Custódio Fernando Morais Toledo
Marcelo Bellini Garcia

SISTEMA DE PLANEJAMENTO DE PROCESSOS AUTOMATIZADO

Trabalho de Graduação apresentado ao Depto. de Engenharia Mecânica da Escola Politécnica da USP para obtenção do título de Engenheiro Mecânico - Habilitação em Automação e Sistemas (MECATRÔNICA).

Orientadores:
Prof. Dr. Paulo Eigi Miyagi
Prof. Dr. José Reinaldo Silva

São Paulo
Dezembro de 1992

Aos meus pais, Custódio e Vera.

Fernando

Aos meus pais, Miguel e Aldaiza, e a minha noiva, Fabiane.

Marcelo

AGRADECIMENTOS

Ao Prof. Marcos de Sales Guerra Tsuzuki, idealizador deste projeto e nosso orientador em seu início, que mesmo estando durante este ano no Japão, continuou a contribuir com suas idéias.

Aos nossos orientadores, Prof. Dr. Paulo Eigi Miyagi e Prof. Dr. José Reinaldo Silva, pela efetiva orientação durante este ano.

Ao Prof. Dr. Oswaldo Horikawa, por sua contribuição na fase de Seleção do Processo de Usinagem e desenvolvimento de heurísticas.

Aos Professores do Laboratório de Automação e Sistemas, Antônio Carlos de Lima, Júlio Arakaki, Newton Maruyama e Ricardo Matone, que também contribuiram para a execução deste trabalho.

Aos colegas da Mecatrônica, pelo companheirismo ao longo destes cinco anos.

Aos nossos familiares e amigos, que sempre nos incentivaram.

Este trabalho foi parcialmente financiado pela FAPESP e CNPq, não representando necessariamente a linha de pensamento destes dois orgãos.

RESUMO

Sistemas computacionais são cada vez mais utilizados nas atividades de projeto (CAD) e fabricação (CAM). Porém, uma eficiente interface entre estes dois processos é ainda motivo de muitas pesquisas. Este trabalho apresenta um Sistema de Planejamento de Processos Automatizado, que realiza a ligação entre CAD e CAM. Este Sistema é baseado no conceito de "features", que são regiões de interesse sobre a superfície de um sólido (p.ex.: furos, rasgos, chanfros, etc...) de modo que a cada "feature" pode-se associar um processo de fabricação. Este sistema procura uma nova abordagem para o problema, buscando ser o mais genérico possível e valorizando a criatividade no processo de concepção do produto.

ABSTRACT

Computational systems are more and more used in design (CAD) and manufacturing (CAM) processes. On the other hand, an efficient interface between both processes is still the aim of many researches. This work introduces an Automatic Process Planning System, which performs the bridge between CAD and CAM. This system is based on the concept of "features", which are regions of interest on a solid surface (e.g.: holes, chamfers, pockets, etc...). To each "feature" a manufacturing process can be associated. A new approach for this problem is reached, becoming the system as generic as possible and increasing the design process creativity.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

No processo de concepção de um produto, o primeiro e mais importante passo é a realização do planejamento do processo, que é o ato de preparar as instruções de operação a fim de produzir uma peça através um projeto de engenharia. Um plano detalhado contém os processos envolvidos na fabricação, os parâmetros do processo (p.ex.: velocidade de corte, velocidade de avanço e profundidade de corte da ferramenta, lubrificação, etc...), máquinas e ferramentas requeridas para a produção da peça e o caminho de corte das ferramentas [Chang 90].

A qualidade do produto e o custo de sua produção são fortemente influenciados pelo planejamento do processo, tanto que esta tarefa foi definida pela "Society of Manufacturing Engineers" como "a sistemática determinação de métodos pelos quais um produto possa ser fabricado economicamente e competitivamente". Atualmente, o método de produção está gradualmente se movendo em direção à automação, tornando-se necessária a criação de sistemas de planejamento de processos auxiliados por computador (CAPP – Computer-Aided Process Planning). Existem três propostas básicas para implementação de sistemas CAPP:

- Planejamento Variante de Processos
- Planejamento Generativo de Processos
- Planejamento Automático de Processos

A técnica variante é baseada na aplicação de tecnologia de grupo. O computador é utilizado para armazenar os planos de um grupo tecnológico por meio de procedimentos e tabelas. Um técnico especializado edita o plano variante a partir de planos já existentes de uma peça similar. Já a técnica generativa cria um novo plano sem referência a nenhum plano já existente, porém existe a necessidade de técnicos especializados para apoiar a decisão em pontos de incerteza.

A técnica automática, por outro lado, procura eliminar a presença de planejadores de processo utilizando o computador em todos os aspectos. Um sistema de planejamento de processos automatizado apresenta, portanto, um grau de complexidade muito maior que as outras duas técnicas citadas.

1.1 CAPPs Automáticos

Um sistema de Planejamento de Processos Automatizado deverá, a partir de um sólido[1] gerado em um modelador de sólidos, criar automaticamente o processo de fabricação deste sólido.

"Feature"[2], que pode ser definida como uma região de interesse em um sólido (p. ex.: furos, rasgos, chanfros, etc...), é um conceito fundamental para a compreensão de sistemas CAPPs Automáticos, que permitirá associar a cada sólido a maneira pela qual ele poderá ser fabricado [Floriani 89]. Um sólido pode ser então interpretado como sendo uma "forma completa" composta por uma coleção de componentes que são as "features".

Atualmente, existem dois métodos para se implementar um sistema CAPP Automático. O primeiro método transfere a tarefa de planejamento do processo para a fase de projeto, incorporando informações de "features" no modelo computacional. Este ambiente de projeto limita o projetista às "features" disponíveis, que são por definição possíveis de serem fabricadas. O segundo método valoriza a criatividade do projeto, que fica limitado somente à capacidade do sistema de CAD e à habilidade do projetista. Esta concepção de sistemas CAPP exige, no entanto, a criação de módulos intermediários para reconhecimento e sequenciação das "features" do sólido. Assim, este "design" de sistemas CAPP Automáticos pode ser descrito em termos de cinco módulos principais :

- **Reconhecimento Automático de "Features"**, que tem por função reconhecer e localizar uma "feature" em um sólido;
- **Escalonamento Automático de "Features"**, que tem por função gerenciar o módulo de refinamento[3] de "features", gerando as possíveis sequências de usinagem;
- **Seleção do Processo de Usinagem**, que tem por função gerar o processo de fabricação da peça, utilizando-se das sequências fornecidas pelo escalonador;
- **Planejamento do Método de Fixação**, que tem por função gerar os planos de fixação para os possíveis planos de processo;

[1] Convencionou-se neste trabalho denominar a peça a ser manufaturada pelo termo *sólido*, já que no ambiente em que este projeto está inserido a peça será obtida de um modelador de sólidos.

[2] Uma definição de "feature" mais apropriada a este trabalho será apresentada na seção 2.5.

[3] Convecionou-se neste trabalho chamar a união destes dois primeiros módulos de Refinamento Automático de "features".

- **Geração do Caminho de Corte**, que tem por função gerar o código NC para a máquina.

1.2 Estrutura do Trabalho

Este trabalho é dividido em 8 capítulos. O capítulo 2 apresenta alguns conceitos necessários à compreeensão e implementação do sistema. O capítulo 3 descreve a arquitetura do sistema.

O capítulo 4 se relaciona ao módulo de *Reconhecimento Automático de "Features"*, desde a concepção do módulo até sua implementação, discutindo-se algoritmos e estruturas principais. O capítulo 5 segue a mesma linha do capítulo 4, mas trata agora do módulo de *Escalonamento Automático de "Features"*. Estes dois módulos juntos constituem a base do sistema CAPP Automático.

O capitulo 6 apresenta o módulo de *Seleção do Processo de Usinagem*, o tratamento das informações obtidas do módulo de *Escalonamento Automático de "Features"* por meio de heurísticas. Sua saída é a *Folha de Processo*, que também é aqui discutida.

O capítulo 7 apresenta os resultados obtidos, comentando a integração do sistema e apresentando uma simulação detalhada de seu funcionamento. Finalmente, o capítulo 8 desenvolve as conclusões pertinentes ao trabalho.

1.3 Sugestão para a Leitura deste Trabalho

Ao leitor que tenha a intenção de conhecer o funcionamento deste sistema sem considerar detalhes técnicos de implementação, a leitura dos capítulos 4, 5 e 6 pode ser ignorada. Já o leitor que desejar um aprofundamento maior no assunto, é sugerida a leitura na ordem estabelecida no trabalho.

Capítulo 2

Conceitos Fundamentais

Neste capítulo apresentamos inicialmente o conceito de modelagem de sólidos por fronteira, utilizado na modelagem dos sólidos do modelador de sólidos didático e portanto, nas "features" que irão compor o sólido. A seguir, introduzimos o conceito acerca de grafos planares, que apresenta uma melhor visualização de vértices, arestas, faces e seus interrelacionamentos.

Apresentamos a seguir, o conceito de relações de adjacência e funções de modelagem, base para a implementação do sistema de *Reconhecimento Automático de "Features"*. Apresentamos por último o conceito de "features", seu relacionamento em um sólido e a conceituação de árvore de "features", base do sistema de *Escalonamento Automático de "Features"*.

2.1 Modelagem de Sólidos – Boundary Representation

Boundary Representation (B-rep) [Mäntylä 88] é uma técnica de modelagem de sólidos que se caracteriza por uma importante propriedade, que consiste na clara separação entre as informações topológicas e as informações geométricas de um sólido.

As informações topológicas se referem às informações de adjacência (p. ex.: capacidade de descrever como cada face está conectada às suas faces adjacentes de maneira que um volume totalmente fechado seja definido). As informações geométricas se relacionam à descrição da geometria da superfície, descrição da geometria das arestas e a localização dos vértices.

A separação entre topologia e a geometria permite economizar tempo computacional e obter maior precisão numérica, já que não há necessidade de obter a adjacência das faces por técnicas numéricas que analisam as proximidades geométricas de seus componentes.

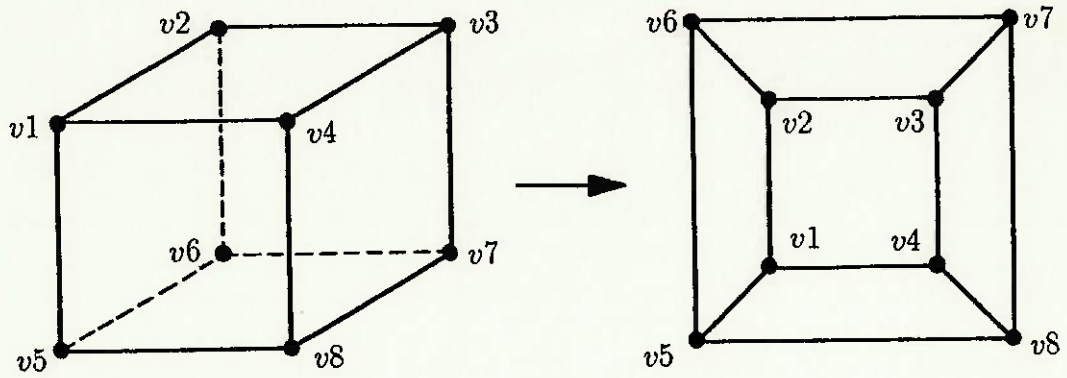


Figura 2.1: Grafo Conexo

2.2 Representação por Grafos Planares

Um grafo é dito planar quando as arestas do grafo não se cruzam, com exceção nos vértices. Todos os objetos que pertencem ao domínio de representação dos sólidos podem ser representados por um grafo planar. Os grafos planares são divididos em grafos planares conexos e grafos planares desconexos.

Para os grafos planares conexos (Figura 2.1) a fórmula de Euler é válida:

$$v - e + l = 2$$

$$\begin{cases} \text{v - número de vértices do grafo} \\ \text{e - número de arestas do grafo} \\ \text{l - número de laços do grafo} \end{cases}$$

No caso de grafos planares desconexos (Figura 2.2) utiliza-se uma extensão da fórmula de Euler:

$$v - e + le = 2 - 2.h + li$$

$$\begin{cases} \text{le - número de laços externos} \\ \text{li - número de laços internos} \\ \text{h - número de furos} \end{cases}$$

Quando for necessário se referir a vértices, arestas, laços, o grafo planar será utilizado preferencialmente pois apresenta uma melhor visualização da forma que eles se inter-relacionam.

2.3 Relações de Adjacência

As relações de adjacência estão relacionadas à topologia do objeto, ou seja, à forma com que os elementos primitivos (vértices, arestas, laços) se relacionam. Elas são a base para

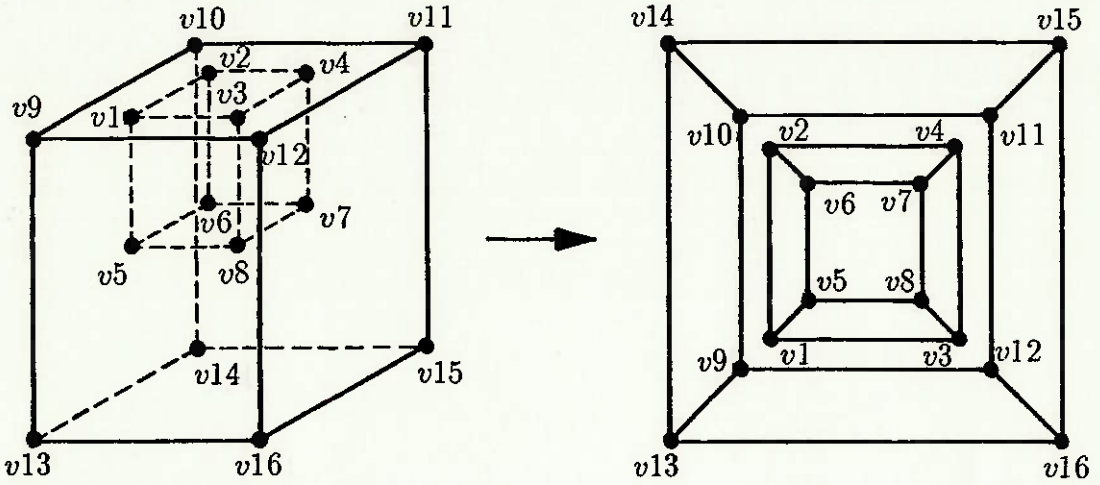


Figura 2.2: Grafo Desconexo

as funções de modelagem que serão utilizadas nos algoritmos de comparação de sólidos e extração de "features". Pela simbologia definida por [Weiler 85], é possível representar nove relações de adjacência sobre os primitivos. Uma extensão desta simbologia permite a representação de faces com contornos múltiplos. Cada face possui obrigatoriamente um laço externo e zero ou mais laços internos.

As relações de adjacência são listas ou conjuntos de elementos organizados de alguma forma. A terminologia utilizada é indicada abaixo:

$\{a_1, a_2, \ldots\}$ - indica um conjunto não ordenado de elementos

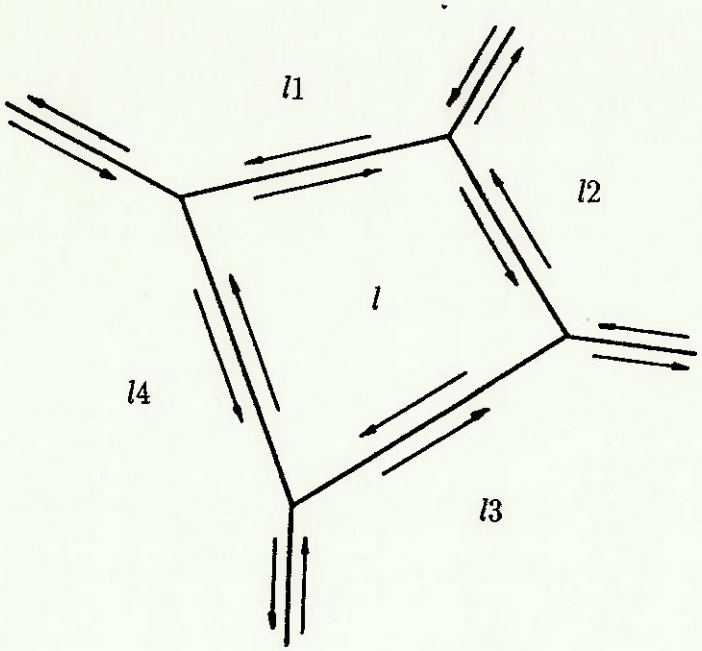
$(a_1, a_2, \ldots)$ - indica uma lista linear ordenada de elementos

$[a_1, \{a_2, a_3, \ldots\}]$ - indica uma lista semi-ordenada em que o primeiro elemento é determinado e os demais não possuem ordem específica

$< a_1, a_2, \ldots >$ - indica uma lista circular ordenada de elementos

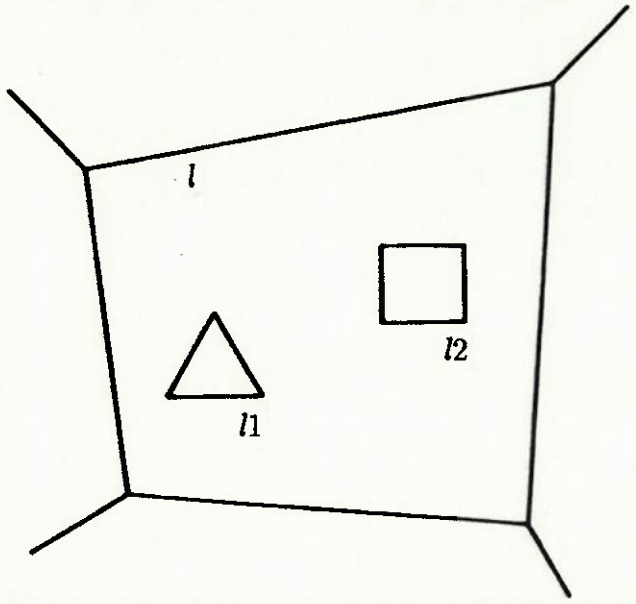
São utilizadas duas relações de adjacência nos algoritmos implementados.A relação de adjacência $l < L >$ (Fig. 2.3) que se refere à lista ordenada circular de laços ao redor do laço de referência, e a relação de adjacêcia $l\,[L]$ (Fig. 2.4) que se refere a lista semi-ordenada de laços , cujo primeiro elemento da lista é o laço externo da face de referência. Os demais elementos não possuem ordem específica e representam os laços internos da face de referência.

Abaixo definimos um simbolismo para representar a ordenação dos elementos de uma lista ordenada circular A com n elementos.



$$l\,<L>=<l1, l2, l3, l4>$$

Figura 2.3: Relação de Adjacência $l\,<L>$



$$l\,[L] = [l, \{l1, l2\}]$$

Figura 2.4: Relação de Adjacência $l\,[L]$

$$nt(a_i, A_n)^k = a_{i+k} \qquad \begin{cases} i < n \\ k \in Z \end{cases}$$

$$A_n =< a_0, a_1, a_2, \ldots, a_i, \ldots >$$

O índice não indica a posição absoluta do elemento na lista, mas indica apenas a posição relativa entre os elementos da lista.

Definição 1 *Duas listas ordenadas circulares serão consideradas equivalentes se e apenas se existir um elemento a_i de A_n e um elemento b_j de B_n tal que:*

$$nt(a_i, A_n)^k = nt(b_j, B_n)^k \qquad k = 0, \ldots, n$$

Bijeção é uma relação binária R de um conjunto A para um conjunto B, aonde todo elemento do conjunto B é a imagem de apenas um elemento do conjunto A e dois elementos do conjunto A não possuem a mesma imagem.

Em acordo aos tipos de elementos primitivos apresentados, a bijeção entre dois sólidos é definida pela associação de quatro bijeções entre os quatro conjuntos de elementos primitivos. A bijeção de um conjunto de vértices a um conjunto de vértices será referenciada por fv. Analogamente, as bijeções entre conjuntos de arestas, laços e faces serão referenciadas por fa, fl e ff respectivamente. É importante observar que caso exista uma bijeção entre dois sólidos é porque ambos os sólidos possuem o mesmo número de vértices, arestas, laços e faces. Estas considerações nos permitem realizar as seguintes definições:

Definição 2 *Seja α o conjunto de elementos primitivos do sólido A e β o conjunto de elementos primitivos do sólido B. Caso exista uma bijeção g de α para β, implica na existência das bijeções fv, fa, fl e ff (Figura 2.5).*

Definição 3 *Dada uma bijeção $g : \alpha \rightarrow \beta$, define-se a imagem de uma lista circular ordenada $A_n =< a_0, a_1, a_2, \ldots, a_i, \ldots >$ pela seguinte expressão $g(A_n) =< g(a_0)_0, g(a_1)_1, g(a_2)_2, \ldots, g(a_i)_i, \ldots >$. Neste trabalho, utilizaremos a seguinte simbologia para representar a imagem de uma lista circular ordenada:*

$$nt(g(a_i)_i, g(A_n))^k = g(nt(a_i, A_n)^k)_{i+k} \qquad k = 0, \ldots, n$$

Segundo a bijeção g é possível associar três listas de elementos primitivos, que serão representadas por:

- $l < L >$: lista de laços que circunda o laço l

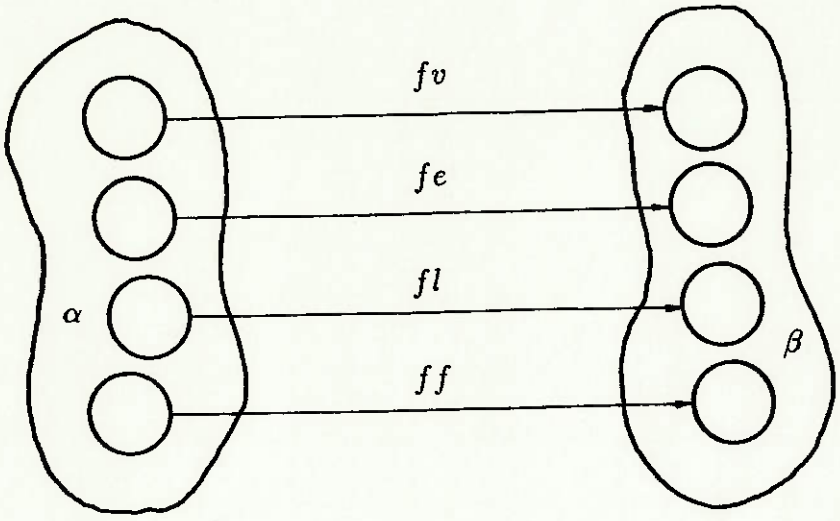


Figura 2.5: Bijeção entre dois Sólidos

- $g(l < L >)$: imagem da lista de laços que circunda o laço l
- $g(l) < L >$: lista de laços que circunda a imagem do laço l

Definição 4 *Caso as listas circulares ordenadas $g(l < L >)$ e $g(l) < L >$ sejam equivalentes, então diz-se que o elemento primitivo l e sua imagem sob a bijeção g possuem a classe $l < L >$ de relação de adjacência equivalente (Figura 2.6).*

Definição 5 *Caso todos os elementos primitivos l de um sólido A e suas respectivas imagens sob a bijeção g possuam a relação de adjacência de classe $l < L >$ equivalente, então diz-se que o sólido A e sua imagem sob a bijeção g possuem a classe $L < L >$ de relação de adjacência equivalente.*

Definição 6 *Caso um sólido A e sua imagem sob a bijeção g possuam todas as classes de relação de adjacência equivalentes, então diz-se que o sólido A e sua imagem sob a bijeção g são topologicamente equivalentes.*

Definição 7 *Se um sólido A e um sólido B são topologicamente equivalentes sob uma bijeção g, então diz-se que a bijeção g é de classe G^*.*

Definição 8 *Se um sólido A e um sólido B possuem as classes $L < L >$ e $L[L]$ de relação de adjacência equivalentes sob uma bijeção fl, então diz-se que a bijeção fl é de classe fl^*.*

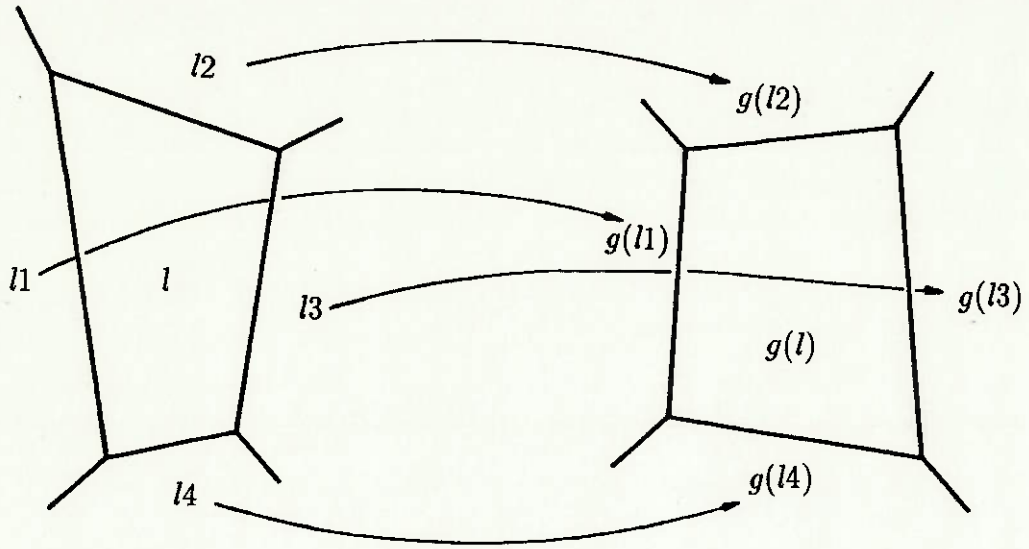


Figura 2.6: Classe $l < L >$ de Relação de Adjacência equivalente

Teorema 1 *A condição necessária e suficiente para que exista uma bijeção g de classe G^* de um sólido A para um sólido B é que exista uma bijeção fl de classe fl^* do conjunto de laços do sólido A para o conjunto de laços do sólido B [Tsuzuki 88].*

O Teorema 1 é conhecido como a "condição necessária e suficiente para que dois sólidos sejam topologicamente equivalentes".

2.4 Funções de Modelagem

Conforme a "condição necessária e suficiente para que dois sólidos sejam topologicamente equivalentes", devemos procurar por uma bijeção fl de classe fl^*. O processo de busca por uma bijeção de classe fl^* é, em seu pior caso, exaustivo e proporcional ao fatorial do número de laços do sólido. Isto ocorre porque não foi definido nenhum critério que permita analisar em estados intermediários se estamos no caminho correto.

Com o intuito de realizar uma busca não exaustiva, foram definidas funções de modelagem baseadas nas relações de adjacência $l < L >$ e $l[L]$. As funções de modelagem permitem analisar em estados intermediários se estamos no caminho correto pois à medida que ocorrem as associações dos laços dos dois sólidos, as funções de modelagem acumulam informações sobre o pedaço de um sólido que foi associado à um outro pedaço de um outro sólido, permitindo analisar se está ocorrendo uma equivalência topológica até o momento.

Simultaneamente, as funções de modelagem permitem definir uma maneira para representar "features", aonde algumas condições permanecem indefinidas.

Definição 9 *Dados os conjuntos de laços Al e Al', aonde Al é o conjunto de laços do sólido A e Al' ⊂ Al. Define-se a função λ para um laço la ∈ Al pela seguinte expressão:*

$$\lambda(l, Al') = \begin{cases} l & se\ l \in Al' \\ d & se\ l \notin Al' \end{cases}$$

Definição 10 *Dados os conjuntos de laços Al e Al', aonde Al é o conjunto de laços do sólido A e Al' ⊂ Al; define-se a função de modelagem Λ* para um laço la ∈ Al e que possui a relação de adjacência la $< L >=< l_1, l_2, \ldots, l_n >$, pela seguinte expressão:*

$$\Lambda^*(la, Al') =< \lambda(l_1, Al')_1, \lambda(l_2, Al')_2, \ldots, \lambda(l_n, Al')_n >$$

Definição 11 *Dados os conjuntos de laços Al e Al', aonde Al é o conjunto de laços do sólido A e Al' ⊂ Al; define-se a função de modelagem Λ⁺ para um laço la ∈ Al e que possui a relação de adjacência la $[L] = [l_e, \{l_{i1}, \ldots, l_{in}\}]$, pela seguinte expressão:*

$$\Lambda^+(la, Al') = [\lambda(l_e, Al')_e, \{\lambda(l_{i1}, Al')_{i1}, \ldots, \lambda(l_{in}, Al')_{in}\}]$$

Para a representação de "features" o subconjunto de laços Al' deverá conter todos os laços que compõem a "feature" e as relações de adjacência $l < L >$ e $l\,[L]$ relativas aos laços das "features" são definidas pelas funções de modelagem Λ^* e Λ^+ (Fig. 2.7). Informações geométricas podem ser aglutinadas às funções de modelagem, como o ângulo entre faces, coordenadas de vértices e comprimento de arestas. Uma definição detalhada de "features" é realizada na seção 2.5.

Definição 12 *Dada uma bijeção $g : \alpha \rightarrow \beta$ e o conjunto $Al'^d = Al' \cup \{d\}$, aonde $Al' \subset Al$; define-se o mapeamento $h : Al'^d \rightarrow Bl'^d$ pela seguinte expressão:*

$$h(\lambda(la, Al')) = \begin{cases} g(\lambda(la, Al')) & se\ \lambda(la, Al') \neq d \\ d & se\ \lambda(la, Al') = d \end{cases}$$

Definição 13 *Dados os sólidos A e B e uma bijeção $g : \alpha \rightarrow \beta$. Define-se a imagem de uma lista circular ordenada $\Lambda^*(la, Al') =< \lambda(l_1, Al')_1, \lambda(l_2, Al')_2, \ldots, \lambda(l_n, Al')_n >$ segundo o mapeamento $h : Al'^d \rightarrow Bl'^d$, pela seguinte expressão:*

$$h(\Lambda^*(la, Al')) =< h(\lambda(l_1, Al')_1)_1, h(\lambda(l_2, Al')_2)_2, \ldots, h(\lambda(l_n, Al')_n)_n >$$

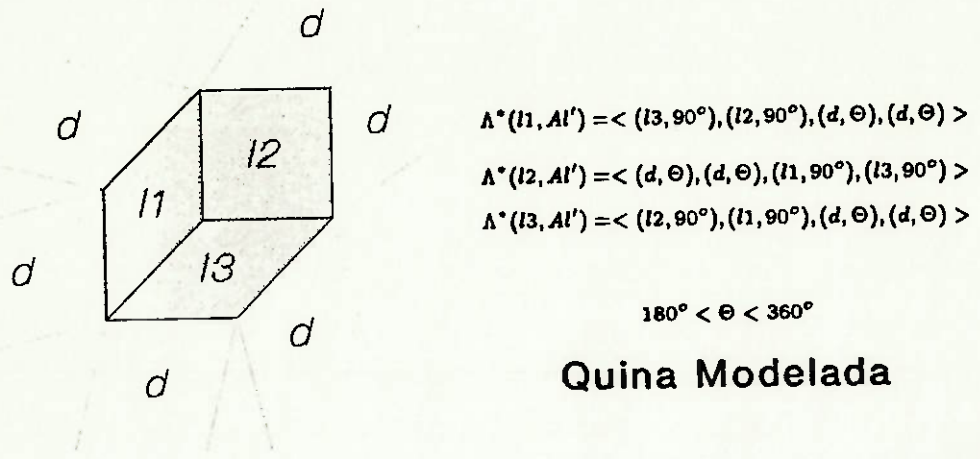


Figura 2.7: Representação de uma "feature"

Definição 14 *Dada uma bijeção* $h : Al'^d \rightarrow Bl'^d$, *caso as listas* $h(\Lambda^*(l, Al'))$ *e* $\Lambda^*(h(l), Bl')$ *sejam equivalentes, então diz-se que o laço* l *e sua imagem sob a bijeção* h *possuem a função* Λ^* *equivalente.*

Definição 15 *Caso todos os elementos primitivos* l *de* Al' *e suas respectivas imagens sob a bijeção* h *possuam a função* Λ^* *equivalente, então diz-se que* Al' *e sua imagem sob a bijeção* h *possuem a função* Λ^* *equivalente.*

Definição 16 *Se* Al' *e* Bl' *possuem a função* Λ^* *equivalente sob uma bijeção* h, *então diz-se que a bijeção* h *é de classe* h^*.

O conceito de função Λ^+ equivalente é definido similarmente, e a busca por uma bijeção fl^* será direcionada pela busca de bijeções $h : Al'^d \rightarrow Bl'^d$ de classe h^*.

2.5 "Features"

Uma "feature" é definida por [Floriani 89] como sendo uma região de interesse sobre a superfície de um objeto (p. ex. saliências, rasgos, furos, chavetas, etc...). Assim, um sólido pode ser interpretado como sendo uma "forma completa" composta por uma coleção de componentes que são as "features".

Já segundo [Chang 90], uma "feature" é um subconjunto de elementos geométricos em uma peça de engenharia que possui características especiais de projeto ou manufatura.

Ainda segundo [Ferreira 91], uma "feature" é a quntidade de material que deve ser usinada.

Embora estas definições não estejam incorretas, não considerou-se nenhuma delas completa. Assim, apresentamos a definição abaixo, que é mais apropriada dentro deste projeto:

Definição 17 *Uma "feature" é um conjunto aberto de faces (isto é, possui algumas condições topológicas indefinidas), onde suas condições geométricas relevantes*[1] *são totalmente definidas. Associado a cada um destes conjuntos haverá ao menos um processo de manufatura de tal forma que a extração de material do sólido através deste(s) processo(s), leva à obtenção deste conjunto em um sólido.*

Esta definição de "feature" é mais completa, já que deste modo pode-se modelar uma "feature" sem ambiguidades e com certa flexibilidade, permitindo que "features" ligeiramente diferentes entre si – mas com mesmo processo de manufatura – possam receber o mesmo tratamento em sua modelagem.

Na figura 2.8, a "feature" *quina* pode ser encontrada nos sólidos 1 e 2 (região hachurada), não importando as partes restantes dos sólidos. No caso do sólido 3, haverá uma equivalência topológica entre sua região hachurada e a *quina*, porém a geometria do sólido ao redor do hachurado não é equivalente à *quina*[2]. Isto impede a equivalência entre a região hachurada do sólido 3 com a *quina*, o que é coerente pois os processos de manufatura são diferentes. Na verdade, esta região hachurada do sólido 3 é um pedaço de uma "feature" denominada *furo não passante*.

Ainda na figura 2.8, o sólido 4 apresenta uma região hachurada com geometria ligeiramente diferente da região hachurada do sólido 1. No entanto, a modelagem da *quina* permite tratar as duas regiões hachuradas como sendo equivalentes, ou seja, tratam-se da mesma "feature" na definição proposta. Isto é coerente pois o processo de manufatura de ambas as regiões hachuradas é o mesmo.

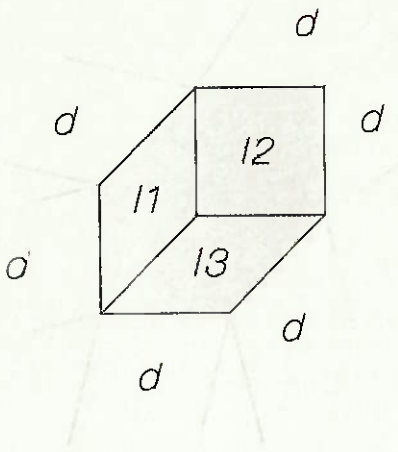
Outro aspecto importante é que uma "feature" pode ser extraída de um sólido de acordo com a definição que segue:

Definição 18 Extração de Uma "feature": *a extração de uma "feature" α de um sólido β pode ser considerada como o preenchimento do volume que a "feature" α ocupa no sólido β.*

A "Inversa de uma feature" α pode ser definida como o volume sólido que, unido ao sólido β, preenche o volume que delimitava a "feature" α no sólido β, ou seja, é o volume sólido que unido ao sólido β "extrai" a "feature" α do sólido β.

[1] No nosso caso, entende-se por relevante as condições de ângulo entre faces.

[2] As faces da região hachurada formam 90° com suas faces vizinhas, estando, portanto, fora da faixa $180° < \Theta < 360°$ definidas na modelagem da *quina*.



$\Lambda^*(l1, Al') =< (l3, 90°), (l2, 90°), (d, \Theta), (d, \Theta) >$

$\Lambda^*(l2, Al') =< (d, \Theta), (d, \Theta), (l1, 90°), (l3, 90°) >$

$\Lambda^*(l3, Al') =< (l2, 90°), (l1, 90°), (d, \Theta), (d, \Theta) >$

$180° < \Theta < 360°$

Quina Modelada

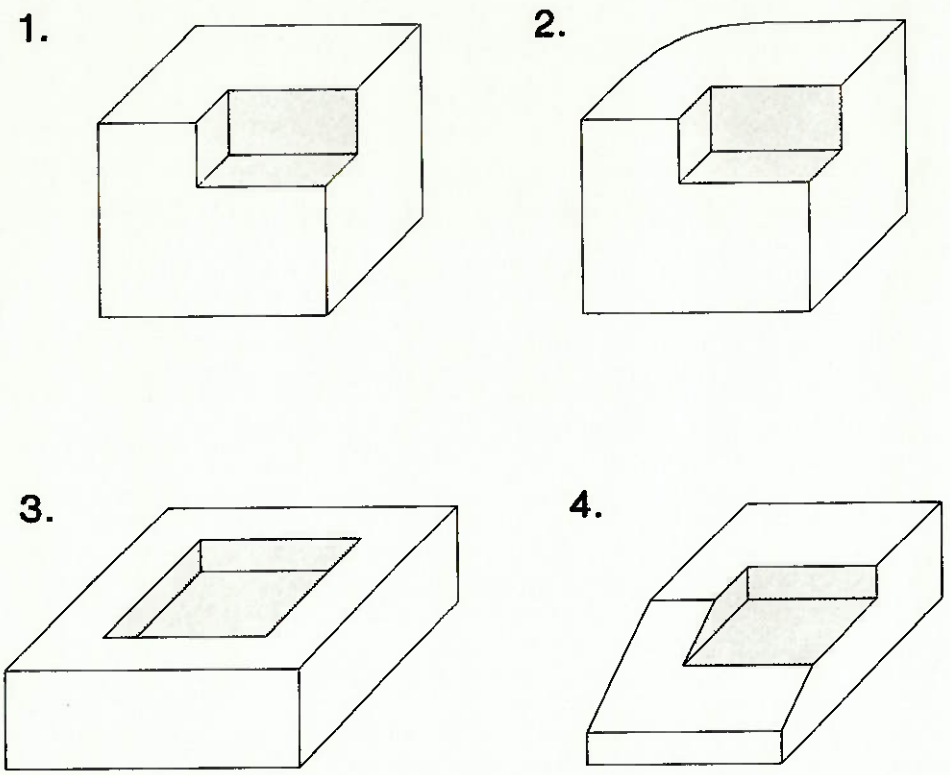


Figura 2.8: Definição de "feature"

2.6 Relacionamentos entre " Features"

Dentro deste sistema, um dos principais objetivos será escalonar as "features" que compõe a peça modelada. Durante o escalonamento, aparecerá um relacionamento intrínseco entre as "features", que será muito importante para o projeto. Assim, definimos os relacionamentos entre "features" como sendo:

Definição 19 *"Features" Independentes: dadas duas "features" α e β, α será dita independente em relação a β se α puder ser extraída do sólido modelado independentemente da extração de β.*

Definição 20 *"Features" Aninhadas: dadas duas "features" α e β, α será dita aninhada em relação a β se α só puder ser extraida do sólido modelado após a extração de β.*

Definição 21 *"Features Interferentes": uma região do sólido pode ser reconhecida por vários conjuntos distintos de "features". Como estes conjuntos representam a mesma região do sólido, ocupando o mesmo espaço físico, chamamo-os "features" interferentes.*

2.7 Conceituação: Árvore de "Features"

O objetivo de um Sistema de Planejamento de Processos Automatizado é planejar um processo de usinagem para uma determinada peça que se deseja manufaturar. Para atingir este objetivo, é necessário que se avaliem todas as possibilidades de se planejar este processo, escolhendo a alternativa mais viável. A *Árvore de "features"* será a estrutura de dados que conterá as informações necessárias para realizarmos esta avaliação.

Basicamente, uma *Árvore de "features"* pode ser considerada como uma árvore que possui associada a cada nó uma "feature". Deste modo, a *Árvore de "features"* permite que as "features" sejam dispostas hierarquicamente, segundo sua ordem de extração do sólido. Portanto, o que devemos fazer é reconhecer todas as "features" que compõem o sólido, em todas as sequências de extração possíveis[3], sendo que cada uma destas sequências corresponde, teoricamente, a um modo possível da peça ser usinada. Cada uma das sequências de extração será viável ou não para a usinagem da peça de acordo com critérios estabelecidos que serão analisados em módulos posteriores que aplicam técnicas de Inteligência Artificial.

Dentro do contexto de se usinar uma peça, deve-se possuir na *Árvore de "features"* apenas "features" possíveis de serem manufaturadas. Para isto, estabelece-se um *Arquivo*

[3]Uma sequência de extração das "features" é aquela onde se reconhecem e se extraem "features" do sólido sucessivamente, até que o sólido se encontre na forma de matéria-prima (p. ex.: um paralelepípedo), ou seja, todas as "features" que compõem o sólido foram extraídas em uma determinada ordem.

de "features", o qual conterá todas as "features" que poderão ser reconhecidas pelo sistema e, posteriormente, fabricadas[4]

Acerca de *Árvore de "features"*, enunciaremos os seguintes lemas:

Lema 1 *Uma "feature" pode estar associada a nós distintos de uma mesma Árvore de "features".*

Lema 2 *Em consequência ao lema 1, uma "feature" pode estar presente na Árvore de "features" em posições relativas diferentes em relação a uma segunda "feature".*

Lema 3 *Em uma Árvore de "features", uma "feature" α associada a um nó Ω será independente ou aninhada em relação a todas as "features" associadas a nós de mesmo nível do nó Ω.*

Lema 4 *Sejam duas "features" α e β associadas a nós distintos de uma Árvore de "features". As "features" α e β serão independentes entre si se os diversos nós associados às duas "features" aparecerem na Árvore de "features" segundo as três posições relativas possíveis.*[5]

Lema 5 *Sejam duas "features" α e β associadas a nós distintos de uma Árvore de "features". A "feature" α será aninhada em relação à "feature" β se nenhum dos nós associados à "feature" α estiver ao mesmo nível ou acima dos nós associados à "feature" β.*

Lema 6 *Sejam duas "features" α e β associadas a nós distintos de uma Árvore de "features". A "feature" α será interferente em relação à "feature" β se nenhum dos nós associados à "feature" α estiver abaixo ou acima dos nós associados à "feature" β.*

Lema 7 *Em uma árvore de "features", se uma "feature" α é aninhada em relação a uma "feature" β, então a "feature" α é independente em relação a todas as "features" independentes em relação a "feature" β.*

---

[4]Isto não significa uma limitação para o sistema, pois há a possibilidade de se criar novas "features" sempre que o sistema chegar a um ponto onde não exista mais nenhuma "feature" no *Arquivo de "Features"* que possa ser reconhecida pelo sistema (no caso de a sequência ainda não ter obtido, por exemplo, um paralelepípedo). Isto pode ser realizado utilizando-se o módulo *Editor de "Features"*, descrito no capítulo de Especificação do Sistema.

[5]As três posições relativas possíveis são: acima, abaixo e no mesmo nível.

Capítulo 3

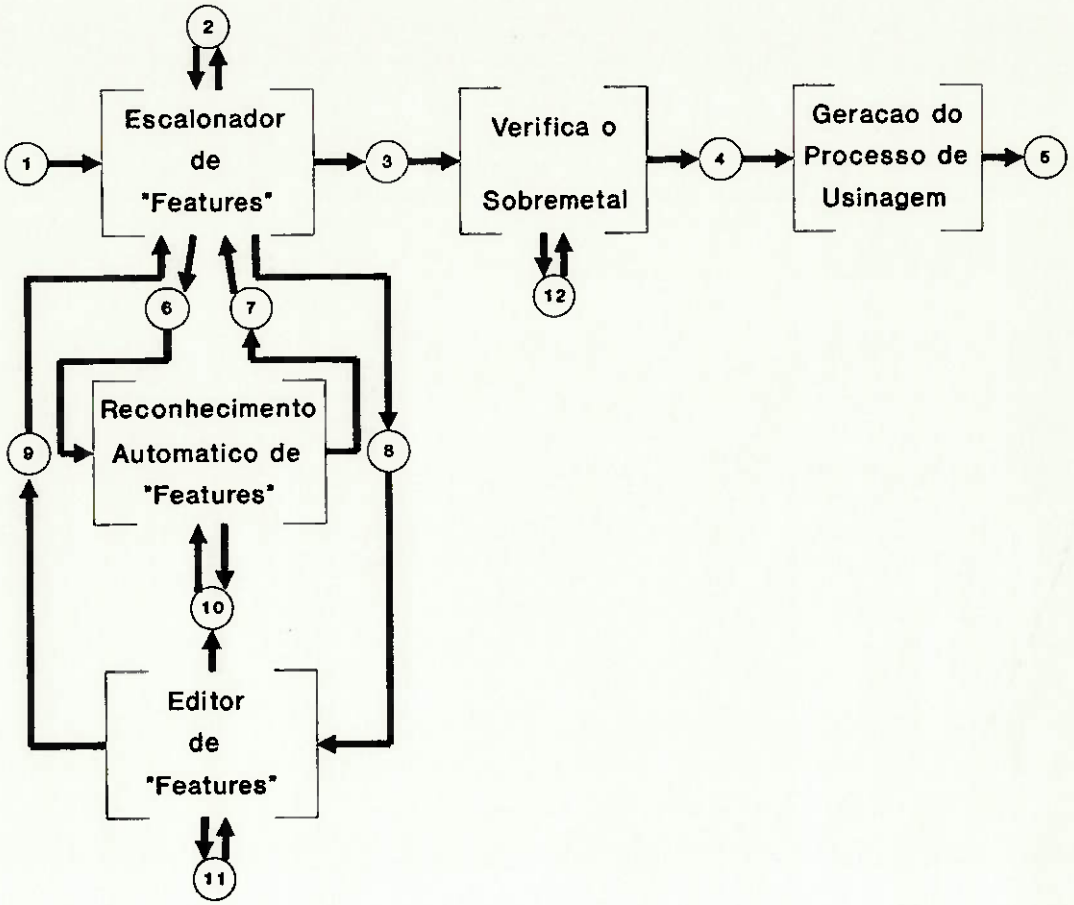
Especificação do Sistema

Este capítulo tem por objetivo apresentar a arquitetura do sistema e detalhar cada um de seus módulos. A técnica de representação utilizada – PFS (Production Flow Schema) [Miyagi 88], [Hasegawa 88] – se caracteriza por descrever conceitualmente os principais blocos (partes ativas e partes passivas) do sistema, os seus conteúdos e suas interrelações e, foi adotado neste caso por descrever explicitamente a interação dos diferentes módulos do sistema. Os elementos ativos estão representados entre "[" e "]" e os elementos passivos por "◯".

3.1 PFS – Refinamento Automático de "Features"

A estrutura do módulo de *Refinamento Automático de "Features"* é apresentada através do diagrama PFS da Figura 3.1. Cada um de seus elementos ativos e seus interrelacionamentos são apresentados a seguir:

- **Escalonador de "Features"**: Este módulo tem por função determinar as possíveis sequências de usinagem do sólido modelado. A proposta inicial era que fossem determinadas todas as sequências possíveis. Isto se tornou inviável já que se percebeu que o algoritmo pode gerar um conjunto de opções explosivo. Deste modo, decidiu-se aplicar heurísticas já neste módulo, trazendo uma parte do planejamento do processo para esta fase. Este módulo tem como entrada o sólido modelado e como saída uma estrutura de árvore de "features" que contém as melhores sequências de usinagem determinadas por heurísticas.
- **Reconhecimento Automático de "Features"**: Este módulo tem por função realizar o reconhecimento e localização de "features" no sólido de interesse. Este módulo tem por entrada a "feature" a ser reconhecida e o sólido representados por Funções de Modelagem. A saída deste módulo pode ser *"feature" reconhecida* ou *"feature" não reconhecida*. Se a resposta for *"feature" reconhecida*, este módulo a



1	Sólido segundo funões de modelagem
2	Arquivo de "Features" disponíveis p/ Reconhecer
3	Árvore de "features"
4	Árvore de "features" mais sobremetal
5	Folha de processo
6	Feature e Sólido Para Comparação no Reconhecedor
7	Resposta do Reconhecedor quanto à Comparação
8	Sólido com "features" restantes
9	Reinicializador do sistema
10	Banco de dados de "features"
11	Usuário
12	Banco de dados de peças em bruto

Figura 3.1: PFS – Sistema de Refinamento Automático de "Features".

localiza no sólido e: 1. extrai a "feature", realizando em seguida a extração de suas informações geométricas; 2. marca as faces da "feature" reconhecida. Este módulo é gerenciado pelo módulo de *Escalonamento de "Features"*.

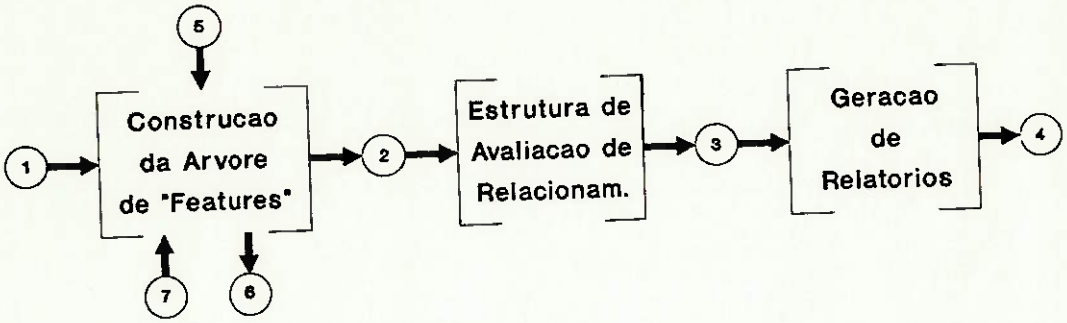
- **Editor de "Features"**: Este módulo tem por função oferecer uma interação entre o usuário e o sistema para a criação de "features" que serão inseridas no *Banco de Dados de "Features"* (10). Este módulo tem por entrada um sólido modelado e como saída a permissão para reinicialização do sistema após a execução das tarefas por parte do usuário.
- **Verifica o Sobremetal**: Este módulo verifica qual peça disponível no *Banco de Dados de Peças em Bruto* (12) é a melhor opção quanto ao menor volume de sobremetal. Além disso, insere a sequência de usinagem para a retirada do sobremetal antes de começar a usinagem das "features". Este módulo tem por entrada a estrutura da árvore de "features" e o modelo do objeto com as "features" extraídas. A saída é a árvore de "features" completa.
- **Geração do Processo de Usinagem**: este módulo tem por função gerar um processo de usinagem tendo como entrada a Árvore de "Feature".

O interrelacionamento entre estas partes é apresentado a seguir. O objeto modelado é a entrada para o *Sistema de Refinamento Automático de "Features"*. O módulo de *Escalonamento de "Features"* realiza o sequenciamento das "features" extraídas do sólido. Para isto, este módulo necessita do *Sistema de Reconhecimento de "Features"*. Se após o processo de extração de "features" não for possível obter um primitivo básico (paralelepípedos, cilindros, etc...), isto significa que o *Banco de Dados de "Features"* está incompleto. Com a utilização do *Editor de "Features"*, o usuário poderá adicionar as "features" restantes do sólido de entrada do sistema ao *Banco de Dados de "Features"*, bem como associar a estas "features" restantes um processo de manufatura. Desta forma, poder-se-á obter o primitivo básico do sólido inicial, e a "feature" anteriormente desconhecida constará agora no *Banco de Dados de "Features"*. Deve-se ressaltar, que para executar esta tarefa junto ao *Editor de "Features"*, o usuário deve ter conhecimento de técnicas de planejamento de processo.

Quando o primitivo básico é obtido, o módulo *Verifica o Sobremetal* determina qual peça disponível no *Banco de Dados de Peças em Bruto* é a melhor opção quanto ao menor volume de sobremetal. Terminado este passo, todo o sequenciamento das "features" estará disponível para o *Sistema de Geração do Processo de Usinagem.*

3.2 PFS – Escalonador Automático de "Features"

A estrutura do módulo de *Escalonamento Automático de "Features"* é apresentada através do diagrama PFS da Figura 3.2. Cada um de seus elementos ativos e interrelacionamentos são apresentados a seguir:



1	Sólido Modelado Segundo Funções de Modelagem
2	Estrura da Árvore de "Features"
3	Estruturas da Árvore de Relacionamentos
4	Sequência de "Features" e Relacinamentos
5	Arquivo de "Features" disponíveis p/ Reconhecer
6	Feature e Sólido Para Comparação no Reconhecedor
7	Resposta do Reconhecedor quanto à Comparação

Figura 3.2: PFS – Escalonador Automático

- **Construção da Árvore de Features**: Este sub-módulo do *Escalonador Automático de "Features"* tem por função estabelecer a hierarquia entre as "features" extraídas do sólido para que, posteriormente, possa ser gerado o processo de usinagem. Opera retirando as "features" de (5), enviando por (6) para o *Reconhecedor Automático de "Features"* que irá analisar se aquela "feature" existe no sólido naquele momento. Se existir, então retorna por (7) a indicação da a existência da "feature" mais suas características gerais quanto a dimensões, características geométricas, etc..., que serão inseridas na árvore a fim de serem utilizadas no restante da construção da árvore bem como para construir a estrutura de avaliação de relacionamentos e posteriormente gerar o plano de processo. Em seguida, tenta reconhecer novamente o mesmo tipo de "feature". O *Sistema de Marcar Faces* (seção 3.3) não permite que a mesma "feature" seja novamente encontrada no sólido. Se for encontrada outra do mesmo tipo e em uma posição diferente do sólido, o processo é repetido até não encontrar mais nenhuma do mesmo tipo, passando a outro tipo de "feature", até terminar com todas do *Arquivo de "Features"*. Assim, um nível de árvore é criado. A seguir, uma "feature" da árvore é escolhida segundo algum critério, extrai-se a "feature" do sólido, e a partir deste nó da árvore inicia-se o processo novamente, até que se encontre um elemento primitivo (p.ex.: um paralelepípedo);

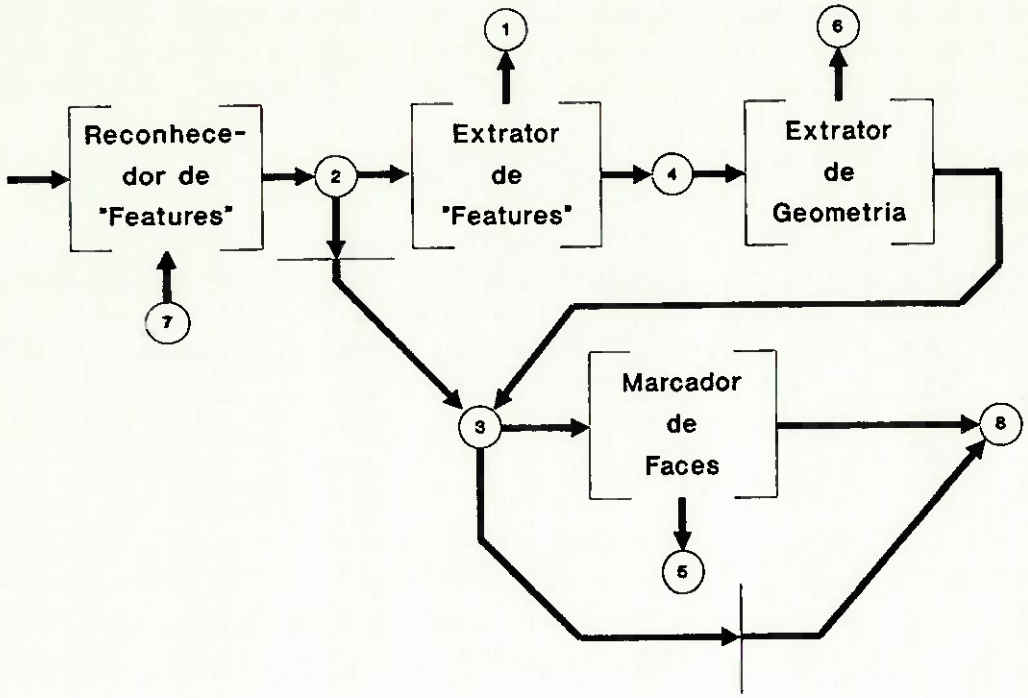
- **Estrutura de Avaliação de Relacionamentos**: Este sub-módulo do Escalonador Automático de "features" tem por objetivo determinar uma estrutura de dados que torne simples a identificação dos relacionamentos entre as "features", por meio de regras simples. Para isto se utilizará a estrutura hierárquica criada no sub-módulo descrito acima.

- **Relatórios de Saída**: Este sub-módulo do Escalonador Automático de "features" tem por função gerar relatórios que permitem ao usuário verificar o conteúdo de cada nó das estruturas, de forma a conferir e completar informações geradas pelo sistema. Após passar por ele, os dados gerados pelo Escalonador Automático de "Features" são passados ao módulo de seleção do processo de usinagem.

3.3 PFS – Reconhecedor Automático de "Features"

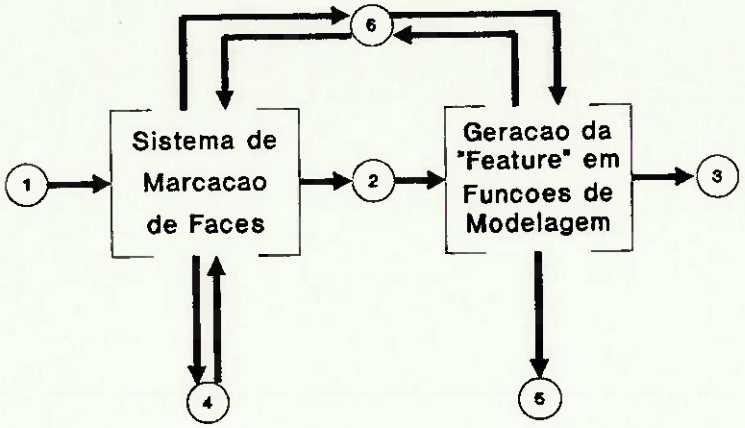
A estrutura do módulo de *Reconhecimento Automático de "Features"* é apresentada através do diagrama PFS da Figura 3.3. Cada um de seus elementos ativos e interrelacionamentos são apresentados a seguir:

- **Reconhecedor de "Features"**: Este sub-módulo tem por objetivo realizar o reconhecimento e localização da "feature" no sólido. A saída é *"Feature" reconhecida* ou *"Feature" não reconhecida.*



1	Sólido e "feature" extraída
2	"Feature" reconhecida ou não e "flags"
3	"Flag" para marcar faces
4	Sólido com a "feature" reconhecida
5	Faces marcadas
6	Geometria extraída
7	Banco de dados de "features"
8	Resposta do reconhecedor

Figura 3.3: PFS – Sistema de Reconhecimento Automático de "Features".



1	"Flag" de acionamento do editor
2	Faces Marcadas
3	Reinicializador do sistema
4	Usuário
5	Banco de dados de "features" com nova "feature" modelada
6	Sólido com "features" restantes

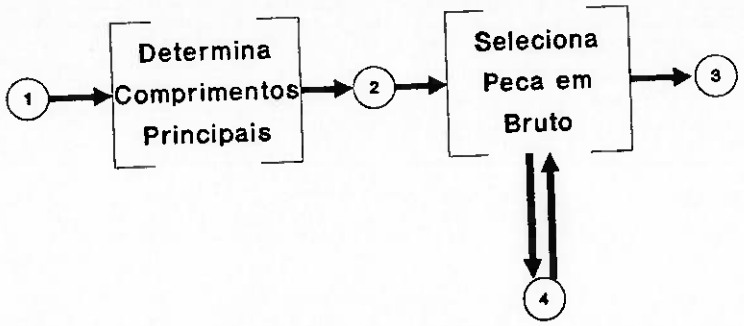
Figura 3.4: PFS – Editor de "Features".

- **Marcador de Faces**: Se a "feature" for reconhecida e o "flag"[1] para marcar faces estiver setado, o *Marcador de Faces* será acionado. A finalidade deste sub-módulo é permitir que várias "features" de um mesmo tipo possam ser reconhecidas em um mesmo nível da *Árvore de "Features"*.
- **Extrator de "Features"**: Se a "feature" for reconhecida e o "flag" para extrair a "feature" do sólido estiver setado, o *Extrator de "Feature"* gera um arquivo contendo o sólido com a "feature" extraída em funções de modelagem.
- **Extrator de Geometria**: Após a extração da "feature", as informações geométricas necessárias para a geração de um processo de usinagem são obtidas do sólido.

3.4 PFS – Editor de "Features"

A estrutura do módulo do *Editor de "Features"* é apresentada através do diagrama PFS da Figura 3.4. Cada um de seus elementos ativos e interrelacionamentos são apresentados a seguir:

[1]Este "flag" e os outros do módulo de reconhecimento, são gerenciados pelo escalonador.



1	Árvore de "features"
2	Comprimentos x, y, z do sólido
3	Árvore de "features" mais sobremetal
4	Banco de dados de peças em bruto

Figura 3.5: PFS – Verifica Sobremetal.

- **Sistema de Marcação de Faces**: O usuário (4) interage com o sistema, marcando as faces no sólido [2] [3] que formarão a "feature" desejada.
- **Geração da "Feature" em Funções de Modelagem**: Baseado nas faces marcadas (2) e no sólido original (1), este sub-módulo gera a "feature" em Funções de Modelagem (seção 2.4).

3.5 PFS – Verifica o Sobremetal

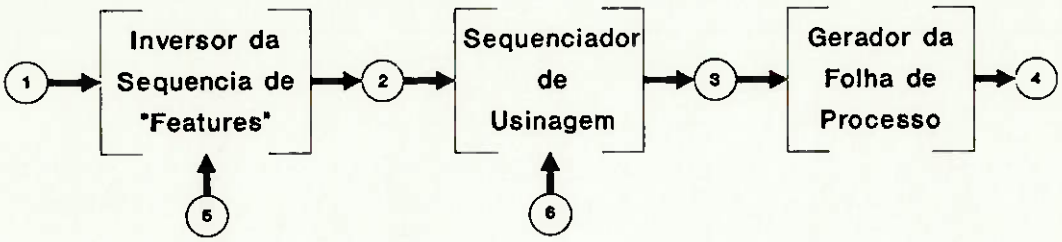
A estrutura do módulo de *Verificação de Sobremetal* é apresentada através do diagrama PFS da Figura 3.5. Cada um de seus elementos ativos e interrelacionamentos são apresentados a seguir:

- **Determina Comprimentos Principais**[4]: Este sub-módulo tem por objetivo determinar os comprimentos máximos da peça nas direções x, y, z.

[2]Sólido gerado pelo próprio usuário e que contenha a "feature" que lhe interessa. Neste caso, o sistema é utilizado somente para gerar "features" para o banco de dados.

[3]Sólido que restou do processo de geração do Plano de usinagem. Neste caso o editor é acionado quando não se atingiu um sólido primitivo (p.ex. um paralelepipedo) após a construção da árvore de "features".

[4]Nesta fase do projeto utilizaremos somente peças em bruto de formato de paralelepípedo.



1	Sequência de "features" do escalonador e relacionamentos
2	Sequência invertida de "features"
3	Sequência de "features" para usinagem
4	Folha de processo
5	Heuristicas de sequência de usinagem
6	Heuristicas de manufatura

Figura 3.6: PFS – Seleção do Processo

- **Seleciona Peça em Bruto**: A partir dos comprimentos principais do sólido é selecionada uma peça em bruto que consta do *Banco de Dados de Peças em Bruto*. Este banco de dados deve conter – em uma situação real – somente peças disponíveis em estoque, ou com possibilidade de ser adquirida.

3.6 PFS – Seleção do Processo de Usinagem

A estrutura do módulo de *Seleção do Prcesso de Usinagem* é apresentada através do diagrama PFS da Figura 3.6. Cada um de seus elementos ativos e interrelacionamentos são apresentados a seguir:

- **Inversor Da Sequência**: este sub-módulo tem por função construir uma estrutura baseada numa das sequências de extração da árvore de "features". Esta estrutura será uma sequência de "features" invertida em relação à sequência da estrutura da árvore, para que se possa gerar o plano de processo. A inversão desta sequência é necessária e os motivos são apresentados no capítulo 6;
- **Gerador Da Sequência De Usinagem**: este sub-módulo tem por função trocar a posição de algumas das "features" da sequência, aplicando heurísticas de modo a

racionalizar tempos de troca de ferramentas, de máquinas, etc.... Este módulo é necessário para evitar uma sequência de extração aleatória, que poderia levar a um processo de fabridação longo e de alto custo;

- **Gerador Da Folha De Processo**: este sub-módulo tem por função criar uma folha de processo aplicando heurísticas para se determinar velocidade de corte, ferramentas de desbaste e acabamento, etc.... Este módulo deve possuir uma conexão com um banco de dados eficiente e um programa de tecnologia de grupo, para que se racionalize a busca dos parâmetros.

Capítulo 4

Reconhecimento Automático de "Features"

Este módulo tem por função realizar o reconhecimento, localização e/ou extração e marcação de "features" no sólido de interesse. Este módulo tem por entrada a "feature" a ser reconhecida e o sólido representados por funções de modelagem.

4.1 Algoritmo de Reconhecimento de "Features"

Os algoritmos para comparação de sólidos e reconhecimento de "features" implementam a busca por bijeções entre dois subconjuntos de laços. Os algoritmos são estruturados em três principais módulos, conforme o algoritmo abaixo:

- Passo 1: seleciona a primeira associação [1].
- Passo 2: acrescenta uma associação ao conjunto de associações.
- Passo 3: realiza o "backtracking"[2] para uma associação considerada incorreta.

Estrutura do algoritmo

<Passo 1: Seleciona a primeira associação>
ok = **true**
k = *1*
repeat

[1] Associação de um laço do sólido A com um laço do sólido B

[2] Ao se construir um grafo de busca, pode-se verificar que a escolha de um determinado caminho foi inadequada. O "backtracking" é utilizado para se retornar a um determinado nó anterior e escolher um novo caminho a partir deste[Nilsson 82].

```
        if ok == true
            ok = false
            <Passo 2: Seleciona uma nova associação>
            if <Passo 2 com sucesso>
                    ok = true
                    k = k + 1
        else
            k = k - 1
            <Passo 3: realiza o "backtracking">
            if <Passo 3 com sucesso>
                    ok = true
                    k = k + 1
    until k == 0 or k == n
```

- ok representa o estado do algoritmo
- k representa o número de associações já realizadas
- n representa o número total de associações a serem selecionadas

No Passo 1 deve ser selecionada a primeira associação (la^1, lb^1). É muito importante que sejam consideradas heurísticas ao se selecionar a primeira associação de laços, permitindo encurtar o tamanho da árvore de busca.

No Passo 2 uma nova associação deve ser selecionada (la^k, lb^k). É conveniente que heurísticas sejam definidas para a escolha do laço la^k, aumentando a eficiência das comparações entre as funções de modelagem Λ^* e Λ^+.

O Passo 3 deverá ser executado caso não seja possível encontrar uma bijeção no passo 2. O laço la^k previamente selecionado, deve ser associado a um novo laço lb^k, tal que as mesmas condições do Passo 2 sejam satisfeitas. Nos Passos 2 e 3, é conveniente que a escolha do laço lb^k seja feita pelas mesmas regras que foram adotadas para a escolha do laço la^k.

O tamanho da árvore de busca pode ser encurtado, pela aglutinação de informações topológicas, como o número de arestas do laço, às funções de modelagem.

Para o algoritmo de comparação entre sólidos e reconhecimento de "features", o nível máximo é definido pelo número de laços dos sólidos que estão sendo comparados. Para o algoritmo de reconhecimento de "features", o nível máximo é definido pelo número de laços da "feature" a ser reconhecida.

4.2 Implementação

Foram implementados quatro versões do Sistema de Reconhecimento Automático de "Features" (versões 0.0, 1.0, 1.1 e 1.2) até a finalização da versão final. A implementação dos

programas foi baseada no algoritmo apresentado na seção anterior e detalhes de maior importância da versão 1.2 são descritas à seguir (Os detalhes das versões 0.0, 1.0 e 1.1 são explicados em [Toledo 91a]). Abaixo é apresentada uma tabela destacando as diferenças entre cada versão.

Versão	Topologia conexa	Topologia desconexa	Informações Geométricas	Processo de busca
0.0	sim	não	não	arbitrário
1.0	sim	não	não	heurística
1.1	sim	sim	não	heurística
1.2	sim	sim	sim	heurística

Nesta versão são utilizadas as funções Λ^* e Λ^+ baseadas nas relações de adjacência $l < L >$ e $l[L]$. Portanto qualquer tipo de sólido pode ser comparado a nível de topologia. As informações geométricas utilizadas no processo de reconhecimento de "features" são discutidas na seção 4.5.

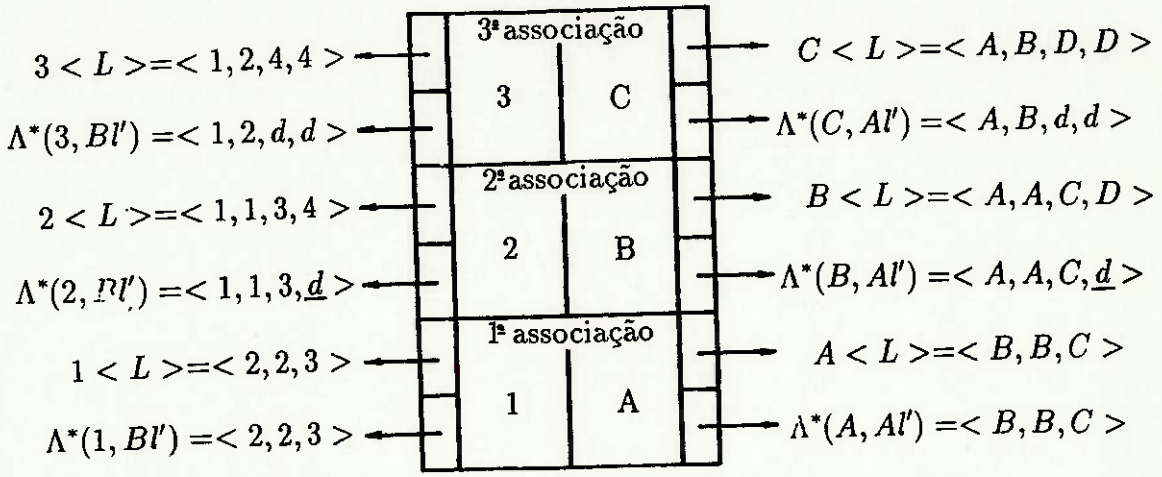
A seguir são apresentados os módulos principais do programa destacando os mecanismos utilizados para aumentar sua eficiência.

4.3 Módulos Principais - Versão 1.2

4.3.1 Passo 1

No **Passo 1** (do algoritmo apresentado na seção 2.5) é realizada a escolha da primeira associação (la^1, lb^1). Este passo é muito importante pois se for realizada uma escolha errada do primeiro par, este equívoco poderá ser descoberto somente em um estágio mais avançado do processo de comparação e todo o tempo gasto até este momento será perdido. Sendo assim, é interessante criar um bom mecanismo para a escolha do primeiro par e então reduzir a quantidade de possíveis associações no primeiro passo, contribuindo para uma conclusão mais rápida do processo de comparação.

O mecanismo implementado para o passo 1 do comparador de sólidos é baseado no levantamento estatístico da quantidade de faces com um mesmo número de arestas. Com este levantamento estatístico pode-se determinar a quantidade de arestas (N_a) das faces com mesmo número de arestas que aparecem menos vezes no sólido (ou "feature") A. Será escolhido como possível primeira face de A uma face qualquer do sólido A que tenha N_a arestas. No sólido B será montada uma lista com todas as faces desse sólido que apresentam N_a arestas. Portanto, no pior caso, a face escolhida do sólido A será associada com todas as faces do sólido B que têm N_a arestas.



No exemplo desta figura, o próximo par a ser associado será o par $(D, 4)$, pois o primeiro "d" encontrado em uma função Λ^* ou Λ^+ corresponde a face D na relação de adjacência $la^2 < L >$ e a face 4 na relação de adjacência $lb^2 < L >$.

Figura 4.1: Exemplo - Passo 2

4.3.2 Passo 2

No **Passo 2**, a escolha de um novo par de faces não é feita de forma aleatória. O próximo par a ser escolhido é o primeiro "d" ("don't care" – face não associada que consta em alguma relação de adjacência das faces já associadas) encontrado nas funções Λ^* ou Λ^+ (percorridas alternadamente) das faces já associadas partindo do primeiro par. A figura 4.1 ilustra através de um exemplo o processo descrito.

Com este mecanismo de escolha da próxima associação, o grau de aleatoriedade diminui de maneira significativa, permitindo que o algoritmo convirja para uma solução mais rapidamente. O caminho de escolha das faces se assemelha a um espiral que envolve a primeira face escolhida (Figura 4.2). Utilizando este mecanismo, o processo de busca pela "condição necessária e suficiente para que dois sólidos sejam topologicamente equivalentes" é, em seu pior caso, proporcional ao quadrado do número de laços do sólido. Dependendo da eficiência na escolha da primeira associação (Passo 1), o processo de busca chega a ser, em seu pior caso, proporcional ao número de laços do sólido. Na figura 4.3 pode-se observar a grande eficiência do algoritmo implementado

4.3.3 Passo 3

O **Passo 3** realiza o "backtracking" para uma associação considerada incorreta. No algoritmo implementado, todas as faces associadas a partir da função Λ^* ou Λ^+ que gerou a associação considerada incorreta são desassociadas. Os novos pares serão escolhidos a partir de uma nova posição de concatenação das listas circulares (relação de adjacência $l < L >$ e função de modelagem Λ^*) desta face (se a associação incorreta foi gerada por

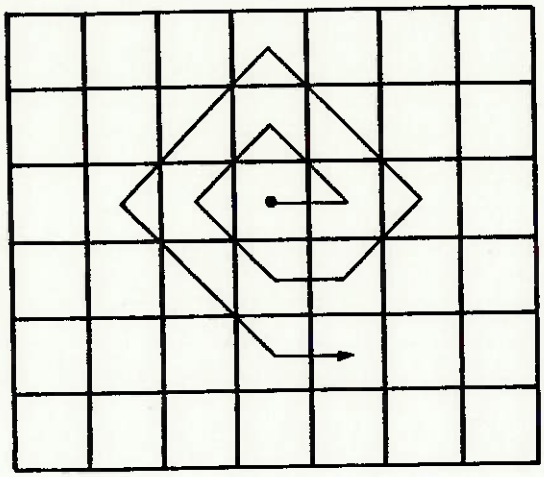
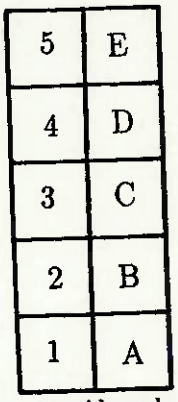


Figura 4.2: Caminho de escolha das faces

Figura 4.3: Eficiência do algoritmo implementado



No exemplo desta figura, considerando que os pares $(D,4)$e$(E,5)$ tenham sido escolhidos a partir das funções do par $(B,2)$, e a associação $(E,5)$ foi considerada incorreta, então os pares $(D,4)$ e $(E,5)$ serão retirados. Uma nova posição de concatenação para o par $(B,2)$ será procurada e as novas associações serão realizadas a partir desta nova situação.

Figura 4.4: Exemplo - Passo 3

uma função Λ^*) ou a partir de uma nova posição de combinação das listas desconexas (se a associação incorreta foi gerada por uma função Λ^+). Esgotando-se todas as possibilidades de concatenação, todo o processo descrito acima será repetido, ou seja, novo "backtracking" será realizado. A figura 4.4 mostra um exemplo do processo descrito.

Se após um "backtracking" o algoritmo retornar ao primeiro par associado e este não apresentar uma nova posição de concatenação, então um novo primeiro par deverá ser escolhido. Esta escolha será feita a partir dos possíveis primeiros pares determinados estatisticamente no primeiro passo.

4.4 Principais Estruturas - Versão 1.2

4.4.1 Armazenamento do Sólido

Estrutura de armazenamento das informações topológicas necessárias do sólido:

```
struct solido
{
      face loop ;
      face *l_l ;
      face *l_d_l ;
      int n ;
      int d ;
} ;
```

- **face:** tipo de dado utilizado para nomear as faces. Foi definido como tipo *int* (*typedef int face*).

- **loop:** laço do sólido.
- ***l_l:** ponteiro para a lista $l < L >$ do laço.
- ***l_d_l:** ponteiro para a lista $l\,[L]$ do laço.
- **n:** tamanho da lista $l < L >$ do laço (equivalente ao número de arestas do laço).
- **d:** tamanho da lista $l\,[L]$ do laço (equivalente ao número de laços externos e internos do laço - *loop*).

As informações necessárias do objeto são obtidas a partir de um arquivo gerado pelo modelador de sólidos contendo informações topológicas e geométricas do objeto. Elas são colocadas em um vetor da estrutura descrita. A estrutura de armazenamento das informações geométricas do sólido é explicada na próxima seção.

4.4.2 Armazenamento da Pilha

Estrutura de uma posição da pilha utilizada para armazenar o estado de comparação:

```
struct pilha
{
      struct a ;
      {
            face a ;
            face *l_l ;
            face *h_lbd_l[2] ;
            face *l_d_l ;
            face *h_lbd_d_l[2] ;
            STR_GEO *geo_l_l ;
      } a ;
      struct b ;
      {
            face b ;
            face *gl_l ;
            face *lbd_gl[2] ;
            face *gl_d_l ;
            face *lbd_d_gl[2] ;
            STR_GEO *geo_gl_l ;
      } b ;
      int
            n,
            d,
            assocs,
            rots,
            descs,
            *combs,
} ;
```

A *struct a* se refere ao sólido A e a *struct b* se refere ao sólido B.

- **a:** laço do sólido A (Domínio).

- ***l_l:** relação de adjacência $l < L >$ do laço do sólido domínio.
- ***h_lbd_l:** Imagem da função Λ^*.
- ***l_d_l:** relação de adjacência $l\,[L]$ do laço do sólido domínio.
- ***h_lbd_d_l:** Imagem da função Λ^+.
- ***geo_l_l:** Lista que contém as informações geométricas de cada face da relação de adjacência $l < L >$ do laço **a**.
- **b:** laço do sólido B (Imagem).
- ***gl_l:** relação de adjacência $l < L >$ do laço do sólido imagem.
- ***lbd_gl:** função Λ^* do laço do sólido B.
- ***gl_d_l:** relação de adjacência $l\,[L]$ do laço do sólido imagem.
- ***lbd_d_gl:** função Λ^+ do laço do sólido B.
- ***geo_gl_l:** Lista que contém as informações geométricas de cada face da relação de adjacência $l < L >$ do laço **b**.
- **n:** número de arestas do laço do sólido A ou B, já que por definição o número de arestas dos laços dos dois sólidos tem que ser o mesmo para ocorrer equivalência.
- **d:** tamanho da relação de adjacência $l\,[L]$ dos laços do par associado nesta posição da pilha.
- **assocs:** quantidade de pares associados a partir da parte conexa desta posição na pilha.
- **rots:** utilizado para verificar se as listas circulares nesta posição já realizaram uma rotação completa.
- **descs:** quantidade de pares associados a partir da parte desconexa desta posição na pilha
- ***combs:** utilizado para verificar se as listas referentes a parte desconexa já realizaram todas as combinações possíveis.

As funções de modelagem das faces já associadas podem se modificar ligeiramente com a associação de um novo par de faces à pilha ou realização de "backtracking". Assim, procurou-se, com a estrutura da pilha (*struct pilha*), armazenar as funções de modelagem das faces associadas, e alterar somente algumas informações de algumas funções de modelagem, descartando-se a necessidade de montar todas as funções de modelagem para cada alteração na pilha. As funções de modelagem do sólido domínio (sólido A) são, na

verdade, armazenadas na pilha na forma de imagem de suas funções de modelagem. Dessa forma, a comparação das funções de modelagem dos dois sólidos é feita de forma direta, sem a necessidade de se calcular a imagem da função de modelagem do sólido A todo instante.

Além disso, as funções de modelagem das faces associadas dos dois sólidos têm uma posição de concatenação para associar seus elementos, já que são listas circulares. O algoritmo implementado busca por essa concatenação assim que houver alguma informação na função de modelagem e armazena na pilha em sua forma concatenada, evitando a realização de operações desnecessárias.

4.5 Informações Geométricas

Apesar do algoritmo estar baseado em informações topológicas obtidas de um modelo B-rep e contidas na estrutura do sólido do Reconhecedor Automático de "Features", as informações geométricas são importantes para se definir exatamente a "feature" e o sólido.

As informações geométricas estão sempre associadas às faces do sólido e estão contidas em uma estrutura paralela à estrutura que contém as informações topológicas do sólido.

Estrutura de armazenamento das informações geométricas do sólido ou da "feature":

```
struct geometria
{
    face loop ;
    STR_GEO *geo_l_l ;
    int n ;
} ;
```

- **loop:** laço do sólido.
- ***geo_l_l:** ponteiro para a lista que contém as informações geométricas de cada face da relação de adjacência $l < L >$ do laço definido em `loop`.
- **n:** tamanho da lista apontada por *geo_l_l (equivalente ao número de arestas do laço).
- **STR_GEO:** estrutura das informações geométricas consideradas.

Se fossem considerados, por exemplo, os comprimentos das arestas e ângulos entre faces, a estrutura STR_GEO teria o seguinte formato:

```
struct STR_GEO
{
    float ang ;
    float comp ;
} ;
```

Convém notar que as informações geométricas consideradas estão totalmente associadas às relações de adjacência $l < L >$ do sólido. A figura 4.5 ilustra como são obtidas as informações geométricas da estrutura em que estão armazenadas e o que elas significam. As informações de ângulo entre faces e comprimento de aresta são as informações geométricas consideradas no exemplo.

As informações geométricas somente são utilizadas no processo de comparação de sólidos depois de considerada a equivalência topológica entre as faces dos dois sólidos (ou um sólido e uma "feature").

4.5.1 Determinação das Informações Geométricas

O Sistema de Reconhecimento Automático de "Features" foi implementado de forma a suportar qualquer tipo de informação geométrica que o usuário determinar. O sistema suporta também a utilização de várias informações geométricas simultaneamente.

Inicialmente, estão implementados três tipos de informações geométricas:

caso 1 – ângulos entre faces.

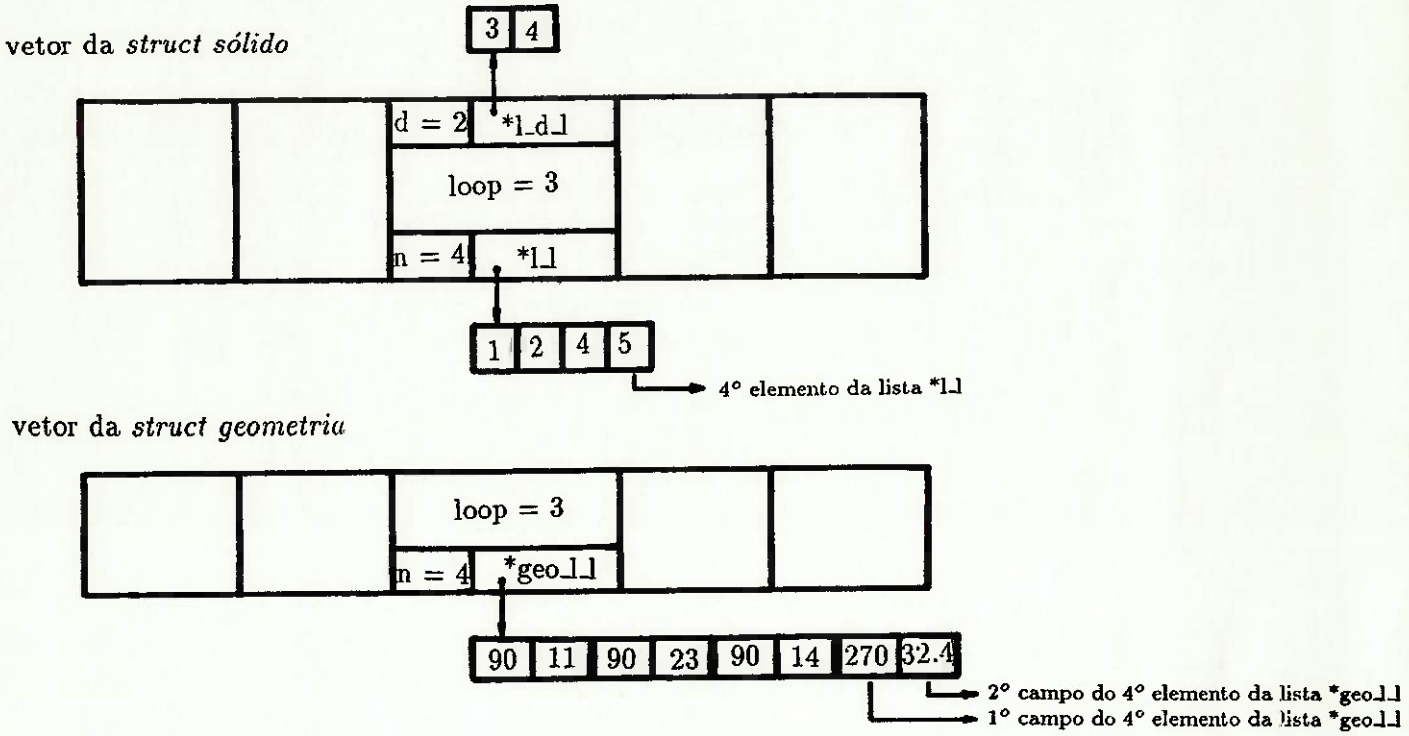
caso 2 – comprimentos das arestas.

caso 3 – ângulos entre faces e comprimentos das arestas.

Se o usuário optar por um dos três casos já existentes, ele deve definir no arquivo *compara.h*, na posição de definição dos casos (já consta no arquivo uma definição inicial) o caso que ele irá utilizar. Por exemplo, se o usuário se decidir pela utilização do caso 2, deverá constar no arquivo *compara.h* a seguinte implementação:

```
#undef caso1
#define caso2
#undef caso3
```

Caso o usuário necessite de outras informações geométricas, ele mesmo pode criar as funções necessárias para que essas informações geométricas sejam consideradas. As modificações são simples, pois as funções a serem criadas serão semelhantes às já existentes, com excessão de pequenos trechos que estão impressos com letras maiores para destacar a mudança. Para acelerar as modificações, pode-se copiar blocos contendo as funções já criadas e apenas realizar as modificações.



Do exemplo desta figura obtemos que o ângulo entre as faces 3 (*loop* = 3, indicado na struct solido) e 5 (4° elemento da lista apontada por *l_l) é 270° (1° campo do 4° elemento da lista apontada por *geo_l_l). Da mesma forma, o comprimento da aresta entre as faces 3 e 5 é 32.4mm (2° campo do 4° elemento da lista apontada por *geo_l_l). Portanto, as informações geométricas de uma determinada face estão localizadas em posições análogas no vetor da *struct geometria*. O número de campos é determinado pelo número de informações geométricas consideradas.

Figura 4.5: Associação da topologia e geometria

4.6 Extrator de "Features"

Para a construção da árvore de "features" é de fundamental importância que se extraiam as "features" reconhecidas no sólido.

Uma solução genérica pode ser obtida pela implementação de um sistema baseado em operações booleanas de um modelador de sólidos. Como no período de implementação do sistema, o Modelador de Sólidos Didático do LAS-EPUSP não estava com algumas das funções necessárias à implementação do *Extrator de "Features"* em funcionamento, buscou-se um outro tipo de solução. A solução encontrada baseia-se em Funções de Modelagem (seção 2.4). As funções de modelagem do sólido de interesse são manipuladas de forma a gerar um sólido sem a "feature" que se desejava extrair.

4.6.1 Processo de Extração

O sistema implementado é baseado em algumas regras que são enumeradas a seguir e para cada uma destas regras é apresentada um exemplo ilustrativo do que representaria as modificações das funções de modelagem em um mundo físico.

1. **Extração de Laços:** O primeiro passo para a extração de uma "feature" é realizada atrvés da seguinte heurística:

   Heurística 1 *Todos os laços da "feature" reconhecida devem ser retirados das Funções de Modelagem correspondentes ao sólido.*

   A "feature" reconhecida é localizada no sólido. Isto é possível pois no processo de reconhecimento da "feature" no sólido é realizada uma busca por uma bijeção dos laços da "feature" para os laços de um pedaço do sólido (Seção 2.3, Teorema 1). Os laços correspondentes a esta "feature" são então retirados das funções de modelagem do sólido (Figura 4.6).

2. **Aglutinação de Arestas:** Há casos em que a extração das "features" não é possível somente utilizando o heurística 1. É necessário a execução de outros passos para que se obtenha a extração completa da "feature" no sólido. Na Figura 4.7, o sólido resultante da aplicação do heurística 1 não é topologicamente equivalente a um paralelepípedo (apesar de visivelmente ser), sendo necessário a retirada do vértice v e a fusão das arestas a'_m e a'_n em uma única aresta. Para a ocorrência disto, foi criado a seguinte Heurística:

   Heurística 2 *Se duas arestas vizinhas fazem fronteira com os mesmos laços, então estas duas arestas são transformadas em uma única aresta.*

   A aplicação desta heurística é exemplificada na Figura 4.8.

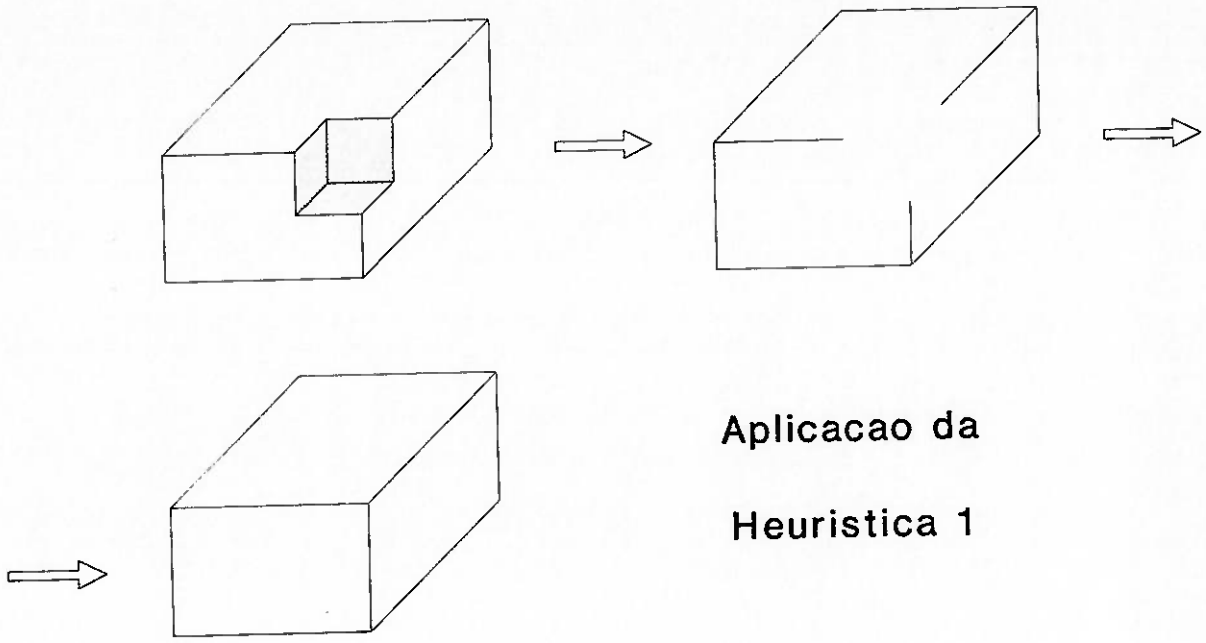


Figura 4.6: Processo de extração de laços do sólido.

Heurística 2 *Se duas arestas vizinhas fazem fronteira com os mesmos laços, então estas duas arestas são transformadas em uma única aresta.*

A aplicação desta heurística é exemplificada na Figura 4.8.

3. **Extração de Laços Internos:** Para extração de "Features" que estejam alojadas internamente a um laço (p.ex.: furos), é necessária a extração dos laços da "feature" no sólido (Heurística 1), bem como do(s) laço(s) internos restantes. Para isto, foi criada a seguinte Heurística:

   Heurística 3 *Todos os laços que não fazem fronteira com nenhum laço devem ser retirados das funções de modelagem do sólido.*

   A aplicação desta heurística é exemplificada na Figura 4.9.

Apesar de não ter sido a intenção inicial tornar este extrator o definitivo do sistema CAPP, já se estuda a possibilidade de acrescentar mais algumas Heurísticas para extração, tornando-o mais genérico para satisfazer às necessidades do sistema. A velocidade de extração de "features" seria muito maior do que a da solução que trabalha em cima de operações booleanas do Modelador de Sólidos Didático desenvolvido no LAS-EPUSP.

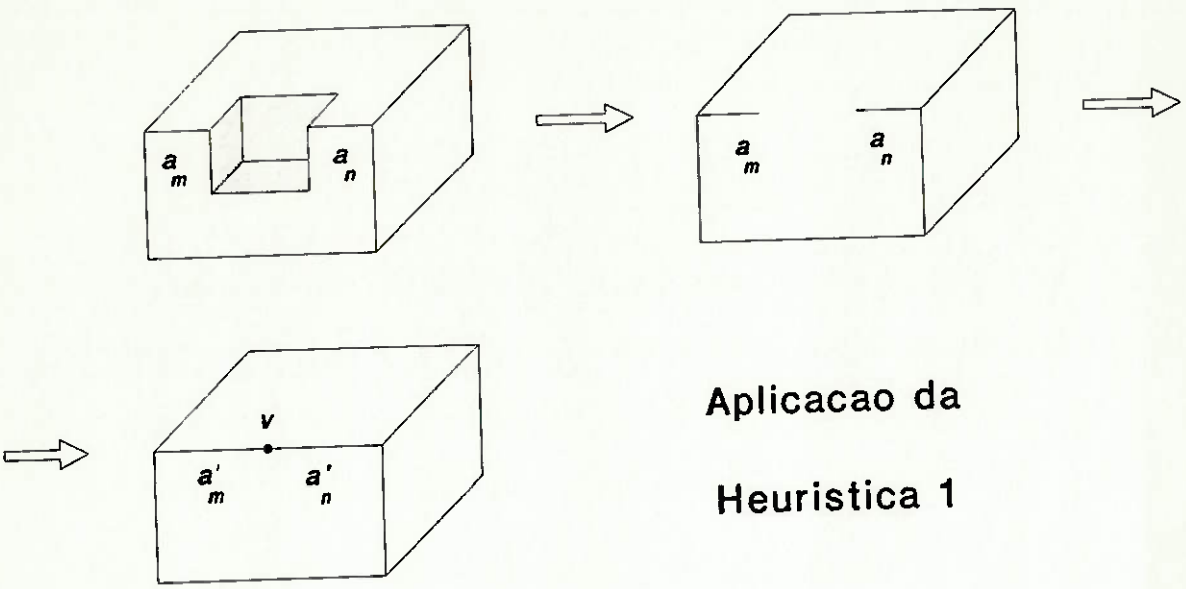


Figura 4.7: Extração incompleta de uma "feature"

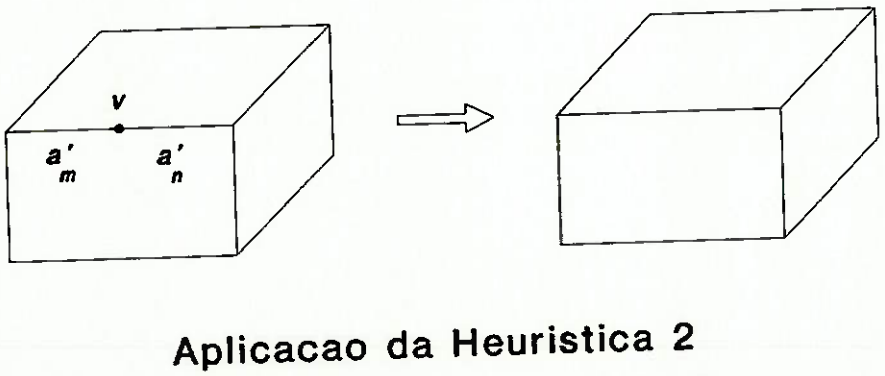


Figura 4.8: Extração completa da "feature" do exemplo da figura anterior.

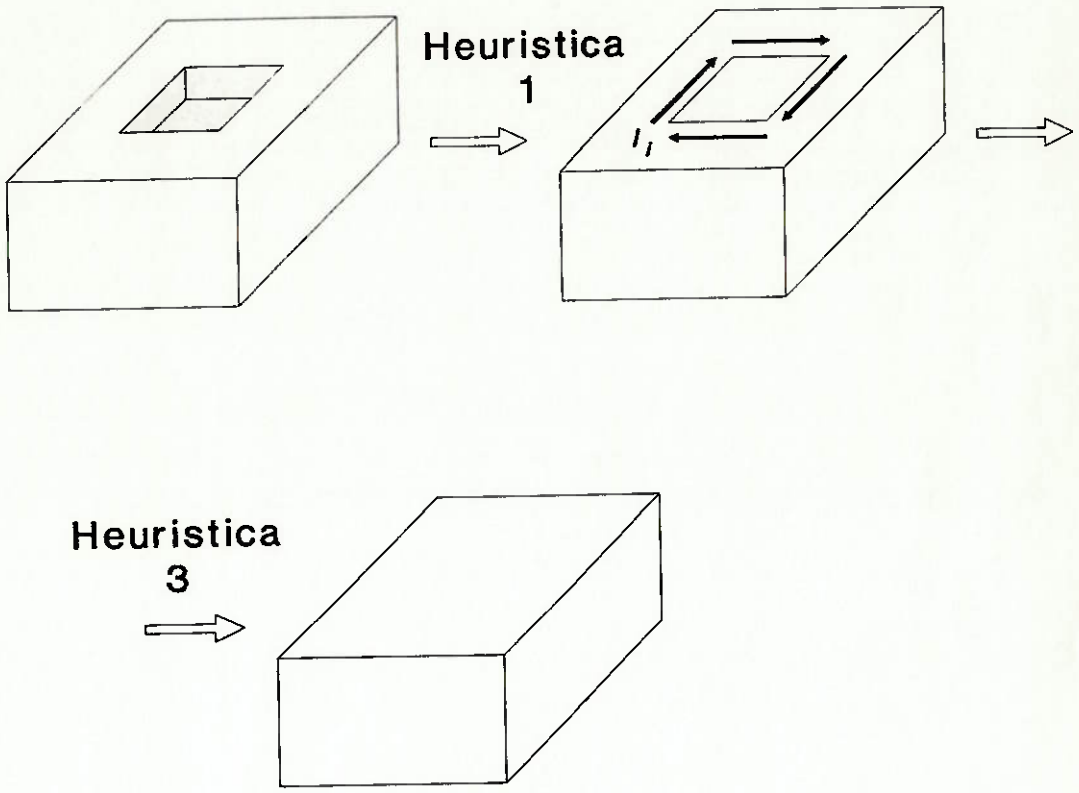


Figura 4.9: Exemplo de aplicação da Heurística 3.

Capítulo 5

Escalonamento Automático de "Features"

Neste capítulo, discutiremos detalhadamente o módulo de escalonamento automático de "features". O capítulo foi dividido em seções que contém em cada uma um dos sub-módulos que compõe o sistema e que foram apresentados préviamente na seção 3.2. Esta discussão vai desde as especificações de cada módulo até sua implementação.

5.1 Arquivo de "Features"

O arquivo de "features" é um arquivo que conterém nomes de arquivos que correspondem às "features"[1] que compõem a nossa base de dados, definindo o conjunto de "features" que o sistema conhece e pode fabricar. O escalonador de "features" seleciona uma "feature" por vez e envia ao reconhecedor automático de "features" de maneira que ele possa comparar esta "feature" com o sólido que está sendo manipulado.

O *Arquivo de "features"* foi implementado de forma que as "features" modeladas seguem um dos três modos descritos acima. Estas "features" são ditas **"features" para reconhecimento**, já que elas são usadas pelo sistema para a comparação com o sólido com o qual se pretende planejar o processo. A cada uma destas **"features" para reconhecimento** é associada uma segunda "feature" dita **inversa da "feature"** , que é exatamente o volume que a **"feature" para reconhecimento** ocupa no sólido. Este volume nos é útil, quando "retirar-se" as "features" do sólido, utilizando *Operações Booleanas*. Na versão atual, este sistema trabalha com um extrator de "features" que é descrito nas seções 3.3 e 4.6, não havendo a necessidade do cálculo da "feature" inversa. Porém, como este extrator não é totalmente genérico, é necessário estudar uma maneira eficiente de torná-lo utilizável para qualquer método. Uma solução baseada no conceito de "features" inversas tamém deverá ser estudado.

[1] Modeladas segundo funções de modelagem. Veja seções 2.4 e 2.5.

Quando se realiza o reconhecimento de "features" as características geométricas de uma "feature" reconhecida são coletadas. Assim, a unicidade de cada "feature" encontrada no sólido é garantida, mesmo que existam "features" do mesmo tipo e com as mesmas características geométricas em posições diferentes no sólido. Por outro lado, tendo as características geométricas da "feature" encontrada disponíveis, bem como o modelo da "feature" inversa, pode-se determinar características geométricas da **inversa da "feature"** correspondente[2].

Além dos dois tipos de "features" descritos acima, é conveniente que esteja associado a cada uma das "features" do *Arquivo de "features"* um nome para que se facilitem futuras análises sobre o sólido modelado.

Assim, podemos dizer que o arquivo de "features" é composto por nomes de arquivos que contém as "features" modeladas. Cada um destes arquivos conterém uma "feature" modelada e pode assim ser utilizado diretamente pelo *Reconhecedor Automático de "features"*.

O número de "features" armazenadas no *Arquivo de "features"* influencia diretamente sobre a velocidade do *Escalonador Automático de "features"*, já que para cada nível da *Árvore de "features"* são realizadas no mínimo tantas comparações quantas forem o número "features" presentes no *Arquivo de "features"*. Entretanto, quanto maior o número de "features" no *Arquivo de "features"* mais genérico será o sistema. Este trabalho não objetiva encontrar um *Arquivo de "features"* que torne o sistema totalmente genérico. A procura por tal conjunto de "features" merece a atenção de um trabalho próprio. A estrutura deste arquivo é aqui estudada já que este é um módulo integrante do *Escalonador Automático de "features"*, de modo que seu funcionamento possa ser avaliado.

5.2 Árvore de "Features"

5.2.1 Estrutura e Nomenclatura da Árvore Binária

Na concepção de uma árvore binária, existem três ponteiros básicos: **left** e **right** que são "filhos" de um nó e **pai**, que é o nó acima de um determinado "filho". Assim, o ponteiro **pai** terá um ponteiro para um "filho" **left** e outro para um "filho" **right**, enquanto cada um destes dois tipos de "filhos" terá um ponteiro apontando para **pai**.

A *Árvore de "features"* não será binária na concepção exata do termo. Isto porque apesar de cada nó da *Árvore de "features"* ter apenas dois "filhos" diretos, poderá ter como "filhos indiretos" tantos nós quantas forem as "features" independentes e aninhadas associadas a este nó.

Deste modo, o "filho" **left** de um nó indicará que descemos um nível, para um filho

[2]Este algoritmo não foi implementado neste trabalho, já que sua complexidade exigiria um trabalho exclusivo para sua execução.

direto. O nó que estiver ligado ao "filho" **left** por um ponteiro **right**, será considerado um "irmão" do nó **left**, estando no mesmo nivel de extração de "features" e tendo por **pai** o mesmo nó de **left**, o mesmo acontecendo com todos os outros nós ligados sucessivamente por ponteiros **right**.

5.2.2 Operadores Básicos para Construção de uma Árvore Binária

A construção de uma árvore binária segundo um certo critério necessita da especificação de alguns operadores básicos. Com base nestes operadores podemos construir uma árvore binária, nos moldes descritos acima, seja qual for o critério adotado. Estes operadores são:

1. **Cria todos os filhos segundo um certo Critério**: dado um nó da árvore, este operador procura todos os filhos deste nó segundo um certo critério definido pelo usuário e insere os filhos encontrados na árvore.

   Parametros ::= **PAI** (nó)
   Critério (função)

   Devolve ::= Primeiro filho a esquerda
   se **NULL**, não tem filho

2. **Busca Primeiro Filho**: dado um nó, verifica se ele possui filho, e retorna o primeiro filho à esquerda (caminha um nível abaixo). *obs: note que esta segunda operação está inclusa na primeira.*

   Parametros ::= **PAI** (nó)

   Devolve ::= Primeiro filho a esquerda
   se **NULL**, não tem filho

3. **Busca Primeiro Irmão**: dado um nó, verifica se ele possui um irmão à direita, e retorna o nó associado ao primeiro irmão à direita.

   Parametros ::= nó

   Devolve ::= Primeiro irmão a direita
   se **NULL**, não tem filho

4. **Busca o Pai**: dado um nó, verifica se ele possui pai e retorna o nó associado a ele.

Parametros ::= nó

Devolve ::= Pai
se **NULL**, não tem pai

O algoritmo de construção da árvore binária fica:

```
Algoritmo(construcao da arvore binaria)
{
   possivel = true
   while (possivel == true)
   {
      (cria todos os filhos)
      if (existe filho)
         (busca primeiro filho)
      else
      {
         if (existe irmao)
            (busca primeiro irmao)
         else
         {
            (assuma que nao existe irmao)
            while ((existe pai) e (nao existe irmao))
               (busca pai)
            if (existe pai)
               (busca primeiro irmao)
            else
               (possivel = false)
         }
      }
   }
}
```

Até aqui, só se discutiu a construção da árvore em si, sem se entrar em detalhes de como serão os critérios para a inserção dos nós. Para o caso da *Árvore de "features"*, estes critérios serão estabelecidos na próxima sub-seção.

Note que o algoritmo aproveita o fato de que a árvore desce de nível sempre pelo lado esquerdo (filho) para realizar a sequenciação de "features" da esquerda para a direita, ou seja, o primeiro ramo da árvore que desce até o último nível é o esquerdo. Assim, a árvore será construida da esquerda para a direita, ramo a ramo. Note ainda que apenas no módulo *Cria todos os filhos segundo um critério* será realizada inserção de nós, facilitando o controle de construção da árvore. Ainda é importante ressaltar que, devido a este módulo, existe uma ordenação implícita dos filhos de um determinado nó, estando ordenados da "esquerda para a direita". Esta ordenação se deve ao fato de que, o primeiro filho encontrado é associado ao nó "left" do pai, o segundo é associado ao nó "right" do primeiro, o terceiro é associado ao nó "right" do segundo e assim por diante.

Outro fato relevante é que quando mencionamos **critério**, queremos dizer **aplicação específica**. Na verdade, para cada operador podemos ter algum critério associado à árvore que desejamos construir. Deve-se notar também que os critérios mais importantes

da árvore deverá estar associado ao operador *cria todos os filhos segundo um critério*, já que é ai que inserimos nós.

5.2.3 Critérios para construção da Árvore de "Features"

Critério associado ao Operador "Cria todos os Filhos segundo um Critério"

O critério principal para construção da *Árvore de "features"*, como dito anteriormente, está contido no operador *Cria todos os filhos segundo um critério.* Este critério é composto de duas partes básicas:

1. reconhecer a "feature" no sólido;
2. marcá-la para procurar por outra do mesmo tipo.

Cada um dos módulos necessários para estabelecer os critérios é apresentado no seguinte algoritmo:

```
Operador(cria todos os filhos segundo um criterio)
{
   (inicializa o arquivo de ''features'')
   while (nao e a ultima)
   {
      (retire a proxima ''feature'')
      if (nao tem mais ''feature'')
         (indica que e a ultima)
      else
      {
         while (reconhece a ''feature'' no solido)
         {
            (insira a ''feature'' no no')
            (marque as faces da ''feature'')
         }
      }
   }
}
```

Cada um dos módulos presentes no algoritmo é descrito abaixo:

1. **Inicializa Arquivo de "features"**: toda vez que for iniciada a procura por todos os filhos, todo arquivo deve ser percorrido, assim, deve ser inicializado;
2. **Retira Próxima "feature" do Arquivo**: toda vez que uma "feature" for retirada do arquivo e for usada para o reconhecimento, a próxima "feature" do arquivo deve ser indicada para que possamos procurar por ela em após o reconhecimento da anterior. Se não existir mais "feature" no arquivo, deve retornar um valor **NULL**, para que o algoritmo seja encerrado;

3. **Reconhece**: este módulo indica o uso do *Reconhecedor Automático de "features"*, que devolve a "feature" reconhecida com suas características geométricas ou um valor que indica que esta "feature" não existe no sólido;

4. **Insere Nó na Árvore**: algoritmo para inserir um nó (ou seja, uma "feature") na *Árvore de "features"*;

5. **Marca Faces da "feature" encontrada**: É possível que tenhamos mais de uma "feature" do mesmo tipo em um sólido. Cada uma destas "features" deve ser tratada de modo independente, e colocada na árvore separadamente. O *Reconhecedor Automático de "features"* não inicia uma comparação entre uma "feature" e um sólido por nenhuma face preferencial, mas pode ficar "viciado", e a procura por sucessivas "features" de mesmo tipo sem alguma precaução pode levar o algoritmo a sempre ficar reconhecendo a mesma "feature", sem sair do lugar. Para evitar isso, deve-se existir um algoritmo que denominamos de "faces marcadas". Este algoritmo "marca" as faces do sólido que fazem parte da "feature", não permitindo assim que uma "feature" já reconhecida seja reconhecida novamente. Este algoritmo será aqui implementado utilizando-se o fato que o reconhecedor armazena as faces da "feature" durante o reconhecimento. Assim, as faces que já foram reconhecidas uma vez são automáticamente excluidas da busca.

Critério Associado ao Operador "Busca Primeiro filho"

Devemos lembrar que a busca pelo primeiro filho corresponde a descermos um nível na *Árvore de "features"*.Quando descemos de nível, a "feature" para a qual descemos deve ser retirada do sólido a fim de que possamos encontrar seus filhos.

```
Operador{busca primeiro filho}
{
   possui = (nao possui filho)
   (pega no' left)
   if ((no' left) <> (nao possui filho))
   {
      (desca para o no')
      (retire a feature do solido)
      possui = (possui no')
   }
   retorna (possui)
}
```

Como pode ser notado, a natureza das operações envolvidas no algoritmo é de grande simplicidade, chegando a ser, na maioria dos casos, uma mera manipulação de ponteiros. Apenas o módulo *Retire a feature do sólido* deverá ser estudado com mais cuidado, já que ali estarão envolvidas operações booleanas, o que pode representar já neste nível uma necessidade de interfacear o *Reconhecedor Automático de "features"* com o modelador de sólidos. Porém, estes aspectos serão abordados na fase de implementação.

Critério Associado ao Operador "Busca Primeiro Irmão"

Assim como no operador anterior o critério associado aqui tem relação com colocar ou retirar "features" do sólido conforme andamos pela árvore. Quando fazemos a busca por um irmão de um nó, devemos lembrar de operar sobre a "feature" que é deixada para trás de forma a recolocá-la no sólido. Isto porque a "feature" irmã deverá ter como um dos filhos a que foi deixada para trás. Além disto, a "feature" para a qual caminhamos deve ser retirada do sólido. Um possível algoritmo para este operador é apresentado abaixo.

```
Operador(busca primeiro irmao)
{
   possui = (nao possui no')
   (pega no' right)
   if ((no' right) <> (nao possui irmao))
   {
      (coloque a feature no solido)
      (caminhe para o irmao)
      (retire a feature do solido)
      possui = (possui irmao)
   }
   retorna (possui)
}
```

Critério Associado ao Operador "Busca o pai"

O critério é análogo ao usado nos dois operadores anteriores, sendo que aqui, quando subimos de nível, devemos recolocar a "feature" no sólido.

```
Operador(busca o pai)
{
   possui = (nao possui pai)
   (pega no' pai)
   if ((no' pai) <> (nao possui pai))
   {
      (coloque a feature no solido)
      (caminha para o pai)
      possui = (possui pai)
   }
   retorna (possui)
}
```

5.3 Estrutura de Avaliação de Relacionamentos

O sub-módulo *Estrutura de Avaliação de Relacionamentos* tem por objetivo criar uma estrutura de dados que mostre claramente o posicionamento de cada "feature" em relação às demais no sólido estudado (ou seja, se a "feature" está abaixo, acima ou no mesmo nível das outras "features" da *Árvore de "features"*). Este posicionamento nos permitirá determinar se estas "features" são independentes, aninhadas ou interferentes.

5.3.1 Estrutura e Nomenclatura

A *Estrutura de Avaliação de Relacionamentos* possuirá dois tipos de nós:

1. **"feature" de referência**: é a "feature" em relação a qual queremos saber a posição relativa das outras "features" contidas no sólido. Assim, sabendo as posições de cada "feature" em relação a todas as outras, poderemos determinar, por meio de regras, qual o relacionamento entre as "features". Teremos portanto "features" **abaixo**, **acima** e no **mesmo nível** que a "feature" de referência.

2. **"feature" de posição**: para cada "feature" de referência, deveremos ter todas as "features" do sólido a ela associadas. As "features" de posição serão estas "features" associadas. Elas serão as "features" do sólido que poderão estar **acima**, **abaixo** ou no **mesmo nível** da "feature" de referência.

Assim, os campos associados à "feature" de referência são:

- **"feature" de referência**;
- **mesmo nível**: indicará uma lista de "features" que foram encontradas no mesmo nível que a "feature" de referência na *Árvore de "features"*;
- **acima**: indicará uma lista de "features" que foram encontradas acima da "feature" de referência na *Árvore de "features"*;
- **abaixo**: indicará uma lista de "features" que foram encontradas abaixo da "feature" de referência na *Árvore de "features"*;
- **próxima**: indica a proxima "feature" de referência na estrutura;
- **anterior**: indica a "feature" de referência anterior na estrutura.

E os campos associados à "feature" de posição são:

- **"feature" de posição**;
- **próxima**: indica a proxima **"feature" de posição** numa lista de "features", que pode indicar **acima**, **abaixo** ou **mesmo nível**;
- **anterior**: indica a **"feature" de posição** anterior, numa lista de "features", que pode indicar **acima**, **abaixo** ou **mesmo nível**.

5.3.2 Operadores Básicos para Construção

A construção da *Estrutura de Avaliação de Relacionamentos* necessita da definição de alguns operadores básicos. Estes operadores são:

1. **Procura por todos os descendentes**: Saindo de uma **"feature" de referência** qualquer, percorre toda a *Árvore de "features"* para baixo daquele nível da **"feature" de referência**, incluindo todas as "features" encontradas (nos nós que percorre) na lista de "features" **abaixo**.
2. **Procura por todos os ascendentes**: "subindo" a *Árvore de "features"* por sucessivos ponteiros **pai**, inclui todas as "features"encontradas ne lista de "features" **acima**.
3. **Procura por todos os irmãos**: percorre o nível que a "feature" de referência se encontra na *Árvore de "features"*, incluindo todas as "features" encontradas na lista de "features" de **mesmo nível**.
4. **Caminha para cima**: "sobe" um nível na *Árvore de "features"*.
5. **Caminha para o lado**: caminha para o irmão **right**.
6. **Caminha para baixo**: caminha para o irmão **left**.

Tendo em vista estes operadores, o algoritmo para a construção da *Estrutura de Avaliação de Relacionamentos* fica:

```
Algoritmo(Estrutura de Avaliacao de Relacionamentos)
{
   (pegue a primeira feature)
   while (nao acabou a arvore de features)
   {
      if (feature encontrada nao esta na estrutura)
         (coloque em no da lista de feat. de referencia)
      (procura por todos os descendentes)
      (procura por todos os ascendentes)
      (procura por todos no mesmo nivel)
      (caminha para baixo)
      if (impossivel caminhar para baixo)
      {
         (caminha para o lado)
         if (impossivel caminhar para o lado)
         {
            (admita que seja possivel caminhar para cima)
            while ((impossivel caminhar para o lado) e
                  (e possivel caminhar para cima  )   )
            {
               (caminhe para cima)
               if (impossivel caminhar para cima)
                  (arvore de features acabou)
               else
                  (caminha para o lado)
```

```
            }
         }
      }
   }
}
```

O algoritmo percorre toda a *Árvore de "features"*, começando pela primeira "feature" que foi encontrada, primeiro estudando o relacionamento de cada "feature" da árvore com a primeira retirada do sólido, depois "descendo" para todas que estão abaixo dela. Quando já não existe mais nenhuma "feature" abaixo dela, que já não tenha sido percorrida e seu relacionamento estudado, passa então para a irmã à direita da primeira "feature" e repete o processo, até que todas as "features" do sólido tenham tido seu relacionamento estudado.

Note que a árvore é "atravessada" da esquerda para a direita, primeiro percorrendo todas as "features" possíveis à esquerda, passando então para a primeira irmã possível, e subindo de acordo com o fim dos sucessivos níveis de "features". Devemos ressaltar que o que entendemos por atravessar a árvore neste algoritmo é no sentido de passar por todas as "features" da árvore, de forma a estabelecer o relacionamento para todas as outras. Os algoritmos específicos para encontrar as "features" de relacionamento serão apresentados na próxima seção.

Operador Procura por todos os Descendentes

Este operador percorre toda a árvore abaixo de um determinado nó (que será a "feature" de referência), retornando as "features" pelas quais vai passando. Estas "features" são inseridas na *Estrutura de Avaliação de Relacionamentos*, na posição de **abaixo**, referente à "feature" de referência. Um possível algoritmo para este operador é apresentado abaixo. Sua entrada é o nó de referência, de onde parte e para onde deverá voltar.

```
Operador(procura por todos os descendentes)
{
   (caminha para baixo)
   while (feature diferente da feature de referencia)
   {
      if (feature nao esta na lista de abaixo)
         (coloque feature na lista de abaixo)
      (caminhe para baixo)
      if (impossivel caminhar para baixo)
      {
         (caminhe para o lado)
         if (impossivel caminhar para o lado)
         {
            (admita que e possivel caminhar para cima)
            while ((impossivel caminhar para o lado) e
                   (possivel caminhar para cima)      )
            {
               (caminha para cima)
               if (e impossivel caminhar para cima)
                  (chegou ao no de referencia)
```

```
                else
                   (caminha para o lado)
             }
          }
       }
    }
}
```

Note que a partir do operador **caminha para baixo**, o algoritmo é o mesmo utilizado para o algoritmo de construção da *Estrutura de Avaliação de Relacionamentos*, com a diferença que o ponto de parada é diferente (neste algoritmo, o ponto de parada é a "feature" de referência). Com isto, posteriormente o algoritmo dos dois poderá ser basicamente o mesmo, se executadas as devidas adaptações para que sirva em ambos os casos. Esta adaptação será vista na fase de implementação.

Operador Busca Por Todos os Ascendentes

Este operador percorre toda a *Árvore de "features"*, partindo da "feature" de referência, para "cima", ou seja, percorre todas as "features" que foram retiradas após a sua retirada. Para isto, basta que percorra todos os **pais** acima dela, sucessivamente. Um possível algoritmo para este operador é mostrado abaixo:

```
Operador(busca por todos os ascendentes)
{
   (caminha para cima)
   while (possivel caminhar para cima)
   {
      if (nao esta na lista de acima)
         (coloque na lista de acima)
      (caminha para cima)
   }
}
```

Operador Procura por Todos os Irmãos

Este operador tem por função percorrer o nível em que se encontra a "feature" de referência, a fim de colocar na lista de **mesmo nível** as "features" encontradas. Um algoritmo para este operador é apresentado abaixo:

```
Operador(procura por todos os irmaos)
{
   (caminha para cima)
   (caminha para baixo)
   while (nao acabarem as features de mesmo nivel)
   {
      if (nao e a feature de referencia)
      {
```

```
            if (nao esta na lista de mesmo nivel)
               (coloque na lista de mesmo nivel)
         }
         (caminha para o lado)
         if (nao e possivel caminhar para o lado)
            (acabaram as features de mesmo nivel)
      }
   }
```

Note que a operação de subir para o nível imediatamente superior, seguida da operação de descer é um dos modos possveis de garantir que todas as "features" de um mesmo nível sejam colocadas na lista de "features" de **mesmo nível**. É claro que passaremos pela "feature" de referência, mas um teste simples fará com que ela não seja inclusa na lista[3] [4].

5.4 Relatórios de Saída

Os relatórios[5] de saída tem por função mostrar como cada uma das estruturas foi cnstruída ao longo da operação do sistema. Estes relatórios tiveram uma função extremamente importante na implementação do sistema, já que seu comportamento podia ser completamente analisado. Para o usuário podem ter a função de verificar passo a passo o desenvolvimento do escalonamento, podendo mudar algum parâmetro de interesse baseado nestas observações.

Nesta fase são gerados relatórios para a árvore de "features" e para a lista bi-ligada, uma estrutura intermediária que é utilizada para verificar a existência de uma determinada "feature" na árvore, para que não seja inserida como uma "feature" diferente na árvore.

5.5 Implementação

Para que se realizasse a implementação do Sistema de *Escalonamento Automático de "features"*, em primeiro lugar houve a definição de como o Sistema seria modularizado. A modularização permite uma clara separação das várias funções envolvidas, além de facilitar a debugação e testes, já que cada módulo pode ser debugado/testado independentemente, facilitando a descoberta de erros de implementação local de cada módulo, bem como reduzindo (praticamente eliminando) erros de integração do Sistema.

[3] As operações de "caminhar" envolvem apenas uma tomada de sentido simples para percorrer a árvore, sempre trabalhando apenas com um dos três ponteiros definidos para os nós da Árvore de "features", não merecendo atenção especial

[4] Todos os algoritmos aqui apresentados ficaram sujeitos a alterações durante a fase de implementação.

[5] Inicialmente estes relatórios teriam função de guardar os relacionamentos obtidos para que fossem utilizados pelos módulos seguintes do sistema. Porém, com a especificação dos módulos seguintes, esta função se mostrou pouco eficiente.

Assim, podemos dizer que o Sistema foi dividido nos seguintes módulos:

1. "Arqfeat.c", que contém as funções responsáveis por gerenciar o *Arquivo de "features"*;
2. "Arvfeat.c", que contém as funções responsáveis por construir a *Árvore de "features"*;
3. "Organiza.c", que contém as funções responsáveis por construir a *Estrutura de Avaliação de Relacionamentos*;

A cada um destes arquivos ".c" associamos um arquivo "header" (".h"), que contém os protótipos de cada função desenvolvida no arquivo ".c", bem como a declaração das variáveis externas, constantes, etc.., necessárias para o desenvolvimento do módulo.

Em linhas gerais, o sistema foi idealizado para funcionar do seguinte modo: cada uma das "features" do *Arquivo de "features"* será modelada segundo funções de modelagem. Existirá um arquivo denominado "arqfeat.nos" que conterá os nomes dos arquivos aonde se encontram os nomes dos arquivos que contém cada uma das "features" que o sistema pode reconhecer modelada. Para cada uma das "features" do *Arquivo de "features"* o módulo *Árvore de "features"* verificará se ela existe ou não no sólido, e procederá conforme o estabelecido nas seções que tratam da organização do projeto. Para cada um dos nós da árvore existirá um arquivo associado que armazenará, por meio de funções de modelagem, o sólido modelado, após a extração da respectiva "feature" que é associada ao nó. Além disto, associa-se o nome da "feature" e outros fatores que serão descritos posteriormente. A seguir é montada a Estrutura de Avaliação de Relacionamentos, com base na árvore e em uma lista de "features" que é criada quando se constrói a Árvore de "features". Por último, é criado o relatório de saída.

5.5.1 Módulo de Árvore de "Features"

Quando foi discutida a organização do projeto, havia ficado estabelecido que quando construíssemos a árvore, no momento em que, por exemplo, subíssemos um nível, a "feature" do nó de onde partissemos deveria ser recolocada no sólido a fim de que fosse reconstruido o sólido como ele era antes de se extrair a "feature". Outras operações para percorrer a árvore também exigiam este tipo de manipulação, de cálculo de inversas, retirar e extrair "features", etc.... No entanto, a nível de implementação foi realizada uma otimização de modo que o algoritmo ficasse mais rápido e eficiente.

Esta otimização baseou-se no seguinte procedimento: toda vez que uma "feature" for reconhecida num sólido, ela é adicionada a um nó correspondente da árvore. É extraida do sólido automáticamente, e o sólido resultante é gravado num arquivo, que é associado ao nó da árvore. Como o extrator é baseado em funções de modelagem, cada um destes sólidos será gravado com base em funções de modelagem.

Note que para cada nó da *Árvore de "features"* temos um determinado sólido, que é "filho" do sólido original, ou seja, que é o sólido original menos as "features" que estão associadas aos nós que estão acima do nó estudado. Este sólido representa completamente o estado atual da árvore e suprime-se uma série de operações para reconstruir o sólido. Assim, só é necessária uma operação, para cada nó, de extração de "feature", o que economiza muito tempo computacional. Cada sólido criado a partir da extração de uma "feature" (que é associada a um nó) será associado ao nó da "feature" extraída em questão.

Estando explicadas estas alterações vamos agora passar a estudar as estruturas envolvidas na construção da *Árvore de "features"*, o que esclarecerá qualquer outra dúvida.

Estruturas Envolvidas na Implementação da Árvore de "features"

São duas as estruturas que deverão ser analisadas nesta seção: **NoArvore** e **lista**. A estrutura **NoArvore** representa o nó da *Árvore de "features"* e a estrutura **lista** representa um nó de uma lista bi-ligada que contém uma determinada "feature" encontrada no sólido (lembre-se que uma mesma "feature" pode ser associada a diversos nós de uma mesma *Árvore de "features"*).

A estrutura **NoArvore** é implementada da seguinte maneira:

```
typedef struct NoArvore NO ;

struct NoArvore
{
   NO    *left,
         *right,
         *pai ;
   char  NomeDaFeature[12],
         Feature[12],
         Solido[12],
   int   CuboIgual,
         VerificaHeuristica,
         Ferramenta,
         NumeroFacesFeature ;
   face  *FacesFeature ;
   COMP  *arestas ;
}
```

Cada um dos campos desta estrutura é discutido abaixo:

- **NO *left**: este campo é um ponteiro que aponta para o filho do nó a que este ponteiro pertence;
- **NO *right**: este campo é um ponteiro que aponta para o irmão do nó a que este ponteiro pertence;
- **NO *pai**: este campo é um ponteiro que aponta para o pai do nó a que este ponteiro pertence;

- **char NomeDaFeature[12]**: este campo é uma "string" de 12 caracteres que guarda o tipo da "feature" que está associada ao nó que a string pertence. Se um tipo de "feature" é encontrada no sólido, este campo é preenchido com seu nome;

- **char Feature[12]**: este campo guarda o nome do arquivo correspondente à "feature" encontrada no sólido. Devemos lembrar que uma mesma "feature" pode ser associada a diversos nós da árvore. Assim, vários nós poderão ter um mesmo conteúdo neste campo;

- **char Solido[12]**: Quando uma "feature" é retirada de um sólido, um novo sólido é criado. Este sólido deve ser armazenado em um arquivo que receberá um determinado nome. O nome deste arquivo é armazenado neste campo. O sólido criado aqui será utilizado quando voltarmos a este nó e formos procurar por seus filhos. Devemos lembrar que cada nó da árvore corresponde a um diferente estado do sólido, portanto nunca haverão campos deste tipo com o mesmo conteúdo;

- **int CuboIgual**: toda vez que um nó é criado, este campo recebe um valor 0. Quando o sólido que corresponde ao nó é igual a um cubo, ou seja, atingimos um ponto de parada, este campo recebe o número de comparações que o reconhecedor realizou para identificar o cubo;

- **VerificaHeuristica**: no próximo capítulo, iremos demonstrar a aplicação de uma heurística para racionalizar a construção da árvore. Deste modo, este campo se fará necessário para se identificar a quais nós foi aplicada esta heuristica. Este campo serve para que façamos esta identificação;

- **Ferramenta**: com base nos valores encontrados no campo "arestas", pode-se calcular uma broca para desbaste de "features" de superfície retangular. Este campo serve para guardar esta ferramenta. No futuro, quando integrado com um sistema TG, este campo deverá ser modificado;

- **NúmeroFacesFeature e *facesFeature**: estes dois campos juntos tem a função de guardar o número de faces que a "feature" do nó tem e quais são estas faces no sólido modelado. Assim, não é necessário se utilizar o reconhecedor para comparar "features", o que economiza um grande tempo computacional;

- **arestas**: guarda o valor de arestas no 3 eixos da "feature", de tal forma que possamos calcular as ferramentas, número de passes, etc.... Deve-se ressaltar que este trabalho foi limitado a "features" retangulares.

Ainda há mais uma estrutura a ser analisada (apesar desta estrutura ter nada a ver com a árvore em si, ela é montada durante a construção da árvore, daí ela ser analisada aqui). Esta estrutura tem por função armazenar cada uma das "features" encontradas no sólido apenas uma vez (mesmo que apareça várias vezes). Este armazenamento facilitará a procura por "features" iguais associdas a diferentes nós da *Árvore de "features"*. Esta

lista evitará um grande trabalho computacional que é o de procurar por "features" iguais na *Árvore de "features"*.

A estrutura é a seguinte:

```
typedef struct lista INV ;

struct lista
{
   INV   *prox,
         *ant ;
   char  Feature[12] ;
   int   NumeroFacesFeature ;
   face  *FacesFeature ;
}
```

- **INV *prox**: aponta para o próximo nó da lista bi-ligada;
- **INV *ant**: aponta para o nó anterior da lista bi-ligada;
- **char Feature[12]**: Análogo ao campo **Feature[12]** na estrutura anterior, toda vez que uma "feature" é encontrada no sólido é checado se ela já existe na árvore em outro nó. Se existir, então associa ao nó da árvore este arquivo já existente. Este campo, na verdade, existe apenas para identificar a "feature" em questão, já que a comparação propriamente dita fica a cargo dos dois campos descritos abaixo;
- **NumeroFacesFeature e facesfeature**: estes campos é onde se faz efetivamente a comparação. Compara-se o conteúdo do resultado da comparação do reconhecedor com o conteúdo de cada um destes campos.

5.5.2 Módulo de Estrutura de Avaliação de Relacionamentos

Este módulo foi implementado exatamente como foi planejado na organização do projeto.

Estruturas Principais

A *Estrutura de Avaliação de Relacionamentos* é construida a partir de dois tipos de estruturas: **relacoes** e **nivel**. A estrutura **relacoes** guarda uma "feature" da qual se quer saber qual está abaixo, acima ou no mesmo nível em relação a ela mesma. A estrutura **nivel** guarda uma "feature" que está acima, abaixo ou no mesmo nível de uma "feature" contida em um nó do tipo **relacoes**.

A estrutura **relacoes** é apresentada abaixo:

```
typedef struct relacoes REL ;
typedef struct nivel  LEVEL ;

struct relacoes
{
```

```
  REL    *ant,
         *prox ;
  LEVEL  *mesmo,
         *acima,
         *abaixo ;
  char   feature[12] ;
}
```

- **REL *ant**: aponta para um nó que contém uma "feature" analisada anteriormente à "feature" contida neste nó;
- **REL *prox**: aponta para um nó que contém uma "feature" analisada posteriormente à "feature" contida neste nó;
- **LEVEL *mesmo**: aponta para o primeiro nó de uma lista que contém todas as "features" no mesmo nível da "feature" contida neste nó. Note que este nó é do tipo **struct nivel**, bem como o serão os dois próximos campos. O que diferencia cada nó é o ponteiro do tipo **struct nivel** que aponta para o primeiro nó da lista;
- **LEVEL *acima**: aponta para o primeiro nó de uma lista que contém todas as "features" acima da "feature" contida neste nó;
- **LEVEL *abaixo**: aponta para o primeiro nó de uma lista que contém todas as "features" abaixo da "feature" contida neste nó;
- **char feature[12]**: contém o nome do arquivo da inversa da "feature" correspondente a este nó.

A estrutura de **nivel** é apresentada abaixo:

```
struct nivel
{
   LEVEL  *proxima,
          *anterior ;
   REL    *ant ;
   char   feature[12] ;
}
```

- **LEVEL *anterior**: aponta para "feature" anterior na lista de "features" de um determinado nível;
- **LEVEL *proxima**: aponta para a próxima "feature" na lista de "features" de um determinado nível;
- **REL *ant**: o primeiro nó da lista de "features" de um determinado nível deverá apontar para o nó que tem o nó do tipo **struct relacoes** da "feature" que queremos analisar;
- **char feature[12]**: contém o nome do arquivo que contém a inversa da "feature" da "feature relacionada ao nó;

Capítulo 6

Seleção do Processo de Usinagem

Este capítulo destina-se a demonstrar como o processo de usinagem é gerado. Descrevem-se as heurísticas envolvidas, desde a escolha de ferramentas e máquinas até os parâmetros do processo como velocidade de corte, profundidade de corte, etc.... Primeiro é dada uma pequena introdução ao conceito de heurísticas. A seguir, é apresentada a primeira heurística, a de volume de "features", que leva a um novo algoritmo para construção da árvore de "features". Introduz-se então as heurísticas para sequência de usinagem e processo de usinagem.

6.1 O Conceito de Heurística

Heurísticas são critérios, métodos ou princípios para decidir qual entre várias alternativas de ação será a mais efetiva para se chegar a um determinado resultado. Elas representam um compromisso entre dois requisitos: a necessidade de tornar tal critério simples e ao mesmo tempo o desejo de vê-las optar corretamente entre boas e más opções [Pearl 84].

Como exemplo pode-se citar um mestre de xadrez que tem como opção vários movimentos. Ele escolherá um movimento que lhe dê uma posição que pareça mais forte do que as outras opções. Este critério de determinar o movimento mais forte é mais simples do que determinar o movimento que lhe daria um "checkmate". O fato de nem sempre os grandes mestres ganharem mostra que suas heurísticas não podem garantir a seleção do movimento mais efetivo.

É a natureza de boas heurísticas que indiquem de modo simples qual das opções devem ser escolhidas e que não garantam necessariamente o caminho mais efetivo, mas que o façam em uma frequência suficiente.

6.2 Novo Algoritmo para a Árvore de "Features"

O algoritmo para construção da árvore de "features", como foi proposto e implementado, gera todas as sequências de usinagem possíveis para um determinado sólido, mas levaria a um resultado explosivo. Para um sólido composto de n "features" independentes, é gerado o seguinte número de nós:

$$N_{nos} = n!(\frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{(n-1)!})$$

Assim, para 10 "features" independentes, são 9.900.000 nós gerados!

Deste modo, verificou-se a necessidade de se obter uma alternativa mais racional. Esta alternativa baseou-se na seguinte heurística:

Heurística 4 *Em um determinado nível da árvore de "features", escolher para criar todos os filhos a "feature" de maior volume.*

Esta heurística permite que uma menor troca de ferramentas (já que se houverem conjuntos de "features" interferentes ela escolherá a "feature" de maior volume e que muitas vezes deverá ter uma única ferramenta associada) e a redução drástica do número de nós da árvore. A aplicação desta heurística leva ao seguinte algoritmo para construção da árvore de "features":

```
Algoritmo(Novo Algoritmo Para a Arvore de ''Features'')
{
   verifica heuristica = ok
   (va para inicio da arvore -- head)
   (crie todos os filhos do primeiro no')
   while (solido nao e cubo) e (heuristica == heuristica ok)
   {
      auxilio = auxilio arvore = (filho do no' head)
      if (verifica heuristica == ok)
      {
         auxilio = auxilio arvore
         (crie todos os filhos do no auxilio arvore)
         auxilio = (filho de auxilio)
      }
      verifica heuristica = heuristica maior volume (no)
   }
}
```

Este algoritmo contrói apenas um dos ramos da árvore, e fornece o seguinte número de nós (para um sólido de n "features" independentes):

$$N_{nos} = n + (n-1) + (n-2) + (n-3) + \cdots + 1$$

Assim, para 10 "features" independentes, ocorre a criação de 55 nós!

Figura 6.1: Desempenho do sub-módulo *Árvore de "Features"*.

Deve-se enfatizar que todos os outros sub-módulos e algoritmos do módulo de escalonamento continuam a atuar sem modificação. O desempenho dos dois algoritmos para construção da árvore pode ser observado pela figura 6.1. A mudança da construção da árvore leva a que os relacionamentos sejam indicados segundo as seguintes heurísticas:

Heurística 5 *Para que se determinem os relacionamentos em uma árvore de "features" basta apenas que um de seus ramos sejam totalmente desenvolvildos.*

Heurística 6 *Se uma "feature" α aparece no mesmo nível e abaixo (ou no mesmo nível e acima) de uma "feature" β, então α e β são independentes.*

Heurística 7 *Se uma "feature" α aparece apenas abaixo de uma "feature" β, então α é.aninhada em relação a β.*

Heurística 8 *Se uma "feature" α aparece apenas no mesmo nível de uma "feature" β, então α e β são interferentes.*

6.3 Determinação da Sequência de Usinagem

A fim de determinarmos a sequência de usinagem, deve-se observar as seguintes heurísticas:

Heurística 9 *Se uma "feature" α for aninhada em relação a uma "feature" β, então α deve ser usinada antes de β.*

Heurística 10 *"Features" independentes podem ser usinadas em qualquer ordem relativa.*

A heurística 9 deve ser utilizada já que geralmente, quando se extraem duas "features", uma aninhada em relação à outra, pode ocorrer que a aninhada tenha que ser usinada antes para que haja possibilidade da ferramenta entrar para usinar a "feature" que a aninha. Este caso pode ser observado pela figura 6.2.

Assim, nota-se que a sequência de "features" obtidas do escalonador é invertida nos pontos onde ocorrem "features" aninhadas, com relação a usinagem. Deste modo, cria-se uma nova sequência, denominada *Sequência de Usinagem*, que é simplesmente a sequência obtida pela inversão da sequência do escalonamento. A sequência de usinagem já poderia ser utilizada para gerarmos o processo, já que respeita as heurísticas 9 e 10. Porém, observando a seguinte heurística:

Heurística 11 *Para que o processo de usinagem se torne eficiente, é melhor que se reduza o número de "set-ups" (troca) de ferramentas. Assim, se duas "features" apresentam o mesmo tipo de ferramenta, elas devem ser usinadas em sequência.*

Deve-se então agrupar as "features" pelas suas ferramentas[1], de forma a reduzir o número de trocas de ferramentas. Como a heurística 10 é sempre respeitada, deve-se tomar cuidado para que a heurística 9 sempre prevaleça. No caso de alteração, deve-se "carregar" a "feature" aninhada junto com a "feature" que foi mudada de posição, de tal forma que se continue usinando primeiro a "feature" aninhada.

6.4 Processo de Usinagem e Folha de Processo

Neste trabalho optou-se por restringir os sólidos a apenas um plano de usinagem esta restrição se deve ao fato de não serem disponíveis os módulos de geração de caminho de corte e de fixadores. Outra restrição é que só se trabalha com "features" de superfície de usinagem retangulares, já que o modelador de sólidos didático não trabalha com arestas curvas. Levando em conta estas restrições, introduzimos as seguintes heurísticas de manufatura:

[1] Esta heurística, que está ajudando a determinar a sequência de usinagem é válida para as ferramentas de desbaste. No caso de ferramentas de acabamento, utiliza-se a mesma sequência aqui obtida, já que o tempo de acabamento é bem maior. As ferramentas para desbaste são calculadas assim que se extrai as dimensões da "feature", e o método de cálculo será apresentado na próxima seção.

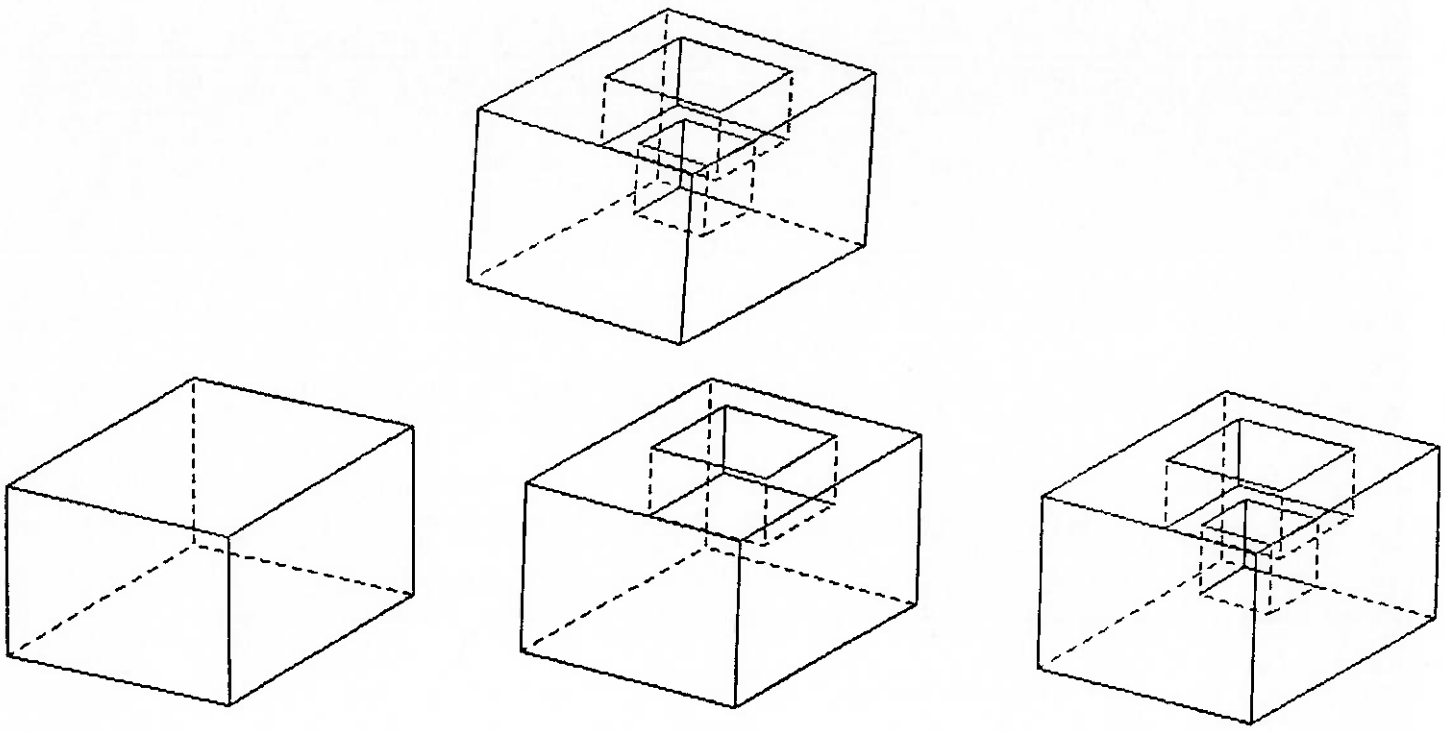


Figura 6.2: "Features" aninhadas: como usiná-las.

Heurística 12 *Uma "feature" deve sempre ser usinada, no desbaste, com a maior ferramenta possível*[2].

Heurística 13 *A usinagem de uma "feature" de formato retangular se dá no desbaste: com uma broca do maior diâmetro possível de se utilizar e, se necessário*[3]*, com a utilização de uma fresa de mesmo diâmetro da broca; no acabamento: por uma fresa que dê um raio de arredondamento tão preciso quanto se deseje*[4].

Heurística 14 *A profundidade de corte radial de uma fresa deve ser, no máximo, igual a metade do diâmetro da fresa.*

Heurística 15 *A profundidade de corte axial de uma fresa pode ser, no máximo, igual a duas vezes o diâmetro da fresa.*

Assim, o programa deve trabalhar percorrendo a sequência de usinagem, inserindo na folha de processo: o número da operação; o tipo da "feature" que está usinando; o

[2]Neste caso, esta ferramenta será sempre uma broca.

[3]Comercialmente, é difícil se encontrar brocas de diâmetro maior do que 30 mm.

[4]Neste trabalho não apresentamos opção para raio de arredondamento. Aqui o raio será sempre de 2.5mm – um raio bastante preciso – ou seja, uma fresa de diâmetro 5mm.

arquivo onde se encontra aquela "feature" específica; a ferramenta com a qual se está executando a operação (que pode ser broca, fresa de desbaste e fresa de acabamento); a máquina onde se realiza a operação; se haverá uso de liquido de corte ou não; a velocidade de corte calculada[5]; a rotação da máquina (tabelado, em RPM); a velocidade de avanço; a profundidade de corte radial; a profundidade de corte axial.

Todos os parâmetros tabelados foram escolhidos para o material do sólido como sendo o aço e o material das ferramentas como sendo aço rápido. Como a evolução do projeto, pode-se acrescentar uma base de dados eficiente associada a um sistema de tecnologia de grupo, para que se possa ter uma escolha mais eficiente dos parâmetros.

6.5 Implementação

6.5.1 Sequência de Usinagem

A sequência de usinagem é uma lista ligada que tem a seguinte estrutura:

```
typedef struct sequencia_de_processo SEQ ;

struct sequencia_de_processo
{
   NO  *NoDaArvore ;
   SEQ *prox ,
       *ant ;
}
```

O campo **NoDaArvore** é um ponteiro que aponta para o nó da árvore que corresponde à "feature" desejada. Deste modo evita-se guardar dados repetidamente. A estrutura é montada de forma simples. Percorre-se a sequência estabelecida pelo escalonamento até se encontrar a última "feature" (que tem associada a si o sólido que é um cubo), e vai-se associando os endereços dos ponteiros correspondentes a nós da lista que vão sendo criados conforme se "sobe". Assim, ao fim do processo, existe uma lista que associa os endereços dos ponteiros de forma inversa ao que estava montada a estrutura. Este procedimento permite que se tenha uma sequência de usinagem que já é possível de ser implementada.

Deve-se agora agrupar as "features" que tem ferramentas iguais para reduzir o número de "set-ups". Este agrupamento é feito da seguinte maneira: escolhe-se um nó da sequência e a percorre, procurando "features" com mesma ferramenta. Se encontrar uma "feature" com a mesma ferramenta, associa o endereço do nó desta "feature" ao nó onde estava a o endereço do nó da "feature" que se tomou por referência e deslocam-se todos os nós ao longo da estrutura. A seguir, procura-se por "features" aninhadas a esta "feature"

[5] A velocidade de corte para uma peça de aço e ferramenta de aço rápido é tabelada e vale de 30 a 45 m/min para desbaste e a metade deste valor adotado para desbaste no acabamento

e realiza-se o mesmo processo, trazendo-as para "cima" da estrutura e procurando novas "features" aninhadas. O algoritmo é apresentado abaixo:

```
Algoritmo(Sequencia de Usinagem)
{
   (cria sequencia inicial invertida)
   (saida do loop = 0)
   (auxilio recebe o inicio da sequencia)
   (auxilio1 recebe seguinte ao auxilio)
   while (saida do loop == 0)
   {
      while ((auxilio1 nao e NULL) e
            (ferramenta de auxilio <> auxilio1))
      {
         (auxilio1 recebe proximo de auxilio1)
      }
      if (auxilio1 e NULL)
      {
         (auxilio recebe proximo de auxilio)
         (auxilio1 recebe proximo do auxilio)
         if (auxilio1 for igual a NULL)
         {
            (saida do loop recebe 1)
         }
      }
      else
      {
         if (precisa trocar auxilio e auxilio1)
         {
            (movimenta auxilio1 acima de auxilio)
            (move aninhadas se necessario)
            (va para o inicio da sequencia)
         }
         else
            (auxilio1 recebe proximo de auxilio1)
      }
   }
}
```

6.5.2 Processo de Usinagem e Folha de Processo

O processo de usinagem e a folha de processo são gerados simultaneamente. Os cálculos são realizados como descritos na seção 6.4. O processo de usinagem é mostrado a cada passagem de ferramenta. Quando do cálculo das passagens das fresas para desbaste, primeiro fazemos todo o desbaste radial a uma determinada profundidade de corte, descendo em seguida e assim sucessivamente.

Capítulo 7

Resultados

7.1 Implementação e Integração

Este sistema foi implementado em linguagem C. Tomou-se o cuidado de não se utilizar funções que não pertencessem ao C "standard". Os "include files" que traziam protótipos de funções para DOS foram excluidos de forma que o sistema tanto rodasse em estações de trabalho como em PC's. Atualmente existem versões rodando tanto para estações padrão RISC 6000 IBM como para micros padrão IBM-PC. Sendo o sistema base para um sistema de grande porte (até o momento, este sistema apresenta 15 módulos e aproximadamente 6000 linhas de código), é conveniente que utilize como plataforma as estações de trabalho, que apresentam desempenho muito superior.

A modularização prévia do sistema, com o desenvolvimento dos PFS's, garantiu um perfeito conhecimento de todas as interfaces existentes no sistema, evitando erros nesta fase, economizando no tempo de implementação. Um aspecto que pode ser melhorado quanto à implementação é acabar com a gravação e leitura de arquivos, que são frequentes no módulo de escalonamento, passando todos os dados para memória dinâmica.

7.2 Exemplo de Aplicação

Este exemplo é baseado no sólido da figura 7.1. A modelagem deste sólido, bem como a das "features" que compõem o banco de dados de "features" é apresentada no apêndice C.

O primeiro passo do sistema é construir a árvore de "features", realizando o escalonamento de "features", encontrando-se um primitivo, que neste caso será um cubo, e que é apresentado na figura 7.2.

O arquivo de "features" disponíveis para reconhecimento é apresentado na figura 7.3. Este arquivo contém as "features" na seguinte ordem: 1) quina; 2) degrau interno;

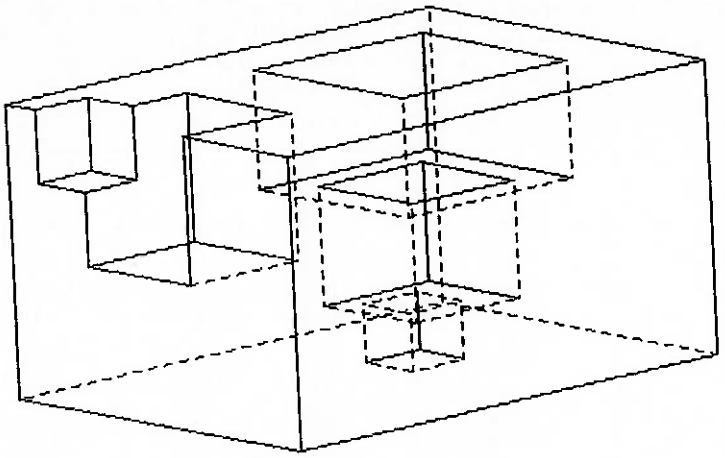


Figura 7.1: Sólido exemplo

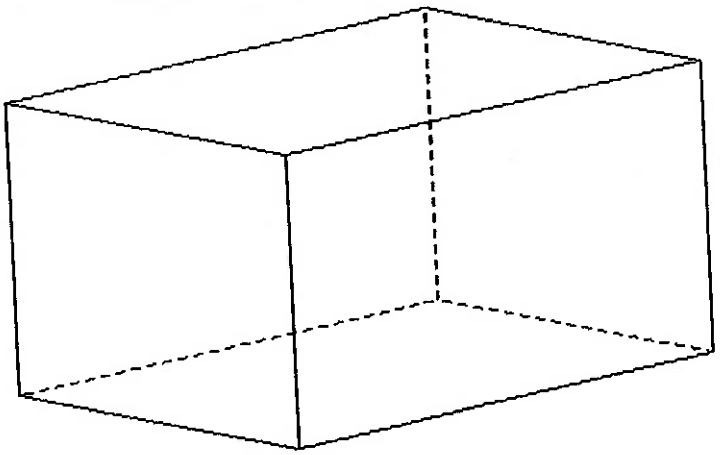


Figura 7.2: Cubo primitivo

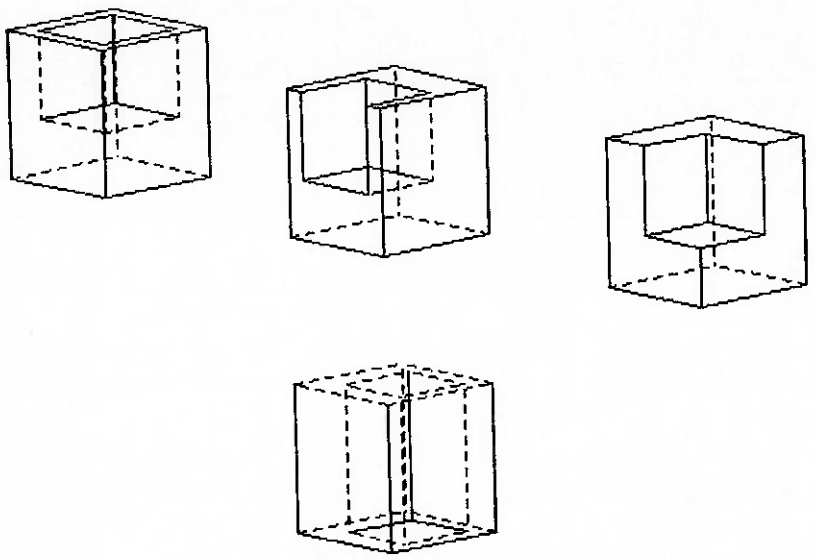


Figura 7.3: Arquivo de "features"

3) furo não passante; 4) furo passante. Assim, a primeira "feature" que o sistema tenta reconhecer é uma quina. Este reconhecimento é positivo, a "feature" é reconhecida, extraida e suas faces são marcadas. O reconhecimento e extração desta quina neste nível da árvore são representados pela figura 7.4. O sólido superior indica o nó "pai" (de onde realizamos o reconhecimento), a quina representa a "feature" que se tenta reconhecer, e o sólido inferior representa o novo sólido criado após a extração da quina, e que gera um novo nó da árvore.

A seguir, o sistema procura por outra "feature" do tipo quina no sólido. Como se pode notar, no sólido existe apenas uma quina que já foi reconhecida e marcada. Assim, a figura 7.5 representa o insucesso de se encontrar uma nova "feature" do tipo quina.

A próxima "feature" que o sistema tenta reconhecer neste nível é o degrau interno. Mas o degrau é aninhado em relação à quina e portanto, não será encontrado ainda neste nível. Esta tentativa de reconhecimento é representado pela figura 7.6. Neste nível ainda repetimos o processo para para o furo passante e o furo nã passante. Pelas figuras 7.7 e 7.8 pode-se ver a construção final deste nível da árvore. Este nível da árvore é representado pela figura 7.8.

A "feature" de maior volume neste nível é a quina, portanto, o nó da árvore que será escolhido para continuar o processo é o gerado na figura 7.4. O próximo nível da árvore é então representado pela figura 7.9. Seguindo o mesmo processo, os nós da árvore serão criados segundo as figuras 7.10, 7.11 e 7.12. A árvore de "features" é representada pela figura 7.13.

Construída a árvore, o próximo passo é determinar a sequência para gerar o processo de usinagem. A figura 7.14 representa os vários passos para se determinar a sequência de

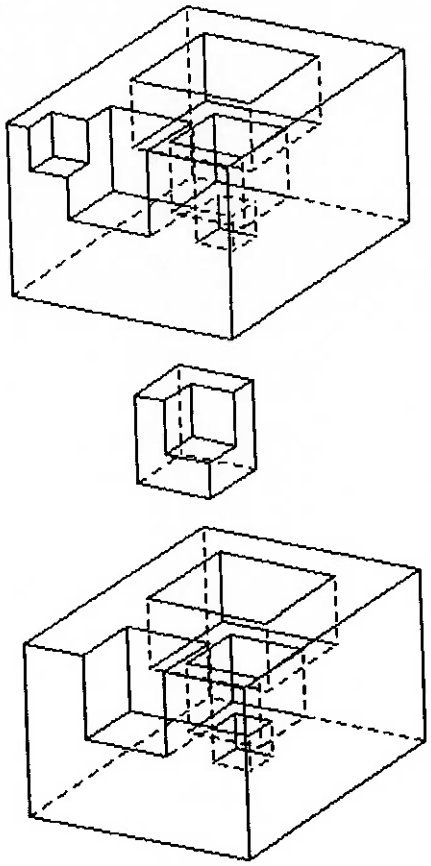


Figura 7.4: Reconhecimento e Extração da Quina no Primeiro Nível

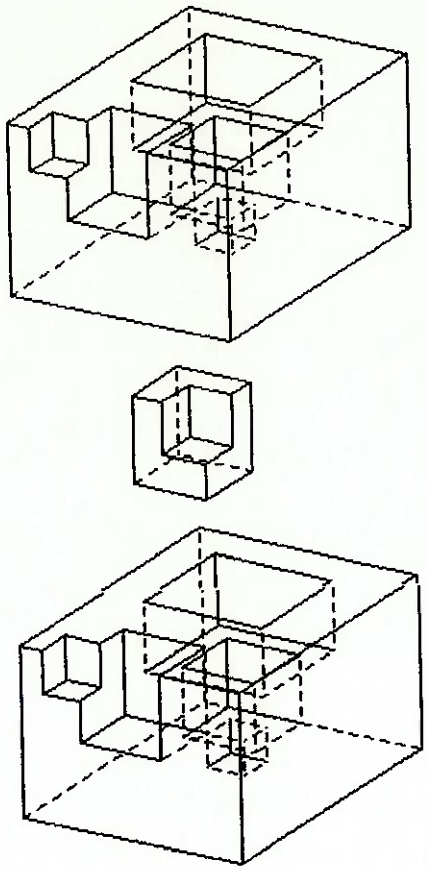


Figura 7.5: Tentativa de Reconhecimento da Quina na Segunda Tentativa

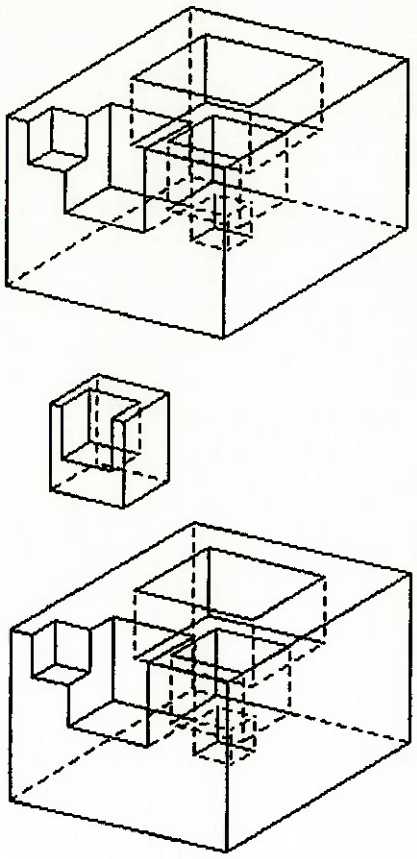


Figura 7.6: Tentativa de Reconhecimento do Degrau Interno no Primeiro Nível

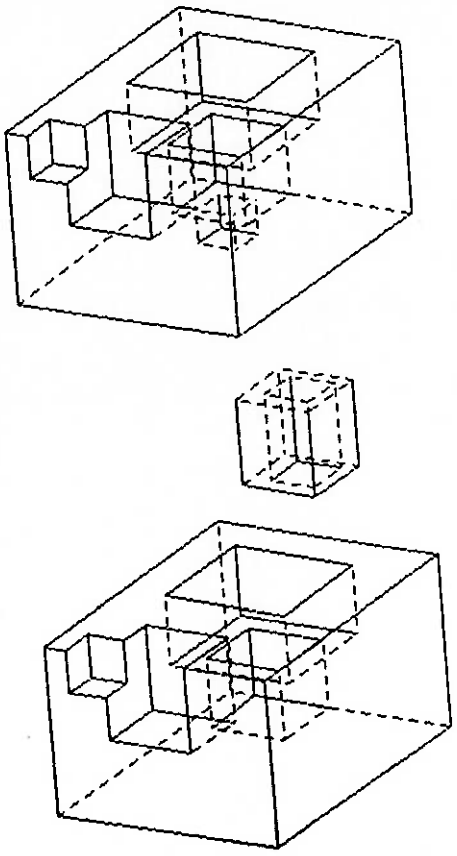


Figura 7.7: Reconhecimento do Furo Passante no Primeiro Nível

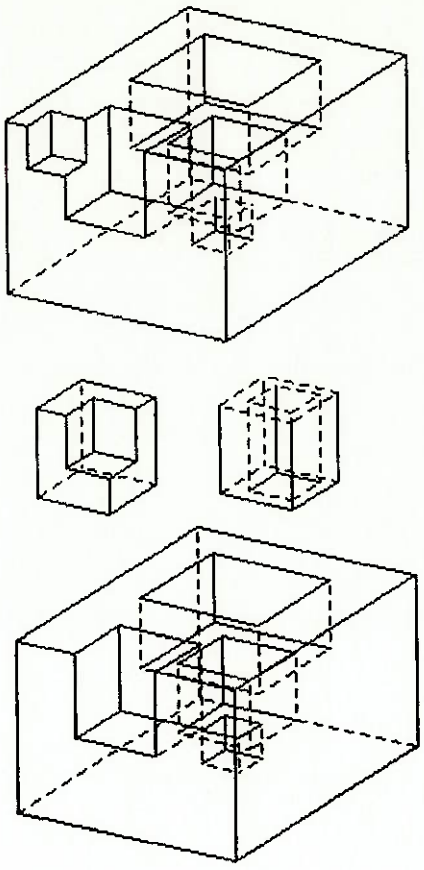


Figura 7.8: Primeiro Nível da Árvore

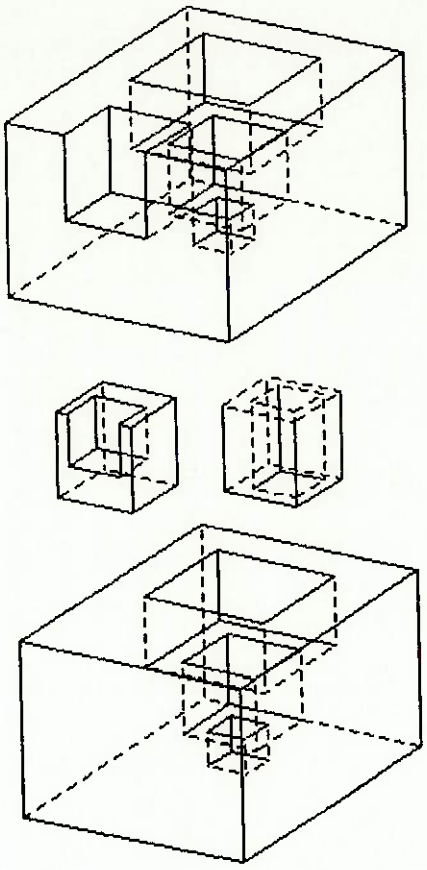


Figura 7.9: Segundo Nível da Árvore

usinagem. A figura 7.14 (a) mostra a sequência obtida pelo escalonamento. A figura 7.14 (b) mostra a inversão para resolver o problema das "features" aninhadas. A figura 7.14 (c) mostra o rearranjo da sequência para que se diminua o número de troca de ferramentas na usinagem. A folha de processo gerada pelo sistema é apresentada na figura 7.15.

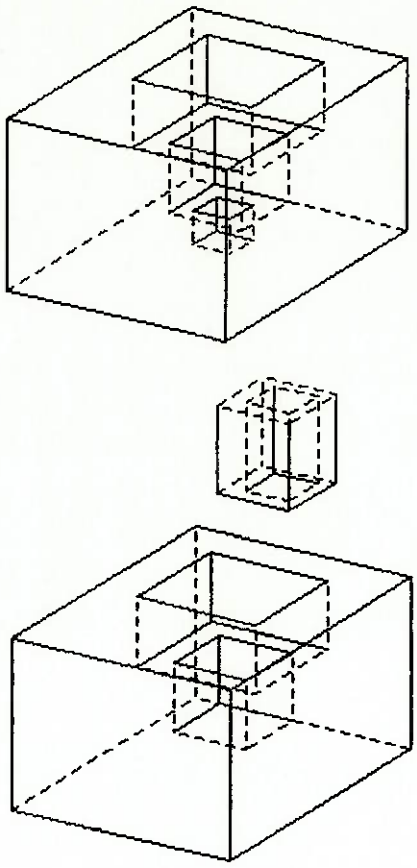


Figura 7.10: Terceiro Nível da Árvore

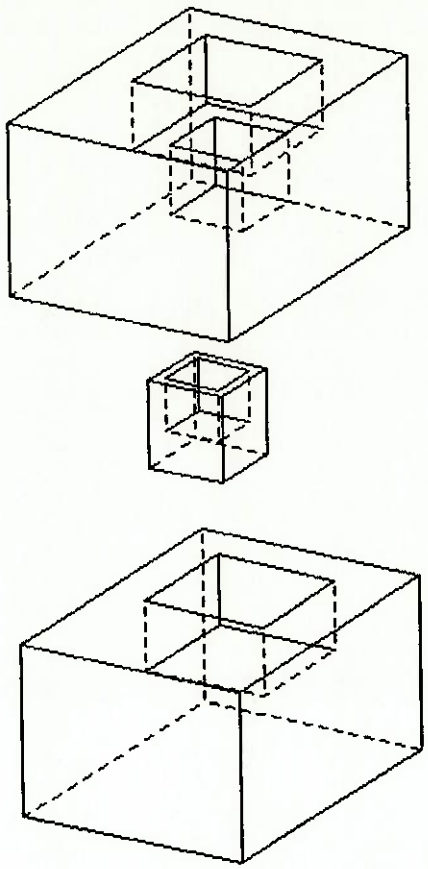


Figura 7.11: Quarto Nível da Árvore

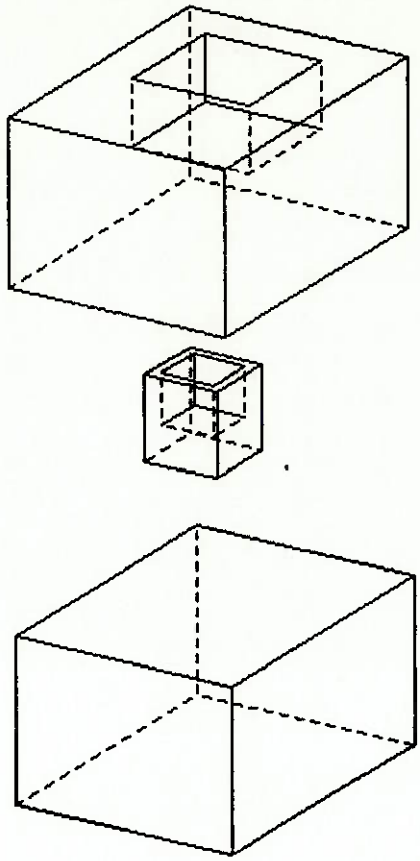


Figura 7.12: Quinto Nível da Árvore

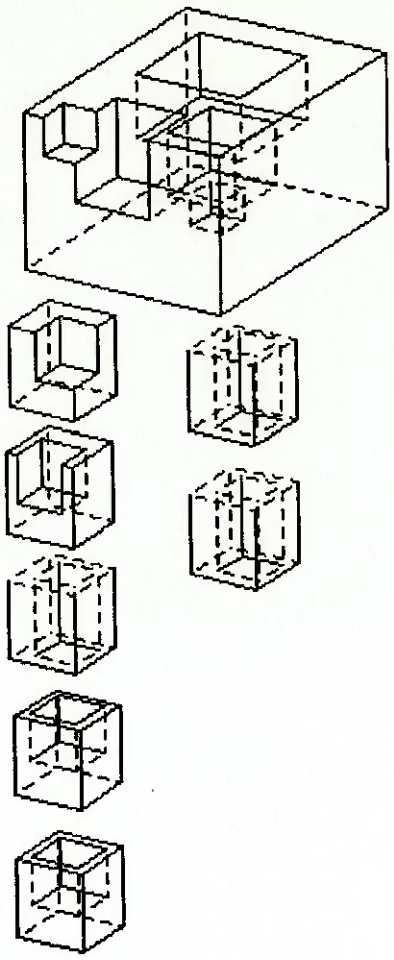


Figura 7.13: Árvore de "Features"

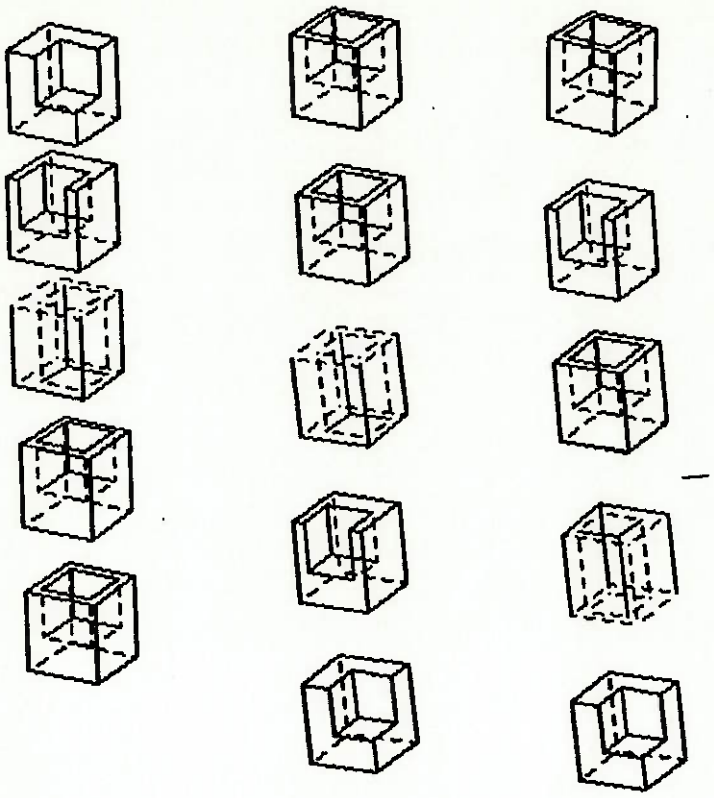


Figura 7.14: Rearranjo da Sequência

Folha de Processo do Solido do Arquivo: exemplo.nos
Material Utilizado: ACO

OP.	TIPO FEATURE	ARQ.FEATURE	FERRAMENTA	MAQUINA	LQC	V.CORTE	RPM	AVANCO	CORTE RADIAL	CORTE AXIAL
					(s/n)	(m/min)		(mm/vol)	(mm)	(mm)
1	furo-np.nos	feat_5.sai	B000000030	M000001	S	45	999	000001		
2	degr-in.nos	feat_3.sai	B000000030	M000001	S	45	999	000001		
3	furo-np.nos	feat_4.sai	B000000019	M000001	S	45	999	000001		
4	furo-pa.nos	feat_2.sai	B000000009	M000001	S	45	999	000001		
5	quina.nos	feat_1.sai	B000000009	M000001	S	45	999	000001		
6	furo-np.nos	feat_5.sai	FD00000030	M000001	S	45	999	000001	4.500	40.000
7	degr-in.nos	feat_3.sai	FD00000030	M000001	S	45	999	000001	4.500	40.000
8	furo-np.nos	feat_5.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
9	furo-np.nos	feat_5.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
10	furo-np.nos	feat_5.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
11	furo-np.nos	feat_5.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
12	degr-in.nos	feat_3.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
13	degr-in.nos	feat_3.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
14	degr-in.nos	feat_3.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
15	degr-in.nos	feat_3.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
16	furo-np.nos	feat_4.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
17	furo-np.nos	feat_4.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
18	furo-pa.nos	feat_2.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000
19	quina.nos	feat_1.sai	FA00000005	M000001	S	22	999	000001	1.000	10.000

Figura 7.15: Folha de Processo do Sólido Exemplo

Capítulo 8

Conclusão

O módulo de *Reconhecimento Automático de "Features"* foi completamente implementado e pode-se dizer que atinge plenamente a condição de ser totalmente genérico. O sistema está preparado para as modificações que devem ocorrer no modelador de sólidos didático (como o acréscimo de arestas curvas). As técnicas de inteligência artificial acrescidas aumentam em muito o desempenho do sistema, como foi demonstrado no gráfico da figura 4.3. O extrator aqui implementado não fazia parte do escopo original do projeto. A solução adotada funciona muito bem para nosso objetivo, mas para algumas "features" ainda deve ser estudado (ele funciona apenas para "features" que para serem extraídas não necessitam que arestas ou faces sejam acrescidas ao sólido, como a quina, o degrau interno, o furo passante e o furo não passante. Para "features" como um degrau ao longo de uma aresta, onde deve ser acrescida uma aresta – ou seja, prolongar as faces – o algoritmo não funciona). Uma solução mais genérica pode ser obtida pela implementação de um algoritmo que seja baseado em operações booleanas do modelador de sólidos, porém com um gasto de tempo computacional muito maior. Talvez, seja interessante tornar o extrator implementado mais genérico.

O módulo de *Escalonamento Automático de "Features"* foi também plenamente implementado. A aplicação da heurística de maior volume (heurística 4, capítulo 6) foi de fundamental importância para que o sistema reduzisse o espaço de busca, tornando o processamento do algoritmo suporável, isto é, polinomial. O algoritmo deixa de criar nós por um método combinatório e passa a criá-los de acordo com um método de seleção através de heurísticas, o que o torna mais efetivo.

O módulo de *Refinamento de "Features"*, composto pelos módulos de *Reconhecimento Automático de "Features"* e *Escalonamento Automático de "Features"*, constitui a base do sistema de planejamento de processos automatizado. A escolha adequada das "features" que compõem o *Arquivo de "Features"*, e que podem ser extraídas do sólido modelado, depende exclusivamente da aplicação e influencia diretamente sobre o desempenho do sistema. Um grande número de "features" no *Arquivo de "Features"* aumenta o campo de uso do sistema, entretanto aumenta o tempo de computação e requer maior

volume de memória.

O módulo de *Seleção do Processo de Usinagem* atingiu seu objetivo: determinar a sequência de usinagem e o processo de manufatura do sólido. Pode-se dizer que a parte mais importante deste módulo do sistema foi a determinação das heurísticas. Deve-se lembrar que algumas restrições foram feitas quanto a aplicação do sistema (usinagem em apenas um plano de "features" de formato retangular). Estas restrições foram feitas na medida em que se pretendia mostrar o caminho para desenvolver o sistema. Muitas outras heurísticas poderiam ser adicionadas, tornando maior o domínio de aplicação do sistema. Um cuidado a ser tomado é para que não se incluam heurísticas em excesso, tornando o sistema redundante em alguns aspectos. Tanto o módulo de *Escalonamento Automático* como o de *Seleção do Processo de Usinagem* devem, em um próximo passo, ser implementados novamente, agora se utilizando PROLOG, a fim de tornar o sistema ainda mais genérico pela introdução de novas heurísticas.

Pode-se dizer, quanto à fase de integração, que o fato do sistema ter sido convenientemente especificado em termos de diagramas PFS, foi de vital importância para que o sistema se ajustasse perfeitamente. Outro fato importante a destacar é a portabilidade do sistema, que atualmente está compilado e operando tanto para micros padrão IBM-PC como para estações de trabalho IBM RS-6000.

Apêndice A

Outros Conceitos Envolvidos

A.1 Grafos

Esta seção apresenta a teoria necessária para tratar-se da implementação de uma árvore, fundamental para o desenvolvimento deste projeto.

Assim, primeiro apresentaremos o conceito de grafos, base para a definição de árvores e, posteriormente, apresentaremos o conceito de árvores propriamente dito.

A.1.1 Conceituaçãa de Grafos

Um grafo consiste de dois subconjuntos: um subconjunto V, não vazio e finito, chamado conjunto dos nós e um subconjunto E, também não vazio e finito, chamado conjunto dos arestas. Pode-se notar então que a todo vértice do grafo é possível associar dois nós. Dois nós conectados por uma aresta são chamados de nós adjacentes. Quando a aresta possui uma direção definida é chamada por aresta orientada, caso contrário é chamada por aresta não orientada. Um grafo que possui todas as suas arestas orientadas é chamado grafo orientado (ou digrafo[1]). O grafo pode ainda ser dito mixado, se for composto de arestas orientadas e arestas não orientadas.

Sejam os nós u e v. Se uma aresta é orientada de u para v então u é dito nó inicial e v é dito nó terminal. Uma aresta que liga estes dois nós é dita incidente a estes dois nós. Uma aresta que liga um nó a ele mesmo é chamada de "loop". Quando duas arestas unem o mesmo par de nós elas são ditas paralelas e o grafo que as contém é denominado um multigrafo. Um nó que não possui conexão a nenhum outro é denominado nó isolado, e um grafo que contém apenas este tipo de nó é chamado de grafo nulo.

Em um grafo direcionado, para qualquer nó v, o número de arestas que tem v por nó inicial é chamado de "grau de saída" do nó v. O número de arestas que tem v por nó

[1]Do original em Inglês directed graph - digraph.

terminal é chamado de "grau de entrada" do nó v. A soma dos valores "grau de saída" e "grau de entrada" é chamado de "grau total".

A definição de grafos não faz referência ao comprimento ou a forma e posicionamento das arestas que unem dois nós quaisquer, bem como não prescreve qualquer ordem para o posicionamento dos nós. Pode acontecer que dois diagramas que pareçam totalmente diferentes representem o mesmo grafo, o que acontece com as figuras A.1 (a) e (a').

Seja um digrafo $G = (V, E)$. Considere a sequência de arestas E de G de tal forma que o nó terminal de qualquer aresta da sequência é o nó inicial da próxima aresta. Um exemplo de tal sequência é:

$$((v_{i1}, v_{i2}), (v_{i2}, v_{i3}), ..., (v_{ik-2}, v_{ik-1}), (v_{ik-1}, v_{ik}))$$

Note que todos os nós da sequência de arestas E precisam ser distintos mas que nem todo conjunto de nós escritos em qualquer ordem fornecem a sequência do modo desejado. Assim, cada nó da sequência deve ser adjacente aos nós que aparecem imediatamente antes e após ao nó, com exceção ao primeiro e último nós. Qualquer sequência de arestas de um digrafo tal que o nó terminal de qualquer uma das arestas na sequência é o nó inicial da próxima aresta da sequência, define um caminho[2] do grafo. Um caminho atravessa os nós da sequência; o caminho se origina no nó inicial da primeira aresta e termina no nó terminal da última aresta da sequência. O número de arestas que aparecem na sequência de um caminho é chamado comprimento[3] do caminho. Um caminho que se inicia e termina no mesmo nó é chamado de circuito[4]. Um digrafo que não possui circuitos é chamado de digrafo acíclico. Naturalmente, estes grafos não podem ter "loops".

A.1.2 Representação de Grafos

Um grafo pode ser coompletamente determinado por suas adjacências e incidências. Estas informações podem ser convenientemente colocadas na forma de uma matriz, facilmente implementada em um programa de computador. Deste modo, para um determinado grafo, existem várias matrizes, incluindo a matriz de adjacência e a matriz de incidência, que serão aqui apresentadas.

Matriz de Adjacência A

A matriz $A = [a_{ij}]$ de um grafo G, com p nós é a matriz $(p \times p)$ na qual $a_{ij} = 1$ se v_i é adjacente a v_j e $a_{ij} = 0$ se v_i não for adjacente a v_j. Deste modo, existe uma correspondência direta entre um grafo com p nós e uma matriz binária simétrica $(p \times p)$ com diagonal zero. A figura A.2 mostra um grafo G e a correspondente matriz de adjacência A.

[2] "path".
[3] "length".
[4] "cycl.

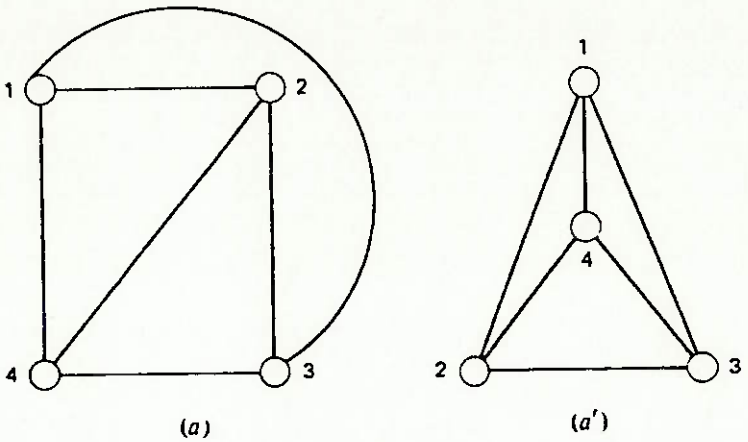


Figura A.1: Exemplos de Diagramas Diferentes que Representam o mesmo Grafo

Uma observação imediata é que a soma dos valores de uma linha i da matriz é igual ao grau do nó v_i. Devido à correspondência entre grafos e suas matrizes de adjacência, análises de propriedades de um grafo podem ser realizadas diretamente sobre estas matrizes.

Matriz de Incidência B

Uma segunda matriz, associada a um grafo G, no qual arestas e nós sã nomeados, é a matriz de adjacência $B = [b_{ij}]$. Esta matriz $(p \times q)$ tem $b_{ij} = 1$ se v_i e x_j são incidentes. De outro modo, $b_{ij} = 0$. Quaisquer $(p-1)$ linhas de B determinam G desde que cada linha é a soma de todas as outras mod 2.

A.1.3 Árvores

Árvores são úteis para descrever qualquer estrutura que envolve hierarquia. Podemos citar alguns exemplos conhecidos como: árvores de família; a classificação decimal de livros em uma livraria; a hierarquia de posições em uma organização; uma expressão algébrica que envolve operações para as quais são determinadas regras de precedência.

No que diz respeito a grafos, uma árvore direcionada é um digrafo acíclico que possui um nó chamado raiz, com "grau de entrada" igual a zero, enquanto todos os outros nós possuem "grau de entrada" igual a um. Uma árvore deve ter pelo menos um nó. Cada nó da árvore é chamado de nó terminal (ou folha), se tiver "grau de saída" igual a zero. O nível de um nó da árvore é o comprimento do seu caminho desde a raiz. Assim, o nível da raiz é igual a zero, enquanto o nível de qualquer nó é igual à sua distância até a raiz. Observe que em qualquer caminho de uma árvore nunca se repete o mesmo nó e que o comprimento de um caminho de um nó qualquer da árvore até um outro nó (se este

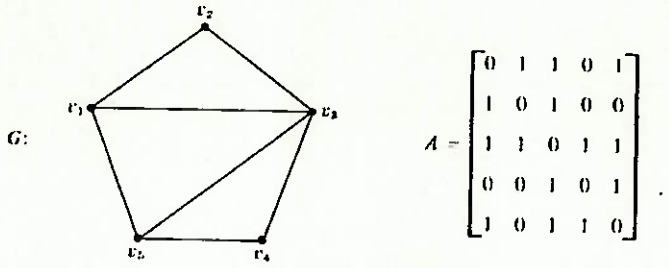


Figura A.2: Um Grafo e sua Matriz de Adjacência

caminho existir) é a distância entre os nós, já que uma árvore é acíclica. A figura A.3 mostra alguns exemplos de árvores.

Com base nos conceitos apresentados acima [Tremblay 84] definiremos **posições relativas** entre dois nós:

Definição 22 *Seja um caminho onde existem dois nós x e y. Se x for o nó inicial do caminho e y o nó terminal do caminho e o valor do comprimento de x em relação à raiz for menor que o valor do comprimento de y em relação à raiz, então x estará acima de y.*

Definição 23 *Sejam dois nós x e y. Se o comprimento dos nós em relação à raiz for igual, então os nós x e y estão ao mesmo nível.*

Ainda com base nos conceitos apresentados por [Tremblay 84] e apoiado nas definições apresentadas acima, definiremos uma nomenclatura própria às árvores, que será utilizada de agora em diante:

Definição 24 *Um nó x, encontrado diretamente acima de um nó y, será dito* **pai** *do nó y.*

Definição 25 *Um nó x, encontrado diretamente abaixo de um nó y, será dito* **filho** *do nó y.*

Definição 26 *Um nó x, encontrado no mesmo nível de um nó y, será dito* **irmão** *do nó y, desde que x e y possuam o mesmo* **pai**.

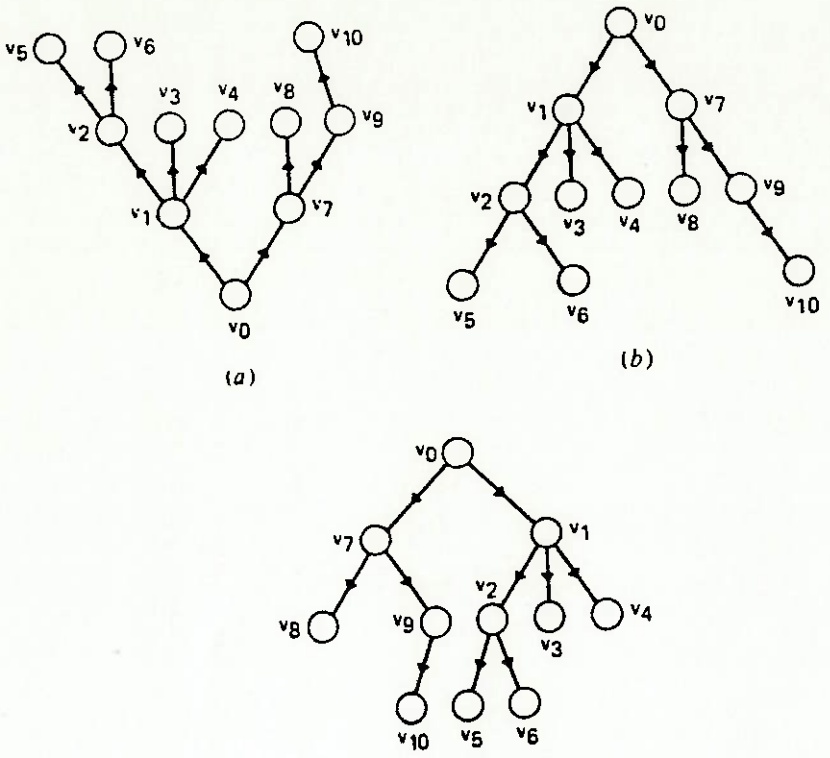


Figura A.3: Árvores

Um outro conceito importante é que as árvores sao recursivas. Assim, uma árvore contém um ou mais nós de tal modo que um nó é chamado raiz enquanto todos os outros nós são divididos em um número finito de árvores ditas sub-árvores.

Aplicaremos agora restrições acerca dos "nós de saída" de cada nó da árvore. Se em uma árvore o "grau de saída" for menor ou igual a um valor M, então a árvore será dita "árvore M-ária". No caso de não haver restrição, a árvore é chamada de **Árvore Genérica**. Para $M = 2$, teremos as chamadas **Árvores Binárias**. A figura A.4 apresenta exemplos de árvores binárias. Árvores binárias são muito úteis em um grande número de aplicações, inclusive no sistema proposto neste trabalho, como poderá ser notado adiante.

A.2 Operadores de Euler

O principal objetivo de se utilizar Operadores de Euler é facilitar a manipulação das complicadas estruturas B-REP, construindo os modelos passo a passo pelo uso de um conjunto mínimo de operadores que manipule a estrutura de dados B-REP e esconda detalhes de implementação. Os Operadores de Euler tornam possível a construção incremental de um objeto, de maneira semelhante a desenhar o objeto linha a linha. Eles também facilitam alterações locais de forma, característica que é muito importante para projetistas de modeladores de sólidos. A estrutura B-REP pode ainda ser utilizada de forma a determinar

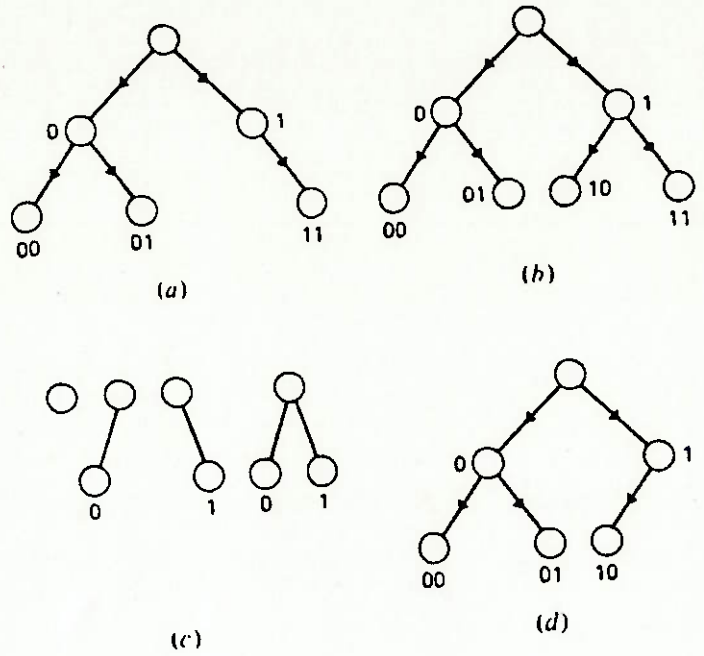


Figura A.4: Árvores Binárias

se o sólido criado é um sólido válido.

Um sólido de s peças desconexas e constituído por f faces, e arestas, v vértices e h furos deve obedecer à fórmula de Euler-Poincaré:

$$v - e + f = 2 * (s - h)$$

Esta fórmula pode ser modificada de forma que se torne consistente com as convenções da estrutura B-REP, pela adição de um novo parâmetro: l, que representa o número de laços. Todos os laços, com exceção de um em cada face, podem ser removidos pela introdução de arestas-ponte conectando os laços entre si. Nenhum novo vértice é introduzido por esta operação, mas o número de arestas depois de removidos os $l - f$ laços deverá ser $e' = e + l - f$. Substituindo e' por e na equação, temos:

$$v - e + 2 * f = 2 * (s - h) + l$$

Diferenciando-se os laços internos (chamados de anéis, que definem o contorno interno das faces) dos laços externos (que definem os contornos externos das faces) e assumindo-se que um sólido possua r anéis, obtemos:

$$v - e + f = 2 * (s - h) + r$$

A fórmula de Euler-Poincaré restringe as combinações válidas entre primitivos a um subconjunto de todas as combinações possíveis.

Vários autores demonstram que cinco operadores são suficientes para construir todos os objetos. Estes cinco operadores podem ser escolhidos de várias maneiras. Um conjunto possível de cinco operadores de Euler é apresentado abaixo:

1. **mvsf(Make Vertex Solid Face)**: Cria um sólido inicial, consistindo apenas de uma face inicial e um vértice;
2. **mev(Make Edge Vertex)**: Adiciona uma nova aresta, conectando um vértice a um novo vértice;
3. **mef(Make Edge Face)**: Divide uma face por meio de uma nova aresta conectando dois vértices;
4. **kemr(Kill Edge Make Ring)**: Dinide o contorno de uma face em dois componentes pela remoção da aresta-ponte;
5. **kfmrh(Kill Face Make Ring Hole)**: Une duas faces de maneira que o contorno da primeira face se torne componente do contorno da segunda face.

Durante a construção de modelos com Operadores de Euler, a topologia é mantida válida segundo a equação de Euler-Poincaré. Ao final de uma sequência de Operadores de Euler assume-se que a geometria do sólido está correta, mas durante os estágios intermediários não há como manter a geometria e a topologia consistentes devido a presença de faces não planares, que, frequentemente, não são possíveis de serem representadas por nenhuma forma matemática. Portanto, os Operadores de Euler são seguros por si, mas devem ser colecionados em sequências que forneçam um significado[Tsuzuki 91b].

Um exemplo de como funcionam estes operadores pode ser visto na tabela A.1, que mostra os operadores de Euler que representam um cubo de lado 10.

mvsf	1 1 2 1 0 10.0 0.0 0.0
svme	1 2 2 2 1 1 1 1 1 0.0 0.0 0.0
svme	1 2 1 1 3 1 1 1 1 10.0 10.0 0.0
svme	1 3 2 2 4 1 1 1 1 0.0 10.0 0.0
mef	1 1 4 2 3 1 1 1 2
svme	1 1 2 2 5 2 2 1 1 0.0 0.0 10.0
svme	1 2 3 3 6 2 2 1 1 10.0 0.0 10.0
mef	1 5 6 1 2 2 2 2 3
svme	1 3 4 4 7 2 2 1 1 10.0 10.0 10.0
mef	1 6 7 2 3 2 3 2 4
svme	1 4 1 1 8 2 2 1 1 0.0 10.0 10.0
mef	1 7 8 3 4 2 4 2 5
mef	1 8 5 4 6 2 5 3 6

obs1: Este é um exemplo real, retirado do modelador de sólidos didático e os valores apresentados são aqui apenas ilustrativos, representando o número de faces, arestas e vértices, comprimento de arestas, câmera do objeto, ...

obs2: O operador SVME é análogo ao operador MEV, sendo de mais alto nível, apresentando informações de câmera, grupo, etc....

Tabela A.1: Operadores de Euler para um Cubo de lado 10

Apêndice B

Simulação de um Reconhecimento de "Feature"

Este apêndice tem por objetivo mostrar como se realiza o processo de reconhecimento automático de "features". A figura B.1 mostra um sólido e uma "feature" e a seguir é apresentado o processo de reconhecimento da "feature" no sólido.

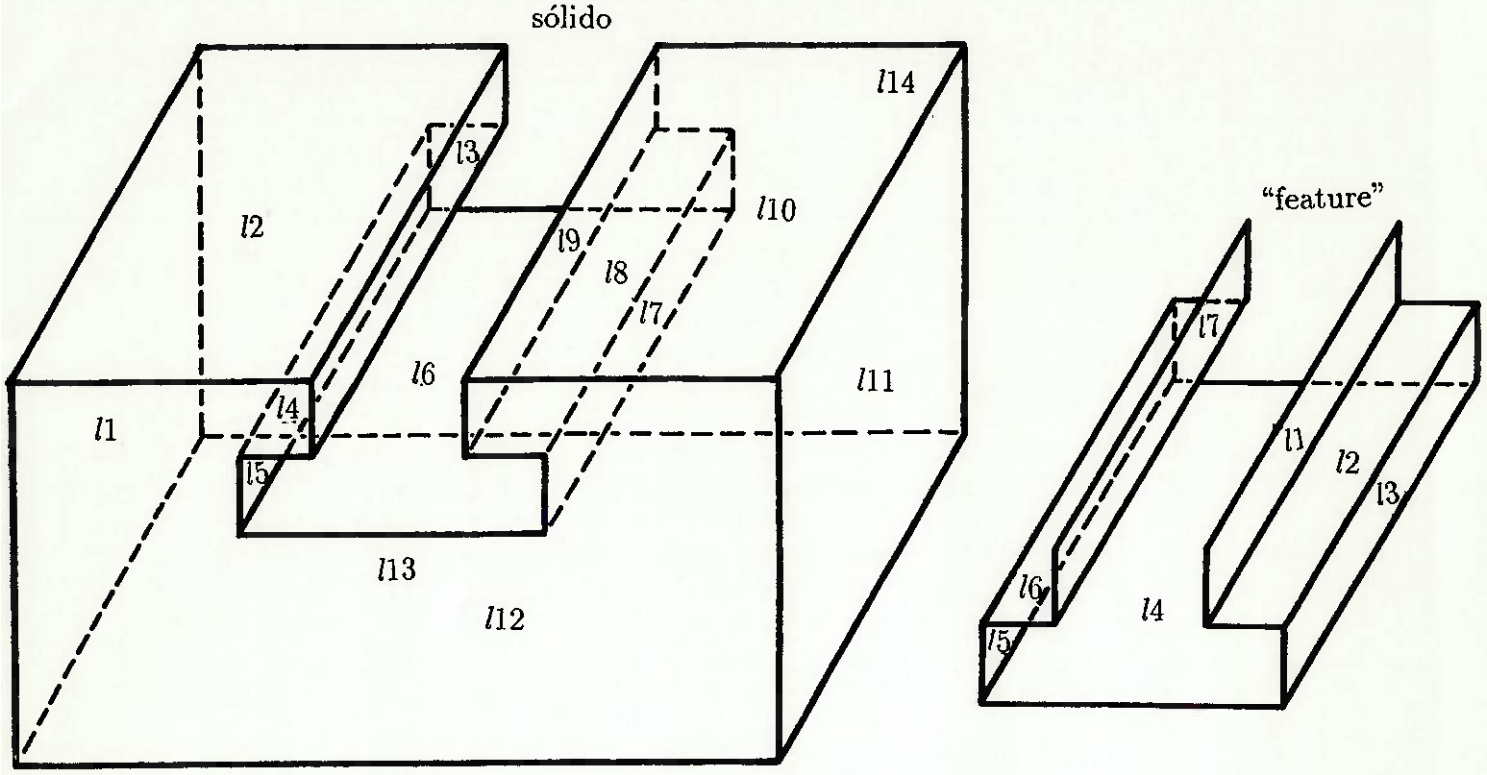


Figura B.1: "Feature" a ser reconhecida no sólido

"Feature" modelada[1] para o Reconhecedor:

```
7         \* Numero de faces da "feature"              *\

1         \* Numero da face da "feature"               *\
4 1       \* Numero de lacos da Funcao de Modelagem e 1[L]*\
0 0 0 2   \* Parte topologica da Funcao de Modelagem   *\
1         \* Relacao de adjacencia desconexa 1[L]      *\
0 0 0 270\* Parte geometrica da Funcao de Modelagem    *\

2
4 1
0 1 0 3
2
0 270 0 90

3
4 1
2 0 4 0
3
90 0 90 0

4
4 1
3 0 5 0
```

[1]Neste exemplo utilizamos como informaçã geométrica somente os ângulos entre faces

```
4
90 0 90 0

5
4 1
4 0 6 0
5
90 0 90 0

6
4 1
5 0 7 0
6
90 0 270 0

7
4 1
0 0 0 6
7
0 0 0 270
```

Sólido modelado para o Reconhecedor:

```
14

1
4 1
2 14 12 13
1
270 270 270 270

2
4 1
1 13 3 14
2
270 270 270 270

3
4 1
2 13 4 14
3
270 270 270 270

4
4 1
3 13 5 14
4
270 270 90 270

5
4 1
13 6 14 4
5
270 90 270 90

6
4 1
13 7 14 5
6
270 90 270 90

7
4 1
13 8 14 6
7
270 90 270 90

8
4 1
13 9 14 7
8
270 270 270 90

9
4 1
13 10 14 8
9
270 270 270 270

10
4 1
13 11 14 9
10
270 270 270 270

11
```

```
4 1
13 12 14 10
11
270 270 270 270

12
4 1
13 1 14 11
12
270 270 270 270

13
12 1
12 11 10 9 8 7 6 5 4 3 2 1
13
270 270 270 270 270 270 270 270 270 270 270 270

14
12 1
1 2 3 4 5 6 7 8 9 10 11 12
14
270 270 270 270 270 270 270 270 270 270 270 270
```

Relatório de saída do Reconhecedor:

```
PROCESSO DE COMPARACAO PASSO A PASSO


POSSIVEIS f.l.lb :
 1 2 3 4 5 6 7 8 9 10 11 12

POSSIVEL f.l.lb  :
 1

POSSIVEL f.l.la  :
 1

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  2 14 12 13 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1

PASSO1 : primeiro par -> 1 1

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  2 14 12 13 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  14 12 13 2 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1
```

```
PASSO3 : nova concatenacao

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 2
1  14 12 13 2 | 0 0 0 2
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 1
1  1 | 1


2  0 1 0 3 | 0 1 0 0
2  14 1 13 3 | 0 1 0 0
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
2  2 | 2
2  2 | 2


numero de elementos na pilha = 2

PASSO2 : novo par -> 2 2

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 2
1  14 12 13 2 | 0 0 0 2
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 1
1  1 | 1


2  0 1 0 3 | 0 1 0 0
2  14 1 13 3 | 0 1 0 0
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
2  2 | 2
2  2 | 2


numero de elementos na pilha = 2

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  14 12 13 2 | 0 0 0 0
```

```
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1

PASSO3 : INSUCESSO

*************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  12 13 2 14 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1

PASSO3 : nova concatenacao

*************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  12 13 2 14 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1

PASSO2 : INSUCESSO

*************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  13 2 14 12 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1

PASSO3 : nova concatenacao
```

```
************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 12
1  13 2 14 12 | 0 0 0 12
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 1
1  1 | 1


2  0 1 0 3 | 0 1 0 0
12  13 1 14 11 | 0 1 0 0
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
2  2 | 12
12  12 | 12


numero de elementos na pilha = 2

PASSO2 : novo par -> 2 12

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 12
1  13 2 14 12 | 0 0 0 12
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 1
1  1 | 1


2  0 1 0 3 | 0 1 0 0
12  13 1 14 11 | 0 1 0 0
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
2  2 | 12
12  12 | 12


numero de elementos na pilha = 2

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
1  13 2 14 12 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
```

```
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 1
1  1 | 1


numero de elementos na pilha = 1

PASSO3 : INSUCESSO


POSSIVEL f.l.lb  :
 2

POSSIVEL f.l.la  :
 1

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
2  1 13 3 14 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 2
2  2 | 2


numero de elementos na pilha = 1

PASSO3 : novo primeiro par -> 1 2

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
2  1 13 3 14 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 2
2  2 | 2


numero de elementos na pilha = 1

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
2  13 3 14 1 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 2
2  2 | 2
```

```
numero de elementos na pilha = 1

PASSO3 : nova concatenacao

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 1
2  13 3 14 1 | 0 0 0 1
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


2  0 1 0 3 | 0 2 0 0
1  13 2 14 12 | 0 2 0 0
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
2  2 | 1
1  1 | 1


numero de elementos na pilha = 2

PASSO2 : novo par -> 2 1

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 1
2  13 3 14 1 | 0 0 0 1
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


2  0 1 0 3 | 0 2 0 0
1  13 2 14 12 | 0 2 0 0
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
2  2 | 1
1  1 | 1


numero de elementos na pilha = 2

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha
```

```
1  0 0 0 2 | 0 0 0 0
2  13 3 14 1 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


numero de elementos na pilha = 1

PASSO3 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
2  3 14 1 13 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


numero de elementos na pilha = 1

PASSO3 : nova concatenacao

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
2  3 14 1 13 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


numero de elementos na pilha = 1

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
2  14 1 13 3 | 0 0 0 0
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2
```

```
numero de elementos na pilha = 1

PASSO3 : nova concatenacao

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 3
2  14 1 13 3 | 0 0 0 3
-----------------------------------------------------------
 0.000      0.000      0.000      270.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


2  0 1 0 3 | 0 2 0 0
3  14 2 13 4 | 0 2 0 0
-----------------------------------------------------------
 0.000      270.000    0.000      90.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
2  2 | 3
3  3 | 3


numero de elementos na pilha = 2

PASSO2 : novo par -> 2 3

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 3
2  14 1 13 3 | 0 0 0 3
-----------------------------------------------------------
 0.000      0.000      0.000      270.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


2  0 1 0 3 | 0 2 0 0
3  14 2 13 4 | 0 2 0 0
-----------------------------------------------------------
 0.000      270.000    0.000      90.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
2  2 | 3
3  3 | 3


numero de elementos na pilha = 2

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
```

```
2  14 1 13 3 | 0 0 0 0
-----------------------------------------------------------
 0.000      0.000      0.000      270.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
1  1 | 2
2  2 | 2


numero de elementos na pilha = 1

PASSO3 : INSUCESSO


POSSIVEL f.l.lb  :
 3

POSSIVEL f.l.la  :
 1

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
3  2 13 4 14 | 0 0 0 0
-----------------------------------------------------------
 0.000      0.000      0.000      270.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
1  1 | 3
3  3 | 3


numero de elementos na pilha = 1

PASSO3 : novo primeiro par -> 1 3

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
3  2 13 4 14 | 0 0 0 0
-----------------------------------------------------------
 0.000      0.000      0.000      270.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
1  1 | 3
3  3 | 3


numero de elementos na pilha = 1

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
3  13 4 14 2 | 0 0 0 0
-----------------------------------------------------------
 0.000      0.000      0.000      270.000
 270.000    270.000    270.000    270.000
```

```
-----------------------------------------------------
1  1 | 3
3  3 | 3


numero de elementos na pilha = 1

PASSO3 : nova concatenacao

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 2
3  13 4 14 2 | 0 0 0 2
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 3
3  3 | 3


2  0 1 0 3 | 0 3 0 0
2  13 3 14 1 | 0 3 0 0
-----------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
2  2 | 2
2  2 | 2


numero de elementos na pilha = 2

PASSO2 : novo par -> 2 2

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 2
3  13 4 14 2 | 0 0 0 2
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 3
3  3 | 3


2  0 1 0 3 | 0 3 0 0
2  13 3 14 1 | 0 3 0 0
-----------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
2  2 | 2
2  2 | 2


numero de elementos na pilha = 2

PASSO2 : INSUCESSO
```

```
************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
3  13 4 14 2 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 3
3  3 | 3


numero de elementos na pilha = 1

PASSO3 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
3  4 14 2 13 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 3
3  3 | 3


numero de elementos na pilha = 1

PASSO3 : nova concatenacao

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
3  4 14 2 13 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 3
3  3 | 3


numero de elementos na pilha = 1

PASSO2 : INSUCESSO

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 0
3  14 2 13 4 | 0 0 0 0
-----------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------
1  1 | 3
```

```
3  3 | 3


numero de elementos na pilha = 1

PASSO3 : nova concatenacao

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 4
3  14 2 13 4 | 0 0 0 4
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 3
3  3 | 3


2  0 1 0 3 | 0 3 0 0
4  14 3 13 5 | 0 3 0 0
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   90.000
-----------------------------------------------------------
2  2 | 4
4  4 | 4


numero de elementos na pilha = 2

PASSO2 : novo par -> 2 4

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 4
3  14 2 13 4 | 0 0 0 4
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 3
3  3 | 3


2  0 1 0 3 | 0 3 0 5
4  14 3 13 5 | 0 3 0 5
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   90.000
-----------------------------------------------------------
2  2 | 4
4  4 | 4


3  2 0 4 0 | 4 0 0 0
5  4 13 6 14 | 4 0 0 0
-----------------------------------------------------------
 90.000    0.000     90.000    0.000
 90.000    270.000   90.000    270.000
-----------------------------------------------------------
```

```
3  3 | 5
5  5 | 5


numero de elementos na pilha = 3

PASSO2 : novo par -> 3 5

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 4
3  14 2 13 4 | 0 0 0 4
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 3
3  3 | 3


2  0 1 0 3 | 0 3 0 5
4  14 3 13 5 | 0 3 0 5
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   90.000
-----------------------------------------------------------
2  2 | 4
4  4 | 4


3  2 0 4 0 | 4 0 6 0
5  4 13 6 14 | 4 0 6 0
-----------------------------------------------------------
 90.000    0.000     90.000    0.000
 90.000    270.000   90.000    270.000
-----------------------------------------------------------
3  3 | 5
5  5 | 5


4  3 0 5 0 | 5 0 0 0
6  5 13 7 14 | 5 0 0 0
-----------------------------------------------------------
 90.000    0.000     90.000    0.000
 90.000    270.000   90.000    270.000
-----------------------------------------------------------
4  4 | 6
6  6 | 6


numero de elementos na pilha = 4

PASSO2 : novo par -> 4 6

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 4
3  14 2 13 4 | 0 0 0 4
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
```

```
-----------------------------------------------------------
1  1 | 3
3  3 | 3


2  0 1 0 3 | 0 3 0 5
4  14 3 13 5 | 0 3 0 5
-----------------------------------------------------------
 0.000     270.000   0.000     90.000
 270.000   270.000   270.000   90.000
-----------------------------------------------------------
2  2 | 4
4  4 | 4


3  2 0 4 0 | 4 0 6 0
5  4 13 6 14 | 4 0 6 0
-----------------------------------------------------------
 90.000    0.000     90.000    0.000
 90.000    270.000   90.000    270.000
-----------------------------------------------------------
3  3 | 5
5  5 | 5


4  3 0 5 0 | 5 0 7 0
6  5 13 7 14 | 5 0 7 0
-----------------------------------------------------------
 90.000    0.000     90.000    0.000
 90.000    270.000   90.000    270.000
-----------------------------------------------------------
4  4 | 6
6  6 | 6


5  4 0 6 0 | 6 0 0 0
7  6 13 8 14 | 6 0 0 0
-----------------------------------------------------------
 90.000    0.000     90.000    0.000
 90.000    270.000   90.000    270.000
-----------------------------------------------------------
5  5 | 7
7  7 | 7


numero de elementos na pilha = 5

PASSO2 : novo par -> 5 7

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 4
3  14 2 13 4 | 0 0 0 4
-----------------------------------------------------------
 0.000     0.000     0.000     270.000
 270.000   270.000   270.000   270.000
-----------------------------------------------------------
1  1 | 3
3  3 | 3


2  0 1 0 3 | 0 3 0 5
4  14 3 13 5 | 0 3 0 5
```

```
-----------------------------------------------------------
 0.000      270.000    0.000      90.000
 270.000    270.000    270.000    90.000
-----------------------------------------------------------
2  2 | 4
4  4 | 4


3  2 0 4 0 | 4 0 6 0
5  4 13 6 14 | 4 0 6 0
-----------------------------------------------------------
 90.000     0.000      90.000     0.000
 90.000     270.000    90.000     270.000
-----------------------------------------------------------
3  3 | 5
5  5 | 5


4  3 0 5 0 | 5 0 7 0
6  5 13 7 14 | 5 0 7 0
-----------------------------------------------------------
 90.000     0.000      90.000     0.000
 90.000     270.000    90.000     270.000
-----------------------------------------------------------
4  4 | 6
6  6 | 6


5  4 0 6 0 | 6 0 8 0
7  6 13 8 14 | 6 0 8 0
-----------------------------------------------------------
 90.000     0.000      90.000     0.000
 90.000     270.000    90.000     270.000
-----------------------------------------------------------
5  5 | 7
7  7 | 7


6  5 0 7 0 | 7 0 0 0
8  7 13 9 14 | 7 0 0 0
-----------------------------------------------------------
 90.000     0.000      270.000    0.000
 90.000     270.000    270.000    270.000
-----------------------------------------------------------
6  6 | 8
8  8 | 8


numero de elementos na pilha = 6

PASSO2 : novo par -> 6 8

************************************************************

Estado atual da pilha

1  0 0 0 2 | 0 0 0 4
3  14 2 13 4 | 0 0 0 4
-----------------------------------------------------------
 0.000      0.000      0.000      270.000
 270.000    270.000    270.000    270.000
-----------------------------------------------------------
1  1 | 3
3  3 | 3
```

```
2  0 1 0 3 | 0 3 0 5
4  14 3 13 5 | 0 3 0 5
-----------------------------------------------------
 0.000      270.000   0.000     90.000
 270.000    270.000   270.000   90.000
-----------------------------------------------------
2  2 | 4
4  4 | 4


3  2 0 4 0 | 4 0 6 0
5  4 13 6 14 | 4 0 6 0
-----------------------------------------------------
 90.000     0.000     90.000    0.000
 90.000     270.000   90.000    270.000
-----------------------------------------------------
3  3 | 5
5  5 | 5


4  3 0 5 0 | 5 0 7 0
6  5 13 7 14 | 5 0 7 0
-----------------------------------------------------
 90.000     0.000     90.000    0.000
 90.000     270.000   90.000    270.000
-----------------------------------------------------
4  4 | 6
6  6 | 6


5  4 0 6 0 | 6 0 8 0
7  6 13 8 14 | 6 0 8 0
-----------------------------------------------------
 90.000     0.000     90.000    0.000
 90.000     270.000   90.000    270.000
-----------------------------------------------------
5  5 | 7
7  7 | 7


6  5 0 7 0 | 7 0 9 0
8  7 13 9 14 | 7 0 9 0
-----------------------------------------------------
 90.000     0.000     270.000   0.000
 90.000     270.000   270.000   270.000
-----------------------------------------------------
6  6 | 8
8  8 | 8


7  0 0 0 6 | 0 0 0 8
9  13 10 14 8 | 0 0 0 8
-----------------------------------------------------
 0.000      0.000     0.000     270.000
 270.000    270.000   270.000   270.000
-----------------------------------------------------
7  7 | 9
9  9 | 9


numero de elementos na pilha = 7

PASSO2 : novo par -> 7 9
```

```
**************************************************************

SOLIDOS EQUIVALENTES

numero de comparacoes = 19
```

Apêndice C

Modelagem do Exemplo Proposto

Este apêndice mostra como é feita a modelagem do sólido do exemplo apresentado no capítulo 7 e as modelagens das "features" do Banco de Dados de "features".

Modelagem do Exemplo Proposto:

```
31

1
4 1
2 26 4 5
1
90 40 270 40 90 40 90 40

2
4 1
3 26 1 5
2
90 40 270 40 90 40 90 40

3
4 1
4 26 2 5
3
90 40 270 40 90 40 90 40

4
4 1
1 26 3 5
4
90 40 270 40 90 40 90 40

5
4 2
4 3 2 1
5 6
90 40 90 40 90 40 90 40

6
```

```
4 1
7 8 9 10
6
270 20 270 20 270 20 270 20

7
4 1
8 6 10 11
7
90 20 270 20 90 20 90 20

8
4 1
9 6 7 11
8
90 20 270 20 90 20 90 20

9
4 1
10 6 8 11
9
90 20 270 20 90 20 90 20

10
4 1
7 6 9 11
10
90 20 270 20 90 20 90 20

11
4 2
10 9 8 7
11 28
90 20 90 20 90 20 90 20

12
10 2
23 15 14 13 18 19 21 15 22 24
12 26
270 100 270 100 270 10 270 10 270 40 270 40 270 40 270 100 270 100 270 100

13
4 1
12 14 27 18
13
270 10 90 10 90 10 270 10

14
4 1
12 15 27 13
14
270 10 270 10 90 10 90 10

15
10 1
23 25 22 12 21 20 18 27 14 12
15
270 70 270 100 270 70 270 100 270 40 270 40 270 40 270 10 270 10 270 100

18
6 1
27 15 20 19 12 13
18
270 10 270 40 90 40 90 40 270 40 270 10
```

```
19
4 1
18 20 21 12
19
90 40 90 40 90 40 270 40

20
4 1
18 15 21 19
20
90 40 270 40 90 40 90 40

21
4 1
19 20 15 12
21
90 40 90 40 270 40 270 40

22
4 1
15 25 24 12
22
270 70 270 100 270 70 270 100

23
4 1
12 24 25 15
23
270 100 270 70 270 100 270 70

24
4 1
12 22 25 23
24
270 100 270 70 270 100 270 70

25
4 1
24 22 15 23
25
270 100 270 100 270 100 270 100

26
4 1
1 2 3 4
26
270 40 270 40 270 40 270 40

27
4 1
15 18 13 14
27
270 10 270 10 90 10 90 10

28
4 1
29 30 31 32
28
270 10 270 10 270 10 270 10

29
4 1
30 28 32 33
29
90 10 270 10 90 10 90 10
```

```
30
4 1
31 28 29 33
30
90 10 270 10 90 10 90 10

31
4 1
32 28 30 33
31
90 10 270 10 90 10 90 10

32
4 1
29 28 31 33
32
90 10 270 10 90 10 90 10

33
4 1
32 31 30 29
33
270 10 270 10 270 10 270 10
```

Modelagem da Quina:

```
3

1
4 1
0 0 3 2
1
270 0 270 0 90 0 90 1

2
4 1
0 0 1 3
2
270 0 270 0 90 0 90 0

3
4 1
1 0 0 2
3
90 3 270 0 270 0 90 2
```

Modelagem do Degrau Interno:

```
4

1
4 1
0 0 4 2
1
270 0 270 0 90 0 90 3

2
4 1
0 1 4 3
2
270 0 90 0 90 0 90 0

3
4 1
0 0 2 4
3
270 0 270 0 90 0 90 0

4
4 1
0 3 2 1
4
270 2 90 1 90 0 90 0
```

Modelagem Do Furo Não Passante

```
5

1
4 1
2 0 4 5
1
90 0 270 0 90 0 90 1

2
4 1
3 0 1 5
2
90 0 270 0 90 0 90 2

3
4 1
4 0 2 5
3
90 0 270 0 90 3 90 0

4
4 1
1 0 3 5
4
90 0 270 0 90 0 90 0

5
4 1
4 3 2 1
5
90 0 90 0 90 0 90 0
```

Modelagem do Furo Passante:

```
4

1
4 1
4 0 2 0
1
90 0 270 0 90 0 270 0

2
4 1
1 0 3 0
2
90 3 270 1 90 0 270 0

3
4 1
4 0 2 0
3
90 0 270 2 90 0 270 0

4
4 1
3 0 1 0
4
90 0 270 0 90 0 270 0
```

Bibliografia

[Chang 90] T. C. Chang, "Expert Process Planning for Manufacturing", *Addison-Wesley Publishing Company, Inc.*, 1990.

[Choi 84] B. K. Choi, M. M. Barash, D. C. Anderson, "Automatic recognition of machined surfaces from 3D solid model", *Computer Aided Design*, 16(2):81-86, March 1984.

[Ferreira 91] J. C. E. Ferreira, S. Hinduja, "Giving a Tool-Orientation to Manufacturing Features in 21/2-D Components", *XI Congresso Brasileiro de Engenharia Mecânica*, São Paulo, Dezembro de 1991.

[Floriani 89] L. de Floriani, E. Bruzzone, "Building a feature-based object description from a boundary model", *Computer Aided Design*, 21(10):602-610, December 1989.

[Fox 88] D. L. Fox, "Dynamic Memory Management in C", *BYTE*, June 1988.

[Garcia 91a] M. B. Garcia, "Escalonamento Automático de Features", *Primeiro Relatório, Processo Fapesp 91/1031-6*, Outubro 1991.

[Garcia 91b] M. B. Garcia, M. S. G. Tsuzuki, "Escalonamento Automático de Features", *VIII Colóquio de Iniciação Científica do IME-USP*, Outubro 1991.

[Garcia 91c] M. B. Garcia, M. S. G. Tsuzuki, "Escalonamento Automático de Features", *X CICTE*, São Carlos - SP, Dezembro 1991.

[Garcia 92] M. B. Garcia, "Escalonamento Automático de Features", *Segundo Relatório, Processo Fapesp 91/1031-6*, Abril 1992.

[Gavancar 90] P. Gavancar, M. R. Henderson, "Graph-based extraction of protusions and depressions from boundary representations", *Computer Aided Design*, 22(7):442-450, September 1990.

[Harary 71] F. Harary, "Graph Theory", *Addison-Wesley Publishing Company, Inc.*, Second Printing, April 1971.

[Hasegawa 88] K. Hasegawa, K. Takahashi, P. E. Miyagi, "Applications of the Mark Flow Graph to Represent Discrete Event Production Systems and System Control", *Transactions of the Society of Instrument and Control Engineers*, vol. 24, No. 1, Tokio, 1988.

[Henderson 86] M. R. Henderson, "Extraction and Organization of Form Features", *Elsevier Science Publishers B. V. (North-Holland*, 1986.

[Joshi 88a] S. Joshi, T. C. Chang, "Graph-Based heuristics for recognition of machined features from a 3D solid model", *Computer Aided Design*, 20(2):58-66, March 1988.

[Joshi 88b] S. Joshi, N. N. Vissa, T. C. Chang, "Expert process planning system with solid model interface", *INT. J. PROD. RES.*, 26(5):863-885, 1988.

[Mäntylä 88] M. Mäntylä, "An Introdution to Solid Modeling", *Computer Science Press*, 1988.

[Markowsky 80] G. Markowsky, M.A. Wesley, "Fleshing Out Wire Frames", *IBM J. Res. Development - vol. 24*, September 1980 .

[Mathews 88] J. Mathews, "Threaded Binary Trees", *Dr. Dobb's Journal*, March 1988.

[Mortenson 85] M.E. Mortenson, "Geometric Modeling", *John Wiley & Sons*, 1985.

[Miyagi 88] P. E. Miyagi, K. Hasegawa, K. Takahashi, "A Programming Language for Discrete Event Production Flow Schema and Mark Flow Graph", *Transactions of the Society of Instrument and Control Engineers*, vol. 24, No. 1, Tokio, 1988.

[Nilsson 82] N. J. Nilsson, "Principles of Artificial Intelligence", *Springer-Verlag*, 1982.

[Pearl 84] J. Pearl, "Heuristics" *Addison-Wesley Publishing Company, Inc.*, 1984.

[Schildt 90] H. Schildt, "Turbo C Avançado (Guia do Usuário)", *Mc. Graw Hill*, 1990.

[Tichenor 88] M. Tichenor, "Virtual Arrays in C", *Dr. Dobb's Journal*, May 1988.

[Toledo 91a] C. F. M. Toledo, "Reconhecimento Automático de Features", *Primeiro Relatório, Processo Fapesp 91/0270-7*, Setembro 1991.

[Toledo 91b] C. F. M. Toledo, M. S. G. Tsuzuki, "Reconhecimento Automático de Features Baseado na Representação B-rep", *VIII Colóquio de Iniciação Científica do IME-USP*, Outubro 1991.

[Toledo 91c] C. F. M. Toledo, M. S. G. Tsuzuki, "Reconhecimento Automático de Features", *X CICTE*, São Carlos - SP, Dezembro 1991.

[Toledo 92a] C. F. M. Toledo, "Reconhecimento Automático de Features", *Segundo Relatório, Processo Fapesp 91/0270-7*, Março 1992.

[Toledo 92b] C. F. M. Toledo, "CAPP Automático", *Terceiro Relatório, Processo Fapesp 91/0270-7*, Agosto 1992.

[Tsuzuki 88] M. S. G. Tsuzuki, "A Study on the Representation of Three Dimensional Solids", *Master Thesis*, Yokohama National University, Japan, March 1988.

[Tsuzuki 91a] M. S. G. Tsuzuki, P. E. Miyagi, "Modelador de Sólidos Didático", Seminário sobre computação gráfica em A.E.C., *SOBRACON*, Novembro 1991.

[Tsuzuki 91b] M. S. G. Tsuzuki, "Modelagem de Sólidos: Representação por Fronteira (B-REP)", *Congresso Brasileiro de Engenharia Mecânica*, 10, São Paulo, SP, 1991. COBEM 91, v. 13, p. 611-614.

[Tsuzuki 91c] M. S. G. Tsuzuki, "Reconhecimento Automático de "Features" baseado na Representação B-REP", *Congresso Brasileiro de Engenharia Mecânica*, 10, São Paulo, SP, 1991. COBEM 91, v. 13, p. 607-610.

[Tsuzuki 91d] M. S. G. Tsuzuki, P. E. Miyagi, L. A. Moscato, "Automatic Recognition of Features", *International Conference on Computer Graphics an Visualization Techniques*, 1, SESIMBRA, Portugal, 1991. COMPUGRAPHICS 1991, SESIMBRA, Portugal, 1991, v. 2, p. 92-101.

[Tremblay 84] J.P. Tremblay, P.G. Sorenson, "An Introduction to Data Structures with Applications", *Mc Graw Hill*, second edition, 1984.

[Weiler 85] K. Weiler, "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments", *IEEE Computer Graphics & Applications*, 5(1):21-39, January 1985.

[Woo 85] T. C. Woo, "A Combinatorial Analysis of Boundary Data Structure Schemata", *IEEE Computer Graphics and Applications*, 5(3):19-27, March 1985.