,Ltjfp,Ltafp,Ltpfp,Trabtop,Trablip,[[]],Sucaux2,Suc):-
                        custo(Trabtoa,Trabtop,0,C),
          append([ s(Da^Ltjfa^Ltafa^Ltpfa^Trabtoa^Trablia,
                     Dp^Ltjfp^Ltafp^Ltpfp^Trabtop^Trablip,C)],Sucaux2,
                     Suc ),!.

combi(Da,Ltjfa,Ltafa,Ltpfa,Trabtoa,Trablia,
        Dp,Ltjfp,Ltafp,Ltpfp,Trabtop,Trablip,[[H|HR]|T],Sucaux,Suc):-

              actualiza_tot_liv([H|HR],Trabtop,Trabtopp,Trablip,Trablipp),
              append([H|HR],Ltafp,Ltaf1b),
              desini([H|HR],HDESI),
              subtract(Ltpfp,HDESI,Ltpf1b),

                        custo(Trabtoa,Trabtopp,0,C),
          append( [ s(Da^Ltjfa^Ltafa^Ltpfa^Trabtoa^Trablia,
                     Dp^Ltjfp^Ltaf1b^Ltpf1b^Trabtopp^Trablipp,C)],Sucaux,
                     Sucaux1 ),
          combi(Da,Ltjfa,Ltafa,Ltpfa,Trabtoa,Trablia,
                 Dp,Ltjfp,Ltafp,Ltpfp,Trabtop,Trablip,T,Sucaux1,Suc).


/*   Fim de:                                                          */
/*   Geracao de todos os estados sucessores dum estado                */

/*   ##################################################    */

/*   Eliminar as marcas F e G em arc(A,B,C,D,E,F,G),                  */
/*   pondo_as indeterminadas                                          */


desini([],[]):-!.
desini([arc(A,B,C,D,E,F,G)|R],[arc(A,B,C,D,E,_,_)|R1]):-desini(R,R1).


/*   Fim de:                                                          */
/*   Eliminar as marcas F e G em arc(A,B,C,D,E,F,G),                  */
/*   pondo_as indeterminadas                                          */

/*   ##################################################    */
```

```
/*   ######################################################   */


/*   Eliminar redundancias entre estados                      */

geral(Ldesdobram1,Ldesdobram):-
        fraccionar(Ldesdobram1,[[]],Lacom,[[]],Lnaocom),
        compressao(Lacom,[[]],Lcom),
        append(Lnaocom,Lcom,Ldesdobram).


/*   ######################################################   */


fraccionar([[]],[[]],[[]],[[]],[[]]):-!.
fraccionar([[]],[[]|Lacom],Lacom,[[]],[[]]):-!.
fraccionar([[]],[[]],[[]],[[]|Lnaocom],Lnaocom):-!.
fraccionar([[]],[[]|Lacom],Lacom,[[]|Lnaocom],Lnaocom):-!.
fraccionar([H|T],Lacomaux,Lacom,Lnaocomaux,Lnaocom):-
        fraccionar1(H,H,[],Lista),
        klength(Lista,KL),
        KL=:=0 ->

            append(Lnaocomaux,[H],Lnaocomaux1),
            fraccionar(T,Lacomaux,Lacom,Lnaocomaux1,Lnaocom);

            append(Lacomaux,[H],Lacomaux1),
            fraccionar(T,Lacomaux1,Lacom,Lnaocomaux,Lnaocom).


/*   ######################################################   */
```

```
/*   ######################################################   */


fraccionar1([],_,Lista,Lista):-!.

fraccionar1([HH|TH],[HH|TH],Laux,Lista):-
        fraccionar2(HH,TH,[],Frac2),
        append(Laux,Frac2,Laux1),
        fraccionar1(TH,TH,Laux1,Lista).


/*   ######################################################   */


fraccionar2(_,[],Frac2,Frac2):-!.

fraccionar2(arc(A,B,C,D,E,F,G),[arc(H,I,J,K,L,M,N)|R],Laux,Frac2):-
      A==H,
      C==J,
      D==K,
      F=\=M,
      G=\=N,
          append(Laux,[arc(A,B,C,D,E,F,G)],Frac2),!;
          append(Laux,[],Frac2),!.


/*   ######################################################   */


compressao([[]],[[]|Resultado],Resultado):-!.

compressao([H|T],Laux,Resultado):-
        append(Laux,[H],Laux1),
        append(T,[[]],T1),
        comparar(H,T1,[[]],TL),
        compressao(TL,Laux1,Resultado).


/*   ######################################################   */
```

```
/*   ##################################################      */


comparar(_,[[]],[[]],[[]]):-!.
comparar(_,[[]],[[]|TL],TL):-!.
comparar([H|T],[HT|RT],Raux,TL):-
        compara1([H|T],HT,L),
        klength(L,KL),
        KL=:=0 ->
             comparar([H|T],RT,Raux,TL);
             append(Raux,[HT],Raux1),
             comparar([H|T],RT,Raux1,TL).


/*   ##################################################      */


compara1([],LL,LL):-!.
compara1([arc(A,B,C,D,E,F,G)|Rarco],[arc(H,I,J,K,L,M,N)|Rarco1],LL):-
       A==H,
       C==J,
       D==K,
       F==M,
       G==N,
       compara1(Rarco,Rarco1,LL),!;
       A==H,
       C==J,
       D==K,
       F==N,
       G==M,
       compara1(Rarco,Rarco1,LL),!;
       compara2(arc(A,B,C,D,E,F,G),Rarco1,[],Listresto),
       append([arc(H,I,J,K,L,M,N)],Listresto,Listresto1),
       compara1(Rarco,Listresto1,LL).


/*   ##################################################      */
```

```
/*   ######################################################      */


compara2(_,[],Listresto,Listresto):-!.
compara2(arc(A,B,C,D,E,F,G),[arc(H,I,J,K,L,M,N)|Rarco2],Comparaux,
Listresto):-
        A==H,
        C==J,
        D==K,
        F==M,
        G==N,
        compara2(arc(A,B,C,D,E,F,G),Rarco2,Comparaux,Listresto);
        A==H,
        C==J,
        D==K,
        F==N,
        G==M,
        compara2(arc(A,B,C,D,E,F,G),Rarco2,Comparaux,Listresto),!;
        append(Comparaux,[arc(H,I,J,K,L,M,N)],Comparaux2),
        compara2(arc(A,B,C,D,E,F,G),Rarco2,Comparaux2,Listresto),!.


/*   Fim de:                                                    */
/*   Eliminar redundancias entre estados                        */


/*   ######################################################      */
```

```
/*  FIM DE :                                                    */
/*  REGRAS DE PRODUCAO E ESTRATEGIA DE CONTROLE                 */
/*  ____________________________________________________________*/
```

```
/*  HEURISTICA                                                    */
/*  ____________________________________________________________  */
```

```
/*   ###################################################      */


/*    h(Estado,Heuristica_associada_ao_estado)                 */

h( Diah^Ltjfh^Ltafh^Ltpfh^Trabto^Livres,H ):-
                                heuristica(Diah,Ltpfh,Trabto,H).


heuristica( Diah,Ltpfh,Trabto,H):-
                    ntarefas(Ntarefas),
                     juntar(Ltpfh,1,Ntarefas,[],Relax,[],Critic),
                    lista_trabalhadores(Ltrab),

        programacao_linear(Diah,Trabto,Ltrab,Relax,Critic,0,H).


/*    Fim de:                                                  */
/*    h(Estado,Heuristica_associada_ao_estado)                 */


/*   ###################################################      */
```

```
/*  ################################################  */

/*  Contabiliza na lista dos trabalhos por fazer,L3, os                */
/*  "arcs" que sao relaxados e os "arcs" que sao criticos              */
/*  dicriminados por tarefas                                           */

juntar(_,Contarefas,Ntarefas,Relax,Relax,Critic,Critic):-Contarefas>Ntarefas,!.
juntar(Ltpfh,Contarefas,Ntarefas,Relaux,Relax,Critaux,Critic):-

    findall(arc(Act,Nact,N1,N2,T,_,_),
        (member(arc(Act,Nact,N1,N2,T,_,_),Ltpfh),
        T==Contarefas,
        rapidez(Act,Nact,Rap),
        Rap=:=0),
        Lrela),

    findall(arc(Act,Nact,N1,N2,T,_,_),
        (member(arc(Act,Nact,N1,N2,T,_,_),Ltpfh),
        T==Contarefas,
        rapidez(Act,Nact,Rap),
        Rap=:=1,
        tolerancia(arco(Act,N1,N2,T),Tol),
        Tol=\=0),
        Lrela1),

     append(Lrela,Lrela1,Lrela2),
     klength(Lrela2,KLrela),
     append(Relaux,[juntar_relax(Contarefas,KLrela)],Relaux1),

     findall(arc(Act,Nact,N1,N2,T,_,_),
         (member(arc(Act,Nact,N1,N2,T,_,_),Ltpfh),
         T==Contarefas,
         rapidez(Act,Nact,Rap),
         Rap=:=1,
         tolerancia(arco(Act,N1,N2,T),Tol),
         Tol=:=0),
         Lcrit),

     klength(Lcrit,KLcrit),
     append(Critaux,[juntar_critic(Contarefas,KLcrit)],Critaux1),
     Contarefas1 is Contarefas+1,
     juntar(Ltpfh,Contarefas1,Ntarefas,Relaux1,Relax,Critaux1,Critic).


/*  Fim de:                                                            */
/*  Contabiliza na lista dos trabalhos por fazer,L3, os                */
/*  "arcs" que sao relaxados e os "arcs" que sao criticos              */
/*  dicriminados por tarefas                                           */

/*  ################################################  */
```

```
/*  ################################################  */


/*   Resolve o problema de programacao linear associado a          */
/*   cada tipo de trabalhador,como contributo desse tipo de        */
/*   trabalhador para a heuristica associada a cada estado.        */
/*   Para a resolucao do problema de programacao linear            */
/*   estabelece_se uma ligacao a rotina DDPLRS da biblioteca       */
/*   IMSL.                                                         */


programacao_linear(_,_,[],_,_,H,H):-!.

programacao_linear(Diah,Trabto,[CLtrab|RLtrab],Relax,Critic,Heuris,H):-
        trabalhador_tarefas(CLtrab,Ltarefas),
        sobreposicoes(Diah,Lsobreposicoes),
        relaxadas(Diah,Lrelaxadas),
        criticas(Diah,Lcriticas),

            findall(sobreposicao(Taref,Sobre),
                    (member(sobreposicao(Taref,Sobre),Lsobreposicoes),
                     member(Taref,Ltarefas)),
                    Lsobre),
            computa(Ltarefas,Lsobre,[],Lowerbound),

            member(total(CLtrab,Val),Trabto),
            computar(Ltarefas,Lrelaxadas,Lcriticas,Relax,Critic,[Val],B),
            dia_final(Fim),
            Ndias is Fim-Diah,
            n_equacoes_restricao(CLtrab,M),
            irtype(CLtrab,Irtyp),
            n_variaveis(CLtrab,NVAR),
            upperbound(CLtrab,Upperbound),
            custos(CLtrab,C),
            matrix_a(CLtrab,A),


        tell('tstmin.dad'),write( M),nl,
                            write(NVAR),nl,
                            writelist(Lowerbound),
                            writelist(Upperbound),
                            writelist(B),
                            writelist(C),
                            writelist(Irtyp),
                            writelist(C),
                            writelist(A),
                            write(Ndias), told,
```

```
    system(" run tstmin "),

    see('tstmin.res'),read(Obj),processfile([],Lresp),seen,

    calculo(CLtrab,Obj,Val,Heur),

    Heur1 is Heuris+Heur,

      programacao_linear(Diah,Trabto,RLtrab,Relax,Critic,Heur1,H).


/*  Fim de:                                                          */
/*   Resolve o problema de programacao linear associado a            */
/*   cada tipo de trabalhador,como contributo desse tipo de          */
/*   trabalhador para a heuristica associada a cada estado.          */
/*   Para a resolucao do problema de programacao linear              */
/*   estabelece_se uma ligacao a rotina DDPLRS da biblioteca         */
/*   IMSL.                                                           */


/*  ##################################################              */
```

```
/*   ##################################################      */

/*    Composicao do vector de limite inferior "lowerbound"      */
/*    por tipo de trabalhador                                   */

computa([],_,Lowerbound,Lowerbound):-!.
computa([CLtaref|RLtaref],Lsobre,Lower,Lowerbound):-
        n_equipas(CLtaref,Nequipas),
        computaa(CLtaref,1,Nequipas,Lsobre,Lower,Lower1),
        computa(RLtaref,Lsobre,Lower1,Lowerbound).

computaa(CLtaref,Cont,Nequipas,Lsobre,Lower,Lower1):-
        Cont=:=Nequipas,
        member(sobreposicao(CLtaref,Valor),Lsobre),
        append(Lower,[Valor],Lower1),!;
        append(Lower,[0],Lower2),
        Cont1 is Cont+1,
        computaa(CLtaref,Cont1,Nequipas,Lsobre,Lower2,Lower1).

/*   Fim de:                                                    */
/*    Composicao do vector de limite inferior "lowerbound"      */
/*    por tipo de trabalhador                                   */

/*   ##################################################      */

/*    Composicao do vector B por tipo de trabalhador            */

computar([],_,_,_,_,B,B):-!.
computar([CLtaref|RLtaref],Lrelaxadas,Lcriticas,Relax,Critic,Baux,B):-
            member(relaxada(CLtaref,Vx),Lrelaxadas),
            member(juntar_relax(CLtaref,Vxx),Relax),
            V is Vx+Vxx,
            member(critica(CLtaref,Vc),Lcriticas),
            member(juntar_critic(CLtaref,Vcc),Critic),
            Vvv is Vc+Vcc,
            Vv is V+Vvv,
            append(Baux,[Vv],Baux1),
            computar(RLtaref,Lrelaxadas,Lcriticas,Relax,Critic,Baux1,B).

/*   Fim de:                                                    */
/*    Composicao do vector B por tipo de trabalhador            */

/*   ##################################################      */
```

```
/*  ################################################  */

/*  Calculo da contribuicao para a heuristica associada  */
/*  ao estado dum determinado tipo de trabalhador        */

calculo(_,Ntrab,Val,0):-Val>Ntrab,!.

calculo(CLtrab,Ntrab,Val,Heur):-Ntaux is Ntrab-Val,
                                salario(CLtrab,S),
                                Heur is Ntaux*S.

/*  Fim de:                                              */
/*   Calculo da contribuicao para a heuristica associada */
/*  ao estado dum determinado tipo de trabalhador        */

/*  ################################################  */
```

```
/*   ##################################################      */


/*   Interface de escrita entre os dados em estrutura de lista      */
/*   em prolog e os dados em formatagem fortran utilizados          */
/*   na rotina DDPLRS da biblioteca IMSL                            */


writelist([]).
writelist([X|L]):-write(X),write('.0  '),nl,
                  writelist(L).

/*   ##################################################      */

processfile(Laux,Resp):-read(Term),
                        process(Term,Laux,Resp).

process(end_of_file,Resp,Resp):-!.

process(Term,Laux,Resp):-append(Laux,[Term],Laux1),
                         processfile(Laux1,Resp).


/*   Fim de:                                                        */
/*   Interface de escrita entre os dados em estrutura de lista      */
/*   em prolog e os dados em formatagem fortran utilizados          */
/*   na rotina DDPLRS da biblioteca IMSL                            */


/*   ##################################################
```

```
/*    FIM DE :                                                    */
/*    HEURISTICA                                                  */
/*    ____________________________________________________________*/
```

```
/*  CUSTO DE TRANSICAO ENTRE ESTADOS                                */
/*  ________________________________                                */
```

```
/*   ##################################################      */


/*    calculo do custo de transicao entre estados           */


custo([],[],C,C):-!.

custo([total(Trab,Tot)|Rtotal],[total(Trab,Tot1)|Rtotal1],C1,C):-

    Delta is Tot1-Tot,
    Delta>=0 ->

        salario(Trab,Sal),
        Del is Delta*Sal,
        C2 is C1+Del,
        custo(Rtotal,Rtotal1,C2,C) ;

        custo(Rtotal,Rtotal1,C1,C).


/*   fim de :                                               */
/*    calculo do custo de transicao entre estados           */


/*   ##################################################      */
```

```
/*   FIM DE :                                                    */
/*   CUSTO DE TRANSICAO ENTRE ESTADOS                            */
/*   ________________________________________________________    */
```

/* ALGORITMO A* */
/* ---------- */

```
/*  ######################################################  */

bestfirst(Start,Solution):- biggest(Big),
                            start(Start),
                            expand([],l(Start,0/0),Big,_,yes,Solution),!.

/*  ######################################################  */

expand(P,l(N,_),_,_,yes,[N|P]):- goal(N).

expand(P,l(N,F/G),Bound,Tree1,Solved,Sol):-
        F=<Bound,
        ante_sucessao(N,Listsucc),
        ( bagof(M/C,(member(s(N,M,C),Listsucc),
                     not member(M,P)),Succ),!,
          succlist(G,Succ,Ts1),
          bestf(Ts,F1),
          expand(P,t(N,F1/G,Ts),Bound,Tree1,Solved,Sol);
          Solved=never).

expand(P,t(N,F/G,[T|Ts]),Bound,Tree1,Solved,Sol):-
        F=<Bound,
        bestf(Ts,BF),
        min(Bound,BF,Bound1),
        expand([N|P],T,Bound1,T1,Solved1,Sol),
        continue(P,t(N,F/G,[T1|Ts]),Bound,Tree1,Solved1,Solved,Sol).

expand(_,t(_,_,[]),_,_,never,_):-!.

expand(_,Tree,Bound,Tree,no,_):- f(Tree,F),F>Bound.

/*  ######################################################  */

continue(_,_,_,_,yes,yes,Sol).

continue(P,t(N,F/G,[T1|Ts]),Bound,Tree1,Solved1,Solved,Sol):-
    (Solved1=no,inserta(T1,Ts,NTs);
    Solved1=never,NTs=Ts),
    bestf(NTs,F1),
    expand(P,t(N,F1/G,NTs),Bound,Tree1,Solved,Sol).

/*  ######################################################  */
```

```
/*  ###################################################  */

succlist(_,[],[]).

succlist(G0,[N/C|NCs],Ts):- G is G0+C,
                            h(N,H),
                            F is G+H,
                            succlist(G0,NCs,Ts1),
                            inserta(l(N,F/G),Ts1,Ts).

/*  ###################################################  */

inserta(T,Ts,[T|Ts]):- f(T,F),
                       bestf(Ts,F1),
                       F=<F1,!.

inserta(T,[T1|Ts],[T1|Ts1]):- inserta(T,Ts,Ts1).

/*  ###################################################  */

f(l(_,F/_),F).

f(t(_,F/_,_),F).

/*  ###################################################  */

bestf([T|_],F):- f(T,F).

bestf([],Big):- biggest(Big).

/* ###################################################  */

min(X,Y,X):- X=<Y,!.

min(X,Y,Y).

/*  ###################################################  */
```

```
/*  FIM DE :                                                    */
/*  ALGORITMO A*                                                */
/*  ___________________________________________________________ */
```

```
/*  AUXILIARES                                                    */
/*  ____________________________________________________________  */
```

```
/*   ######################################################      */


combinations([H|T],[H|U]):-combinations(T,U).
combinations([_|T],U):-combinations(T,U).
combinations([],[]).

/*   ######################################################      */

member(A,[A|_]).
member(A,[_|L]):-member(A,L).

/*   ######################################################      */

reverse([],[]).
reverse([H|T],L):-reverse(T,R),
                  append(R,[H],L).

/*   ######################################################      */


insert(X,[],[X]).
insert(X,[H|T],[X|[H|T]]).

/*   ######################################################      */


append([],L,L).
append([H|T],L,[H|U]):-append(T,L,U).

/*   ######################################################      */
```

```
/*   ######################################################      */


append1(L,[[]],L):-!.
append1([],L,L).
append1([H|T],L,[H|U]):-append(T,L,U).


/*   ######################################################      */


subtract(L,[],L):-!.
subtract([H|T],L,U):-member(H,L),!,subtract(T,L,U).
subtract([H|T],L,[H|U]):-!,subtract(T,L,U).
subtract(_,_,[]).


/*   ######################################################      */


klength([],0).
klength([P,..T],N):-klength(T,M),
                    N is M+1.


/*   ######################################################      */


findall(X,G,Xlist):-call(G),
                    assertz(stack(X)),
                    fail;
                    assertz(stack(bottom)),
                    collect(Xlist).

collect(L):-retract(stack(X)),!,
            (X==bottom,!,L=[];L=[X|Rest],collect(Rest)).


/*   ######################################################      */
```

```
/*  FIM DE :                                                    */
/*  AUXILIARES                                                  */
/*  ----------------------------------------------------------  */
```

BIBLIOGRAFIA

Bellman, R. E. (1957)- **Dynamic Programming** - Princeton University Press, Princeton, N. J.

Bellman, R. E.; Dreyfus S. (1962) - **Applied Dynamyc Programming** - Princeton University Press, Princeton, N. J.

Bratko, I. (1986)- **Prolog Programming for Artificial Intelligence** - Addison-Wesley Publishing Company

Checkland. P. (1985) - **From Optimizing to Learning: A Development of Systems Thinking for the 1990S** - - J. Opl.Res. Soc. Vol. 36 Nº 9 pp. 757-767

Christofides, N.; Mingozzi, A.; Toth, P. and Sandi, C. (eds) (1979) - **Combinatorial Optimization** - Jonh Willey & Sons

Clocksin, W. F.; Mellish, C. S. (1985) - **Programming in Prolog**, 2ª Edição - Springer-Verlog

Coelho, H. (1982) - **Programação em Lógica - O Estilo, O Método e O Instrumento**, Centro de Informática do LNEC

Coelho, H.; Cotta, J.C. e Pereira M. (1982) - **How to Solve it with Prolog**, 3ª Edição, Laboratório Nacional de Engenharia Civil - Lisboa

Grant, J. (1986) - **Lessons For O. R. From A. I.: A Scheduling Case Study** - J. Opl. Res. Soc. Vol. 37 Nº 1 pp. 41-57

Kanal, L.; Kumar V. (eds) (1988) - **Search in Artificial Intelligence** - Springer-Verlog

Kaufman (1967) - **Graphs, Dynamic Programming and Finite Games** - Mathematics in Science and Engineering Vol. 36 - Academic Press

Lahrichi, A. (1982) - **Ordennancements. La Notion de «Parties Obligatoires» et son Application aux Problèmes Cumulatifs** - Revue Française d´Automatique, d´Informatique et de Recherche Opérationelle - RAIRO, Vol. 16, Nº. 3, pp. 241 a 262

Lee, R. M. (1986) - **A Logic Programming Aproach to Building Planning and Simulation Models** - Working Paper - Universidade do Texas

Nilsson, N. J. (1980) - **Principles of Artificial Intelligence**, Tioga Publ. Co., Palo Alto. CA,

Nilsson, N. J. (1971) - **Solving Methods in Artificial Intelligence**, McGraw - Hill International Book Company

Pearl, J. (1984) - **Intelligent Search Strategies for Computer Problem Solving** - Addison - Wesley Publishing Company

Phelps (1986) - **Artificial Intelligence - An Overview of Similarities with OR** - J. Opl. Res. Soc. Vol. 37 Nº 1 pp. 13-20

Rich, E. (1983) - **Artificial Intelligence** - McGraw - Hill International Book Company

Shapiro, J. (1979) - **Mathematical Programming: Structures and Algorithms** - Jonh Willey & Sons

Smith. D. (1982) - **Network Optimisation Practice - A Computational Guide** - Ellis Horwood Ltd.

Soubrier, J.P. (1982) - **Un Algorithme de Résolution de Problèmes d´Ordonnancement Dynamiques** - Revue Française d´Automatique, d´Informatique et de Recherche Opérationelle - RAIRO, Vol.16, Nº 3, pp. 219 a 239