Haroldo César Frota Bezerra

Um estudo comparativo entre algoritmos indutivos incrementais e não-incrementais utilizados na geração de bases de conhecimento a partir de exemplos

Campina Grande
1995

Haroldo César Frota Bezerra

Um estudo comparativo entre algoritmos indutivos incrementais e não-incrementais utilizados na geração de bases de conhecimento a partir de exemplos

Dissertação apresentada ao Curso de Mestrado em Informática da Universidade Federal da Paraíba, como requisito parcial à obtenção do título de Mestre em Informática.

Área de concentração: Inteligência Artificial

Orientador: Prof. Giuseppe Mongiovi
Universidade Federal da Paraíba

Campina Grande
Universidade Federal da Paraíba
1995

B574e Bezerra, Haroldo César Frota.
Um estudo comparativo entre algoritmos indutivos incrementais e não-incrementais utilizados na geração de bases de conhecimento a partir de exemplos / Haroldo César Frota Bezerra. - Campina Grande, 1995.
135 f.

Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia.

1. Inteligencia Artificial. 2. Aquisição do Conhecimento. 3. Métodos Indutivos. 4. Dissertacao - Informática. I. Mongiovi, Giuseppe. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 004.8(043)

UM ESTUDO COMPARATIVO ENTRE ALGORITMOS INDUTIVOS INCREMENTAIS E NÃO-INCREMENTAIS UTILIZADOS NA GERAÇÃO DE BASES DE CONHECIMENTO A PARTIR DE EXEMPLOS.

HAROLDO CESAR FROTA BEZERRA

DISSERTAÇÃO APROVADA EM 07/12/95

GIUSEPPE MONGIOVI, Dr.
Presidente

FERNANDO ANTONIO DE CARVALHO GOMES,Dr.
Componente da Banca

EDILSON FERNEDA, Dr.
Componente da Banca

Campina Grande, 07 de dezembro de 1995

Aos meus pais Prisco e Tacila.
(In memorian)

AGRADECIMENTOS

Gostaria de citar e agradecer ao Banco do Nordeste e a alguns dos muitos amigos que contribuiram para a concretização deste trabalho.

- Ao Prof. Giuseppe Mongiovi, meu orientador, pelo desprendimento, dedicação, competência, objetividade, paciência, boa vontade e amizade, que estiveram sempre presentes em todas as fases deste trabalho.

- À minha esposa Sheilla, pelo incentivo, ajuda, e paciência.

- Aos professores Tarcísio Pequeno, Mauro Pequeno, José Carlos e Laédio que me recomendaram à esta Universidade.

- Aos meus irmãos Roberto Cláudio e Tarquínio, e aos amigos do Banco do Nordeste, Laédio, Oliver, José Carlos, Gabriel, Gil e Edmilson, que me incentivaram e me ajudaram a realizar este curso de mestrado.

- A Reginaldo, amigo do BNB e parceiro em artigos, e a Washington, amigo e companheiro de trabalhos de Universidade, que em muito contribuiram com suas sugestões para melhoria desta dissertação.

- A todos os amigos do curso, principalmente Belo, Robson e Mario Ernesto, parceiros de trabalhos, e aos professores Hélio, Peter, Antão, Edilson e Bernardo, pelos incentivos e ensinamentos.

- Aos colegas do BNB, Jansen, Luis César, Eugênio, Arisio, Sandra, Luis Medeiros, Dulce, Gonzaga, Eduardo, Telma, Araripe e Bezerra, que em Fortaleza ou em Campina Grande sempre mostraram-se prestativos na solução de meus problemas.

LISTA DE FIGURAS

FIGURA 1.1 Arquitetura básica de um sistema especialista 5

FIGURA 2.1 Estratégia de processamento dos algoritmos não-incrementais 19

FIGURA 2.2 Estratégia de processamento dos algoritmos incrementais 20

FIGURA 2.3 Árvore de decisão gerada pelo ID3 para o domínio "padrão global" 24

FIGURA 2.4 Árvore após o processamento do primeiro exemplo 28

FIGURA 2.5 Árvore após o processamento do segundo exemplo 28

FIGURA 2.6 Árvore após o processamento do terceiro exemplo 29

FIGURA 2.7 Árvore após o processamento do quarto exemplo 29

FIGURA 2.8 Árvore após o processamento do quinto exemplo 30

FIGURA 2.9 Árvore após o processamento do sexto exemplo 30

FIGURA 2.10 Árvore após o processamento do sétimo exemplo 31

FIGURA 2.11 Árvore após o processamento do oitavo exemplo 31

FIGURA 2.12 Árvore de decisão utilizada para conceituar relevância topológica 35

LISTA DE TABELAS

TABELA 2.1 Conjunto de treinamento para o domínio "padrão global" 23

TABELA 3.1 Resumo das análises de custo dos algoritmos incrementais e não-incrementais 46

TABELA 4.1 Quantidade n de exemplos em domínios estáticos acima do qual o ID3 é mais eficiente que o ID5 59

TABELA 4.2 Quantidade N de exemplos em domínios dinâmicos (com b=4) acima do qual o ID3 é mais eficiente que o ID5 61

TABELA 4.3 Valores das parcelas componentes dos custos do ID5 68

LISTA DE GRÁFICOS

GRÁFICO 3.1 - Custo de processamento dos algoritmos ID3 e ID5 em função da quantidade ***n*** de instâncias, para domínios estáticos 48

GRÁFICO 3.2 - Número de instâncias **N** a partir do qual o ID3 é mais eficiente que o ID5 em função do tipo de domínio aqui representado por ***d*** e ***b***, para um fator $\alpha = 18$ 49

GRÁFICO 3.3 - Custo de um processamento dos algoritmos ID3 e ID5 em função da quantidade ***n*** de novas instâncias, para domínios dinâmicos após o processamento prévio de ***i*** instâncias 50

GRÁFICO 3.4 - Valor de **N** em função de ***i***, a partir do qual o ID3 é mais eficiente que o ID5 em domínios dinâmicos após o processamento de ***i*** instâncias 50

GRÁFICO 4.1 - Custo do ID3 e do ID5 no domínio estático (10 atributos e 4 valores por atributos) em função do número n de exemplos processados.....60

GRÁFICO 4.2 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=10, em função do número de exemplos processados 62

GRÁFICO 4.3 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=20, em função do número de exemplos processados 62

GRÁFICO 4.4 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=30, em função do número de exemplos processados.....63

GRÁFICO 4.5 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=40, em função do número de exemplos processados 63

GRÁFICO 4.6 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=50, em função do número de exemplos processados 63

GRÁFICO 4.7 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=60, em função do número de exemplos processados 64

GRÁFICO 4.8 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=70, em função do número de exemplos processados 64

GRÁFICO 4.9 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=80, em função do número de exemplos processados 64

GRÁFICO 4.10 - Custo do ID3 e ID5 para o domínio dinâmico (D104002) com i=90, em função do número de exemplos processados 65

GRÁFICO 4.11 - Comparativo dos custo dos algoritmos ID3 e ID5 em função do número de exemplos N, em um domínio com 20 atributos e 5 valores por atributos 70

GRÁFICO 4.12 - Segmento do GRAF. 4.11 com valores de n variando de 2 a 24 em maior escala 70

LISTA DE CÓDIGOS

PSEUDO-CÓDIGO 2.1 - Algoritmos TDIDT 22

PSEUDO-CÓDIGO 2.2 - O algoritmo ID3 22

PSEUDO-CÓDIGO 2.3 - O algoritmo ID4 25

PSEUDO-CÓDIGO 2.4 - O algoritmo ID5 26

PSEUDO-CÓDIGO 2.5 - A rotina Incorpore Ex. na árvore 26

PSEUDO-CÓDIGO 2.6 - A rotina Transforme árvore subindo atributo selecionado 27

PSEUDO-CÓDIGO 2.7 - O ID5R 32

PSEUDO-CÓDIGO 2.8 - O IDL 33

SUMÁRIO

RESUMO xv

1 INTRODUÇÃO

1.1 Inteligência artificial 1
1.2 Sistemas baseados em conhecimento e Engenharia do conhecimento 2
1.3 Sistemas especialistas 4
1.4 Métodos automáticos de aquisição do conhecimento 8
1.5 Aprendizado indutivo a partir de exemplos 13
1.6 Algoritmos indutivos utilizados para geração de bases de conhecimentos a partir de exemplos 13
1.7 Algoritmos Incrementais 14
1.8 Objetivos do trabalho 15
1.9 Organização da dissertação 16

2 OS ALGORITMOS INDUTIVOS INCREMENTAIS E NÃO-INCREMENTAIS

2.1 Intodução 18
2.2 Os algoritmos da família TDIDT 21
2.3 O ID3 22
2.4 O ID4 24
2.5 O ID5 25
2.6 O ID5R 32
2.7 O IDL 33
2.8 Conclusões 37

3 ANÁLISE COMPARATIVA TEÓRICA ENTRE ALGORITMOS INCREMENTAIS E NÃO INCREMENTAIS

3.1 Introdução 38

3.2 Comparação de desempenho de processamento entre algoritmos incrementais e não-incrementais na geração de árvores de decisão 39
3.3 Análise do algoritmo ID3 40
3.4 Análise do algoritmo ID3$^+$ 41
3.5 Análise do algoritmo ID5R 41
3.6 Análise do algoritmo ID5 44
3.7 Análise do algoritmo ID4 45
3.8 Análise do algoritmo IDL 45
3.9 Resumo da análise de desempenho dos algoritmos 45
3.10 Comparação de desempenho em domínios estáticos 47
3.11 Comparação de desempenho em domínios dinâmicos 49
3.12 Conclusões 52

4 ANÁLISE COMPARATIVA PRÁTICA ENTRE ALGORITMOS INCREMENTAIS E NÃO-INCREMENTAIS

4.1 Introdução 54
4.2 Criação de domínios para análise 54
4.3 Forma de obtenção dos valores dos custos de execução dos algoritmos.....55
4.4 Resultados encontrados 58
4.4.1. Domínios estáticos 58
4.4.2. Domínios dinâmicos 60
4.5 Comportamento do custo em função das métricas definidas 66
4.5.1 Componentes do custo do ID5 67
4.6 Conclusões 71

5 CONCLUSÕES E FUTUROS TRABALHOS

5.1 Conclusões 74
5.2 Futuros trabalhos 77

APÊNDICE A - Resultados da análise experimental em domínios estáticos.....79

APÊNDICE B - Resultados da análise experimental em domínios dinâmicos.....93

APÊNDICE C - Código fonte do algoritmo ID5 utilizado na análise experimental.....117

Referências Bibliográficas.....130

RESUMO

A fase crítica do processo de construção de sistemas especialistas é a responsável pela geração da base de conhecimento, porque ela é cara, difícil e de grande influência sobre a qualidade final desses sistemas.

A indução a partir de exemplos, gera bases de conhecimento em forma de árvores de decisão ou de regras, através de algoritmos chamados de algoritmos indutivos.

Algoritmos indutivos podem ser do tipo não-incremental ou incremental.

Algoritmos não-incrementais recebem como entrada um conjunto de exemplos e geram como saída uma base de conhecimento.

Algoritmos incrementais recebem como entrada uma base de conhecimento e um conjunto de novos exemplos e geram como saída uma base de conhecimento atualizada.

Neste trabalho apresentamos uma análise comparativa de desempenho entre essas duas famílias de algoritmos geradores de árvore de decisão.

Mostramos através de análise comparativa teórica e experimental que, ao contrário do que os autores dos algoritmos incrementais disseram sobre esses algoritmos, somente em raras situações de domínios e de quantidade de exemplos esses algoritmos mostraram-se mais eficientes que os não-incrementais, não atingindo assim os objetivos a que se propunham.

1 Introdução

1.1 Inteligência artificial

A Inteligência Artificial é uma ciência relativamente recente e seu nome surgiu em meados da década de 50 e foi dado por John MacCarthy numa conferência, realizada em Darmouth, que estudava o uso do computador na simulação de aspectos da inteligência humana.

Ela interage com a lingüística quando do estudo de interfaces de sistemas com seres humanos, com a filosofia e a psicologia quando do estudo de aspectos da inteligência, do raciocínio, do ensino e da aprendizagem, com a engenharia e a ciência da computação quando da concepção de máquinas ou computadores inteligentes. Isto provoca o aparecimento de pesquisadores com os mais diversos perfis, cada um deles procurando dar uma definição que espelhe melhor as metas e métodos de suas pesquisas.

Apresentar uma definição que seja aceita por todos os estudiosos da área e que abranja os seus diversos enfoques não tem sido uma tarefa fácil. Não há uma definição que seja aceita como um consenso por todos os cientistas da área.

Para exemplificar citamos algumas dessas definições:

. Patrick H. Winston [Winston 84].

> *"Inteligencia Artificial é o estudo das idéias que permitem aos computadores tornarem-se inteligentes."*

. Eugene Charniak [Charniak 85].

> *"Inteligencia Artificial é o estudo das faculdades mentais através do uso do computador."*

. Peter Bock [Bock 85].

> *"Inteligencia Artificial é a capacidade que uma máquina-humana (autômata) possui para emular ou simular métodos humanos para a aquisição e aplicação do conhecimento e raciocínio através da dedução e indução."*

Finalmente uma definição que é creditada ao professor Marvin Misky do MIT (Massachusetts Institute of Tecnology), e apresentada em [Boden 77].

> *"Inteligência Artificial é a ciência que constrói máquinas que executam atividades que exigem inteligência se forem feitas pelo homem."*

Verificamos nessas definições a presença das duas escolas de pensamento identificadas por Firebaugh [Firebaugh 89]. Uma escola que considera a "*mente como objeto e a máquina como ferramenta*" (segunda definição) e a outra que considera o "*computador como objeto e a mente como modelo*" (primeira, terceira e quarta definições).

Dentro dessas duas escolas de pensamento, a segunda, que considera *a máquina como objeto e a mente como modelo*, tem apresentado resultados de maiores sucessos. Esses resultados são conseqüência principalmente de trabalhos na área da robótica e da engenharia do conhecimento.

1.2 Sistemas baseados em conhecimento e engenharia do conhecimento

Nas décadas de 60 e 70 as pesquisas na área de inteligência artificial direcionavam-se para a busca de métodos gerais de desenvolvimento de um resolvedor único de problemas (GPS - *General Problem Solver*). O insucesso na busca desse método geral redirecionou as pesquisas no sentido de se buscar métodos específicos de solução de problemas

[Jackson 86]. Percebeu-se, que somente um grande poder de inferência não seria suficiente para o desenvolvimento de um sistema inteligente que apresentasse alguma real utilidade. A qualidade e a quantidade do conhecimento tratado pelo sistema era tão ou mais importante que seu poder de inferência.

Dessa constatação, surgiu a necessidade de se explicitar e separar o conhecimento do código que manipula esse conhecimento, i.e., que realiza inferência sobre ele. Começaram a ser desenvolvidos então sistemas com essas características, e por isso foram chamados de *sistemas baseados em conhecimento*.

O processo de desenvolvimento dos *sistemas baseados em conhecimento*, com particular ênfase a aquisição do conhecimento humano para incorporá-lo aos sistemas computadorizados, foi definido no início da década de 70 como *Engenharia do Conhecimento*. Os técnicos responsáveis pela captação desse conhecimento e posterior elaboração desses sistemas foram batizados de *Engenheiros do Conhecimento*.

Com o surgimento dos primeiros sistemas baseados em conhecimento pensou-se na possibilidade da construção de um sistema que simulasse o comportamento de um ou de vários especialistas humanos. Com uma boa máquina de inferência e uma grande base de conhecimento, poderia-se construir um sistema generalista que soubesse tudo sobre qualquer domínio. Esta tentativa também não teve sucesso, pois o tamanho desta base de conhecimento e o espaço de busca a ser trabalhado pela máquina de inferência teriam dimensões inviáveis de serem tratadas. Além disso, a obtenção de uma base de conhecimento com tais característica era praticamente impossível.

Assim, passou-se então a construir sistemas baseados em conhecimento para resolver problemas específicos em domínios menores e bem delineados. As áreas inicialmente abordadas foram as da medicina, geologia, agronomia e financeira.

A esses sistemas, por se restringirem a uma estreita área de atuação mas tratada com profundidade, passou-se a dar o nome de *sistemas especialistas*.

1.3 Sistemas especialistas

Dentre as pesquisas da área de Inteligência Artificial, aquelas relacionadas com sistemas especialistas têm sido as principais responsáveis pelo crescimento da área. Este crescimento é creditado à motivação dos cientistas obtida em conseqüência da grande credibilidade junto à comunidade usuária desses sistemas, pois eles têm conseguido viabilizar e justificar a solução de variados problemas reais.

Michaelsen apresenta uma definição bastante abrangente para sistemas especialistas que é citada em [Firebaugh 89]:

> *"Sistemas Especialistas são sistemas de computação que podem aconselhar, analisar, classificar, comunicar, consultar, projetar, diagnosticar, explicar, investigar, prever, formar conceitos, identificar, interpretar, justificar, aprender, gerenciar, monitorar, planejar, apresentar, recuperar, escalonar, testar e ensinar. Eles tratam de problemas que normalmente seria necessário o uso de especialistas humanos na sua solução."*

É bastante visível que esta definição não pode ser considerada como uma definição restrita. Ela dá ênfase à abrangência que os sistemas especialistas têm demonstrado, embora nenhum destes sistemas em particular tenha conseguido apresentar todas essas características.

Podemos considerar a arquitetura de um sistema especialista como sendo formada por três principais módulos:

- Base de conhecimento.
- Sistema de controle, composto de:
  - máquina de inferência
  - módulo de explanação.
- Interface com o usuário.

Uma boa representação gráfica dessa arquitetura foi apresentada em [Alexandre 94] e a reproduzimos na FIG. 1.1:

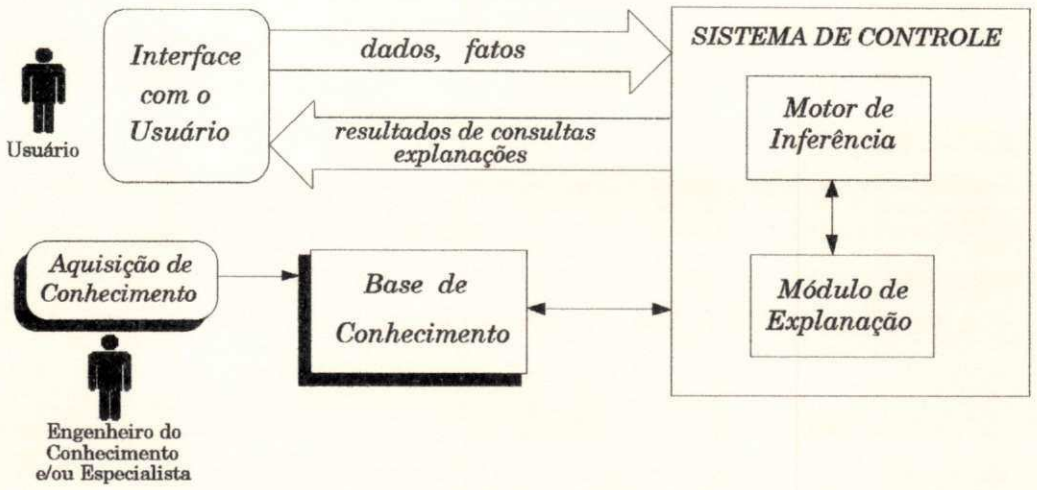

FIGURA 1.1 Arquitetura básica de um sistema especialista.

Da mesma maneira que nos sistemas tradicionais, podemos dividir o processo de construção de sistemas especialistas em várias etapas ou fases. Karen L. McGraw sugere dividir esse processo nas seguintes fases [MacGraw 89] :

- Identificação das características do problema
- Conceituação do domínio
- Organização e formalização do conhecimento
- Implementação
- Testes
- Validação e
- Manutenção

Nos sistemas tradicionais, as maiores dificuldades estão concentradas nas fases iniciais de captação das necessidades dos usuários. Elas irão nortear as fases seguintes e contribuir diretamente para a qualidade final do sistema. As pesquisas em engenharia de software para sistemas tradicionais têm constantemente procurado metodologias e técnicas visando melhorar a captação das reais necessidades dos usuários objetivando a produção de sistemas com menor custo e maior qualidade.

Da mesma forma, em sistemas especialistas, as fases iniciais responsáveis pela criação da base de conhecimento, são as fases críticas no processo de construção de um sistema especialista. Essas fases iniciais tornam-se críticas em função da grande dificuldade do processo de aquisição de conhecimento, de seu alto custo e principalmente

porque a experiência tem mostrado a sua grande e direta influência na qualidade final do sistema.

Essa dificuldade, que se transformou num verdadeiro ponto de estrangulamento do processo de elaboração de um sistema especialista, é conhecida da comunidade científica da área como *"The Feigenbaum bottleneck"*, por ter sido identificada por Edward Feigebaum [Feigebaum 81].

Vários são os métodos que têm sido usados no processo *de aquisição de conhecimento*. Podemos classificar esses métodos em dois grandes grupos [Boy 87]:

- Métodos cognitivos.
- Métodos automatizados.

Os *métodos cognitivos* baseiam-se em entrevistas feitas pelo engenheiro do conhecimento junto ao especialista. Esses métodos têm muita semelhança com as técnicas de levantamento de informações utilizadas pelos analistas de sistemas tradicionais nas fases de anteprojeto e de projeto lógico. Devido às dificuldades de comunicação entre o engenheiro de conhecimento e o especialista, provocadas frequentemente pelo pouco conhecimento do domínio por parte do engenheiro do conhecimento, e pela pouca disponibilidade de tempo do especialista, esses métodos, em geral, tornam-se bastante lentos. Por outro lado um bom especialista normalmente é raro e consequentemente caro o que torna também esses métodos muito dispendiosos.

A experiência tem mostrado que um engenheiro de conhecimento, que domina bem as técnicas de entrevistas, com um bom conhecimento do domínio e que mantenha um bom relacionamento com o especialista, consegue acrescentar à base de conhecimento, em média, apenas de uma a quatro regras por dia [Shapiro 87].

Essas dificuldades inerentes aos métodos cognitivos têm orientado os esforços das pesquisas para os *métodos automatizados* que procuram eliminar essas dificuldades, visando diminuir o tempo gasto na geração da base de conhecimento e como conseqüência seus custos, sem entretanto comprometer sua qualidade.

John Boose [Boose 90] classifica os *métodos automatizados* em duas categorias:

- métodos semi-automáticos (baseados em entrevistas).
- métodos automáticos (baseados em aprendizagem automática).

Os métodos *semi-automáticos* procuraram contornar as dificuldades dos métodos cognitivos através da tentativa de automatização das entrevistas, objetivando, dessa maneira, diminuir a necessidade da presença do engenheiro do conhecimento. Com esse método, o especialista interage diretamente com a máquina através de entrevistas conduzidas pelo computador.

Alguns sistemas como o ETS (Expertise Transfer System) [Boose 84], AQUINAS [Boose 87], SALT [Marcus 87], MOLE [Eshelman 88], KRITON [Diederich 88], KITTEN [Shaw 88], utilizando métodos semi-automáticos foram implementados e considerados como uma evolução em relação aos métodos cognitivos. Entretanto, a utilização desses sistemas mostrou ainda graves deficiências. Várias dessas deficiências são citadas em [Mongiovi 95], entre elas podemos destacar:

- As entrevistas podem ficar muito longas e cansativas.
- A maioria desses sistemas só conseguem gerar regras atômicas.
- Esses sitemas são centrados quase que exclusivamente no especialista, utilizando muito pouco ou quase nada as experiências anteriores.

Outra abordagem de automatização de aquisição do conhecimento surgiu, desta vez mais abrangente, procurando levar em consideração todas as formas possíveis pelas quais uma máquina possa captar conhecimento do ambiente. Com esta abordagem têm surgido muitos trabalhos de pesquisa e essa área foi denominada de *aprendizagem automática* ou "*Machine Learning*" [Michalski 86].

Quando utilizamos técnicas de *aprendizagem automática* para a obtenção de conhecimento e esse conhecimento poder ser representado de uma forma explícita, temos um *método automático de aquisição de conhecimento* [Cirne 91].

1.4 Métodos automáticos de aquisição de conhecimento

Um método de aquisição de conhecimento é dito automático quando capta conhecimento, de uma forma automática, a partir do ambiente que o envolve e o representa, de uma forma explícita, através de uma base de conhecimento [Cirne 91].

Antes de descrevermos os principais paradigmas da aprendizagem automática, vamos descrever, de forma resumida, os mecanismos básicos de inferência lógica, visto que esses paradigmas utilizam regras de interferência lógica para viabilizar a automação da aquisição de conhecimento.

Mecanismos básicos de inferência lógica

Os *métodos de aprendizagem automática* utilizam alguns mecanismos básicos de *inferência lógica*. As três regras de inferência lógica mais utilizadas são:

- Dedução
- Indução
- Abdução

A **dedução** consiste em se chegar a uma conclusão a partir de premissas previamente colocadas. Ela se caracteriza por preservar a verdade e é uma inferência do geral para o particular.

O exemplo a seguir esclarece melhor esse conceito:

> *" Todo algoritmo indutivo que gera ávore de decisão como base de conhecimento, apresenta o problema sintático".*
>
> *" O ID3 é um algoritmo indutivo que gera árvore de decisão".*

Então podemos inferir que:

> *"O ID3 apresenta o problema sintático".*

Em linguagem da lógica das proposições temos:

Se $\mathcal{A}$ é verdade, e $\mathcal{A}$ implica $\mathcal{B}$ é verdade, então $\mathcal{B}$ é verdade.

$$\frac{\mathcal{A}, \mathcal{A} \rightarrow \mathcal{B}}{\mathcal{B}}$$

A **indução** chega à conclusão a respeito de todos os elementos de um conjunto a partir de observações feitas apenas sobre alguns elementos desse conjunto. Esta conclusão poderá ou não ser verdadeira. É uma inferência do particular para o geral.

Para esclarecer esse conceito, consideremos o seguinte exemplo:

Sejam as afirmações:

" O ID3 é um algoritmo indutivo que gera ávore de decisão".
" O ID4 é um algoritmo indutivo que gera ávore de decisão".
" O ID5 é um algoritmo indutivo que gera ávore de decisão".

Então podemos inferir que:

" Todo algoritmo indutivo gera árvore de decisão".

O resultado da indução pode ser verdadeiro ou não. Neste exemplo apresentado, o resultado da indução não é verdadeiro, pois temos algoritmos indutivos que não geram árvores de decisão.

Em linguagem da lógica das proposições temos:

Se $\mathcal{A}$ é verdade, e $\mathcal{B}$ é verdade, então $\mathcal{A}$ implica $\mathcal{B}$.

$$\frac{\mathcal{A}, \mathcal{B}}{\mathcal{A} \rightarrow \mathcal{B}}$$

A **abdução** é uma forma de se chegar a uma inferência plausível ou uma conclusão apenas aceitável, que é consistente com as informações disponíveis mas que pode ou não ser verdadeira.

O exemplo a seguir esclarece melhor essa idéia:

" Todo professor deveria ser melhor remunerado"

"O Haroldo deveria ser melhor remunerado"

Então podemos inferir:

"O Haroldo é professor"

A afirmação final poderá ou não ser verdadeira, aqui neste exemplo, da mesma forma que no exemplo anterior de indução, não é verdadeira.

Em linguagem da lógica das proposições, temos:

Se $\mathcal{A}$ implica $\mathcal{B}$ é verdade, e $\mathcal{B}$ é verdade, então $\mathcal{A}$ é verdade.

$$\frac{\mathcal{A} \rightarrow \mathcal{B}, \mathcal{B}}{\mathcal{A}}$$

A abdução, da mesma forma que a indução, não preserva a verdade, mas quando utilizada associada à técnicas estatísticas, consegue identificar a inferência mais provável entre as várias possíveis.

Paradigmas de aprendizagem automática

Jaime Carbonel [Carbonell 89] classifica os métodos de aprendizagem automática em quatro principais paradigmas:

- Conexionista
- Genético
- Analítico
- Indutivo

O paradigma **conexionista** foi inspirado no funcionamento do cérebro humano. É composto de modelos baseados em redes neurais artificiais, que tentam imitar a maneira de funcionamento do cérebro humano. Os modelos conexionistas são também chamados de redes neurais artificiais (RNA), redes neurais ou sistemas de processamento paralelo distribuidos. O aprendizado é realizado através de ajuste de pesos em uma rede neural.

A abordagem conexionista é criticada por não conseguir explicitar a linha de raciocínio utilizada na solução de problemas [Mongiovi 95], mas apresenta bons resultados em domínios com uma grande massa de dados, e é bastante útil para sistemas evolutivos.

O paradigma **genético** foi inspirado na teoria da evolução e seleção natural de Darwin [Goldberg 89]. É composto de algoritmos chamados genéticos que partem de uma população de indivíduos para produzir uma população melhor adaptada ao meio. Os indivíduos correspondem às variações das descrições de conceitos de uma mesma espécie, e combinações destes indivíduos são testadas contra uma função objetivo (critério de seleção). A qualidade dos resultados deste paradigma depende diretamente de um grande número inicial da população e é utilizado convenientemente apenas quando o espaço de hipóteses é muito grande, não estruturado e quando o custo do teste das hipóteses é baixo [Mongiovi 95].

O paradigma **analítico** é baseado em aprendizagem a partir de poucos exemplos e um substancial conhecimento do domínio em questão. Utiliza experiência de problemas passados para guiar quais cadeias dedutivas percorrer quando da resolução de novos problemas, ou para fornecer regras de controle que tornem mais eficientes as pesquisas do domínio. Segundo Engelbert Nguifo [Nguifo 93], os métodos que se utilizam deste paradigma podem ser classificados em:

- *por analogia ou baseados em casos* (CBL:Case-Based Learning)
- *baseados em explanações* (EBL:Explaination-Based Learning)

Os *métodos baseados em casos* se baseiam na transferência de conceitos de um objeto (base) para outro (meta), a partir da informação de que eles são semelhantes. Esses métodos utilizam os três mecanismos básicos de inferência: a dedução, a indução e a abdução.

Os *métodos baseados em explanações* utilizam uma aprendizagem onde a generalização é justificada por explicações. Ela consiste de uma fase de explicação (utilizando o mecanismo de dedução), seguida algumas vezes de uma fase de generalização (utilizando o mecanismo de indução) que permite atualizar o conhecimento até então aprendido.

O paradigma **indutivo** procura induzir uma descrição geral de um conceito a partir da descrição de um conjunto de exemplos e contra-exemplos desse conceito. O objetivo é construir uma descrição geral do conceito no qual todas as instâncias podem ser rederivadas por instanciação mas nenhuma das instâncias negativas (contra-exemplos) podem ser derivadas pelo mesmo processo. Este paradigma é particularmente indicado para se adquirir novas descrições de conceitos e regras para sistemas especialistas [Vasco 93].

O paradigma indutivo por sua vez pode ser classificado em duas subclasses:

- Aprendizagem indutiva a partir de observações.
- Aprendizagem indutiva a partir de exemplos.

A *aprendizagem indutiva a partir de observações* é também conhecida como *generalização descritiva.* Seu objetivo é encontrar uma descrição geral que caracterize uma coleção de observações, i.é, criar descrições especificando as propriedades dos objetos pertencentes a uma determinada classe.

A *aprendizagem indutiva a partir de exemplos* é também conhecida como *aquisição de conceitos.* Nela os exemplos observados são caracterizações de situações, processos, etc, pré-classificados em uma ou mais classes ou conceitos. Cada hipótese induzida pode ser vista como um conjunto de regras de reconhecimento de um conceito, tal que se um objeto satisfaz uma dessas regras, então ele representa este conceito.

Portanto, enquanto a indução a partir de exemplos (*aquisição de conceitos*) induz hipóteses para classificar objetos em classes, a indução a partir de observações (*generalização descritiva*) produz descrição que especifica propriedades de objetos pertencente a uma certa classe.

1.5 Aprendizagem indutiva a partir de exemplos

Na aprendizagem indutiva a partir de exemplos pode-se usar dois tipos de generalização:

- generalização parte-global.
- generalização instância-classe.

Com a *generalização parte-global* o objetivo é conceituar um objeto a partir de exemplos que representam partes do objeto.

Com a *generalização instância-classe,* o objetivo é fazer uma descrição geral de uma classe quando informamos uma coleção de instâncias da classe.

Nos restringiremos, em nosso trabalho, à *aprendizagem automática indutiva a partir de exemplos utilizando generalização instância-classe,* e a chamaremos de agora em diante apenas como aprendizagem indutiva a partir de exemplos.

1.6 Algoritmos indutivos utilizados para geração de bases de conhecimento a partir de exemplos

Diversos algoritmos foram desenvolvidos utilizando indução a partir de exemplos para automatizar o processo de aquisição de conhecimento. Esses algoritmos geram bases de conhecimento representadas na forma de árvores de decisão ou diretamente na forma de regras.

Os algoritmos que geram árvore de decisão são os mais conhecidos e mais comumente utilizados. São normalmente os algoritmos membros da família TDIDT (*Top Down Induction of Decision Tree*) [Quinlan 86]. Fazem parte dessa família de algoritmos o ID3 [Quinlan 83], o C4 [Quinlan 87], o CART [Breiman 84], entre outros.

Podemos classificar os algoritmos indutivos com relação a estratégia de construção da base de conhecimento, em:

- algoritmos incrementais.
- algoritmos não-incrementais.

1.7 Algoritmos Incrementais e não-incrementais

O conceito de incrementabilidade desses algoritmos vem do fato deles atualizarem a base de conhecimento (incrementando conhecimento à base de conhecimento existente) a cada novo exemplo processado.

Para melhor entendermos a distinção entre os algoritmos incrementais e os não-incrementais, consideremos um domínio em que temos uma base de conhecimento anteriormente gerada através de um conjunto inicial de exemplos, e novas instâncias de exemplos estão disponíveis para processamento.

Com um *algoritmo não-incremental*, para incorporarmos o efeito desses novos exemplos à base de conhecimento, teríamos que desprezar toda a base de conhecimento já adquirida e darmos como entrada para o algoritmo generalizador a união do conjunto inicial de exemplos com o conjunto das novas instâncias de exemplos.

Com um *algoritmo incremental,* usaríamos como entrada para o algoritmo generalizador apenas o conjunto de novas instâncias e a base anteriormente gerada. Ele atualiza a base de conhecimento a cada instância de novo exemplo que é processado.

As FIG.2.1. e FIG.2.2 do capítulo 2, ilustram bem essas duas filosofias de processamento.

É vísivel que, para esta situação, a estratégia utilizada pelos algoritmos incrementais é em princípio mais eficiente, pois não há necessidade de esforço adicional (reprocessamento do conjunto de exemplos iniciais) para reaprender o conhecimento já adquirido (base de conhecimento anteriormente gerada). Essa, é a nosso ver, a principal motivação para o surgimento dos *algoritmos incrementais.*

Portanto, para os domínios dinâmicos[1] os *algoritmos incrementais* surgiram para dar maior eficiência, "mais rapidez", ao processo de geração automática de bases de conhecimento.

---

[1] Domínios dinâmicos são domínios que apresentam uma disponibilidade contínua de novos conjuntos de treinamento.

Os *algoritmos incrementais*, da mesma forma que os *não-incrementais*, podem gerar bases de conhecimento representadas através de árvore de decisão ou de regras, sendo que os mais conhecidos e usados são os das árvores de decisão.

1.8 Objetivos do trabalho

Como vimos no item anterior, os algoritmos incrementais surgiram como uma evolução dos algoritmos não-incrementais no sentido de melhoria de desempenho de processamento.

Geralmente, um algoritmo incremental se baseia em algum outro algoritmo não-incremental e procura chegar ao mesmo resultado que chegaria o não-incremental através de um processo que seria mais eficiente. Um exemplo disso são o ID4 [Schlimmer 86], ID5 [Utgoff 88], ID5R [Utgoff 89], IDL [Van de Velde 89] que se baseiam no ID3 [Quinlan 83] e geram árvore de decisão, e o AQ15 [Michalski 86b] que se baseia nos algoritmos não-incrementais da família dos AQs (AQ7-AQ11) [Michalski 83] e gera regras.

Alguns desses algoritmos incrementais vão um pouco além e incorporam outras melhorias em relação ao algoritmo no qual ele foi inspirado. Este é, por exemplo, o caso do IDL que incorpora um critério adicional (relevância topológica), não utilizado no ID3, para escolha de um atributo para o nó, visando possibilitar uma eventual diminuição do tamanho da árvore através de poda. Estas melhorias entretanto não são inerentes à incrementabilidade, e poderiam ser incorporadas também aos algoritmos não incrementais.

Quando os autores desses algoritmos incrementais apresentaram seus trabalhos à comunidade científica, procuraram justificar sua utilidade através de comparações com seus antecessores. Geralmente essas comparações são feitas em relação ao desempenho de processamento (custo) e em relação ao resultado final do algoritmo (base de conhecimento).

Estudando alguns desses algoritmos e analisando o resultado dessas comparações, principalmente em relação ao desempenho, verificamos que elas eram muito simplistas pois:

- As análises levavam em consideração apenas o pior caso;
- Mesmo considerando o pior caso, não levavam em consideração as diversas situações do domínio;
- As métricas utilizadas não nos pareciam as mais adequadas;
- Os autores se mostravam muito parciais na comparação dos resultados de seus algoritmos.

Com base nessas observações preliminares, resolvemos direcionar nosso trabalho visando fazer uma comparação mais aprofundada entre esses algoritmos, com o objetivo de procurar identificar mais corretamente sua melhor aplicabilidade. Ou seja, até que ponto os algoritmos incrementais são uma solução? Ou será que são um problema? O objetivo maior deste trabalho é procurar uma resposta para essas indagações.

Por serem mais frequentemente utilizados, por serem mais citados na literatura, e por uma questão de amplitude do trabalho, restringiremos o escopo deste estudo aos algoritmos incrementais que geram árvore de decisão, deixando a outra família de algoritmos (geradores de regras), basicamente representada pelo AQ15, para futuros trabalhos.

1.9 Organização da dissertação

Visando documentar nosso trabalho, organizamos esta dissertação em capítulos que a seguir descrevemos:

Capítulo-2 - *Os algoritmos indutivos incrementais e não-incrementais*, onde descreveremos os principais algoritmos indutivos incrementais utilizados para geração de bases de conhecimento representadas através de árvore de decisão.

Capítulo-3 - *Análise comparativa teórica entre algoritmos incrementais e não-incrementais*, onde compararemos os custos de processamento dos principais algoritmos incrementais com o algoritmo ID3, escolhido para representar os não-incrementais, com base em métricas e complexidades apresentadas por seus autores.

Capítulo-4 - *Análise comparativa experimental entre algoritmos incrementais e não-incrementais*, onde compararemos os custos de processamento do algoritmo ID5 (escolhido para representar os incrementais por razões que serão mostradas no capítulo 3) com o ID3 (escolhido para representar os não-incrementais), e faremos uma análise crítica das métricas de custo utilizadas pelos autores dos algoritmos incrementais.

Capítulo-5 - *Conclusões e futuros trabalhos*, onde mostraremos em que situações os algoritmos incrementais, aqui analisados, são mais eficientes que os não-incrementais, e relacionaremos alguns trabalhos possíveis de serem desenvolvidos visando dar continuidade aos estudos aqui apresentados.

Apêndice A - *Resultados da análise experimental em domínios estáticos*, onde mostraremos os valores dos custos dos algoritmos ID3 e ID5 resultantes dos diversos experimentos realizados com domínios estáticos.

Apêndice B - *Resultados da análise experimental em domínios dinâmicos*, onde mostraremos os valores dos custos dos algoritmos ID3 e ID5 resultantes dos diversos experimentos realizados com domínios dinâmicos.

Apêndice C - *Código fonte do algoritmo ID5 utilizado na análise experimental.*

2 Os algoritmos indutivos incrementais e não-incrementais

2.1 Introdução

Um dos fatores do sucesso do processo de aquisição automática de conhecimento é a sua capacidade de gerar conhecimento de fácil entendimento e utilização por parte do ser humano [Michalski 86a].

A indução a partir de exemplos é um dos métodos de aquisição automática de conhecimento que consiste em inferir uma descrição geral de um certo conceito a partir de um conjunto de exemplos e contra-exemplos desse conceito, denominado conjunto de treinamento, extraído de um especialista ou do mundo real [Firebaugh 89].

A árvore de decisão é uma das formas de representação do conhecimento gerada pelos métodos indutivos, em que seus nós não-terminais representam as características mais relevantes de um domínio (os atributos), os ramos denotam os valores assumidos por esses atributos, e as folhas representam os elementos de classificação (os conceitos).

Alguns domínios que se utilizam da aprendizagem automática de conhecimento dispõem apenas de um único conjunto de treinamento fixo (não mutável) e a estes damos a denominação de *domínios estáticos*. Outros apresentam uma disponibilidade contínua de novos conjuntos de treinamento e, por isso, os denominamos de *domínios dinâmicos*.

Se gerarmos inicialmente uma base de conhecimento (BC) a partir de um conjunto de treinamento (CT), e posteriormente necessitarmos enriquecer essa base de conhecimento, através da incorporação de novos exemplos adquiridos (ΔCT), deparamo-nos com duas possibilidades de solução:

- A primeira, seria unir o conjunto de treinamento anteriormente processado (CT) com o novo conjunto de treinamento (ΔCT), e passar o resultado desta união como entrada para o algoritmo gerar nova base de conhecimento;

- A segunda, seria passar como entrada para o algoritmo somente o novo conjunto de treinamento (ΔCT) e a base de conhecimento anteriormente gerada (BC), para que o algoritmo atualize esta base a cada instância do novo conjunto de exemplos.

Na primeira solução, todo o conhecimento anterior (base de conhecimento) já adquirido pelo sistema é desprezado e uma nova base de conhecimento é então criada.

Com a segunda alternativa de solução, o conhecimento anterior é "conservado" e "incrementado" em função do novo conjunto de treinamento (ΔCT).

Percebe-se, intuitivamente, que a segunda alternativa é mais racional e mais natural que a primeira, pois não há esforço adicional e desnecessário para reaprender o conhecimento já adquirido.

Os algoritmos que se utilizam da segunda alternativa de solução são chamados de *algoritmos incrementais* e os que se utilizam da primeira alternativa são classificados como *não-incrementais*.

As FIG.2.1 e FIG.2.2 ilustram a estratégia de processamento dessas duas classes de algoritmos.

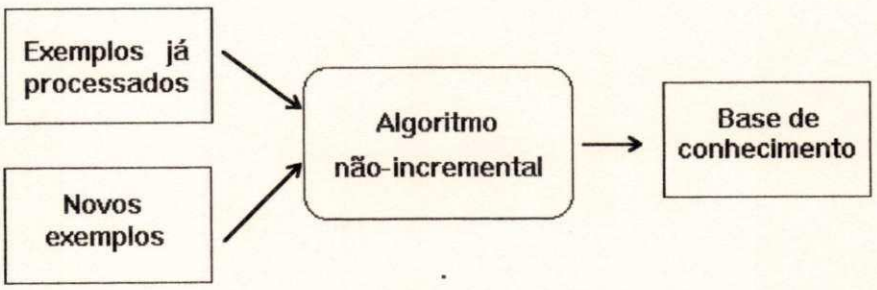

FIGURA 2.1 Estratégia de processamento dos algoritmos não-incrementais

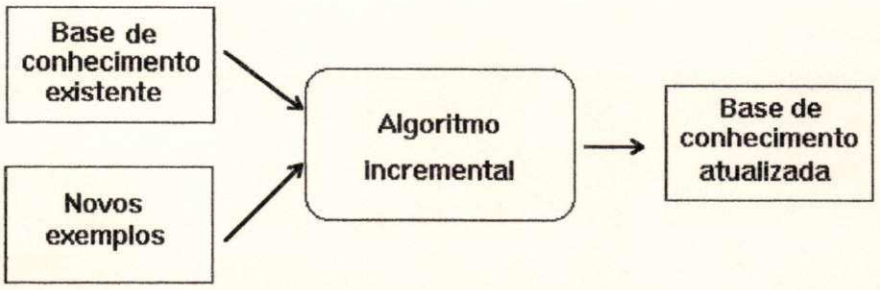

FIGURA 2.2 Estratégia de processamento dos algoritmos incrementais

Os primeiros algoritmos indutivos surgidos, ID3 [Quinlan 83], o C4 [Quinlan 87], CART [Breiman 84], AQ7 e AQ11 [Michalski 83], foram concebidos dentro da filosofia apresentada nessa primeira alternativa de solução, e portanto são ditos *não-incrementais*.

A partir do surgimento desses primeiros algoritmos começaram a surgir novos algoritmos que se utilizavam do conhecimento já adquirido e de novos exemplos para a geração da nova base de conhecimento, i.é., procuravam um modelo de solução (segunda alternativa FIG.2.2) mais natural e mais racional.

Ryszard Michalski, chega, inclusive, a afirmar que sistemas de aprendizagem que utilizam um grande e diversificado número de observações pode ser computacionalmente inviável em sistemas não incrementais (sic!) [Michalski 85].

Vários são os algoritmos não-incrementais que geram suas bases de conhecimento representadas por árvore de decisão. Os mais difundidos são os da família ID3, ou seja oriundos do algoritmo ID3.

Dentre os algoritmos incrementais dessa família, podemos destacar o ID4 [Schlimmer 86], o ID5 [Utgoff 88], o ID5R [Utgoff 89], que se baseiam no ID3 e têm como meta gerar a mesma árvore que o ID3 geraria para o mesmo conjunto de treinamento (CT+ΔCT). Destacamos também o IDL [Van de Velde 89], que se baseia no ID5R mas oferece adicionalmente uma técnica de transformação baseada em critérios

topológicos, criando oportunidades de poda, possibilitando assim, a geração de árvores de menor tamanho.

O AQ15 é um algoritmo incremental descendente da série de algoritmos AQ7-AQ11, que gera suas bases de conhecimento diretamente na forma de regras. Nosso trabalho procurará fazer uma análise comparativa entre os algoritmos que geram árvore de decisão e, portanto, pertencentes à família do ID3.

Neste capítulo descrevemos os algoritmos ID4, ID5, ID5R e o IDL como os algoritmos incrementais que serão analisados e o ID3 como representante dos não-incrementais.

2.2 Os algoritmos da família TDIDT

A árvore de decisão tem sido uma das formas de representação do conhecimento mais amplamente utilizadas pelos métodos indutivos automáticos de aquisição do conhecimento. Os algoritmos utilizados para isso, têm como objetivo mapear um conjunto de exemplos em uma árvore de decisão de tamanho mínimo (largura e altura). Os principais algoritmos que seguem essa filosofia, constroem a árvore de decisão a partir da raiz, descendo até as folhas, sendo por isto classificados como pertencentes à família TDIDT (Top Down Induction of Decision Trees) [Quinlan 86].

O processo de construção da árvore nesses algoritmos inicia-se colocando no nó raiz o atributo selecionado a partir de uma função de avaliação aplicada ao conjunto de exemplos. As funções de avaliação são normalmente funções estatísticas, e dentre elas a mais comumente utilizada é o cálculo da entropia [Klir 88]. O conjunto de treinamento é então subdividido em subconjuntos (um para cada valor do atributo selecionado) e arcos são ligados ao nó (tantos quanto forem esses subconjuntos). O processo continua recursivamente para cada subconjunto até que uma condição de parada seja satisfeita. Neste ponto o elemento de classificação presente no subconjunto é devolvido para o ponto de chamada recursiva ou o algoritmo se encerra.

Um pseudo-código, representativo da família TDIDT poderia ser:

```
Entrada: CT - Conjunto de Treinamento
Saída:   AD - Árvore de Decisão

TDIDT ( CT )
      Crie uma AD vazia
      Se os exemplos possuem o mesmo elemento de classificação E
         Coloque o elemento de classificação E na raiz de AD
         Retorne AD
      Senão
         Calcule o valor da função de avaliação para cada atributo
                 presente em CT
         Selecione o atributo que apresentou o melhor valor
         Coloque o atributo selecionado na raiz de AD
         Para cada valor do atributo selecionado
            Crie um ramo em AD associado ao valor
            Crie um subconjunto CT onde só ocorra o par atributo=valor
            TDIDT ( subconjunto )
            Retire de CT todos os exemplos mapeados pelo ramo construído
         fimPara
fimTDIDT
```

PSEUDO-CÓDIGO 2.1 - Algoritmos TDIDT

2.3 O ID3

O ID3 constrói uma árvore de decisão escolhendo para a raiz um melhor atributo-teste e dividindo o conjunto de treinamento em conjuntos menores, um para cada valor do atributo escolhido, para os quais são construidas sub-árvores recursivamente. Para escolher este melhor atributo-teste é usada uma medida de informação chamada entropia. Aqui chamaremos esta medida de *E-score*.

O ID3 é um algoritmo da família TDIDT, sendo portanto seu pseudo-código (PSEUDO-CÓDIGO 2.2) um caso particular do pseudo-código TDIDT (PSEUDO-CÓDIGO 2.1) em que a função de avaliação é a entropia.

```
Entrada: CT - Conjunto de Treinamento
Saída:   AD - Árvore de Decisão

ID3 ( CT )
     Crie uma AD vazia
     Se os exemplos possuem o mesmo elemento de classificação E
          Coloque o elemento de classificação E na raiz de AD
          Retorne AD
     Senão
          Calcule a entropia para cada atributo presente em CT
          Selecione o atributo que apresentou a menor entropia
          Coloque o atributo selecionado na raiz de AD
          Para cada valor do atributo selecionado
             Crie um ramo em AD associado ao valor
             Crie um subconjunto de CT onde só ocorra o par atributo=valor
             ID3 ( subconjunto )
             Retire de CT todos os exemplos mapeados pelo ramo construído
          fimPara
fimID3
```

PSEUDO-CÓDIGO 2.2 - O algoritmo ID3.

Exemplo do ID3

Seja o domínio representado pelo conjunto de treinamento da TAB-2.1 adaptado de [Quinlan 83] e que denominamos "padrão global".

TABELA 2.1 Conjunto de treinamento para o domínio "padrão global".

EXEMPLO	ALTURA	COR CABELO	COR OLHOS	CLASSE
1	baixa	louro	castanhos	N
2	alta	escuro	castanhos	N
3	alta	louro	azuis	P
4	alta	escuro	azuis	N
5	baixa	escuro	azuis	N
6	alta	ruivo	azuis	P
7	alta	louro	castanhos	N
8	baixa	louro	azuis	P

Para o conjunto de treinamento da TAB-2.1, o ID3 gera a árvore de decisão da FIG-2.3.

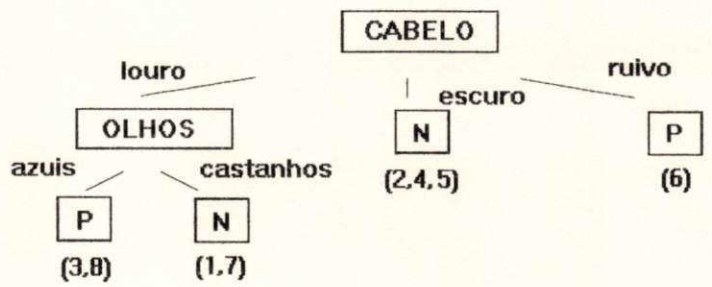

FIGURA 2.3 Árvore de decisão gerada pelo ID3 para o domínio "padrão global".

Obs: Os números entre parênteses abaixo dos nós folha, referem-se aos exemplos mapeados, e servem apenas para documentação.

2.4 O ID4

O ID4 objetiva construir incrementalmente (estratégia da FIG.2.2) uma árvore igual àquela que o ID3 construiria para um mesmo conjunto de treinamento (CT+ΔCT). Para viabilizar essa construção de forma incremental o ID4 acrescenta informações em cada nó visando possibilitar o cálculo do *E-score* de todos os possíveis atributos-teste e a escolha do melhor atributo para o nó.

Para todo nó no caminho de classificação de um exemplo, após atualizar as informações associadas a esse nó, o ID4 seleciona o melhor atributo-teste do nó. Quando o melhor atributo-teste não é aquele presente no nó, o ID4 descarta a sub-árvore que tem esse nó como raiz e reconstrói a sub-árvore com o novo atributo-teste.

Um pseudo-código para uma implementação do ID4 seria:

```
Entrada: A - Árvore de decisão
         N - O nó raiz da árvore de decisão
         E - Um exemplo
Saída :  A - Árvore de decisão atualizada

ID4 (A, E, N )
  Atualize os contadores do nó N
  Se o nó N for uma folha e sua classe for igual a classe de E
     Retorne A
  Senão
     Selecione o melhor atributo-teste para N usando um
               critério estatístico (entropia)
     Se o melhor atributo-teste for diferente do atributo de N
       Desconsidere as sub-árvores de N
       Construa uma sub-árvore usando o processo TDIDT e o mesmo
                critério estatístico, tendo como raiz o nó N e
                atualizando os contadores de cada nó
       Retorne A
     Senão
       Seja N o nó seguinte no caminho de classificação de E
       ID4 (A, E, N)
fimID4
```

PSEUDO-CÓDIGO 2.3 - O algoritmo ID4.

Maiores detalhes sobre esse algoritmo podem ser encontrados em [Schlimmer 86].

2.5 O ID5

O ID5 (PSEUDO-CÓDIGO 2.4) é semelhante ao ID4 com a diferença que, quando ocorre a necessidade de substituição do atributo-teste de um nó, a árvore não é descartada e reconstruída mas sim transformada. Esta transformação busca compatibilizar a árvore existente com o novo atributo-teste selecionado.

O processo de incorporação de um novo exemplo na árvore existente (*Incorpore_Ex_Arvore*) e o processo de transformação da árvore subindo o atributo selecionado (*Transforme A subindo atributo selecionado)* referenciados no PSEUDO-CÖDIGO.2.4, são comuns ao ID5 e aos algoritmos que serão descritos posteriormente (ID5R e IDL). Assim

sendo, apresentamos um pseudo-código para cada um desses procedimentos (PSEUDO-CÓDIGOS 2.5 e 2.6).

Um pseudo-código para uma implementação do ID5 seria:

```
Entrada: A - Árvore de decisão
         E - Um exemplo
Saída :  A - Árvore de decisão atualizada


ID5 (A,E)
  Incorpore_Ex_Arvore (A,E)
  Seja N igual a raiz de A
  Reestruture (A,N)
fimID5

Reestruture (A,N)
  Se o nó N for uma folha
     Retorne A
  Senão
     Selecione o melhor atributo-teste para N usando um
               critério estatístico
     Se o melhor atributo-teste a for diferente do atributo de N
        Transforme_A_subindo_o_atributo_selecionado (N,a)
     Seja N o nó seguinte no caminho de classificação de E
     Reestruture (A,N)
fimReestruture
```

PSEUDO-CÓDIGO 2.4 - O algoritmo ID5.

```
Entrada: A - Árvore de decisão
         E - Um exemplo
Saída :  A - Árvore de decisão atualizada

INCORPORE_EX_ARVORE (A,E)
  Classifique E atualizando os contadores de instâncias
               positivas e negativas no caminho de classificação
  Seja N igual ao último nó no caminho de classificação de E
  Se E está incorretamente classificado
     Repita
          Expanda o nó N construindo uma árvore usando o
                   processo TDIDT e um critério estatístico,
                   atualizando os contadores de instâncias
     Até classificar E
  Senão
     Se não conseguiu classificar E
        Adicione ao nó N um ramo representando o valor do
                 atributo de N presente em E
        Coloque na folha do ramo criado a classe de E
  Guarde as informações dos pares 'atributo=valor' presentes em
         E que não constam no caminho de classificação
  Retorne A
FIMINCORPORE_EX_ARVORE
```

PSEUDO-CÓDIGO 2.5 - A rotina Incorpore Ex. na árvore.

```
Entrada: A - Árvore
         N - Nó da árvore A
         a - Atributo selecionado para o nó N

Saída: Árvore A transformada

TRANSFORME_ÁRVORE_SUBINDO_ATRIBUTO_SELECIONADO (N, a)
    Para cada filho[i] do nó N
      Se o filho[i] é folha
         Crie um novo nó com o atributo a selecionado.
      Senão
         Se o atributo do filho[i] for diferente do atributo selecionado
            TRANSFORME ÁRVORE SUBINDO ATRIBUTO SELECIONADO (filho[i],a)
    fimpara

    Para cada filho[i] do nó N
      Crie uma subarvore com raiz N para cada filho[i]
      Troque o atributo raiz da subarvore pelo atributo de seu filho[i]
    fimpara

    Faça uma intercalação das subárvores criadas
    Se existe algum nó cujas folhas derivadas têm mesma classe
      Faça uma poda neste nó
    Substitua a subárvore de nó N da árvore A pelas subárvores
               intercaladas
fim TRANSFORME_ÁRVORE_SUBINDO_ATRIBUTO_SELECIONADO
```

PSEUDO-CÓDIGO 2.6 - A rotina Transforme árvore subindo atributo selecionado.

Para viabilizar a implementação do algoritmo, os seguintes elementos são adicionados à estrutura da árvore:

- em cada nó, contadores contendo a freqüencia de cada par atributo-valor para cada classe;
- nos nós folha, uma lista com os pares atributo-valor ainda não presentes na árvore e pertencentes aos exemplos mapeados.

Para o mesmo conjunto de treinamento utilizado para exemplificar o ID3 (TAB-2.1), mostraremos, a seguir (FIG.2.4, FIG.2.5, ... ,FIG.2.11), o resultado da árvore de decisão, gerada pelo ID5, após o processamento[1] de cada um dos oito exemplos da TAB.2.1.

[1] Entende-se por processamento de um exemplo, a incorporação desse exemplo à árvore, seguida de uma eventual transformação dessa árvore.

Nas figuras a seguir, os contadores associados aos nós estão representados entre colchetes, e as listas associadas aos nós folha, por uma questão de simplificação das figuras, não estão detalhadas com os respectivos pares atributo-valor de cada exemplo.

No início do processamento a árvore é nula.

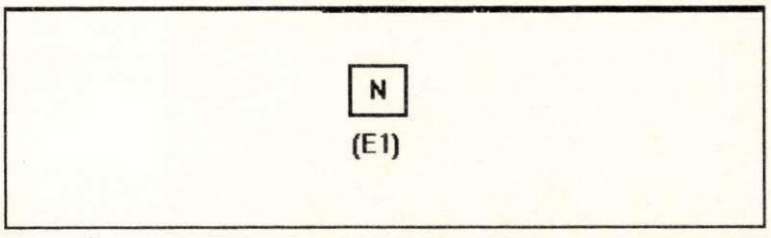

FIGURA 2.4 Árvore após o processamento do primeiro exemplo (E1)

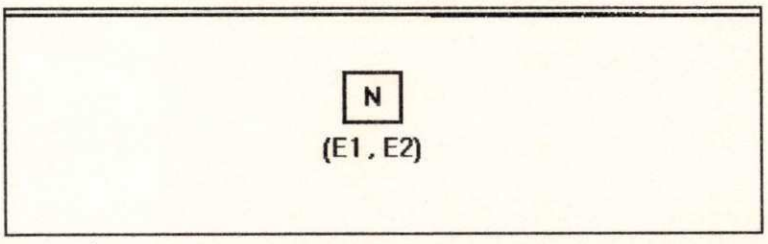

FIGURA 2.5 Árvore após o processamento do segundo exemplo (E2)

A ávore não foi alterada pois:

- classificou corretamente o exemplo 2 sem necessidade de *expansão ou de adicão de algum ramo* (PSEUDO-CODIGO.2.5)
- não sofreu nenhuma *reestruturação* pois continha apenas um nó-folha (PSEUDO-CÓDIGO.2.4)

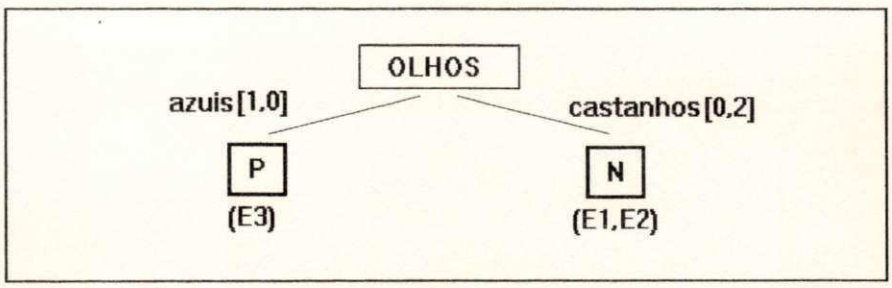

FIGURA 2.6 Árvore após o processamento do terceiro exemplo (E3)

Para classificar corretamente o exemplo 3, houve necessidade de *expandir* a árvore através da incorporacão do atributo OLHOS (PSEUDO-CODIGO.2.5)

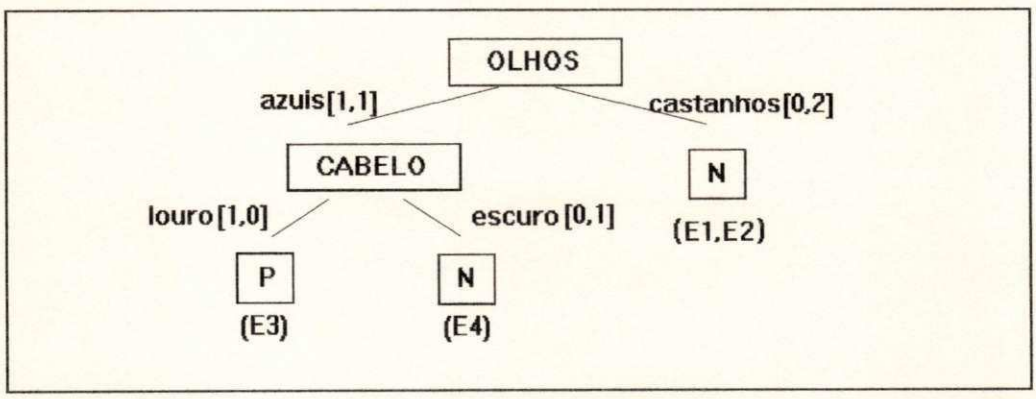

FIGURA 2.7 Árvore após o processamento do quarto exemplo (E4)

Nova *expanção* foi executada através da inclusão do atributo CABELO para viabilizar a correta classificação do exemplo 4.

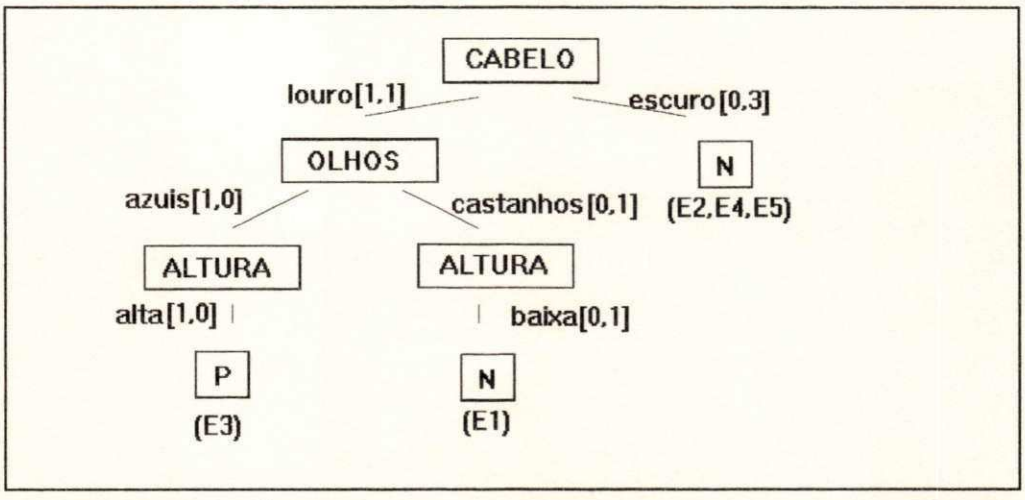

FIGURA 2.8 Árvore após o processamento do quinto exemplo (E5)

Para incorporação do quinto exemplo à árvore, não houve necessidade de nenhuma expansão ou adição de ramo (PSEUDO-CÓDIGO.2.5), entretanto, o atributo OLHOS não mais se apresentava como o melhor atributo para aquele nó, e a árvore sofreu uma *transformação* para viabilizar a *subida do atributo CABELO em substituição ao atributo OLHOS* (PSEUDO-CÓDIGO.2.6).

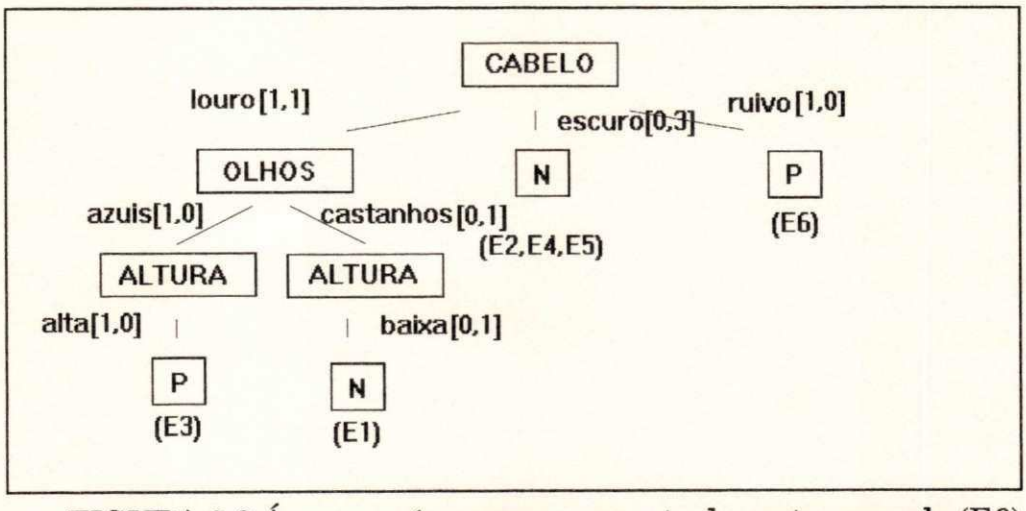

FIGURA 2.9 Árvore após o processamento do sexto exemplo (E6)

A incorporação do sexto exemplo à árvore, provocou uma adição de ramo com valor igual a ruivo associado ao nó do atributo CABELO, e nehuma transformação da ávore foi necessária pois os atributos do caminho de

classificação do exemplo continuaram sendo os melhores para aqueles nós.

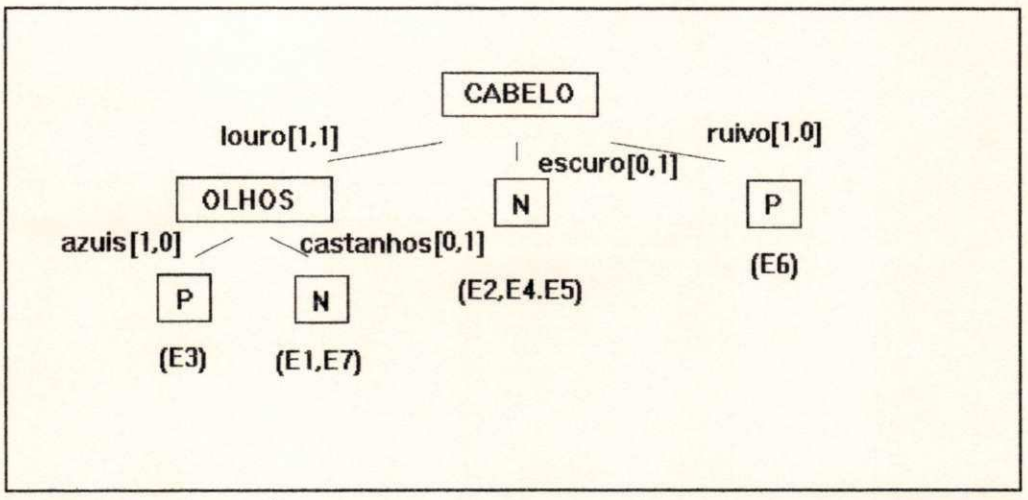

FIGURA 2.10 Árvore após o processamento do sétimo exemplo (E7)

Na incorporação do sétimo exemplo à árvore, houve uma adição de ramo com valor igual a alta no nó do atributo ALTURA que provocou a identificação da posibilidade de poda pois todas as folhas derivadas deste nó tinham a mesma classe.

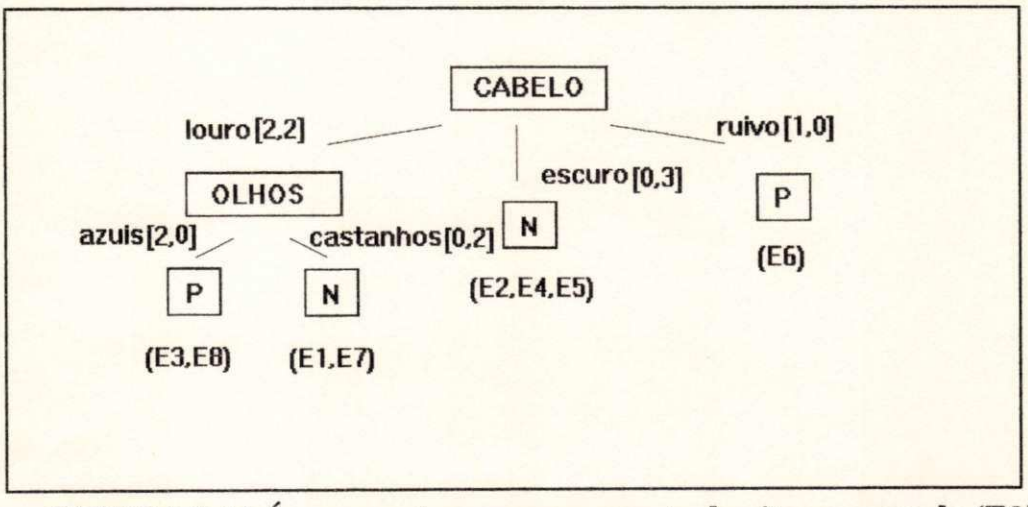

FIGURA 2.11 Árvore após o processamento do oitavo exemplo (E8)

Na incorporação do oitavo exemplo à árvore, nenhuma expansão nem adição de ramo foram necessárias e não houve mudança de atributos pois os mesmos continuaram sendo os melhores para aqueles nós.

Como podemos observar na FIG.2.11 , a árvore gerada pelo ID5 é igual à árvore gerada pelo ID3 apresentada na FIG.2.3.

2.6 O ID5R

O ID5R é semelhante ao ID5, exceto que acrescenta um procedimento que seleciona, com base no cálculo do E-score, o melhor atributo-teste para todos os nós da árvore que sofreu a transformação, enquanto que o ID5 calcula apenas para os nós no caminho de classsificação. Este procedimento visa garantir que todo nó da árvore terá o melhor atributo-teste.

Um pseudo-código para uma implementação do ID5R seria:

```
Entrada: A - Árvore de decisão
         E - Um exemplo
Saída :  A - Árvore de decisão atualizada

ID5R (A, E)
  Incorpore_Ex_Arvore (A, E)
  Seja N igual a raiz de A
  Reestruture (A, N)
fimID5R

Reestruture (A, N)
  Se o nó N for uma folha
      Retorne A
  Senão
      Selecione o melhor atributo-teste para N usando um
                critério estatístico
      Se o melhor atributo-teste for diferente do atributo de N
         Transforme A subindo o atributo selecionado (N,a)
         Para cada filho F de N
               Reestruture (A, F)
         fimPara
      Senão
         Seja N o nó seguinte no caminho de classificação de E
         Reestruture (A, N)
fimReestruture
```

PSEUDO-CÓDIGO 2.7 - O ID5R.

Maiores detalhes desse algoritmo são encontrados em [Utgoff 89].

2.7 O IDL

O IDL é semelhante ao ID5R com a diferença que, para identificar a necessidade de transformação da árvore, utiliza como opção adicional e prioritária um critério topológico chamado *ganho de relevância topológica*. Esse critério é baseado na estrutura da árvore e não em critérios estatísticos, eliminando, assim, o compromisso de geração de uma árvore de decisão semelhante àquela gerada pelo ID3, pois cria oportunidades de poda possibilitando a geração de uma árvore de menor tamanho que a criada pelo ID3. Ou seja, o IDL é um algoritmo incremental que embute simplificação.

Um pseudo-código para uma implementação do IDL seria:

```
Entrada: A - Árvore de decisão
         E - Um exemplo
Saída:   A - Árvore de decisão atualizada

IDL (A, E)
  Incorpore_Ex_Arvore (A, E)
  Seja N igual a folha no caminho de classifcação de E
  Reestruture (A, N)
fimIDL

Reestruture (A, M)
  Se todos os nós do caminho de classificação foram analisados
     Retorne A
  Para cada folha sob M com a mesma classe de E
      Encontre o maior caminho de classificação que satisfaça E
  fimPara
  Calcule o ganho de relevância topológica em relação a M(GRT_M)
           de cada um dos atributos presentes no caminho
           encontrado
  Se algum dos atributos apresenta um GRT_M positivo
     Transforme A subindo o atributo de maior GRT_M para o nó M
  Senão
     Transforme A subindo o atributo de melhor critério
                  estatístico para o nó M
  Se todas as folhas de M possuem a mesma classe
     Despreze as sub-árvores de M
     Coloque em M a classe de E
  Seja P o nó pai de M
  Reestruture (A, P)
fimReestruture
```

PSEUDO-CÓDIGO 2.8 - O algoritmo IDL.

Para uma melhor compreensão do IDL é necessário uma definição do conceito de relevância topológica.

Relevância Topológica

Para definirmos *relevância topológica*, é necessário inicialmente conceituarmos *caminho completo e caminhos parciais de classificação de um exemplo.*

- *Caminho completo de classificação de um exemplo* ou simplesmente *caminho de classificação de um exemplo*, é aquele caminho único, composto pelos pares nó-ramo e o nó folha da árvore envolvidos no processo de classificação desse exemplo.

- *Caminhos parciais do exemplo*, são caminhos em que todos os teste são satisfeitos pelo exemplo, mas não são o caminho de classificação do exemplo.

Assim, a relevância topológica do atributo a para o exemplo e na árvore $\mathcal{T}$, $(\mathcal{R}_{\mathcal{T}(a,e)})$ representa o número de ocorrências do atributo a nos caminhos (completo e parciais) do exemplo e.

Exemplo do cálculo de relevância topológica

Para um melhor entendimento de relevância topológica, consideremos:

- O seguinte exemplo e, formado por uma classe e quatro pares atributo-valor:

  Tempo=chuvoso,
  Temperatura=amena,
  Humidade=alta,
  Ventania=não, e
  classe= P.

- A árvore $\mathcal{T}$, graficamente representada na FIG.2.12 .

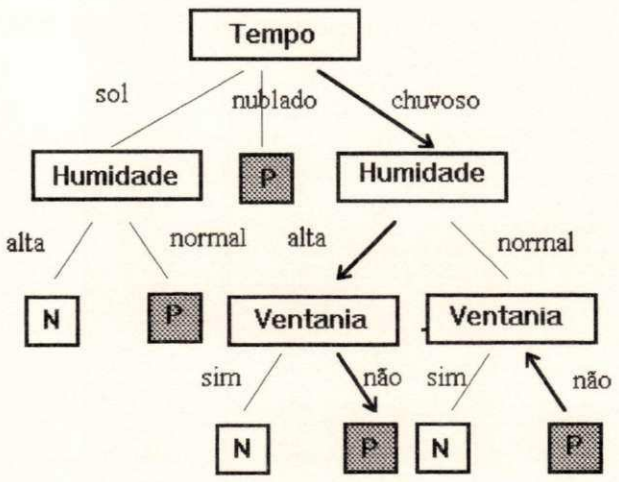

FIGURA 2.12 Árvore de decisão utilizada para conceituar relevância topológica.

Na árvore $\mathcal{T}$, o único *caminho de classificação do exemplo e,* está marcado mais fortemente e com a seta apontando para baixo, ele é:

(Tempo=chuvoso, Humidade=alta, Ventania=não , P)

Os outros caminhos que cobrem o exemplo *e* são os seguintes caminhos parciais:

- (**P**) - no caminho (tempo=sol, humidade=normal,p)
- (**P**) - no caminho (tempo=nublado,p)
- (**P , Ventania=não**) - no caminho (tempo=chuvoso, humidade=normal,ventania=não,p)

Neste caso, considerando os caminhos completos e parciais temos:

- O número de ocorrências do atributo **Tempo** nos caminhos é igual a 1 (um).

- O número de ocorrências do atributo **Humidade** nos caminhos é igual a 1 (um).

- O número de ocorrências do atributo **Ventania** nos caminhos é igual a 2 (dois).

Portanto os valores de suas respectivas relevâncias topológicas são:

$$\mathcal{R}_{\mathcal{T}(Tempo,e)} = 1 \qquad \mathcal{R}_{\mathcal{T}(Humidade,e)} = 1 \qquad \mathcal{R}_{\mathcal{T}(Ventania,e)} = 2$$

Ganho de Relevância Topológica

O *Ganho de relevância topológica* de um atributo a para o exemplo e em relação à sub-árvore cuja raiz é $\mathcal{M}$ e tem o nó S como o seguinte no caminho de classificação de e é dado por:

$$GR_{\mathcal{M}(a,e)} = \frac{\mathcal{R}_{\mathcal{M}(a,e)} - \mathcal{R}_{S(a,e)}}{\mathcal{R}_{\mathcal{M}(a,e)}}$$

Exemplo do cálculo do ganho de relevância topológica

Para exemplificar o cálculo do ganho de relevância topológica vamos considerar:

- A árvore $\mathcal{T}$ da FIG-2.12 , cuja raiz tem o atributo tempo.
- A subárvore S, cuja raiz tem o nó humidade (nó seguinte ao nó raiz de $\mathcal{T}$ no caminho de classificação de e)
- O exemplo e formado pelos pares atributo-valor:

  Tempo=chuvoso,
  Temperatura=amena,
  Humidade=alta,
  Ventania=não, e
  classe= P.
- O atributo *tempo.*

O *ganho de relevância topológica* do atributo *tempo* para o exemplo e em relação a árvore $\mathcal{T}$ é:

$$GR_{\mathcal{T}(tempo,e)} = \frac{R_{\mathcal{T}(tempo,e)} - R_{S(tempo,e)}}{R_{\mathcal{T}(tempo,e)}}$$

Como, $R_{\mathcal{T}(Tempo,e)} = 1$ e $R_{S(tempo,e)} = 0$

Então: $GR_{\mathcal{T}(tempo,e)} = \frac{1-0}{1}$

Portanto: $GR_{\mathcal{T}(tempo,e)} = 1$

2.8 Conclusões

Apesar dos algoritmos ID4, ID5 e ID5R almejarem alcançar a mesma base de conhecimento que o ID3 alcançaria se aplicada a um mesmo domínio, apenas o ID5R consegue em qualquer situação atingir totalmente este objetivo.

Uma análise comparativa preliminar sobre a qualidade das bases de conhecimento geradas por esses algoritmos, nos aponta em princípio para uma equiparação entre eles visto que, exceto o IDL, todos eles, almejam alcançar a mesma base de conhecimento do ID3.

Nos próximos capítulos nos deteremos a analisar o aspecto principal da motivação destes algoritmos, que é "*o desempenho de processamento*".

3 Análise comparativa teórica entre algoritmos incrementais e não-incrementais

3.1 Introdução

Os autores dos algoritmos incrementais quando apresentaram seus trabalhos à comunidade científica procuraram valorizá-los fazendo algumas análises comparativas com outros algoritmos similares. Essas análises entretanto, mostram-se parciais e incompletas, pois: (a) foram feitas de forma isolada, pois não faziam a comparação do algoritmo com um conjunto representativo dos demais; (b) estavam incompletas, pois não combinaram os diversos fatores de custo; (c) não levaram em consideração como a política de chegada de novos exemplos afetava o processo de aprendizagem e (d) não apresentavam uma orientação para escolha do algoritmo mais adequado a uma dada situação de aprendizagem.

Jeffrey C. Schlimmer [Schlimmer 86] definiu uma métrica que é função do número de comparações entre atributos e usou-a para comparar o ID4 (incremental) com o ID3 (não-incremental), contudo, essa comparação foi realizada apenas para o caso da incorporação de apenas um novo exemplo. Paul E. Utgoff [Utgoff 89] comparou o ID5 com o ID5R, utilizando uma métrica semelhante à de Schlimmer e, embora tenha definido uma métrica adicional (número de Cálculos de *E-score*), não a utilizou em suas conclusões. Walter Van de Velde [Van de Velde 89] apresentou os valores do desempenho do IDL em termos das mesmas métricas utilizadas por Utgoff, confrontando com os valores de desempenho apresentados pelo ID5 e ID5R, sem, contudo, fazer comparações conclusivas sobre os mesmos.

Neste capítulo, apresentamos uma análise comparativa mais abrangente, considerando conjuntamente a natureza dos domínios e os

diversos fatores que influenciam nas métricas dos custos dos algoritmos [Bezerra 94a].

3.2 Comparação de desempenho de processamento entre algoritmos incrementais e não-incrementais na geração de árvores de decisão

No nosso estudo comparativo, para caracterizar a dimensão de um domínio, usaremos os seguintes parâmetros: quantidade de atributos, quantidade de valores por atributo e quantidade de exemplos do conjunto de treinamento.

Para a identificação desses parâmetros utilizaremos a notação a seguir:

d = número de atributos.
b = número máximo de valores por atributo.
n = número total de exemplos do conjunto de treinamento.

A análise aqui apresentada é baseada em duas métricas que foram definidas em [Utgoff 89].

A primeira mede o *custo de atualização dos contadores de instâncias positivas e negativas* representado pela *quantidade de adições realizadas nestes contadores*. Cada adição é chamada de " instance-count addition" (*ICA*). Utgoff considera esta métrica similar ao "número de comparações de atributos" apresentada em [Schlimmer 86].

A segunda mede *o custo para selecionar um atributo* representado pela quantidade de cálculos da função de avaliação, aqui chamada de "quantidade de cálculos de *E-score*".

A análise comparativa feita, leva sempre em consideração apenas o pior caso, portanto não considera as características do domínio. Domínios distintos podem levar à árvores de profundidades distintas. Árvores de profundidades maiores geralmente têm maior custo de construção que árvores de profundidades menores.

As análises de complexidade dos algoritmos ID3 (item 3.3), ID3$^+$ (item 3.4), ID5 (item 3.6) e ID5R (item 3.5) apresentadas a seguir, estão em

[Utgoff 89]. A análise do ID4 (item 3.7) apresentada em [Schlimmer 86], foi adaptada para manter uniformidade e facilitar comparações, e a do IDL (item 3.8), está em [Van de Velde 89].

3.3 Análise do algoritmo ID3.

O ID3 verifica inicialmente se todas instâncias em um nó são da mesma classe. Se forem, a árvore é um nó resposta contendo o nome da classe. Se não forem, o algoritmo conta o número de instâncias positivas e negativas para cada valor de cada atributo. Calcula o *E-score* para escolher o atributo-teste no nó, e constrói uma árvore recursivamente para cada subconjunto de instâncias com o mesmo valor do atributo-teste selecionado. Não há necessidade de calcular o "*E-score*" quando há somente um atributo possível no nó.

Na escolha do atributo-teste na raiz, o algoritmo deve contar todos os d valores de atributos de todos n exemplos. Portanto, na raiz (nível 0) haverá $d.n$ *ICAs* e d cálculos de *E-score*. No nível 1 haverá b sub-árvores no pior caso (numero máximo de valores de um atributo) e $(d-1)n$ *ICAs* e $(d-1)b$ cálculos de *E-score*. No nível 2, também no pior caso haverá $b.b$ sub-árvores, e teremos $(d-2).n$ *ICAs* e $(d-2).b^2$ cálculos de *E-score*. Seguindo esse processo para todos os d níveis possíveis da árvore, e somando essas parcelas de cada nível (soma dos termos de uma progressão aritmética), o custo total para construir a árvore no pior caso do ID3 em relação ao número de *ICA* é dado por:

$$\sum_{i=1}^{d} i.n = \frac{d.(d+1)}{2} \cdot n$$

ou seja, a complexidade é $\boldsymbol{O(nd^2)}$

Por outro lado, o custo de construção da árvore no pior caso do ID3 em termos de cálculos de *E-score* será igual a:

$$\sum_{i=2}^{d} (i.b^{d-i}) = \frac{2.b^{d} - b^{d-1} - (d+1).b + d}{(b-1)^2}$$

ou seja, a complexidade é $\boldsymbol{O(b^d)}$

3.4 Análise do algoritmo ID3$^+$

Este algoritmo, chamado aqui de ID3$^+$, constrói uma nova árvore a cada novo exemplo do conjunto de treinamento. Cada novo exemplo é adicionado à tabela de exemplos e então utiliza o ID3 para construir uma nova árvore de decisão. Ele foi apresentado em [Schlimmer 86] com a finalidade de comparar seus resultados com o do ID4.

Com base na análise anterior do ID3, no pior caso o número de *ICA* para o ID3$^+$ é:

$$\sum_{i=1}^{n} \frac{i.d.(d+1)}{2} = \frac{d.(d+1)}{2} \cdot \frac{n.(n+1)}{2}$$

ou seja, a complexidade é $\boldsymbol{O(n^2d^2)}$

e o número de cálculo de *E-score* para o ID3$^+$ é:

$$\sum_{i=1}^{n} \frac{2.b^d - b^{d-1} - (d+1).b + d}{(b-1)^2} = n \cdot \frac{2.b^d - b^{d-1} - (d+1).b + d}{(b-1)^2}$$

ou seja, a complexidade é $\boldsymbol{O(nb^d)}$

3.5 Análise do algoritmo ID5R.

O ID5R constrói a árvore de decisão incrementalmente, baseado nos exemplos observados. Se necessário, a árvore é alterada, para no final obter uma árvore equivalente à árvore que o ID3 construiria para o mesmo conjunto de exemplos.

Informações adicionais são acrescidas a todos os "atributos-teste" possíveis em cada nó, para que seja possível substituir o "atributo-teste" do nó quando necessário. Essas informações são contadores de instâncias positivas e negativas para cada possível valor de cada atributo.

Número de *ICAs* no ID5R para o pior caso.

Três parcelas compõem o custo de atualização da árvore. A *primeira* representa o custo de atualização dos contadores de instância quando se incorpora um exemplo na estrutura da árvore. A *segunda* representa o custo de reestruturação da árvore para que o "atributo-teste" desejado fique na raiz. A *terceira* representa o custo da reestruturação recursiva das sub-árvores para garantir que cada uma das sub-árvores tenha o melhor "atributo-teste" em sua raiz.

Primeira parcela: O custo para incorporar um exemplo na estrutura da árvore no pior caso é dado por:

$$\sum_{i=1}^{d} i = \frac{d.(d+1)}{2}$$

ou seja, $\boldsymbol{O(d^2)}$

Segunda parcela: O custo para que o "atributo-teste" desejado fique na raiz no pior caso, é igual a:

$$\sum_{i=1}^{d} b^{i-1}.2.b^2(d-i) = \frac{2.b^2}{(b-1)^2}.b^{d-1} - b.(d-1) + (d-2)$$

ou seja, $\boldsymbol{O(b^d)}$

Terceira parcela: O custo de reestruturação recursiva das sub-árvores para garantir que cada uma delas tenha o melhor "atributo-teste" em sua raiz no pior caso é dado por:

$$\sum_{i=3}^{d-1} b^{d-i}.b^i = (d-3).b^d$$

ou seja, $\boldsymbol{O(db^d)}$

Considerando as três parcelas, a complexidade do custo resultante em números de *ICA*, no pior caso, para um exemplo será:

$$\boldsymbol{O(db^d)}$$

Consequentemente, para n exemplos, a complexidade do número de *ICA*, no pior caso, será:

$$O(ndb^d)$$

Número de cálculos de *E-score* no ID5R para o pior caso.

Duas parcelas compõem o custo de atualização da árvore de decisão quantificado em termos de cálculos de *E-score*.

A *primeira parcela* representa o custo da incorporação de um exemplo na estrutura da árvore, e é igual a:

$$\sum_{i=1}^{d} i = \frac{d.(d+1)}{2}$$

ou seja, $O(d^2)$

A *segunda* representa o custo de reestruturação recursiva das sub-árvores para garantir que cada uma delas tenha o melhor "atributo-teste" em sua raiz, e é igual a:

$$\sum_{i=3}^{d} b^{i-2}.(d-i+2) = (2.b^{d-1} - b^{d-2} - d.b + d - 1) \cdot \frac{b}{(b-1)^2}$$

ou seja, $O(b^d)$

Considerando as duas parcelas simultaneamente, temos:

$$\frac{d.(d+1)}{2} + \frac{(b^{d+1} - b)}{(b-1)}$$

ou seja, $O(b^d)$

Portanto, para n exemplos a complexidade do número de cálculos de *E-score* realizados pelo ID5R, para o pior caso, será:

$$O(nb^d)$$

3.6 Análise do algoritmo ID5.

O algoritmo ID5 é semelhante ao algoritmo ID5R, diferenciando-se apenas pela ausência do processo de reestruturação recursiva das sub-árvores, visando garantir que cada uma das sub-árvores tenha o melhor "atributo-teste" na raiz. Assim, com a eliminação desta etapa, é diminuído o custo de geração da árvore, trazendo entretanto um inconveniente de não garantir a construção da mesma árvore que o ID3 construiria para o mesmo conjunto de exemplos.

Número de *ICA* no ID5 para o pior caso.

Para o pior caso o número total de *ICA* para um exemplo é:

$$\frac{d.(d+1)}{2} + \frac{(b^{d+1} - b)}{(b-1)}$$

ou seja, $\boldsymbol{O(b^d)}$

e para n exemplos o número de *ICA* é:

$$\boldsymbol{O(nb^d)}$$

Número de cálculos de *E-score* no ID5 para o pior caso.

O número de cálculo de *E-score* para n exemplos no pior caso é:

$$\sum_{i=1}^{d} n.i = \frac{d.(d+1)}{2} . n$$

ou seja, $\boldsymbol{O(nd^2)}$

3.7 Análise do algoritmo ID4.

O ID4, no seu pior caso reconstrói a árvore a partir do nó raiz, tendo portanto nesta situação um desempenho medido em termos de *ICA* e cálculos de *E-score* semelhantes ao ID3$^+$.

Portanto, para o pior caso, a complexidade do custo do ID4 em número de *ICA* é:

$$O(n^2d^2)$$

e em termos de quantidade de cálculos de *E-score* é:

$$O(nb^d)$$

3.8 Análise do algoritmo IDL.

Seguindo a mesma metodologia adotada na análise do ID5 e ID5R, Walter Van de Velde [Van de Velde 89] realizou uma análise do custo da construção da árvore de decisão pelo IDL, encontrando os seguintes resultados para o pior caso:

- número de *ICA*

$$O(n.d.b^d)$$

- número de cálculos de *E-score*

$$O(n.d^2)$$

3.9 Resumo das análises de desempenho dos algoritmos.

A TAB. 3.1 mostra os resultados finais encontrados na análise de custo para geração de uma árvore de decisão através dos algoritmos já referidos. Acrescentamos, na tabela, a coluna Custo Total, para representar a soma dos custos de *ICA* e de *E-scores*. Foi necessário introduzir um fator de compatibilização de unidades de custo (α) que varia com a Função de Avaliação utilizada no cálculo do *E-score*. O valor de α é sempre maior que 1, pois em termos de tempo de processamento, o cálculo de um *E-score* é maior do que o de um *ICA*.

TABELA 3.1 Resumo das análises de custo dos algoritmos incrementais e não-incrementais

Algoritmos	Número de ICA	Quantidade de cálculos de E-score	Custo Total
ID3	$O(n.d^2)$	$O(b^d)$	$n.d^2 + \alpha\, b^d$
ID4	$O(n^2.d^2)$	$O(n.b^d)$	$n^2.d^2 + \alpha\, n.b^d$
ID5	$O(n.b^d)$	$O(n.d^2)$	$n.b^d + \alpha\, n.d^2$
ID5R	$O(n.d.b^d)$	$O(n.b^d)$	$n.d.b^d + \alpha\, n.b^d$
IDL	$O(n.d.b^d)$	$O(n.d^2)$	$n.d.b^d + \alpha\, n.d^2$

As conclusões em termos de comparação de desempenho entre os diversos algoritmos incrementais e o ID3, apresentadas pelos autores dos algoritmos incrementais aqui citados, têm considerado apenas uma métrica isoladamente, em alguns casos considerando o *ICA* e em outros o *E-score*, não fornecendo, assim, uma visão do desempenho total dos algoritmos. Portanto, para proceder uma análise mais realística entre os algoritmos incrementais e não incrementais, neste trabalho, utilizaremos as duas métricas (*ICA+E-score*) conjuntamente e consideraremos o desempenho dos algoritmos em domínios estáticos e domínios dinâmicos, levando em conta neste último caso a chegada de n novas instâncias de cada vez [Bezerra 94b].

Para a comparação dos algoritmos incrementais com o representante dos não-incrementais (ID3), escolheremos dentre os incrementais aquele que apresentar o menor custo total.

Observando a coluna *custo total,* da TAB 3.1, percebe-se facilmente que o ID5 tem um custo total menor que o ID5R e o IDL, visto que sempre teremos $b \geq 2$, $d \geq 2$ e $n \geq 1$.

Da mesma forma, é fácil mostrar que o *Custo ID4* $\geq$ *Custo ID5.* De fato, para cada um desses algoritmos, a coluna do custo total, na TAB.3.1, fornece:

- *Custo (ID4)* = $n^2d^2 + \alpha\, nb^d$
- *Custo (ID5)* = $nb^d + \alpha nd^2$

A diferença de custo entre eles é dada por:

$$\Delta Custo = n^2d^2 + \alpha nb^d - nb^d - \alpha nd^2$$

ou:

$$\Delta Custo = n\,[\, b^d(\alpha - 1) - d^2(\alpha - n)\,]$$

como,

$$(\alpha - 1) \geq (\alpha - n) \quad \text{e} \quad b^d \geq d^2$$

então:

$$\Delta Custo \geq 0$$

ou seja:

$$Custo\,(ID4) \geq Custo\,(ID5)$$

Portanto, escolheremos o ID5 (por apresentar o menor custo) para representar os incrementais para efeito de comparação com o ID3.

3.10 Comparação de desempenho em domínios estáticos.

Para domínios estáticos, onde o processo de aprendizagem é executado uma única vez com n instâncias, o comportamento do custo em função do número n de instâncias é dado por:

Para o ID3 $\quad n d^2 + \alpha b^d$

Para o ID5 $\quad n(b^d + \alpha d^2)$

Em um mesmo domínio e utilizando-se a mesma função de avaliação para os dois algoritmos (d, b e α constantes), verificamos que a taxa de crescimento do custo no ID3 em função de n é constante e igual a d^2, enquanto que para o ID5 é também constante mas igual a $b^d + \alpha\, d^2$. Portanto, para um mesmo domínio, existe um valor de n a partir do qual o ID3 (não incremental) apresenta um desempenho superior ao do ID5 (incremental). O GRAF. 3.1 mostra essa situação.

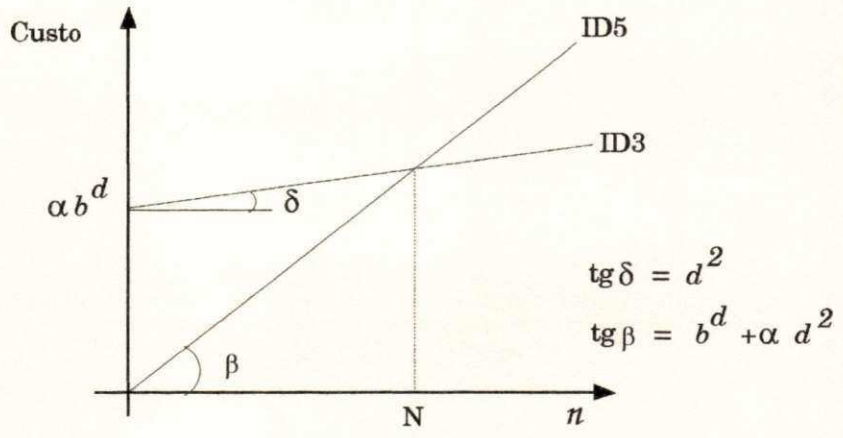

GRÁFICO 3.1 - Custo de processamento dos algoritmos ID3 e ID5 em função da quantidade ***n*** de instâncias, para domínios estáticos [Bezerra 94a].

O valor mínimo de ***n*** a partir do qual o ID3 é mais eficiente que o ID5 é dado por,

$$\mathcal{N} = \frac{\alpha b^d}{b^d + d^2(\alpha - 1)}$$

Como, para um dado α, **N** é função do domínio (valores de ***d*** e ***b***), vamos analisar o comportamento dessa função. É fácil verificar que, com o crescimento de b^d, **N** tende para α, e esse limite é alcançado tão mais rapidamente quanto maior for o valor de ***b***.

Baseados em valores de quantidades de E-scores (entropia), ICAs e de tempo de CPU, apresentados em [Utgoff 89], encontramos um valor de α em torno de 18. O GRAF. 3.2 mostra o valor de **N** em diversos tipos de domínios para α igual a 18.

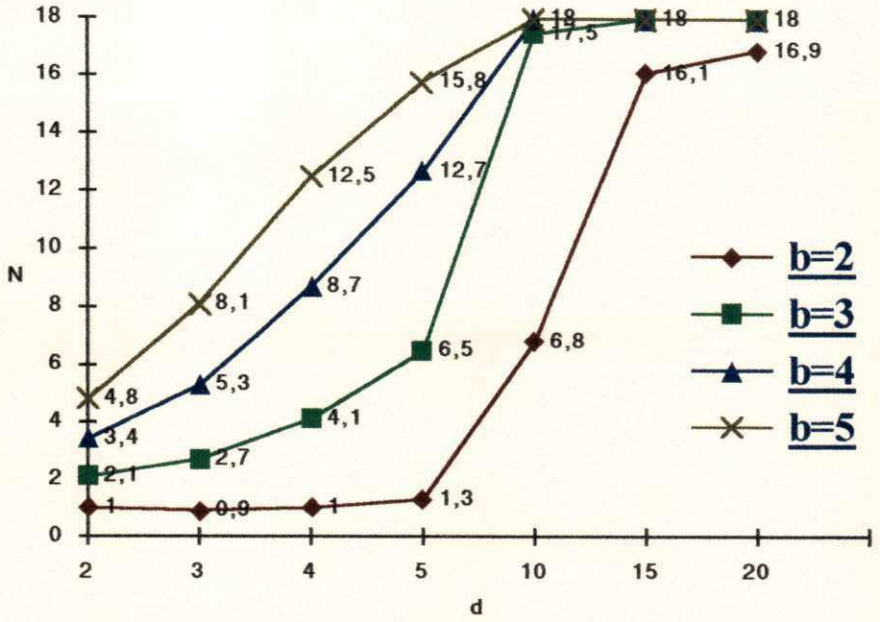

GRÁFICO 3.2 - Número de instâncias **N** a partir do qual o ID3 é mais eficiente que o ID5 em função do tipo de domínio aqui representado por ***d*** e ***b***, para um fator $\alpha = 18$.

Da análise do GRAF.3.2, concluimos que, independentemente do domínio (valores de ***b*** e ***d***), o ID5 só é mais eficiente do que o ID3 em casos de poucas instâncias (máximo igual a α, que no caso vale 18) e nas demais situações o ID3 é superior. Estas situações em que o ID3 é superior , abrangem os casos reais que normalmente possuem centenas ou milhares de instâncias. Esta conclusão, para os domínios estáticos, já era de se esperar, pois a virtude dos incrementais está no seu uso para domínios dinâmicos.

3.11 Comparação de desempenho em domínios dinâmicos.

Em domínios dinâmicos, existe a necessidade de executar o processo de aprendizagem diversas vezes. Na primeira execução, com ***i*** instâncias, o comportamento do desempenho dos algoritmos é igual ao comportamento apresentado em domínios estáticos. Em execuções posteriores, em que são incorporados à base de conhecimento corrente, lotes de ***n*** novas instâncias a cada processamento, o custo total (TAB.3.1) em função do ***n*** é dado por:

Para o ID3 $\quad d^2n + id^2 + \alpha b^d$

Para o ID5 $\quad n(b^d + \alpha d^2)$

onde ***i*** é o número de instâncias previamente utilizadas na formação da base de conhecimento corrente.

O GRAF.3.3 mostra o comportamento do custo, em domínios dinâmicos após um processamento inicial com *i* instâncias.

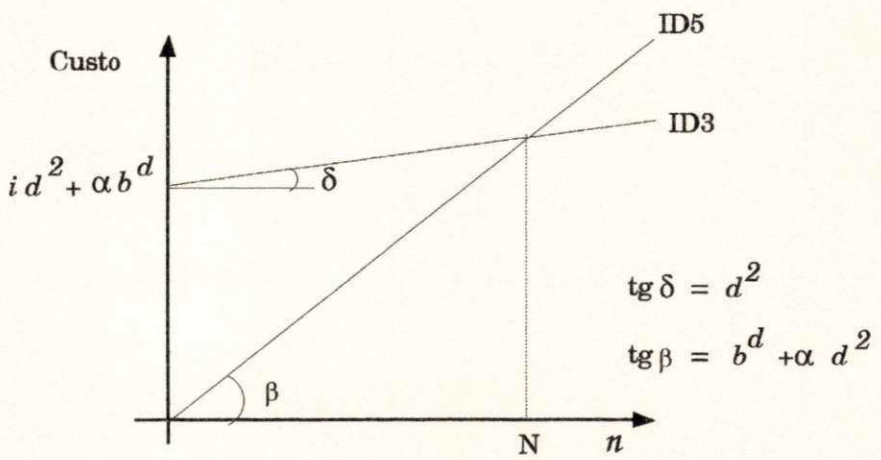

GRÁFICO 3.3 - Custo de um processamento dos algoritmos ID3 e ID5 em função da quantidade ***n*** de novas instâncias, para domínios dinâmicos após o processamento prévio de ***i*** instâncias [Bezerra 94b].

Da mesma forma que nos domínios estáticos verificamos, pelo GRAF.3.3, que existe um **N** a partir do qual o ID3 é mais eficiente do que o ID5, e seu valor é dado por,

$$\mathcal{N} = \frac{\alpha b^d}{b^d + d^2(\alpha - 1)} + \frac{id^2}{b^d + d^2(\alpha - 1)}$$

Vamos agora analisar a variação de **N** (valor a partir do qual o ID3 é superior ao ID5) em função do domínio e do número de ***i*** instâncias já processadas. Para isso vamos fixar o número máximo de valores por atributo em ***b***=3 e variar o número de atributos ***d***. Oportunamente discutiremos a influência de outros valores para ***b***. É fácil verificar que,

também neste caso dos domínios dinâmicos, o valor de **N** tende para α com o crescimento de ***d*** e para um dado valor de ***b***. Só que agora, para valores intermediários de ***d*** (***d*** < 10) o **N** depende do número de instâncias ***i*** previamente utilizadas.

O GRAF. 3.4 mostra o comportamento do **N** para diversos tipos de domínios (***b*** = 3 e ***d*** ≥ 2), diversos valores de ***i*** e α=18.

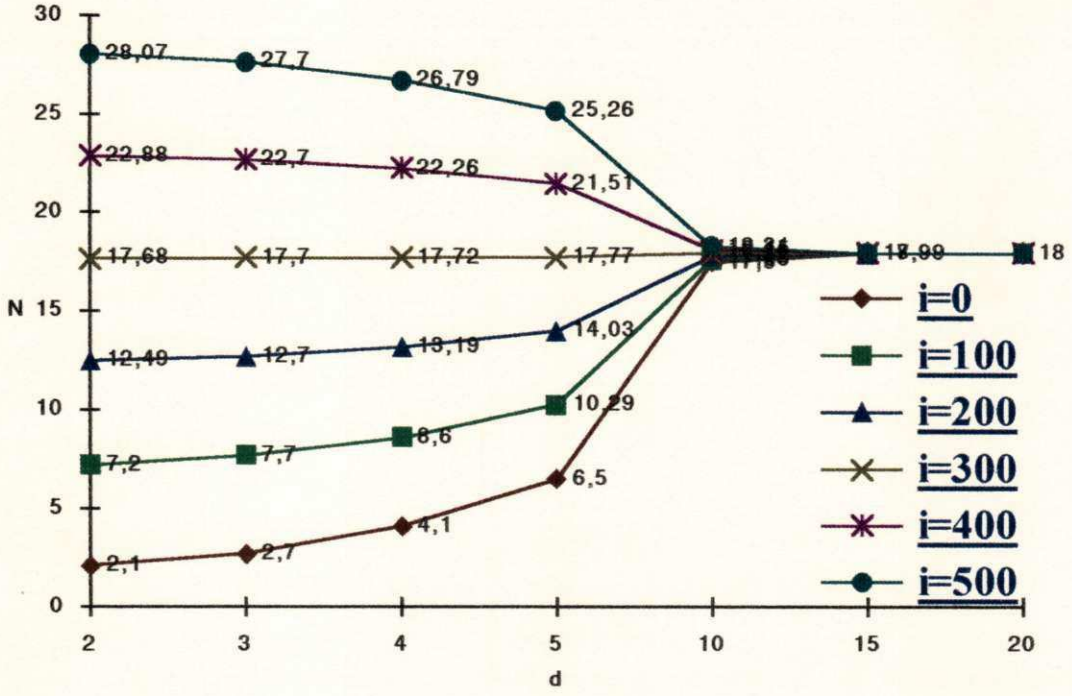

GRÁFICO 3.4 - Valor de **N** em função de ***i***, a partir do qual o ID3 é mais eficiente que o ID5 em domínios dinâmicos após o processamento de ***i*** instâncias.

Para analisar o GRAF.3.4, podemos dividir o domínio, conforme a quantidade de atributos, em três faixas:

- *F1. Domínios de brinquedo*, em geral caracterizados por $2 \leq d < 5$.
- *F2. Alguns domínios de brinquedo e alguns domínios reais,* ou seja, domínios com quantidade de atributos variando no intervalo $5 \leq d < 10$.
- *F3. A maioria dos domínio reais,* ou seja, aqueles domínios com 10 ou mais atributos ($d \geq 10$).

Para a primeira faixa *(F1)*, a superioridade dos incrementais (ID5) depende do tamanho do lote e do número ***i*** de instâncias processadas. Quanto maior for o valor de ***i***, maior será o tamanho do lote que garante a superioridade dos incrementais. Isto é, se o ponto de partida for uma base de conhecimento que espelhe um grande número de exemplos (***i***), o ID5 poderá ser superior mesmo com tamanho de lotes bem maiores que 1 (note que no limite mínimo ***i***=0, o tamanho limite já é 2). Contudo, para cada combinação de valores de ***b***, ***d*** e ***i***, haverá um limite **N** para o tamanho do lote a partir do qual o ID3 apresenta um melhor desempenho.

Para a segunda faixa *(F2)*, as conclusões são identicas às anteriores (faixa *F1*), exceto que agora, para que o ID5 seja superior ao ID3 o tamanho dos lotes poderá ser inferior ao da faixa *(F1)*, se $i > 300$, e superior para $i < 300$, tendendo para o limite α (18 no caso) quando o valor de i fica em torno de 300.

Finalmente, para a terceira faixa *(F3)* verificamos que, independentemente do número de instâncias ***i*** já processadas, o ID5 será superior ao ID3 para tamanho de lotes inferiores a α, e em caso contrário o ID3 será superior.

A análise do GRAF.3.4 foi feita para valores de ***b***=3, entretanto essas mesmas conclusões são válidas para os demais valores de ***b***. A única diferença é que, quanto maior for o valor de ***b***, menor será a faixa de valores de **N** para os diversos ***i*** 's.

3.12 Conclusões.

Os estudos comparativos de desempenho de processamento entre os algoritmos não incrementais, representados pelo ID3 e os incrementais representados pelo ID5, apresentados neste capítulo, analisando-se o pior caso, indicam que:

a) Para *domínios estáticos,* e *dinâmicos em um primeiro processamento* (sem uma árvore inicial, i.e, i =0), o ID3 (não-incremental) é mais

eficiente que o ID5 (representante dos incrementais) para aplicações do mundo real e também na grande maioria das aplicações de brinquedo.

b) Para *domínios dinâmicos, em processamentos posteriores ao primeiro* ($i>0$):

b1) O ID5 deve ser preferido quando o número de atributos não é pequeno *(d >10)* e, ao mesmo tempo, o número de novos exemplos a serem processados *(n)* é menor que o fator α que compatibiliza o custo de ICA com o custo em E-scores. O número *i* não influencia nesta situação.

b2) O ID5 deve também ser preferido quando $n<N$, onde N é uma função de α, *b (número máximo de valores por atributos), d (número de atributos)* e *i (número de exemplos já processados).*

b3) Em todos os demais casos, o ID3 é mais eficiente.

4 Análise comparativa experimental entre algoritmos incrementais e não-incrementais

4.1 Introdução

A análise comparativa teórica dos custos dos algoritmos incrementais e não incrementais, que apresentamos no capítulo anterior, foi baseada em métricas definidas pelos autores desses algoritmos e foram feitas considerando o pior caso. Neste capítulo, através de execuções dos algoritmos em situações reais, procuramos checar a validade das conclusões teóricas apresentadas. Confrontamos as conclusões teóricas com as conclusões obtidas em casos reais e analisamos as métricas utilizadas pelos autores visando representar os custos de seus algoritmos.

4.2 Criação de domínios para análise

Para uma melhor comparação dos resultados da análise prática dos custos dos algoritmos ID3 e ID5 com a análise teórica antes apresentada, escolhemos, para processamento, domínios com as mesmas características dos que foram mostrados nos gráficos dos custos dos algoritmos na análise comparativa teórica. Esses domínios têm uma quantidade de atributos que varia de 2 a 20 e um número de valores por atributo na faixa de 2 a 5. Igualmente ao caso da análise teórica, a quantidade de exemplos estará entre 2 e 100.

O ambiente A4 [Vasco 93] possibilita a criação de domínios através de seu módulo funcional de *modelagem de domínio*, que permite que se cadastre as características de um domínio e uma tabela de exemplos associada a esse domínio. Esse processo é realizado através de uma entrada de dados dos nomes das classes, dos nomes e valores dos atributos e o registro, para cada exemplo, do conjunto de pares atributo-valor e a respectiva classe.

Como necessitávamos criar uma grande quantidade de domínios distintos, optamos por não utilizar a função modelagem de domínio do ambiente A4, e sim automatizar o processo de geração dos arquivos de domínios. Utilizando a linguagem C, elaboramos um programa que a partir da definição de certos parâmetros (quantidade de atributos, número de valores por atributo, quantidade de classes e de exemplos) de um dado domínio, gerasse as características desse domínio juntamente com a sua tabela de exemplos. Para viabilizar a utilização futura de outros métodos disponíveis no ambiente A4, resolvemos gerar os domínios com suas respectivas tabelas de exemplos em "lay-outs" de arquivos compatíveis com esse ambiente. Os códigos atribuídos a cada atributo, a cada valor de atributo e a cada classe, seguiram o padrão do ambiente A4 e foram criados através da função **rand** da linguagem C que gera valores aleatórios.

4.3 Forma de obtenção dos valores dos custos de execução dos algoritmos

Os algoritmos ID3 e ID5 foram implementados utilizando a linguagem C++. O ID3 já havia sido implementado quando da elaboração do ambiente A4, e o ID5 o implementamos seguindo os padrões desse ambiente para que, no futuro, a sua incorporação ao ambiente fosse trivial.

A apuração dos tempos de CPU gastos na execução dos algoritmos foi feita em estações "SUN SPARC station 2", utilizando o Sistema Operacional Solaris 1.1.1 pertencentes à Rede do Departamento de Sistemas de Computação da UFPB.

Iniciamos a medição dos custos de execução dos algoritmos, representados pelos seus tempos de CPU, utilizando a opção **-pg** do compilador C++. Esta opção de compilação permite que o sistema gere uma estatística em um arquivo compactado, contendo o número de vezes que cada rotina do programa é executada, seu tempo de CPU, e o percentual do tempo de CPU de cada rotina em relação ao tempo de CPU total do programa.

Após sucessivas execuções dos algoritmos ID3 e ID5, visando a comparação de seus tempos de execução, constatamos uma situação em que para uma mesma massa de dados de entrada e para um mesmo algoritmo, em momentos distintos, obtinham-se resultados de tempo de CPU diferentes. Isso causou surpresa, pois os tempos de CPU deveriam ser iguais, visto que os algoritmos não continham nenhuma componente aleatória. Preparamos então um pequeno programa de teste contendo apenas um "looping for" visando apuração de seu tempo de CPU, e constatamos a mesma situação (mesmo programa com mesma massa de dados apresentava tempos de CPU distintos se rodado em momentos distintos).

Desprezamos então todos os resultados dos tempos de CPU obtidos através da opção **-pg**, e passamos a utilizar a função **getrusage** do C++ que, entre outras coisas, permite a apuração de tempo de CPU de rotinas de um programa. Constatamos o mesmo problema.

Outra função do C++ foi utilizada. Agora a função **times,** que permite também apurar o tempo de CPU de rotinas de um programa. A mesma situacão ocorreu.

Utilizamos o programa de teste para apuração desses tempos em outro ambiente, agora a Rede do Departamento de Engenharia Elétrica da UFPB utilizando um PC 486-DX2-66 sob o sistema operacional LINUX 1.1.59. A mesma situação aconteceu.

Passamos a testar a mesma situação simulando, na Rede do Departamento de Sistemas de Computação da UFPB, um ambiente "mono-usuário". A mesma situação ocorreu.

Diante desses fatos e após consultarmos vários especialistas em software básico, que diagnosticaram problemas ligados à precisão do relógio das máquinas, continuamos a apuração desses tempos com objetivo de analisarmos melhor o comportamento dessas diferenças de tempo de CPU.

Procedimento adotado na obtenção dos tempos de execução dos algoritmos

Constatamos que as diferenças de tempo ocorriam numa ordem de grandeza insignificante para efeito de nossa análise comparativa, e que essa diferença poderia ser controlada através do cálculo de uma média entre o resultado de vários processamentos do algoritmo com uma mesma situação. Nos casos em que houvese uma variação maior, deveríamos repetir esta situação um número maior de vezes até encontrarmos uma média estável.

Após várias simulações, visando encontrar um número ideal para repetição da execução dos algoritmos, decidimos por repeti-los 10 vezes para cada situação (domínio) visando tirar a média do tempo de execução daquela situação. Portanto, para uma dada situação (domínio), o tempo de execução (CPU) de um algoritmo corresponde à média de 10 valores do tempo de execução (um valor para cada rodada do algoritmo).

Para comparar os tempos de CPU consumidos pelos algoritmos ID3 e ID5 era preciso automatizar o processo de apuração e tratamento desses tempos, devido ao grande número de rodadas que era exigido. As funções **times** e **getrusage** eram mais adequadas para isso, porque seus resultados eram acessíveis em variáveis do próprio programa e portanto mais facilmente controlados para efeito de formatação de uma saida mais apropriada a um processo de automatização. A opção **-pg** do compilador foi descartada para essa finalidade, porque seus resultados são apresentados em um arquivo compactado com muitas informações desnecessárias ao nosso objetivo e, portanto, seu tratamento seria mais complexo que o das funções **times** e **getrusage**. Assim, para a obtenção do **tempo total** de CPU consumido por cada um dos algoritmos ID3 e ID5 (apresentados nos apêndices A e B), optamos pelo uso da função **times** por ser mais específica que a função **getrusage**, pois essa última,

além de obter o tempo de CPU, gera outras informações sobre os recursos da máquina que para nós não eram necessárias.

Já para a análise das métricas utilizadas na estimativa de custo dos componentes do algoritmo ID5, utilizamos a opção **-pg** do compilador por já apresentar um percentual comparativo entre rotinas (métricas) do programa (algoritmo) e não haver necessidade de um grande número de rodadas como o exigido no processo de comparação entre os tempos de CPU dos algoritmos ID3 e ID5.

Codificação de identificação dos domínios

Os domínios apresentados na TAB.4.3 têm suas características representadas pela codificação "***Daavvccqqq***" em que:

.**D** identifica um arquivo de domínio
.***aa*** representa a quantidade de atributos
.***vv*** representa a quantidade de valores por atributo
.***cc*** representa a quantidade de classes
.***qqq*** representa a quantidade de exemplos

Após vários experimentos, verificamos que o número de classes dos domínios não afetava os resultados comparativos entre os tempos de CPU e ID5 e do ID3, sendo semelhantes as diferenças relativas dos tempos de CPU entre esses algoritmos, tanto quando usávamos domínios com duas classes ou com mais de duas classes. Assim, para simplificar a nossa análise, resolvemos fixar em dois o número de classes de cada domínio.

4.4 Resultados encontrados

4.4.1 Domínios estáticos

No Apêndice-A, apresentamos os resultados dos custos dos algoritmos ID3 e ID5 em função da quantidade n de exemplos processados nas diversas situações de tipos de domínios estáticos utilizados em nossa análise experimental.

Em todos os experimentos realizados com domínios estáticos, (com d variando de 2 a 20, b variando de 2 a 5) foi confirmada a conclusão apresentada na análise teórica (vide capítulo 3 item 3.10) de que a partir de uma quantidade N de exemplos, o ID5 apresenta um custo maior que o ID3.

A TAB.4.1 resume os valores de N encontrados nesses experimentos. Percebe-se que o valor máximo encontrado para N foi de 10 exemplos e o mínimo foi de 2 exemplos.

TABELA 4.1 - Quantidade n de exemplos em domínios estáticos acima do qual o ID3 é mais eficiente que o ID5.

valores de d	$b = 2$	$b = 3$	$b = 4$	$b = 5$
2	5	5	5	5
3	3	4	4	3
4	2	2	2	3
5	3	3	3	3
6	2	2	2	2
7	2	2	2	3
8	2	2	2	3
9	2	2	2	4
10	2	2	2	3
11	2	2	3	6
12	2	2	3	6
13	5	5	6	7
14	3	6	7	9
15	8	8	10	10
16	4	4	6	5
17	3	3	6	7
18	4	6	4	6
19	3	5	8	6
20	5	5	6	6

Para ilustrar graficamente (GRAF.4.1) o comportamento dos custos desses algoritmos em domínios estáticos, selecionamos um domínio representativo dos domínios reais. Para isso, escolhemos um domínio com 10 atributos e 4 valores por atributos (D100402), por representar um tamanho médio de domínio [Donato 94].

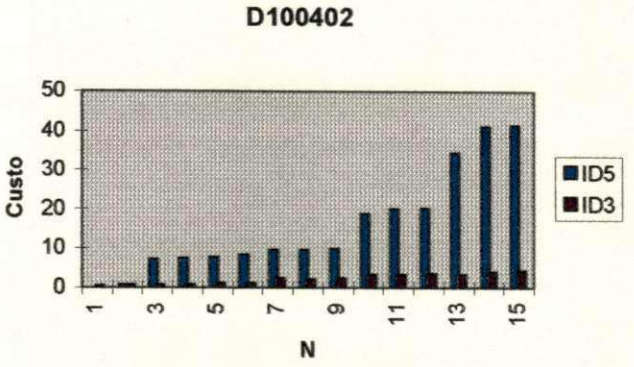

GRÁFICO 4.1 Custo do ID3 e do ID5 no domínio estático (10 atributos e 4 valores por atributos) em função do número *n* de exemplos processados.

Confrontando os resultados apresentados na análise teórica com os dos experimentos em domínios estáticos (Apêndice A), chegamos às seguintes constatações:

- a) Confirmada a existência de um número N de exemplos acima do qual o ID5 é menos eficiente que o ID3;

- b) Confirmada também que a taxa de crescimento do custo do ID3 em função do número (n) de exemplos processados é constante, embora que para certos domínios e, principalmente, para pequenos valores de n apresente oscilações;

- c) A curva representativa dos custos do ID5 em função do número de exemplos processados, apresenta variações bruscas e aleatórias, não apresentando, portanto, uma taxa de crescimento constante como afirma a análise teórica (baseada em métricas definidas pelos autores e considerando o pior caso). Entretanto, a exemplo da análise teórica, em média, essa curva apresenta uma tendência ascendente e um crescimento bem mais acentuado que aquele apresentado pela curva do ID3.

4.4.2 Domínios dinâmicos

No Apêndice-B, apresentamos os resultados dos custos dos algoritmos ID3 e ID5 em função da quantidade *n* de novos exemplos processados

nos diversos domínios. Na escolha desses domínios optamos por fixar em 4 a quantidade de valores por atributo (b médio), variando a quantidade de atributos d numa faixa de 2 a 20, e a quantidade i de exemplos anteriormente processados na faixa entre 10 e 90.

Da mesma forma que em domínios estáticos, também em domínios dinâmicos, em todas situações testadas, foi confirmada a conclusão apresentada na análise teórica (vide capítulo 3 seção 11). Ou seja, a partir de uma quantidade N de exemplos o ID5 apresenta um custo maior que o ID3.

Para ilustrar essa conclusão, mostramos na TAB.4.2 o resumo dos resultados dos experimentos com os valores encontrados para N. Verificamos para as situações analisadas, um valor máximo de N igual a 26 exemplos e o mínimo de 0 (nenhum exemplo).

TABELA 4.2 - Quantidade N de exemplos em domínios dinâmicos (com b=4) acima do qual o ID3 é mais eficiente que o ID5.

d	i=10	i=20	i=30	i=40	i=50	i=60	i=70	i=80	i=90
02	4	4	3	12	10	0	21	26	0
03	2	0	5	0	12	0	19	9	0
04	0	3	0	3	4	2	2	2	2
05	2	1	0	0	1	2	2	0	2
06	2	0	0	2	1	0	0	0	0
07	0	1	0	3	0	2	2	3	2
08	0	0	0	2	0	3	5	2	0
09	2	3	0	0	2	3	3	0	0
10	2	1	0	0	2	0	1	3	1
11	0	0	0	0	0	0	8	8	0
12	0	0	0	0	1	4	0	2	4
13	3	5	0	2	2	4	0	4	0
14	4	3	0	2	0	3	2	7	3
15	0	0	0	0	1	2	2	3	0
16	0	4	0	2	0	0	2	4	6
17	2	0	0	2	0	0	0	2	0
18	0	0	0	0	1	0	2	0	1
19	0	0	0	0	1	0	3	0	0
20	0	0	0	5	1	0	0	0	0
Média	1,2	1,3	0,4	1,8	2,0	1,3	3,9	4,0	1,1

Da mesma forma que na apresentação dos resultados em domínios estáticos, selecionamos um domínio para ilustrar o comportamento dos custos desses algoritmos em domínios dinâmicos (GRAF.4.2 à GRAF.4.10). Para isso escolhemos um domínio com as mesmas características do escolhido para apresentação gráfica da análise experimental em domínios estáticos (D100402). Cada um desses gráficos, apresenta os custos do ID3 e ID5 para esse domínio e para um valor diferente de i (variando de 10 a 90).

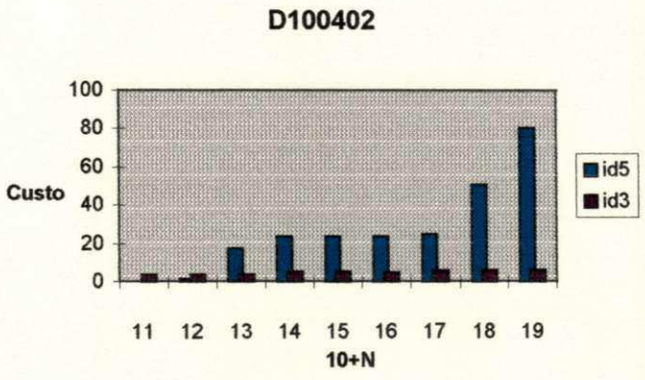

GRÁFICO 4.2 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=10, em função do número de exemplos processados.

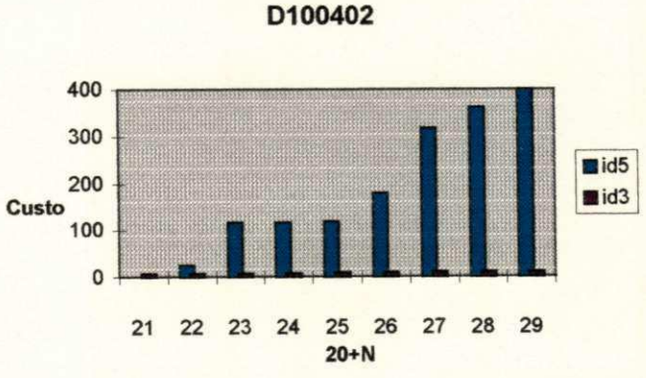

GRÁFICO 4.3 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=20, em função do número de exemplos processados.

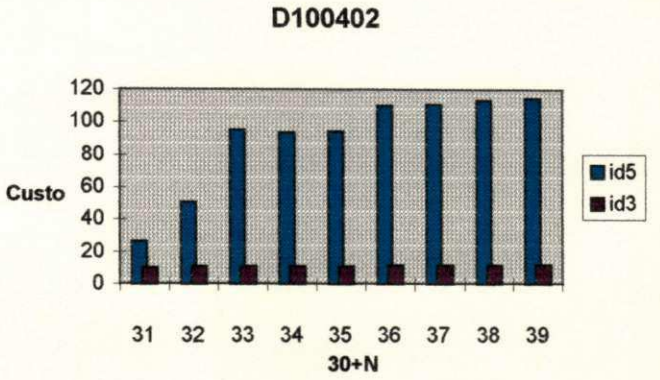

GRÁFICO 4.4 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=30, em função do número de exemplos processados.

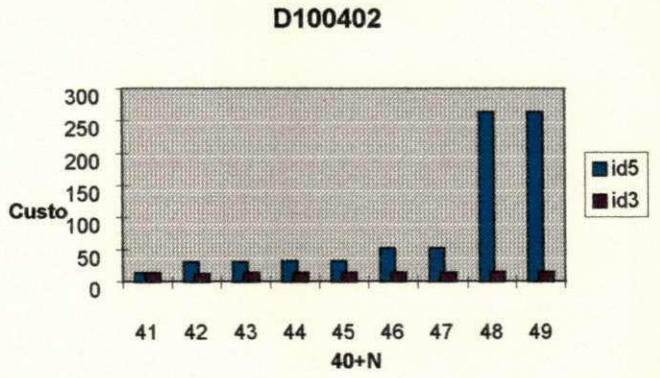

GRÁFICO 4.5 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=40, em função do número de exemplos processados.

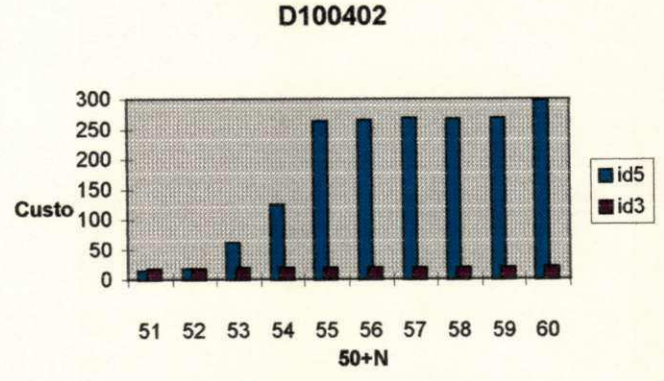

GRÁFICO 4.6 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=50, em função do número de exemplos processados.

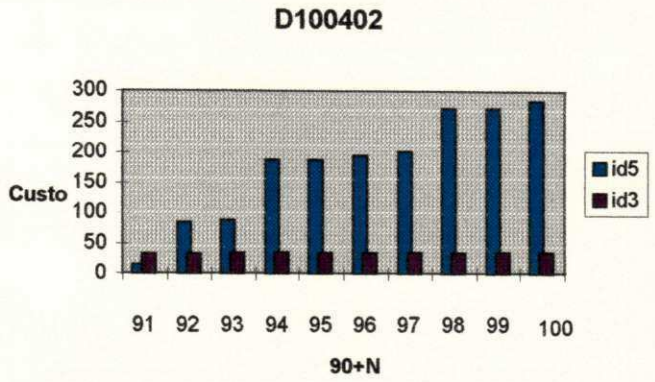

GRÁFICO 4.10 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=90, em função do número de exemplos processados.

Como podemos observar nos GRAF 4.2 à GRAF 4.10, os valores de N (número de exemplos acima do qual o ID3 é melhor que o ID5) são 2, 1, 0, 0, 2, 0, 1, 3, 1, respectivamente, valores esses que correspondem aos apresentados na linha 10 da TAB.4.2.

Para os domínios dinâmicos, encontramos as mesmas constatações verificadas quando do confronto dos resultados da análise teórica com a experimental para os domínios estáticos, ou seja:

- a) Confirmada a existência de um número N de exemplos acima do qual o ID5 é menos eficiente que o ID3;

- b) Confirmada também que a taxa de crescimento do custo do ID3 em função do número n de exemplos processados é constante, embora que para certos domínios e, principalmente, para pequenos valores de n apresente oscilações;

- c) A curva representativa dos custos do ID5 em função do número de exemplos processados, *apresenta variações bruscas e aleatórias*, não apresentando portanto, uma taxa de crescimento constante como afirma a análise teórica (baseada em métricas definidas pelos autores e considerando o pior caso). Entretanto da mesma forma que para os domínios estáticos, também em domínios dinâmicos essa curva, a exemplo da análise teórica, em

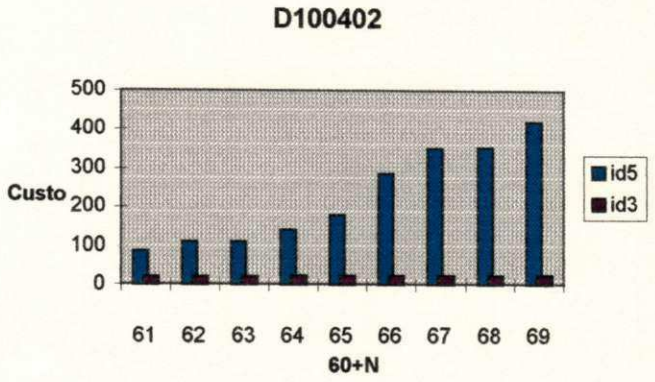

GRÁFICO 4.7 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=60, em função do número de exemplos processados.

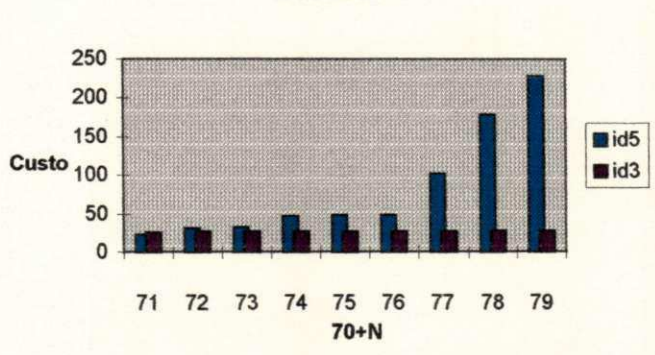

GRÁFICO 4.8 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=70, em função do número de exemplos processados.

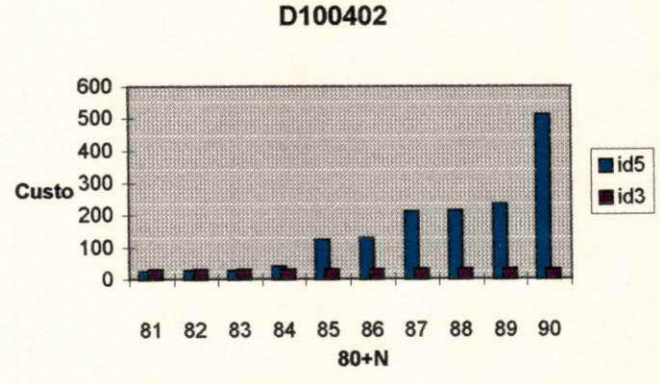

GRÁFICO 4.9 - Custo do ID3 e ID5 para o domínio dinâmico (D100402) com i=80, em função do número de exemplos processados.

média, apresenta uma tendência ascendente e um crescimento bem mais acentuado que aquele apresentado pela curva do ID3.

Analisando os resultados do confronto da análise teórica com a experimental tanto em domínios estáticos como em dinâmicos, três principais aspectos chamaram nossa atenção:

- O primeiro foi a não confirmação da taxa de crescimento constante do custo do ID5 em função do número de exemplos processados.

- O segundo foi a forma abrupta do incremento desse custo quando do processamento de determinados exemplos.

- O terceiro foi a observação de que em domínios dinâmicos, aconteceram situações em que o valor de N foi zero, significando que o ID3 era sempre mais eficiente que o ID5 naquela situação, para qualquer quantidade de exemplos a serem processados.

Essas três constatações, colocavam em suspeita a eficácia das métricas utilizadas pelos autores dos algoritmos incrementais para estimativa dos seus custos. Resolvemos, então, analisar mais profundamente essas métricas.

4.5 Comportamento do custo em função das métricas definidas

Como mostramos anteriormente, o custo do ID3, apesar de algumas oscilações, em média, apresentou um comportamento compatível com os resultados da análise teórica apresentada, pois, para um mesmo domínio, teve um custo aproximadamente proporcional ao número de exemplos processados.

Entretanto, os resultados experimentais apresentados pelos custos do ID5 foram bastante diferentes dos resultados teóricos. As razões para essas diferenças de comportamento estão nas métricas apresentadas pelo autor do ID5, e utilizadas em nossa análise teórica, que não são as mais representativas do comportamento desses custos.

4.5.1 Componentes do custo do ID5

O ID5, e todos os demais algoritmos incrementais aqui apresentados, têm dois principais módulos funcionais. O primeiro módulo tem a função de incorporar o exemplo na base de conhecimento (árvore) e o segundo tem a função de transformar a árvore após a incorporação do exemplo.

Portanto, podemos dividir o custo do ID5 em duas partes. A primeira referente ao custo de incorporação do exemplo na árvore e a segunda referente ao custo de transformação da árvore após incorporação do exemplo.

A incorporação do exemplo na árvore pode ser decomposta em outras duas funções que são: *atualização dos contadores de freqüência* e *classificação do exemplo.*

Por sua vez, a transformação da árvore também pode ser dividida em duas outras funções que são: *cálculo da função de avaliação* e *substituição do atributo do nó.*

Portanto, executar o ID5 significa ativar essas quatro funções principais, que têm cada uma delas um custo associado. Assim, podemos dividir os custos do ID5 em quatro parcelas principais, que são:

1. O *custo para atualização dos contadores de frequência* ($C\mathcal{A}$)
2. O *custo para classificação do exemplo* ($C\mathcal{B}$)
3. O *custo para cálculo da função de avaliação* (CC)
4. O *custo para substituição do atributo do nó* ($C\mathcal{D}$).

As funções responsáveis pelos custos ($C\mathcal{A}$), ($C\mathcal{B}$) e (CC) são executadas sempre que cada novo exemplo é processado, enquanto que a função responsável pelo custo ($C\mathcal{D}$) é executada somente nos casos em que após a incorporação do exemplo na árvore, existir um ou mais atributos que devem ser substituidos por não serem mais os melhores para o nó em questão.

Apresentamos na TAB 4.3. alguns resultados da apuração dos custos referentes a cada uma dessas parcelas quando da geração de bases de conhecimento através do ID5.

As parcelas de custo (CA) e (CC) foram aquelas consideradas como métricas pelos autores dos algoritmos incrementais para efeito de estimativa de custo e portanto foram incorporadas em nossa análise teórica.

Como mostra a TAB.4.3, a parcela (CD) apresentou em média um valor que representa 87% do custo do ID5 e a parcela (CB) apresentou em média um valor que representa 11%, o que demonstra a grande representatividade destas duas novas parcelas em relação às parcelas (CA) e (CC) que representam as métricas definidas pelo autor do ID5, e utilizadas em nossa análise teórica.

TABELA 4.3 - Valores das parcelas componentes dos custos do ID5

Domínio	Custo ID5 (C_t)	At.Cont. (CA)	Classif. (CB)	Calc.Ent (CC)	Subir At (CD)	CD / C_t	CB / C_t	$(CA+CC) / C_t$
030402007	0,26	0	0,05	0,01	0,20	77%	19%	4%
030502010	0,38	0	0,07	0,00	0,30	79%	18%	0%
040202005	0,29	0	0,05	0,01	0,23	79%	17%	3%
040302006	0,29	0	0,07	0,01	0,21	72%	24%	3%
040302013	2,03	0	0,15	0,03	1,85	91%	7%	3%
040502010	0,61	0	0,07	0,02	0,52	85%	11%	3%
040502013	1,49	0	0,09	0,00	1,40	94%	6%	0%
050202012	2,26	0	0,13	0,03	2,09	92%	6%	1%
050402007	0,76	0	0,07	0,01	0,67	88%	9%	1%
050502009	0,75	0	0,09	0,01	0,65	87%	12%	1%
060202008	0,47	0	0,08	0,00	0,39	83%	17%	0%
060202011	1,73	0	0,10	0,04	1,59	92%	6%	2%
060502012	1,30	0	0,12	0,01	1,17	90%	9%	1%
060502010	0,79	0	0,09	0,02	0,68	86%	11%	3%
070502013	1,14	0	0,11	0,02	1,00	88%	10%	2%
090502013	1,83	0	0,15	0,02	1,61	88%	8%	1%
100202010	1,43	0	0,08	0,03	1,30	91%	6%	2%
100302012	1,24	0	0,10	0,04	1,10	89%	8%	3%
170202014	2,53	0	0,17	0,08	2,27	90%	7%	3%
170302014	1,07	0	0,09	0,08	0,89	83%	8%	7%
200402013	4,37	0	0,16	0,07	4,13	95%	4%	2%
Média						87%	11%	2%

Portanto, o custo do ID5 fica melhor representado pela seguinte expressão:

$$Custo\ (ID5) = CA + CB + CC + CD$$

As parcelas CA e CC já foram definidas anteriormente na análise teórica.

A parcela do custo CB ocorre sempre no processamento de um novo exemplo, e o seu valor cresce quando no processo de classificação do exemplo há necessidade de expansão da àrvore.

A parcela do custo CD ocorre eventualmente no processamento de alguns exemplos. Pode ser vista como uma função degrau, permanecendo constante durante certos intervalos e crescendo bruscamente no processamento de um exemplo que cause uma substituição do atributo de um nó. Assim essa parcela poderia ser definida como:

$$\begin{aligned} CD(n) = C_1 &, \text{para } 0 < n \leq N_1 \\ C_2 &, \text{para } N_1 < n \leq N_2 \\ C_3 &, \text{para } N_2 < n \leq N_3 \\ &\cdot \\ &\cdot \\ C_i &, \text{para } N_{i-1} < n \leq N_i \end{aligned}$$

onde os N_i's são os valores de N para o qual ocorre o "salto". Os valores dos C_i's variam em função do domínio e satisfazem a relação $C_{i-1} < C_i \ \forall\ i$.

Essa parcela é a responsável pelo crescimento abrupto do custo do ID5 quando do processamento de alguns exemplos, situação esta citada anteriormente em nossas constatações após a análise experimental.

Portanto, os valores encontrados pelas métricas apresentadas pelos autores dos algoritmos (parcelas CA e CC), em média, representaram apenas 2% do custo total do ID5, o que demonstra a sua total insignificância para a estimativa de custo dos algoritmos incrementais.

As métricas que deveriam ter sido levadas em consideração, adicionalmente, seriam aquelas associadas ao processo de classificação do exemplo na árvore e ao processo de "subida" do atributo,

respectivamente parcelas *CB* e *CD*, que representaram em média 98% dos custos do ID5.

Adicionalmente aos GRAF.4.1 à GRAF.4.10, que apresentam os pontos em que as curvas dos custos dos algoritmos ID3 e ID5 se cruzam (valores de N), apresentamos o GRAF.4.11, referente ao domínio D200502, para dar uma idéia mais ampla do comportamento do crescimento das curvas representativas dos custos do ID3 e do ID5, e ilustrar algumas constatações que fizemos. O GRAF.4.12 é uma ampliação de um setor do GRAF.4.11 para mostrar mais claramente o valor de N (no caso N=5) em que o ID5 passa a ser menos eficiente que o ID3.

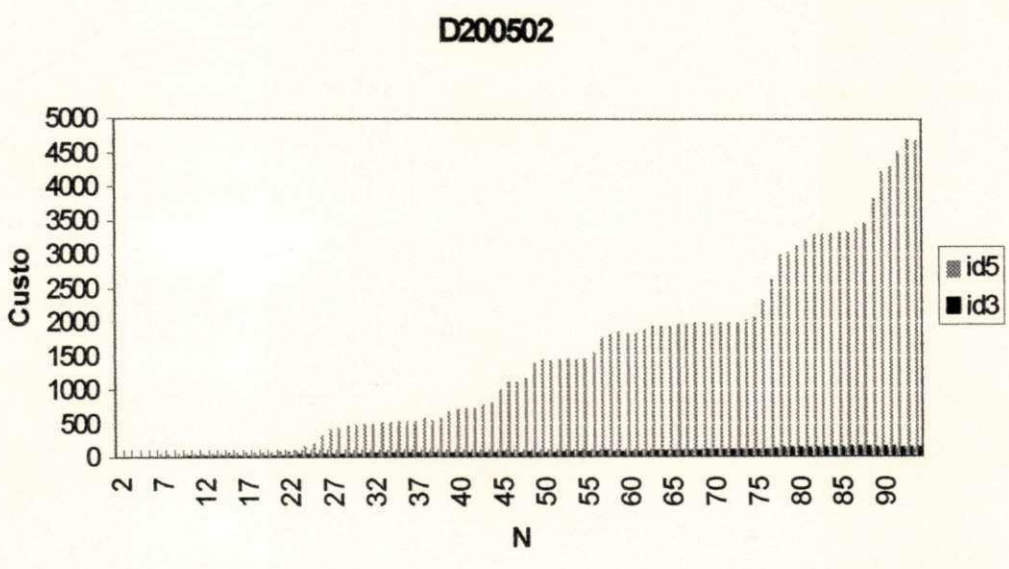

GRÁFICO 4.11 - Comparativo dos custo dos algoritmos ID3 e ID5 em função do número de exemplos N, em um domínio com 20 atributos e 5 valores por atributos

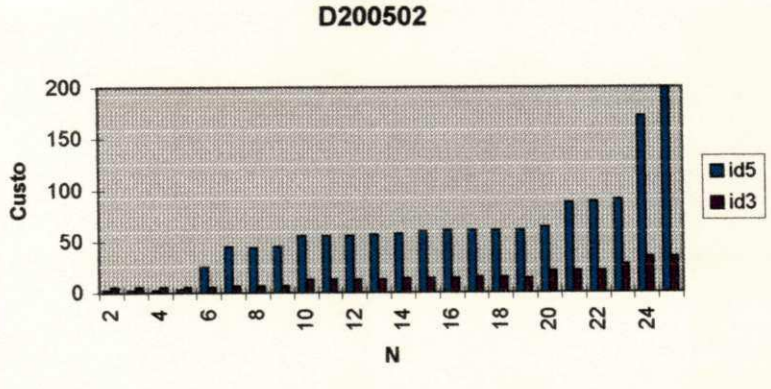

GRÁFICO 4.12 - Ampliação de parte do GRAF. 4.11, com valores de N variando de 2 a 24.

Nos GRAF.4.11 e GRAF.4.12 verificamos facilmente a linearidade da curva representativa dos custos do ID3, e também que a curva dos custos do ID5 não apresenta uma taxa de crescimento constante, pois para determinados valores de n apresenta um aumento brusco de custo. Este aumento brusco acontece quando no processamento do *n-ésimo* exemplo há uma transformação da árvore provocada pela necessidade de troca de um ou mais atributos em um ou mais nós respectivamente.

Essa grande e aleatória variação na taxa de crescimento da curva dos custos do ID5, em função do número de exemplos processados, deve-se à conjugação de dois fatos: a transformação não ocorre para todos os exemplos processados (daí o seu caráter aleatório) e, quando ocorre, em média, ela é responsável por 87% dos custos de processamento de um exemplo (motivo da grande variação). Outro aspecto que o GRAF.4.12 ilustra é o caso de alguns domínios dinâmicos em que o N=0, significando que, nessas situações, o ID3 é mais eficiente que o ID5 para qualquer quantidade de exemplos n >0.

Voltemos nossa atenção para o valor do incremento do custo do ID5, no GRAF.4.12, quando n passa de 23 para 24. Esse incremento representa o custo para o ID5 processar o exemplo de número 24 em domínios dinâmicos com i=23. Verificamos facilmente no GRAF.4.12, que esse incremento (custo para processar o exemplo de número 24 no ID5) é maior que o valor do custo total para o ID3 processar os 24 exemplos. Isso significa que, se fossemos processar esse domínio de uma forma dinâmica com um i=23, teríamos um N=0, ou seja, nessa situação (domínio D200502 com i=23), o ID5 seria sempre menos eficiente que o ID3.

4.6 Conclusões

A análise experimental comparativa dos custos de processamento do ID5 e do ID3 efetuada nos diversos domínios estáticos e dinâmico, cujos resultados estão mostrados nos Apêndice-A e Apêndice-B, nos levou às seguintes conclusões:

a) Confirmando os resultados teóricos, identificamos uma taxa de crescimento constante para os custos do ID3 em função do número n de exemplos;

b) Em divergência com os resultados da análise teórica, identificamos uma taxa de crescimento variável para os custos do ID5 em função de n;

c) Tanto em domínios estáticos como em domínios dinâmicos, existe uma quantidade N de exemplos acima da qual os algoritmos incrementais (ID4,ID5,ID5R e IDL) são menos eficientes que os não incrementais (ID3);

d) Para um mesmo domínio, os custos dos algoritmos incrementais (leia-se ID5) não apresentam um comportamento determinístico.

e) Há um incremento brusco do custo de processamento do ID5 quando do processamento de alguns exemplos. Este incremento acontece quando há necessidade de substituição de algum atributo em um ou mais nós da árvore. Esta situação ocorre de uma forma aleatória em função da situação da árvore e do exemplo a ser processado.

As divergências entre os resultados da análise teórica e da prática para o caso do ID5, citadas nos itens b), d) e e), são consequência das métricas utilizadas na análise teórica dos custos do ID5. Essas métricas foram definidas pelos autores dos algoritmos incrementais para a utilização na análise comparativa de desempenho (custo) com os algoritmos não-incrementais (ID3). Essas métricas foram, a quantidade de *atualização nos contadores de frequência* e a *quantidade de cálculo de E-score*. Elas somadas, apresentaram em nossos experimentos apenas 2%, em média, do custo total do ID5, não sendo portanto representativas desse custo. Os restantes 98% dos custos ficaram divididos em duas funções principais:

- *"subida" de um atributo para um nó visando substituir o atributo do nó* que representou em média 87% dos custos totais do ID5

- *incorporação do exemplo na árvore* que representou, em média, 11% do custo do ID5

Embora as métricas não sendo as mais representativas, a conclusão principal da análise teórica de que há um número de exemplos N a processar acima do qual o ID5 é menos eficiente que o ID3, foi confirmada em todos os experimentos realizados. A única divergência ocorreu na forma de comportamento desse N em função do domínio, que não apresentou um comportamento determinístico e sim aleatório em função do exemplo a processar e da situação da árvore naquele momento.

5 Conclusões e futuros trabalhos

5.1 Conclusões

Vimos que a *incrementabilidade,* em algoritmos indutivos que geram bases de conhecimento representadas através de árvore de decisão, é uma estratégia de processamento que apresenta como característica principal a atualização da árvore de decisão anteriormente gerada, quando ficam disponíveis novos exemplos para processamento. A estratégia de processamento não-incremental, ao contrário, despreza a árvore gerada (conhecimento anteriormente adquirido), e adiciona aos exemplos anteriormente processados os novos exemplos para submete-los novamente a um novo processamento. Como conseqüência dessa filosofia incremental, temos a base de conhecimento sendo atualizada a cada novo exemplo processado.

Essa filosofia surgiu como uma evolução do processo de construção de base de conhecimento tradicional (não incremental), com objetivo principal de trazer maior eficiência (menor custo) a esse processo, pois partia do princípio que, quando surgissem novos exemplos para incorporaração à base de conhecimento, era mais eficiente atualizar o conhecimento adquirido que esquecer todo esse conhecimento e reaprender tudo novamente.

Esses algoritmos incrementais são normalmente baseados em algum outro não-incremental, e buscam gerar a mesma base de conhecimento que esse (não-incremental) geraria se submetido ao mesmo conjunto de treinamento. Portanto, a incrementabilidade em si, não traz nenhuma melhoria em termos de qualidade da base de conhecimento gerada.

Neste trabalho, em nossa análise comparativa com o ID3 (não-incremental), utilizamos o ID5 para representar os incrementais, por apresentar melhor desempenho que os algoritmos ID4, ID5R e IDL.

Os *estudos teóricos* (baseados em métricas apresentadas pelos autores dos algoritmos incrementais), que desenvolvemos e apresentamos nesta dissertação, no capítulo 3, visando comparar o desempenho dos algoritmos incrementais com os não-incrementais, mostraram que, ao contrário do que os autores dos algoritmos afirmaram quando apresentaram seus algoritmos à comunidade científica, os incrementais não atingiram o objetivo a que eles se propunham, pois, em pouquíssimas situações apresentaram-se mais eficientes que os não-incrementais.

Na *análise teórica, para domínios estáticos*, o ID5 (representante dos incrementais) apresentou-se mais eficiente que o ID3 apenas em casos em que a quantidade de exemplos a processar é muito pequena (no máximo igual a 18). Em situações de aplicações reais, em que o número de exemplos a processar é normalmente de centenas ou milhares de exemplos, o ID3 mostrou-se sempre mais eficiente.

Na *análise teórica, para domínios dinâmicos,* o ID5 apresentou-se melhor que o ID3 apenas também quando a quantidade de novos exemplos a processar é muito pequena (oscilando em torno de 18). Para um mesmo domínio, essa quantidade de novos exemplos até onde o ID5 é mais eficiente, cresce com a quantidade de exemplos já incorporados anteriormente à base de conhecimento.

Em nossa *análise experimental* confirmamos a conclusão da análise teórica, que, tanto em situações de domínios dinâmicos como estáticos, existe uma quantidade de exemplos a processar a partir da qual o ID3 é mais eficiente que o ID5. Constatamos também que, para domínios com mais de três atributos (aplicações reais), essa quantidade nunca foi maior que 8, e em diversos casos foi nula. Ou seja, mesmo em domínios dinâmicos, existem situações em que independentemente da quantidade de exemplos a processar o ID3 é sempre mais eficiente que o ID5. A análise experimental nos mostrou também que as métricas apresentadas pelos autores dos algoritmos incrementais e utilizadas em nossa análise teórica, não são as mais representativas dos custos desses

algoritmos, pois, representam em média apenas 2% desses custos. As métricas corretas deveriam levar em consideração o processo de classificação do exemplo e de transformação da árvore após a classificação desse exemplo. O carater aleatório do crescimento do valor do N (quantidade de exemplos acima da qual o ID3 é mais eficiente que o ID5), encontrado em nossa análise experimental, é justificado pelo processo de transformação da árvore que não foi considerado como métrica em nossa análise teórica.

Portanto, nossas análises, tanto teórica como experimental, deixaram claro que as qualidades de eficiência apresentadas pelos autores dos algoritmos incrementais não foram comprovadas, e o algoritmo não-incremental (ID3) na grande maioria dos casos mostrara-se mais eficiente que os incrementais (ID4,ID5,ID5R e IDL).

Isto parece conflitar com a afirmativa de que, a filosofia de construção através de processo incremental (construir reformando) é menos oneroso que o não incremental (construir a partir do início). Em princípio isso é verdadeiro, mas a confirmação disso vai depender da base de conhecimento já existente (o que já existe construído) e dos novos exemplos a incluir (o que deve ser reformado).

Para entendermos melhor o porque desses resultados, vamos fazer uma analogia com a construção civil. É de conhecimento geral que, diante de *uma nova necessidade*, em determinadas situações é mais econômico fazermos uma *reforma* em uma determinada *construção já existente*, e em outras situações, ao contrário, é mais econômico desprezarmos a construção existente e iniciarmos uma *nova construção*. A escolha da alternativa de menor custo, dependerá da construção já existente, e das novas necessidades a serem atendidas.

No nosso caso, a construção já existente é *a base de conhecimento anteriormente gerada*, as novas necessidades são *os novos exemplos disponíveis*, o processo de reforma é o *utilizado pelo algoritmo incremental*, e o processo para fazer uma nova construção é o *utilizado pelo algoritmo não-incremental*. A escolha da melhor alternativa para o nosso caso, dependerá portanto, da árvore inicialmente gerada e dos novos exemplos disponíveis para processamento.

Finalizando, concluimos que:

Os algoritmos incrementais utilizados para gerar base de conhecimento representadas por árvores de decisão (ID4, ID5, ID5R e IDL), não proporcionaram as melhorias de desempenho a que eles se propuseram, pois se mostraram na maioria das situações muito menos eficientes que seus similares não-incrementais (ID3). Entretanto, não podemos descartar essa filosofia de processamento para os algoritmos indutivos, como uma alternativa, pois poderá ser possível a existência de uma outra filosofia de incrementabilidade, diferente da atual, que possa trazer vantagens. Além do mais, a incrementabilidade propicia, de uma forma natural, a evolução de uma base de conhecimento, não requerendo o acúmulo de todos os exemplos obtidos ao longo do tempo. Para certos domínios, esse acúmulo poderá assumir proporções gigantescas, trazendo, assim, sérios problemas com a escolha de conjuntos de treinamento representativos.

5.2 Futuros trabalhos

Ao longo desta dissertação, identificamos pontos que mereciam ser estudados ou aprofundados, mas que infelizmente, por questões de limitação de escopo do trabalho ou mesmo de tempo, não nos foi possível desenvolver, mas que deveriam ser estudados visando dar seqüência a este trabalho. Alguns deles relacionamos a seguir:

. Fazer uma análise comparativa de desempenho e de qualidade da base gerada pelos algoritmos incrementais e não-incrementais geradores de regras (família dos AQs);

. Estudar outras alternativas de modelos visando criar algoritmos incrementais geradores de árvore de decisão que possam realmente ser eficientes;

. Alterar a estrutura de dados geradas por algoritmos não-incrementais, visando compatibiliza-la para uso de algoritmos incrementais, com objetivo de utiliza-los alternadamente em um mesmo domínio;

. Fazer estudos comparativos entre o IDL e o ID3, com relação a qualidade da base de conhecimento gerada, visando analisar o impacto do critério de relevância topológica sobre essa base de conhecimento;

. Detalhar o comportamento do custo dos incrementais (ID4, ID5, ID5R e IDL) em função do domínio levando em consideração as novas métricas identificadas neste trabalho;

. Desenvolver estudos teóricos para os custos dos incrementais e não incrementais para um caso médio.

APÊNDICE A - Resultados da análise experimental em domínios estáticos

Apresentamos neste apêndice os valores dos custos dos algorimos ID3 e ID5 encontrados nos experimentos executados em domínios estáticos, visando identificar em cada um desses domínios, a quantidade de exemplos a partir da qual o ID5 apresenta-se menos eficiente que o ID3.

Os diversos tipos de domínios analisados, estão caracterizados por suas quantidades de atributos, valores por atributo e classes. Para *identificação de cada domínio* utilizamos a codificação "*Daabbcc*" onde:

D identifica um arquivo de domínio,
aa representa a quantidade de atributos,
bb representa a quantidade de valores por atributos e
cc identifica a quantidade de classes.

Para cada domínio analisado, apresentamos uma tabela contendo na primeira linha a identificação do domínio e os títulos ID5 e ID3. Para as demais linhas temos:

- na primeira coluna, a quantidade de exemplos processados,
- nas segunda e terceira colunas, respectivamente o custo do ID5 e ID3 para aquela quantidade de exemplos.

Cada valor de custo do algoritmo apresentado na tabela, corresponde à média dos custos de 10 execuções do algoritmo.

D020202	ID5	ID3
1	0	0
2	0	0,1
3	0	0
4	0	0
5	0,4	0
6	0,9	0,1
7	0,9	0
8	1,6	0,1
9	10,3	0,1
10	9,5	0
11	10,4	0,3
12	10,9	0
13	10,4	0
14	10,7	0
15	9,6	0,2

D020302	ID5	ID3
1	0	0,1
2	0	0,2
3	0	0,3
4	0	0,1
5	0,5	0
6	0,8	0
7	1,2	0,1
8	1	0
9	1,2	0,2
10	0,7	0
11	1,4	0
12	1,3	0,2
13	1,1	0
14	1,9	0,2
15	2,1	0

D020402	ID5	ID3
1	0	0
2	0	0,1
3	0	0,1
4	0	0
5	0,6	0
6	1	0,1
7	0,6	0
8	1,4	0,2
9	1,1	0,1
10	1,7	0,1
11	1,6	0
12	1,6	0
13	1,7	0,1
14	2,8	0,6
15	2,4	0,2

D020502	ID5	ID3
1	0	0,1
2	0	0,4
3	0,1	0,2
4	0	0
5	1,1	0,3
6	1,9	0,2
7	17	0,3
8	17,3	0,4
9	19	0,7
10	33,6	0,5
11	33,8	0,5
12	34,9	0,3
13	56,2	0,4
14	57,1	0,9
15	56,7	0,7

D030202	ID5	ID3
1	0	0,1
2	0	0,2
3	1,2	0,2
4	17,6	0,3
5	17,8	0,2
6	18,1	0,3
7	32,4	0,1
8	32	0,3
9	32,7	0,2
10	32,8	0,3
11	32,4	0,4
12	33,2	0,6
13	33,2	0
14	40,4	0,6
15	41,1	0,3

D030302	ID5	ID3
1	0	0
2	0	0,2
3	0	0,1
4	0,5	0,1
5	1,4	0,3
6	1,3	0,1
7	1,8	0,2
8	2,9	0,2
9	2,7	0,4
10	9,4	0,5
11	16,6	0,6
12	17,1	0,7
13	16,6	0,5
14	16,8	0,8
15	17,5	0,5

D030402	ID5	ID3
1	0	0,1
2	0	0,2
3	0,9	0,2
4	1,2	0,3
5	2	0,1
6	2,4	0,6
7	15,1	0,4
8	39,8	0,7
9	55,3	0,6
10	55	0,7
11	56,3	0,6
12	60,6	0,4
13	60,8	0,6
14	60,4	0,6
15	61,7	0,3

D030502	ID5	ID3
1	0	0,4
2	0	0,2
3	1,1	0,1
4	1,7	0,3
5	1,2	0,3
6	1,7	0,4
7	1,9	0,2
8	2,7	0,5
9	3	0,4
10	20,9	0,6
11	22,4	0,3
12	22,6	0,5
13	22,1	0,7
14	22,6	0,9
15	22,8	1,2

D040202	ID5	ID3
1	0	0,1
2	0,6	0,2
3	6,8	0,2
4	7,6	0
5	15,7	0,2
6	24,1	0,2
7	23,7	0,4
8	23,8	0
9	32,7	0,1
10	41,7	0
11	41,4	0,2
12	41,7	0
13	42,1	0,1
14	42,3	0,3
15	42,5	0,5

D040302	ID5	ID3
1	0	0
2	0,6	0,4
3	1,1	0
4	1,7	0,4
5	1,7	0,4
6	15,3	0,1
7	15,2	0,2
8	15,2	0,3
9	41,7	0,5
10	41,8	0,1
11	72,9	0,4
12	78,7	0,4
13	117,2	0,5
14	145,1	0,8
15	166	0,8

D040402	ID5	ID3
1	0	0,2
2	0,5	0,3
3	1,1	0,2
4	1,4	0,1
5	1,3	0,2
6	2,5	0,5
7	2,4	0,7
8	25,3	0,2
9	53,8	0,4
10	53,6	0,4
11	75,2	0,8
12	110,5	1,1
13	150,2	1
14	150,2	0,8
15	172,9	0,9

D040502	ID5	ID3
1	0	0,1
2	0,3	0,5
3	1,3	0,1
4	1,8	0,7
5	1,3	0,6
6	2,2	0,8
7	2,6	0,9
8	2,5	1
9	3,6	1,2
10	35,5	1,4
11	50,4	1,1
12	51,5	1,1
13	89,3	1,1
14	89,1	1,1
15	88,9	1,3

D050202	ID5	ID3
1	0	0
2	0	0,1
3	0,7	0
4	13,1	0,1
5	12,9	0
6	19,9	0,3
7	21,7	0,5
8	29,5	0,4
9	42,5	0,8
10	75,1	0,5
11	76,5	0,8
12	136	0,8
13	160,3	1
14	169	1,1
15	168,9	1,2

D050302	ID5	ID3
1	0	0,1
2	0	0,4
3	0,8	0,2
4	1,2	0,1
5	1,8	0,3
6	20,2	0
7	39	0,2
8	40,3	0,2
9	48,2	0,1
10	48,7	0,5
11	47,9	0,6
12	55,7	1,1
13	54,9	0,7
14	54,6	1
15	55,8	0,8

D050402	ID5	ID3
1	0	0,1
2	0,1	0,3
3	0,5	0,3
4	1,2	0,3
5	17,2	0,6
6	18	0,5
7	41,6	0,5
8	41,8	0,5
9	56,8	0,7
10	57,1	0,5
11	56,7	1
12	58,6	1,3
13	59,1	1,2
14	58,8	1,3
15	59,5	1,4

D050502	ID5	ID3
1	0	0,6
2	0,1	0,2
3	1,1	0,7
4	15,7	0,5
5	16,1	0,8
6	16,2	0,5
7	17,2	1
8	17,6	1,1
9	42,7	1,5
10	43,1	1,5
11	52,7	1,4
12	62,8	1,6
13	73,2	1,4
14	73	1,8
15	74,9	1,8

D060202	ID5	ID3
1	0	0,1
2	0,6	0,3
3	0,7	0,4
4	0,9	0,2
5	0,8	0
6	2,1	0,2
7	2	0,4
8	26,7	0,4
9	42,3	0,3
10	42,8	0,5
11	103,3	0,5
12	105	0,6
13	117,5	0,5
14	116,5	0,7
15	117,5	0,9

D060302	ID5	ID3
1	0	0,1
2	1	0,3
3	12,3	0,2
4	27,3	0,3
5	28,1	0
6	37	0,4
7	45,8	0,3
8	61,7	0,5
9	86	0,9
10	87,5	0,7
11	88,2	1,2
12	117,1	1
13	135,3	1,2
14	147,8	1,5
15	157,9	1,7

D060402	ID5	ID3
1	0,1	0,3
2	0,7	0,5
3	1,3	0,6
4	16,2	0,2
5	15,8	0,5
6	16,9	0,9
7	17,5	0,9
8	41,6	1,3
9	65,9	1,1
10	66,6	1
11	66,9	1,5
12	91,1	2,1
13	138,6	1,3
14	138,1	1,6
15	147,8	1,7

D060502	ID5	ID3
1	0	0,6
2	0,8	0,5
3	7,3	0,6
4	15,9	0,7
5	16,4	0,6
6	18,2	1,2
7	18,2	1,3
8	17,7	1,5
9	18,5	1,3
10	46,6	2,1
11	46,7	2,4
12	76,6	2,2
13	78	2,3
14	76,4	2,2
15	86,8	2,6

D070202	ID5	ID3
1	0	0,1
2	1	0
3	0,8	0,2
4	0,9	0,1
5	0,5	0,3
6	0,9	0,1
7	1,9	0,6
8	2	0,6
9	3,3	0,9
10	29,2	0,9
11	61,6	1,1
12	74,9	1,1
13	75,8	1,1
14	83,5	1,3
15	94,8	1,3

D070302	ID5	ID3
1	0	0,3
2	0,7	0,2
3	0,8	0,2
4	1,1	0,2
5	13,2	0,5
6	32,7	0,7
7	57,8	0,9
8	87,2	1
9	98,2	0,8
10	110,4	1,6
11	110,4	1
12	110,5	1,5
13	110,7	1,5
14	109,6	1,6
15	111,1	1,6

D070402	ID5	ID3
1	0	0,4
2	0,9	0,7
3	1,1	0,6
4	1,3	0,7
5	1,4	0,6
6	21,5	0,6
7	27,3	1,1
8	40,7	1,2
9	39,8	1,3
10	74,3	1,9
11	102,3	1,8
12	103,1	1,9
13	144,6	1,6
14	187,9	2,1
15	205,1	1,8

D070502	ID5	ID3
1	0,1	0,9
2	0,9	1
3	1,2	0,8
4	1,8	0,8
5	1,8	1
6	2,1	1
7	18	1,8
8	18,4	1,9
9	18,7	2,1
10	19,7	2,7
11	19,6	2,9
12	20,8	2,9
13	67,6	2,9
14	67,5	3
15	67,3	3,1

D080202	ID5	ID3
1	0	0,2
2	0,9	0,2
3	12,9	0
4	19,5	0,1
5	28,5	0,5
6	36,9	0,4
7	37,7	0,4
8	37,4	0,8
9	62,6	0,8
10	62,2	1
11	74,5	0,6
12	86,8	1,3
13	87,5	1,5
14	132	1,7
15	132,2	1,3

D080302	ID5	ID3
1	0,1	0,3
2	0,8	0,5
3	0,9	0,3
4	16,5	0,4
5	17,2	0,4
6	30,3	0,6
7	30	0,9
8	30,7	0,9
9	30,6	0,9
10	30,4	1,4
11	30,7	1,3
12	43,1	1,2
13	54,9	1,1
14	56,4	0,9
15	57,7	1,8

D080402	ID5	ID3
1	0	0,7
2	1,1	0,5
3	1,4	0,8
4	1,8	0,7
5	1,6	0,6
6	3	1,3
7	23,4	1,7
8	24	1,5
9	24,5	2,2
10	25,5	2,4
11	39,9	2,2
12	39,6	2,4
13	78,9	2,4
14	135,1	2,7
15	209,8	2,4

D080502	ID5	ID3
1	0	1,1
2	0,9	1
3	1,3	0,7
4	2	1,2
5	1,6	1
6	2,4	1,3
7	2,4	1
8	3,6	2,5
9	33,7	2,1
10	34,1	2,4
11	42,6	2,4
12	43,4	2,7
13	44,1	2,6
14	67,6	2,6
15	92,3	3,8

D090202	ID5	ID3
1	0	0,2
2	0,9	0,2
3	12,9	0,3
4	13,6	0,2
5	13,1	0,2
6	18,9	0,6
7	45,8	0,6
8	55,7	0,5
9	55,5	0,6
10	70	1,2
11	119,8	1,4
12	120,5	1,2
13	133,5	1,9
14	134,5	2,3
15	133,7	2,1

D090302	ID5	ID3
1	0	0,1
2	0,9	0,4
3	0,9	0,3
4	1,3	0,4
5	1,4	0,5
6	1,5	0,2
7	2,5	0,8
8	29	0,8
9	29	1,3
10	51,8	1,7
11	73,3	1,7
12	74,5	1,7
13	131	1,8
14	130,3	1,6
15	149,6	2,5

D090402	ID5	ID3
1	0	0,7
2	1	0,9
3	1	0,8
4	1,3	0,9
5	1,6	0,5
6	1,5	1
7	2,3	1,9
8	41,6	1,9
9	75,8	1,7
10	75,2	1,8
11	76,2	2
12	110,5	2,8
13	121	3,6
14	121,3	3,7
15	144	4,1

D090502	ID5	ID3
1	0,2	1,2
2	1,1	1,2
3	1,2	1,6
4	1,5	1,4
5	1,9	1,5
6	2,4	1,3
7	3,2	2,5
8	30,8	2,9
9	41	2,8
10	57,1	2,9
11	57,5	3
12	59,1	4,2
13	108,5	5,7
14	108,6	6,2
15	120,7	6

D100202	ID5	ID3
1	0	0,3
2	1	0,4
3	1,1	0,3
4	0,8	0,2
5	1,2	0,3
6	0,9	0,5
7	1,6	0,8
8	2,8	1
9	3,2	0,7
10	84,2	1,9
11	83,8	1,7
12	129,2	1,5
13	131,8	1,7
14	158,6	1,7
15	171,6	1,8

D100302	ID5	ID3
1	0	0,2
2	1	0,3
3	7,9	0,5
4	7,9	0,4
5	7,5	0,5
6	7,8	0,7
7	9,3	1,3
8	9,4	1,4
9	9,5	1,2
10	19,1	1,9
11	18,6	2,1
12	73,9	2,1
13	73,9	2,1
14	74,9	2,7
15	75,1	2,5

D100402	ID5	ID3
1	0	0,6
2	0,8	0,7
3	7,3	0,8
4	7,5	0,8
5	7,6	1,2
6	8,5	1,3
7	9,5	2,4
8	9,5	2,1
9	9,8	2,3
10	18,8	3,4
11	20	3,3
12	20,3	3,6
13	34,3	3,3
14	41,2	4,1
15	41,6	4,3

D100502	ID5	ID3
1	0	1
2	0,7	1,4
3	7,2	1,6
4	8,3	1,7
5	8,3	1,9
6	8,8	1,6
7	39,2	3,2
8	57,6	3,4
9	88,9	3,6
10	117	3,7
11	117	3,5
12	148,5	5,3
13	149,7	5,5
14	150	5,4
15	197,6	7

D110202	ID5	ID3
1	0	0,1
2	0,7	0,3
3	0,9	0,3
4	13	0,5
5	13,5	0,6
6	12,9	0,2
7	13,8	0,6
8	14,1	1
9	22,8	1,3
10	67,2	1,6
11	96,9	1,8
12	123,6	1,4
13	137,8	1,6
14	149,2	1,8
15	152	1,6

D110302	ID5	ID3
1	0	0,4
2	1	0,6
3	1,2	0,6
4	1,7	0,4
5	18,6	0,9
6	18,8	0,5
7	19	0,9
8	18,6	1,1
9	19,7	0,8
10	33,2	1,7
11	70,9	1,6
12	72,4	1,8
13	71	2
14	72,3	1,9
15	123	2,9

D110402	ID5	ID3
1	0	0,9
2	0,9	1,1
3	1,2	0,9
4	1,6	1,2
5	16,8	1,8
6	17,6	1,3
7	39,1	2,4
8	65,8	2,6
9	95,7	2,5
10	128	3,9
11	165,1	3,9
12	173,5	4,4
13	173,5	4
14	174	3,9
15	238,6	5,4

D110502	ID5	ID3
1	0	1,6
2	0,7	1,6
3	1,5	1,8
4	1,5	1,6
5	2	2
6	2,7	2,1
7	3	2,1
8	3	2,3
9	3,5	2,5
10	34,7	2,5
11	40,7	4
12	41,9	4
13	40,9	4,4
14	42,8	4,4
15	42,5	4,5

D120202	ID5	ID3
1	0	0,1
2	0,8	0,7
3	1,1	0,5
4	0,9	0,5
5	1,1	0,5
6	13,8	0,6
7	14,1	0,3
8	15,6	1,6
9	16	1,2
10	15,7	1,2
11	16	1,1
12	17	1,8
13	44,1	1,3
14	102,5	2,6
15	146,1	3

D120302	ID5	ID3
1	0,1	0,6
2	0,8	0,7
3	1,5	0,9
4	1,6	0,9
5	1,4	0,8
6	19,5	0,7
7	19,1	1
8	30,4	1,9
9	31,3	1,8
10	32,6	2,8
11	44,4	2,6
12	44	2,9
13	45	2,7
14	55,9	2,8
15	57,1	2,9

D120402	ID5	ID3
1	0	0,9
2	0,9	1,2
3	1,6	1,3
4	1,4	1,4
5	1,6	1,4
6	21,7	1,4
7	22,7	1,7
8	45,2	2,9
9	44,8	2,9
10	73,8	3,2
11	103,6	3,3
12	151,1	4,5
13	164,9	4,6
14	167,6	6,1
15	167	6

D120502	ID5	ID3
1	0	1,6
2	0,8	2,1
3	1,5	2,1
4	1,8	2,1
5	2,2	2,2
6	24,3	2,3
7	23,8	2,6
8	47,2	4,3
9	47,4	4,3
10	52,7	6,8
11	55	9
12	91,5	7,1
13	109,2	7,6
14	108,8	7,2
15	116,6	7,7

D130202	ID5	ID3
1	0	0,3
2	0,1	0,4
3	0	0,6
4	0	0,1
5	1,1	0,5
6	1,6	1
7	14,3	0,9
8	14,2	1
9	23,5	1,3
10	49,9	1,2
11	49,7	1,3
12	50,3	1,5
13	50,9	1,7
14	50,6	1,7
15	65,8	2

D130302	ID5	ID3
1	0	0,3
2	0,2	0,7
3	0,1	0,8
4	0,1	1,2
5	1,3	1
6	1,4	1
7	1,6	0,9
8	13,9	0,9
9	14,6	1,7
10	25,5	2,8
11	25,5	2,9
12	45,3	3
13	45,4	3,1
14	45,5	3,1
15	46,9	3,3

D130402	ID5	ID3
1	0,1	1,5
2	0	1,3
3	0,1	1,5
4	0	1,5
5	1,3	1,7
6	1,9	1,4
7	2	1,7
8	2,4	1,8
9	3,5	3,4
10	17,9	3,6
11	17,3	3,7
12	18,4	3,9
13	34	4
14	33,8	3,9
15	89,8	5,8

D130502	ID5	ID3
1	0,1	2
2	0,1	2,4
3	0,1	2,3
4	0	2,3
5	1,8	2,4
6	1,8	2,5
7	2,9	2,5
8	27,4	2,8
9	33,6	5,3
10	55,5	5,5
11	90,8	5,7
12	91,5	5,8
13	137,5	5,7
14	138,9	6
15	173,8	8,7

D140202	ID5	ID3
1	0,1	0,3
2	0	0,3
3	0,2	0,1
4	0,1	0,5
5	1	0,3
6	1,2	0,5
7	1,6	1,1
8	17	1,6
9	16,9	1,8
10	32,2	1,4
11	92,9	2,1
12	91,9	2,2
13	120,1	3,2
14	156,8	3,4
15	169,2	3

D140302	ID5	ID3
1	0,1	0,6
2	0	0,9
3	0,1	1
4	0,1	0,9
5	1,1	1,2
6	1,6	1,1
7	12,9	1,1
8	13,5	1
9	13,1	1,3
10	13,7	0,9
11	13,7	1,3
12	14	1,5
13	15	2,9
14	15,6	2,3
15	16,4	2,8

D140402	ID5	ID3
1	0	1,5
2	0	1,5
3	0	1,7
4	0,2	1,8
5	1,7	1,8
6	1,6	2,1
7	20,7	1,9
8	20,2	2,2
9	21,1	2,2
10	21,4	1,9
11	21,7	2,2
12	21,9	2,3
13	23,7	4,1
14	38,8	4,4
15	39	4,4

D140502	ID5	ID3
1	0	2,4
2	0	2,6
3	0,1	2,7
4	0,2	3
5	1,5	2,8
6	2	3
7	2,6	3,3
8	2,8	3,2
9	24,4	3,1
10	24,8	3,1
11	24,7	3,3
12	25,3	3,5
13	26,2	3,6
14	27	6,4
15	28,3	7

D150202	ID5	ID3
1	0	0,3
2	0,1	0,3
3	0,1	0,5
4	0,2	0,7
5	0,2	0,5
6	0	0,3
7	0,1	0,7
8	1,8	1,1
9	56,3	1,7
10	72	2,6
11	72,6	2,3
12	72,4	2,4
13	75,2	3,5
14	147,2	3,9
15	156,5	4

D150302	ID5	ID3
1	0,1	1,1
2	0,1	0,7
3	0	0,9
4	0	1
5	0	1,1
6	0	1,2
7	0	1,2
8	2,6	2,4
9	39,4	2,6
10	39,1	3
11	67,6	4,3
12	67,1	4,2
13	84	4
14	84,1	4,5
15	84,4	4,2

D150402	ID5	ID3
1	**0,1**	**1,5**
2	**0**	**1,7**
3	**0,1**	**1,7**
4	**0,1**	**1,7**
5	**0,1**	**2**
6	**0,1**	**2**
7	**0,4**	**2,4**
8	**2**	**2,2**
9	**3,6**	**4,6**
10	23,5	**6,6**
11	52,5	**5**
12	51,6	**4,8**
13	54,1	**6,7**
14	55,1	**9**
15	55,4	**9**

D150502	ID5	ID3
1	**0**	**2,8**
2	**0**	**3**
3	**0**	**3**
4	**0,2**	**3,1**
5	0,1	**3,4**
6	0	**3,4**
7	**0,3**	3,5
8	**2**	3,9
9	3,9	6,9
10	31,7	7,3
11	39,5	7,2
12	40,6	6,7
13	40,7	11,5
14	94,9	11,2
15	95,4	11,2

D160202	ID5	ID3
1	**0**	0,4
2	**0**	0,3
3	**0**	0,5
4	**1**	0,7
5	8,1	0,9
6	21,1	0,9
7	22,6	2,1
8	46,1	2,2
9	**86,3**	1,8
10	**102**	**2,3**
11	150,7	2,8
12	183,1	3,5
13	232,8	2,9
14	267,5	3,9
15	305,7	3,8

D160302	ID5	ID3
1	**0,1**	**1**
2	**0,1**	**1**
3	**0**	**1**
4	**1,5**	1
5	**20**	1,4
6	20,7	1,3
7	42,5	2,7
8	43,9	2,8
9	51,6	2,8
10	97,8	4,1
11	98,6	4,1
12	132,7	4,1
13	134,6	4,5
14	156,3	4,2
15	176,7	4,5

D160402	ID5	ID3
1	0,1	1,9
2	**0**	1,9
3	**0,1**	2,1
4	**1,9**	2,3
5	**1,9**	2,4
6	**22,2**	2,4
7	**44,5**	2,5
8	**46,5**	5
9	**72,5**	5,1
10	**73**	4,8
11	104,2	5,2
12	104,7	5,4
13	113	5,7
14	112,9	5,5
15	136	5,7

D160502	ID5	ID3
1	0	3
2	**0**	3,1
3	**0**	3,2
4	**1,8**	3,5
5	19,1	4
6	39,9	3,8
7	60,9	7,5
8	88,8	7,8
9	115,8	8,1
10	154,9	8,3
11	154,7	8,3
12	155,8	8,4
13	165	8,5
14	162,4	8,8
15	164,7	8,8

D170202	ID5	ID3
1	**0**	**0,5**
2	**0**	**0,5**
3	**0,8**	**0,7**
4	**1,2**	**0,9**
5	**13,3**	**0,9**
6	**22,3**	**1,5**
7	**22,6**	**1,7**
8	**49,7**	**2,3**
9	**63,3**	**2,2**
10	63,1	**2,4**
11	63,3	**2,5**
12	65,9	**4,2**
13	66	**4**
14	143,3	**3,9**
15	157,4	**3,9**

D170302	ID5	ID3
1	0	**0,8**
2	0,2	**1,1**
3	**1,5**	**1,1**
4	**1,5**	**1,5**
5	**1,7**	**1,4**
6	**2,2**	**1,3**
7	**2,3**	**1,7**
8	**3,4**	**3,2**
9	**4,1**	**3**
10	**4,4**	**3**
11	**4,5**	**3,2**
12	29,5	4,8
13	29,4	5,3
14	63	**5,1**
15	82,4	5,3

D170402	ID5	ID3
1	**0**	**2,1**
2	**0,1**	**2,1**
3	**1,4**	**2,3**
4	**2,1**	**2,6**
5	**2,1**	**2,6**
6	**15,4**	**2,7**
7	**16,2**	**2,8**
8	**40,9**	**5,3**
9	**86,6**	**5,7**
10	**123,8**	**5,7**
11	**122,6**	**5,8**
12	**140,4**	**6**
13	**140,3**	**6,1**
14	**199,8**	**6,4**
15	**198,9**	**6,5**

D170502	ID5	ID3
1	**0**	**3,4**
2	**0,2**	**3,6**
3	**1,7**	**3,6**
4	**2,5**	**3,8**
5	**2,8**	**4,2**
6	**3,1**	**3,9**
7	**26,1**	**4,3**
8	**26,1**	**4,7**
9	**53,4**	**8,7**
10	**83,1**	**9**
11	**83,3**	**9,1**
12	**117,9**	**9,5**
13	**117,7**	**9,6**
14	**118,7**	**9,7**
15	**119,2**	**9,9**

D180202	ID5	ID3
1	0	0,8
2	**0**	0,9
3	**0,9**	0,9
4	**13,9**	0,4
5	**13,5**	0,7
6	**20,3**	1,6
7	**52,5**	2
8	**66,6**	2,2
9	**66,3**	2,6
10	**73,3**	2,6
11	102,1	2,4
12	117,6	3,4
13	130,5	3,2
14	**149**	3,5
15	231,6	4,2

D180302	ID5	ID3
1	**0**	1,2
2	**0**	1,3
3	**1,1**	1,6
4	**1,4**	2
5	1,7	1,9
6	16,4	1,5
7	36,6	3,2
8	36,8	3,3
9	48,9	3,7
10	76,6	3,5
11	109,2	5,4
12	134	3,7
13	134,2	4,3
14	135	4,2
15	134,9	4,1

D180402	ID5	ID3
1	0	2
2	0,2	2,4
3	1,5	2,7
4	15,6	2,7
5	15,7	3
6	15,7	3
7	16	3,1
8	40,5	6
9	41,3	6,2
10	49,2	6,5
11	99,3	6,4
12	117,1	6,8
13	118,9	9,6
14	119,7	9,6
15	119,7	9,7

D180502	ID5	ID3
1	0,1	3,8
2	0,1	3,8
3	1,7	4,1
4	1,8	4,3
5	2,5	4,5
6	19,3	4,9
7	43,6	5,1
8	66,5	9,5
9	76,1	9,7
10	106,9	10,3
11	119,5	10,2
12	118,8	10,6
13	136,9	11
14	177,5	15,6
15	208,4	10,9

D190202	ID5	ID3
1	0	0,6
2	0,1	0,9
3	1,1	0,6
4	1,1	0,6
5	1,1	1,3
6	1,7	0,8
7	1,7	0,9
8	1,6	0,9
9	2,1	0,9
10	2,3	1,1
11	4	2,9
12	16	2,9
13	16,4	2,9
14	16,8	3,2
15	17,2	3

D190302	ID5	ID3
1	0	1,3
2	0	1,5
3	1	1,7
4	1,7	1,7
5	1,7	1,4
6	20,9	1,8
7	20,7	1,6
8	21,6	1,7
9	21,6	1,6
10	22	1,8
11	30,1	5,8
12	54,1	5,9
13	95,3	5,9
14	95,8	6,4
15	105,3	6,3

D190402	ID5	ID3
1	0,1	2,6
2	0,1	2,5
3	1,5	2,9
4	1,6	3
5	2,3	3,4
6	2,4	3,4
7	2,6	3,5
8	24,8	3,5
9	25,7	3,8
10	26,1	3,9
11	38,1	7,4
12	70,9	7,1
13	84,2	7,4
14	85,1	7,4
15	84,8	7,5

D190502	ID5	ID3
1	0	3,9
2	0	4,4
3	1,3	4,7
4	1,6	4,6
5	2,6	5
6	17	5,5
7	16,8	5,6
8	38	5,8
9	38,1	5,9
10	39	6
11	47,7	11,7
12	48,5	11,7
13	49,3	11,8
14	50,4	12
15	50,3	12,1

D200202	ID5	ID3	D200402	ID5	ID3
1	0,1	0,8	1	0	2,9
2	0,8	0,9	2	1,3	3,1
3	0,9	0,9	3	1,5	3,3
4	1	1,2	4	1,7	3,2
5	14,1	0,9	5	1,9	3,5
6	14,9	1,6	6	28,5	6,7
7	47	1,7	7	56,2	7,1
8	62	1,8	8	62,9	7,6
9	77,1	3	9	94,2	7,7
10	79,5	4,5	10	129,9	11,1
11	81,1	4,7	11	172,4	11,3
12	79,2	4,7	12	206,9	11,7
13	106,3	5,1	13	256	11,4
14	106,9	5,2	14	257,9	11,8
15	139,2	5,3	15	290,2	11,8

D200302	ID5	ID3	D200502	ID5	ID3
1	0	1,5	1	0	4,5
2	0,9	1,6	2	1,2	5
3	1,2	1,6	3	1,8	5,2
4	1,2	1,7	4	2,4	5,5
5	15,7	2	5	2,9	5,4
6	15,6	1,8	6	24,5	5,9
7	17,1	4,3	7	43,8	6,3
8	30,8	4,6	8	44,2	6,3
9	51,9	4,7	9	44,9	6,1
10	60,1	4,6	10	54,6	12,3
11	75,7	4,8	11	54,7	12,6
12	77,5	4,7	12	55,6	12,7
13	77	4,9	13	55,7	13
14	77,5	4,8	14	58,2	13,3
15	78,5	4,6	15	58,3	13,4

APÊNDICE B - Resultados da análise experimental em domínios dinâmicos

Neste apêndice apresentamos os valores dos custos dos Algoritmos ID3 e ID5 nos diversos domínios dinâmicos analisados.

Da mesma forma que no Apêndice-A, para cada domínio analisado, apresentamos uma tabela contendo na primeira linha, a identificação do domínio e os títulos ID5 e ID3, e nas demais linhas:

- na primeira coluna, a quantidade de exemplos processados,
- na segunda e terceira colunas respectivamente, o custo do ID5 e ID3 para aquela quantidade de exemplos.

Cada valor de custo apresentado corresponde à média dos custos de 10 execuções do algoritmo.

Apresentamos primeiramente os valores para i = 10, em seguida os resultados para i = 20, 30, 40, 50, 60, 70, 80 e 90 respectivamente.

D020402	id5	id3
11	0,1	0
12	0,1	0,2
13	0,2	0,2
14	1,2	0
15	1,4	0,3
16	1,4	0,6
17	1,1	0,3
18	1	0,5
19	1,1	0,3
20	1,4	0,3

D030402	id5	id3
11	0	0,5
12	5,5	0,8
13	5,4	0,6
14	5,9	0,5
15	6,4	0,2
16	6,5	0,4
17	7,6	0,8
18	7,5	1
19	7,6	0,6
20	7,9	1,1

D040402	id5	id3
11	21,6	0,7
12	57,8	1
13	96,3	0,8
14	96,9	0,9
15	119,1	1
16	119,4	1,1
17	118,1	1,2
18	141,4	1,4
19	156,3	1,7
20	157	1,8

D050402	id5	id3
11	0	0,9
12	1,5	1,2
13	1,3	1,3
14	1,6	1,2
15	1,8	1,2
16	2,4	1,7
17	48,5	2,1
18	80,9	2,2
19	89,8	2,5
20	105,9	2,3

D060402	id5	id3
11	0,8	1,7
12	26,6	1,6
13	72,3	1,3
14	72,3	1,8
15	82,9	1,4
16	128,9	1,7
17	182,8	2,7
18	233,9	2,7
19	234,9	2,5
20	230,2	2,6

D070402	id5	id3
11	29	1,9
12	29	1,7
13	70,8	1,8
14	116,4	2,1
15	133,3	2,1
16	133,8	2,5
17	177,1	2,7
18	176,1	2,9
19	177,3	2,8
20	209,7	3

D080402	id5	id3
11	14,8	2,4
12	15	2,6
13	54,2	2,7
14	111	2,8
15	189,2	2,5
16	188,1	2,9
17	226	3,2
18	225	3,3
19	245,2	3,6
20	244,4	3,5

D090402	id5	id3
11	0	1,9
12	35,4	2,9
13	46,8	3,7
14	48,6	3,8
15	70,5	4,3
16	72	5
17	120,7	3,1
18	143	3,8
19	145,3	4,2
20	205,8	5,8

D100402	id5	id3
11	0,2	3,5
12	1,1	3,4
13	17,2	3,4
14	23,5	4,8
15	23,8	4,7
16	23,4	4,6
17	24,7	5,6
18	50,8	5,8
19	80,4	6,3
20	146,8	6,2

D110402	id5	id3
11	37,6	3,7
12	46,3	4,3
13	45,6	4,4
14	46,2	3,9
15	118,9	5,9
16	118,3	5,8
17	169,6	5,9
18	179,5	5,8
19	180,2	5,8
20	182,8	6,1

D120402	id5	id3
11	29,9	3,4
12	76,5	4,5
13	94,5	4,8
14	93,8	5,8
15	94,9	6,1
16	146,2	6,1
17	147,6	6,8
18	155,2	6,4
19	156,7	6,8
20	157,7	6,4

D130402	id5	id3
11	0,1	3,6
12	0,4	3,5
13	17	4
14	16,8	3,8
15	73,3	5,7
16	74,2	6,1
17	72,7	6
18	73,8	6,1
19	116,4	6
20	128,4	6

D140402	id5	id3
11	0,2	2,3
12	0,3	2,1
13	1,7	4,3
14	16,9	4,2
15	17,1	4,3
16	17,7	4,9
17	18,5	4,9
18	39,2	6,6
19	39,9	6,6
20	41,5	7,8

D150402	id5	id3
11	30,8	4,8
12	29,5	4,8
13	32,2	7,2
14	32,9	9,4
15	34,4	9,6
16	35,2	9,8
17	65,3	9,5
18	159,3	8,1
19	245,2	10,1
20	288,3	10,1

D160402	id5	id3
11	31,2	5,1
12	33,3	5,5
13	41	5,6
14	42,7	6
15	63	5,7
16	64,9	6,2
17	107,3	8,7
18	107,2	8,2
19	106,8	9,2
20	108,2	9,1

D170402	id5	id3
11	0,6	6,2
12	17,7	6,1
13	18,5	6,4
14	81,4	7,2
15	80,7	6,9
16	81,1	6,8
17	127,7	10,2
18	146	9,9
19	157,1	9,7
20	228	12,7

D180402	id5	id3
11	55,2	6,8
12	69,1	6,9
13	70,7	9,5
14	73,2	9,8
15	72,7	10
16	75	10,2
17	73,8	10,6
18	151,5	14,7
19	159,8	14,4
20	223,8	11,5

D190402	id5	id3
11	13,1	7,6
12	48,8	7,5
13	59,3	7,9
14	60,9	7,7
15	62,2	8,2
16	61,3	8
17	123,1	8,2
18	118,7	8,1
19	136,7	11,9
20	170,5	15,3

D200402	id5	id3
11	42,8	11,7
12	81,8	11,9
13	129,1	11,5
14	133,1	11,6
15	166,8	12,3
16	236,7	16,2
17	244,8	16,9
18	248,5	17,1
19	283,8	16,6
20	384,2	20,8

D020402	id5	id3
21	**0**	0,1
22	**0**	0
23	**0,1**	0,4
24	**1,6**	0,5
25	**1,2**	0,6
26	**1,6**	0,2
27	**1,6**	0,5
28	**1,4**	0,4
29	1,8	0,6
D030402	id5	id3
21	11,3	0,9
22	**11,5**	1,2
23	**12**	0,9
24	**12**	0,8
25	**12,1**	0,9
26	**12,2**	0,8
27	**12,8**	0,6
28	12,7	0,8
29	12,8	0,9
D040402	id5	id3
21	0,2	1,7
22	**0,3**	1,7
23	7,1	2,3
24	86,1	2,6
25	123,4	2,7
26	136,6	2,4
27	154,8	2,6
28	155,2	2,2
29	154,8	2,7
D050402	id5	id3
21	**0,1**	2,6
22	**6,7**	2,6
23	7,5	2,7
24	27,2	2,8
25	41,2	3
26	50,6	3,3
27	68,7	3,3
28	68,5	3,3
29	185	4
D060402	id5	id3
21	14,1	2,8
22	84,5	2,8
23	125,4	3,2
24	139,9	3,4
25	239,4	4,4
26	299,1	4
27	354,8	4,4
28	406,8	4,1
29	406,1	4,1
D070402	id5	id3
21	0,1	2,8
22	**9,2**	3,2
23	**9,3**	3,8
24	21	3,6
25	36,7	5
26	36,7	4,5
27	39,5	4,9
28	61,2	5,2
29	83,3	5,8

D080402	id5	id3
21	51,4	3,9
22	73,2	4,2
23	80,1	4,7
24	118,4	5,5
25	117,7	5,5
26	140,9	5,7
27	142,3	5,6
28	167,8	6,1
29	241,5	5,9
D090402	id5	id3
21	0,2	6,5
22	0,7	6,6
23	20,1	6,5
24	35,3	7,4
25	35,5	7,1
26	35,9	7,6
27	44,6	8,7
28	151,4	7,7
29	237,2	6,1
D100402	id5	id3
21	0,6	6,5
22	25	7,7
23	115,9	7,6
24	116,5	7,6
25	118	9,1
26	178,7	9,3
27	317,1	10,3
28	362,3	10,4
29	400	10,5
D110402	id5	id3
21	37,2	8,1
22	38,1	7,9
23	37,5	8,3
24	150,9	9,1
25	153,2	10
26	178,4	9,9
27	208,1	10,9
28	240,4	11
29	267,3	11,4
D120402	id5	id3
21	26,2	7,5
22	120,5	9,3
23	201,7	10,9
24	228,1	11
25	371,2	10,8
26	516,1	13
27	667,9	14,1
28	687,3	14,4
29	688,2	14,7
D130402	id5	id3
21	0,4	7,1
22	1,1	7,1
23	2,5	8,6
24	4,7	11
25	23,9	11,3
26	170,2	14,5
27	200,5	14,4
28	227,3	14,4
29	228,4	15,3

D140402	id5	id3
21	2,1	9,4
22	2,4	9,8
23	35,5	9,9
24	36,7	9,9
25	37,1	10,3
26	98,6	10,4
27	141,3	12,5
28	185,2	12,6
29	184,5	12,6
D150402	id5	id3
21	25,3	11,2
22	32,6	13,5
23	130,9	14,2
24	131,8	14
25	241,3	16,7
26	348	16,4
27	423,1	15,1
28	408,2	15,6
29	443,3	15,1
D160402	id5	id3
21	0,2	9,6
22	0,5	9,8
23	1,7	10,3
24	67,5	16,1
25	133,8	13,5
26	141,7	16,1
27	155	15,9
28	265	18,8
29	283,5	21,8
D170402	id5	id3
21	104,5	14
22	173,6	14,6
23	256,3	13,9
24	247,2	13,5
25	317,5	15,5
26	418,7	14,4
27	426,7	16,9
28	426,1	16,9
29	450,6	19,8
D180402	id5	id3
21	88,6	13,7
22	88,8	13,7
23	128,1	14,1
24	203,9	17,2
25	289,5	14,3
26	328,5	21
27	337,5	21,3
28	387,5	15
29	486,8	24

D020402	id5	id3
31	0	0,2
32	0	0,4
33	0,5	0,4
34	0,6	0,5
35	0,5	0,4
36	0,7	0,7
37	0,8	0,6
38	1	0,5
39	0,7	0,5
D030402	id5	id3
31	0,1	0,8
32	0,4	1,1
33	0,2	1,2
34	0,7	0,9
35	1,4	1,1
36	1,8	1,2
37	28	1,2
38	46,2	1,6
39	46,6	1,4
D040402	id5	id3
31	27	3
32	27,1	2,9
33	33,4	3
34	130,1	3,6
35	137,8	3,5
36	137,8	3,7
37	138,3	3,8
38	139,6	4,1
39	145,5	4,1
D050402	id5	id3
31	24,4	4
32	36,4	4,2
33	159,7	3,7
34	200,9	4
35	229,3	3,9
36	230,9	4
37	230,4	3,6
38	346,7	4,5
39	410,2	4,6
D060402	id5	id3
31	24	4,3
32	25	4,6
33	106	4,9
34	179,6	5,2
35	254,2	5,5
36	279,7	5,5
37	276,7	5,6
38	304,8	5,6
39	302,5	5,4

D070402	id5	id3
31	53,5	6,3
32	91,5	6,4
33	194,1	5,7
34	274,4	6,5
35	369,7	5,9
36	390,3	6,4
37	424,8	6,4
38	423,3	6,1
39	446,2	7
D080402	id5	id3
31	40,7	6,1
32	77,7	7,1
33	165,9	6,3
34	223,7	7,1
35	224,1	7,1
36	237,7	7,8
37	239,2	7,9
38	332,8	8,2
39	351	8,3
D090402	id5	id3
31	120,9	8,5
32	170,7	8,3
33	221,6	9,3
34	267,8	10,4
35	268,1	10,1
36	410,4	8,7
37	438,7	8,6
38	451,1	8,7
39	477,9	9,5
D100402	id5	id3
31	26,1	9,8
32	50,7	10,4
33	94,8	10,8
34	93,9	10,8
35	94,6	10,7
36	110,4	11,1
37	110,9	11,1
38	113,3	11,4
39	114,5	11,6
D110402	id5	id3
31	30,6	10,7
32	65,1	10,8
33	65,7	11,1
34	65,4	10,9
35	96	12,7
36	97,3	12,4
37	96,5	12,8
38	97,4	13
39	97,8	13,3

D120402	id5	id3
31	155,6	14
32	307,1	14,2
33	362,5	14
34	508,3	12,2
35	538,1	14,4
36	575,5	13,9
37	574,4	14,6
38	586,8	14,5
39	589,2	15,8
D130402	id5	id3
31	151,7	12,1
32	152,8	12,3
33	258,4	14,2
34	283,5	16,3
35	355,4	12,7
36	383,5	13
37	382	13
38	384,1	13,2
39	389,1	14,8
D140402	id5	id3
31	13,1	11,9
32	13	12
33	15,7	13,9
34	15,7	14,5
35	102,1	18,5
36	127,5	19
37	128	18,7
38	147	18,8
39	148	18,6
D150402	id5	id3
31	16,2	15,8
32	16,5	16
33	17	16,7
34	17,8	17,4
35	46,8	17,3
36	65	17,8
37	66,2	17,5
38	67,1	19,9
39	66,6	19,7
D160402	id5	id3
31	84,2	17,3
32	118,4	17,6
33	136,4	18
34	224,5	23
35	251,5	21,5
36	313,6	23,6
37	374,8	21,7
38	414,1	24,5
39	415,1	25,3

D170402	id5	id3
31	23,8	17,7
32	55,4	18,3
33	54,4	18
34	55,8	20,9
35	78,5	23,2
36	103,9	23,6
37	104,3	23,7
38	147,3	24,1
39	169,2	23,9
D180402	id5	id3
31	83,8	22
32	120,4	21,7
33	141,2	22
34	216,2	22,1
35	302,7	22,8
36	454,3	26
37	474,9	26,4
38	499,2	27
39	505,2	30
D190402	id5	id3
31	93	24,2
32	94	24,1
33	94,9	24,6
34	110	28,2
35	140,7	28,8
36	140,1	29,4
37	342,2	36,9
38	473,6	33,4
39	504	33,4
D200402	id5	id3
31	49,7	22,8
32	95,1	28
33	119,3	28,6
34	144,1	28,1
35	164,8	32,1
36	166,1	32,2
37	164,9	32,9
38	214,9	31,8
39	213,6	32,6

D020202	id5	id3
41	0	0,4
42	0	0,1
43	0,1	0,1
44	0,2	0,1
45	14	0,2
46	28,2	0,3
47	28,3	0,1
48	27,8	0,1
49	28,2	0,1

D020302	id5	id3
41	0,1	0,5
42	0	0,5
43	0,3	0,8
44	0,1	0,4
45	0,3	0,5
46	0,3	0,5
47	0,2	0,3
48	0,2	0,5
49	0,5	0,5

D020402	id5	id3
41	0,1	0,4
42	0,1	0,9
43	0,3	0,6
44	0,1	0,4
45	0,1	0,8
46	0,2	0,9
47	0,4	0,5
48	0,4	0,7
49	0,4	0,5

D020502	id5	id3
41	0	1,2
42	0,2	1,1
43	0,1	1,2
44	0,7	1,1
45	0,7	1,4
46	0,6	1,6
47	0,8	1,3
48	0,6	1,1
49	1,1	1,5

D030202	id5	id3
41	14,2	0,7
42	13,7	0,7
43	14,7	0,5
44	15,1	0,5
45	14,4	0,6
46	14,3	0,7
47	14,5	0,5
48	15	0,7
49	14,9	0,5

D030302	id5	id3
41	15,6	1,7
42	15,9	1,4
43	64,4	1,3
44	65,2	1,9
45	66,2	1,3
46	65,5	1,1
47	65,2	1,7
48	66,3	1,7
49	64,6	1,7

D030402	id5	id3
41	47,1	2,2
42	71,2	2,3
43	74,6	2,2
44	71,7	1,9
45	72,6	1,9
46	70,1	2
47	70,4	2,3
48	70,7	2,3
49	71,1	2,4

D030502	id5	id3
41	49,1	3,5
42	82,9	4
43	84	3,6
44	99,9	3,9
45	113,3	3,9
46	110,6	4
47	191,8	4,6
48	255,1	3,9
49	318,1	4,8

D040202	id5	id3
41	0,1	1,1
42	0	1,2
43	0,2	1,3
44	0,5	0,7
45	0,4	1,2
46	0,5	1,4
47	0,7	1
48	0,2	0,9
49	2,2	1,5

D040302	id5	id3
41	0,1	2,5
42	0,1	2,4
43	0,4	2,3
44	10	2,6
45	34,9	3,1
46	37	3
47	46,3	3,2
48	78,7	3,6
49	90,7	3,4

D040402	id5	id3
41	0,1	3,6
42	0,2	4
43	27,8	4,1
44	34,6	4,2
45	64,8	4,1
46	79,8	4
47	109,4	4
48	132,6	4,4
49	138,1	4,4
D040502	id5	id3
41	0,3	5,5
42	32,1	5,4
43	66,1	5,6
44	101,5	5,8
45	122	5,6
46	139,2	6,5
47	149,1	6,9
48	150	6,8
49	152,5	7,2
D050202	id5	id3
41	69,9	2,6
42	76,4	2,4
43	113	2,6
44	127,2	2,1
45	139	2,4
46	190,6	2,7
47	196,3	2,5
48	254,8	2,8
49	276,3	2,9
D050302	id5	id3
41	0	3,4
42	1,2	3,8
43	71,3	4,3
44	98,8	4,3
45	98,3	4,6
46	99,5	4,4
47	112,1	4,5
48	112,4	5,1
49	113,1	5
D050402	id5	id3
41	27,1	4,5
42	113,8	4,6
43	111	4,6
44	116,4	4,9
45	134,5	5
46	155	5,5
47	332,7	6,5
48	340	6,8
49	362,1	6,8

D050502	id5	id3
41	33,3	7,2
42	160	7,4
43	181,1	7,5
44	212,7	7,2
45	236	8
46	354,3	9,1
47	399,8	9,2
48	404,5	9,6
49	446	9,5
D060202	id5	id3
41	0	2,9
42	57	2,9
43	58,8	2,7
44	73,1	3
45	75,1	2,9
46	105,6	3
47	105,2	3
48	117,4	3,1
49	118,4	3,3
D060302	id5	id3
41	0,2	3,3
42	0,4	3,3
43	0,5	3,4
44	0,7	3,3
45	1	3,6
46	13	3,7
47	32,3	3,8
48	55,2	4,2
49	67,8	4,1
D060402	id5	id3
41	0,1	5,8
42	36	5,9
43	48,4	6,1
44	65,1	6,7
45	67,1	6,7
46	66,8	6,6
47	63,8	6,5
48	65,2	6,5
49	67,6	7,1
D060502	id5	id3
41	102,6	10,3
42	156,9	9,9
43	183,4	10,2
44	209,7	10,9
45	222	11,1
46	248,1	11,4
47	248,7	11,3
48	278,3	11,6
49	278,9	12,1

D070202	id5	id3
41	0,3	3,7
42	0,3	3,6
43	1	3,5
44	0,8	3,6
45	0,7	3,8
46	22,2	4
47	32,9	4,3
48	32,6	3,9
49	33,2	4

D070302	id5	id3
41	0,6	5,8
42	0,3	5,7
43	29,5	5,5
44	29,5	5,7
45	67,1	5,7
46	67,1	5,9
47	67,3	6,2
48	67,2	5,6
49	78,9	6,3

D070402	id5	id3
41	0,5	6,7
42	1,5	8,1
43	14,1	7,5
44	21,2	7,9
45	34	7,5
46	33,1	7,9
47	33,1	7,9
48	39,2	8,2
49	39,6	8,6

D070502	id5	id3
41	23,2	10,3
42	58,1	10,6
43	57,9	10,8
44	58	10,8
45	58,3	10,7
46	59	11
47	192	11,2
48	256,3	12
49	256,1	12,1

D080202	id5	id3
41	84,3	5,4
42	123,9	6
43	131,2	5,8
44	273,4	5
45	294,4	4,9
46	337,4	5
47	393,3	5,4
48	403,2	5,3
49	412,1	5,8

D080302	id5	id3
41	0,3	5,7
42	28,7	6,4
43	35,4	7,1
44	72,8	6,6
45	72,6	6,9
46	106,1	7,7
47	130,3	8,3
48	156,3	8,5
49	185,9	8,8

D080402	id5	id3
41	0,4	8,5
42	56,3	9
43	56,3	9,2
44	102,8	9,6
45	213,4	9,1
46	266,2	9,2
47	365,6	9,9
48	420,3	10,8
49	546,7	10,2

D080502	id5	id3
41	0,1	10,4
42	32,1	11,5
43	33,7	11,8
44	35,7	12,8
45	53,6	13,1
46	53,5	13,1
47	54,4	13,2
48	92,8	13,7
49	122,3	13,6

D090202	id5	id3
41	11,6	5,5
42	12	5,7
43	97,9	5,9
44	110,2	5,9
45	117,1	5,7
46	117,1	6
47	117,4	6
48	116,8	6,1
49	167,9	6,5

D090302	id5	id3
41	24,7	7,1
42	24,1	7,1
43	24,9	7,4
44	26,8	8
45	121,8	9
46	250	8,6
47	256,8	8,9
48	347,8	9
49	420	8,5

D090402	id5	id3
41	19,8	10,4
42	71,6	12,6
43	84,5	12,5
44	84,2	12,8
45	148,5	14,1
46	150,7	14,5
47	150,7	15,4
48	184	15
49	225,8	14,9
D090502	id5	id3
41	172,5	14,6
42	254,6	15,3
43	305	15,5
44	358,8	16,6
45	384,6	16,8
46	386,6	17,3
47	450,9	17
48	456,8	18,7
49	483,5	18,4
D100202	id5	id3
41	17,5	6,7
42	29,6	6,9
43	44,6	7,3
44	71,1	7,5
45	93,4	8
46	175,8	8,2
47	230,2	8,9
48	321,8	8,7
49	376,5	9
D100302	id5	id3
41	9,1	7,7
42	9,4	8
43	62,7	8,8
44	102,6	10
45	119,5	10,2
46	120,9	10,7
47	121,7	10,7
48	121,3	11
49	168,1	10,9
D100402	id5	id3
41	12,6	12,4
42	29,5	12,3
43	29,9	12,5
44	30,8	12,6
45	30,9	12,8
46	50,7	12,9
47	50,8	13,1
48	263,6	15,1
49	263,6	15,3

D100502	id5	id3
41	6,2	19,3
42	39	19,4
43	40,3	19,5
44	41,7	19,9
45	41,5	20,3
46	53,5	20,2
47	53,2	20,4
48	88,8	21
49	113,4	21,3
D110202	id5	id3
41	163	7
42	273	8,3
43	340,3	7,6
44	529,5	9,5
45	622,6	8,2
46	653,5	8,6
47	801,5	10
48	896,4	10
49	948,4	10,4
D110302	id5	id3
41	8	10
42	8,4	9,9
43	50,7	9,8
44	50,9	9,9
45	50,9	10
46	74,5	10,2
47	74,7	10,3
48	121,8	10,7
49	123,3	11,7
D110402	id5	id3
41	171,8	16,3
42	210,5	16,6
43	267,9	16,9
44	300,5	17,3
45	329,1	17,1
46	380,1	17,3
47	394,9	17,3
48	559,5	19,1
49	638,8	18,6
D110502	id5	id3
41	30,1	19,3
42	196,4	21,2
43	207,2	21,8
44	239,4	21,7
45	258,4	21,9
46	259,4	21,9
47	269,8	24,3
48	399,7	22,5
49	437,1	22,7

D120202	id5	id3
41	94,8	8,4
42	95,1	8,5
43	112,7	8,5
44	129,7	8,9
45	153,1	9,3
46	338,6	9,1
47	399	9,3
48	419,5	9,2
49	663,9	10,5

D120302	id5	id3
41	0,4	11,4
42	37	11,6
43	37,4	11,9
44	55,5	11,9
45	62	12,5
46	189,4	12,7
47	240,2	12,8
48	296,3	12,9
49	314	13,6

D120402	id5	id3
41	41,3	15,9
42	41,6	16,2
43	42,6	16,3
44	104,2	17,6
45	148,8	16,9
46	170,6	16,9
47	222	17,2
48	280,8	18,3
49	287,2	18,7

D120502	id5	id3
41	0,3	20,7
42	22	20,7
43	107,1	23,3
44	117,5	25,1
45	181,1	21,3
46	196,8	21,3
47	198,2	21,4
48	198,9	21,8
49	198,5	21,5

D130202	id5	id3
41	38,7	9
42	162,8	9,5
43	178	10,7
44	196,1	10,4
45	479,9	10,6
46	556	10,6
47	920,6	11
48	1061	11,9
49	1274,5	12,5

D130302	id5	id3
41	48	11,4
42	58,3	11,3
43	71,1	10,9
44	266,4	11,9
45	545,9	15,4
46	638	16,3
47	727,6	12,7
48	752,2	12,9
49	726	12,5

D130402	id5	id3
41	0,9	15,2
42	23,2	15,4
43	24,8	15,2
44	61,1	15,3
45	62,1	15,3
46	68,3	16,6
47	69,8	17,2
48	71	17,1
49	70,6	17

D130502	id5	id3
41	58,6	26,8
42	59,5	26,6
43	172,1	26,4
44	211,1	27,9
45	241,9	27,4
46	238,3	27,1
47	268,8	29,7
48	296,3	31,2
49	488,5	31,7

D140202	id5	id3
41	89,7	9,9
42	212,9	10,6
43	531,9	11,3
44	663,3	10,6
45	716,8	11,3
46	816,2	11,9
47	832,5	12,1
48	895,4	13,1
49	927,2	13,1

D140302	id5	id3
41	0,5	14,6
42	9,3	15,3
43	50	15,6
44	68	15,6
45	122,3	15,8
46	139,5	16
47	133,6	15,8
48	203,1	15,5
49	205,3	16,6

D140402	id5	id3
41	7,5	18,9
42	45,7	20,1
43	47,1	21,8
44	46,7	21,6
45	57,6	21,6
46	83,1	23,8
47	116,4	26,3
48	124,6	26,6
49	123,7	26,5

D140502	id5	id3
41	2	26,8
42	30,9	27,1
43	31,8	27,1
44	31,7	27,3
45	32,5	27,3
46	33	27,8
47	40,2	31
48	231,1	29,7
49	279,8	29,1

D150202	id5	id3
41	0,4	11,6
42	63,2	12,8
43	93,8	12,7
44	125,8	12,7
45	154,3	12,7
46	151,7	12,7
47	178,3	13,6
48	230,4	13,7
49	262,6	13,5

D150302	id5	id3
41	155,5	15,6
42	193,6	15,7
43	356,4	15,4
44	426,5	16
45	657,6	15,5
46	662	16,7
47	761,9	16,8
48	765,6	16,8
49	764,2	18,3

D150402	id5	id3
41	26,7	20
42	37,7	20
43	38,1	19,9
44	38,4	20,3
45	38,9	20,7
46	59,4	20,6
47	60,1	21
48	60,3	20,8
49	80,8	23,1

D150502	id5	id3
41	154,7	37,7
42	184,1	38
43	184,4	37,8
44	190,3	38,7
45	243	38,3
46	382,6	34,7
47	417	34,8
48	453,8	35,1
49	454,1	35,4

D160202	id5	id3
41	81,8	14,2
42	84	14
43	155,5	13,7
44	155,8	14
45	165,6	14,4
46	165,3	14,1
47	169,8	14,5
48	164,9	14,2
49	233,2	14,3

D160302	id5	id3
41	153,3	17,5
42	203,4	17,2
43	388,8	20,6
44	476,5	20,6
45	557,4	21,8
46	725,2	20,6
47	722,7	19,4
48	714,6	19,7
49	710,7	19,7

D160402	id5	id3
41	20,2	24
42	222,3	29,4
43	274,7	29,9
44	276,1	30,8
45	445,2	28,3
46	449,3	28,7
47	488,1	31,1
48	544,7	29,1
49	544,6	29

D160502	id5	id3
41	28,4	44,4
42	62	44,9
43	74	45
44	216,9	37,9
45	268,8	38,3
46	386,7	47,3
47	398,2	46,6
48	401	46,4
49	507,2	39,7

D170202	id5	id3
41	45,2	11,3
42	168,2	12
43	224,1	13,7
44	360,6	14
45	375	14,3
46	372,7	15,1
47	363	14,8
48	363,1	15,7
49	441,8	15,7
D170302	id5	id3
41	81,7	17,5
42	170,2	20,1
43	202,2	20
44	361	21,1
45	414,9	21,3
46	424	23,2
47	558,1	22,6
48	595,7	22,6
49	634,2	22,5
D170402	id5	id3
41	0,5	27,7
42	42,7	30,1
43	42,7	30,4
44	175,5	33,1
45	205,9	33,5
46	259,3	36
47	322,6	37,1
48	456,7	36,9
49	507,5	37,7
D170502	id5	id3
41	29,1	34,1
42	62	39
43	64,7	38,9
44	184,6	40,1
45	219,3	40
46	224,2	44,4
47	224,9	44,4
48	251,3	44,5
49	250,8	46,2
D180202	id5	id3
41	51	14,3
42	67,9	15,2
43	68,6	15,3
44	87,7	16,2
45	99,9	16,2
46	101,6	16,3
47	116,3	15,8
48	130,8	16,8
49	177,5	16,7

D180302	id5	id3
41	0,8	22,4
42	12	22,6
43	53,6	24
44	54,7	24,7
45	99,7	24,8
46	127,9	26,5
47	129,6	26,6
48	126,4	27,1
49	126,3	27,8
D180402	id5	id3
41	52,3	37,8
42	52,5	37,2
43	52,5	37,4
44	53,1	37,3
45	236,6	33,6
46	254,9	37,8
47	257,8	37,4
48	254,2	37,8
49	329,1	39,7
D180502	id5	id3
41	130,1	47,9
42	129,3	48,1
43	156,3	48,9
44	177,1	53,1
45	176,9	52,9
46	211,7	53,5
47	333,2	43,9
48	405,6	53,6
49	426,4	54,5
D190202	id5	id3
41	12,1	13,8
42	56,3	13,4
43	71,8	13,8
44	151	13,9
45	185,3	14,1
46	186,5	14,3
47	203,2	14,1
48	302	14,3
49	359,3	14,5
D190302	id5	id3
41	68,6	22,8
42	69	22,8
43	136,6	26,5
44	171	26,5
45	223	27,6
46	221,5	27,8
47	250,2	27,5
48	249,7	27,9
49	248,2	28,2

D190402	id5	id3
41	168,4	33,1
42	210,1	33,2
43	371,9	34,7
44	423,3	41,7
45	443,7	44,7
46	445	44,8
47	495,7	44,4
48	733,3	37,9
49	776,9	38
D190502	id5	id3
41	176,3	50,7
42	382	53,5
43	406,2	53,7
44	610,4	58
45	807,1	54,4
46	973,3	55,1
47	1162,8	59,2
48	1372,9	60,1
49	1549,1	55,9
D200202	id5	id3
41	0,5	17,3
42	68,7	18,1
43	135,6	18
44	268,6	19,3
45	383,7	18,9
46	488,3	20,6
47	519,2	20,8
48	580,3	20,3
49	583	20,6
D200302	id5	id3
41	46,6	24,5
42	46,7	24,7
43	74,4	25,1
44	101,9	26,9
45	192,4	25,8
46	208,4	26
47	229,7	28
48	325,1	30
49	330,4	34,4
D200402	id5	id3
41	0,8	32,9
42	1	33,3
43	14,4	33,7
44	25,2	34
45	53	37,7
46	98,4	41,6
47	119,1	41,9
48	161,5	46
49	164,3	50

D020402	id5	id3
51	0,1	1
52	0	0,8
53	0,3	0,4
54	0,1	0,4
55	0,4	0,9
56	0,5	0,8
57	0,7	0,9
58	0,5	0,6
59	0,3	0,8
60	0,6	0,6
D030402	id5	id3
51	0,1	2,6
52	0,2	2,3
53	0,5	2,5
54	0,5	2,5
55	0,3	2,6
56	0,9	2,2
57	0,5	2,5
58	1,9	2,9
59	1,7	2,7
60	2,3	2,9
D040402	id5	id3
51	0,2	4,9
52	0	4,4
53	1,7	4,7
54	2,1	5,2
55	11,4	5,1
56	11,9	5,3
57	11,4	5,1
58	12	4,9
59	12	5,4
60	12,6	5
D050402	id5	id3
51	0	6,7
52	25,2	7,4
53	197,3	6
54	197,6	6,1
55	249,4	6,5
56	295,8	6,4
57	307,2	6,8
58	366,9	7,3
59	363,5	7,5
60	383,8	7,6
D060402	id5	id3
51	0,2	7,2
52	28,9	7,2
53	51,2	7,6
54	221	9,2
55	250,5	8,9
56	332,7	8,6
57	358,1	9,1
58	590	9,7
59	638,3	9,6
60	664,8	9,6

D070402	id5	id3
51	17,5	9
52	18	9
53	18	9,1
54	17,2	9,2
55	17,9	9,2
56	18,8	9,3
57	18,8	9,5
58	19,3	9,6
59	30,6	9,9
60	42,3	9,6
D080402	id5	id3
51	46,3	11,4
52	46,2	11,9
53	66,5	12,9
54	119,8	12,1
55	186,7	13,1
56	217,6	13,5
57	220,2	13,4
58	271,8	14
59	266,7	13,3
60	281,6	13,4
D090402	id5	id3
51	0,5	15,5
52	0,1	15,3
53	61,7	15,6
54	80,6	16,6
55	126,3	16,9
56	133,5	17,4
57	135	17,8
58	392,6	18,3
59	480,4	18,6
60	482,1	19,2
D100402	id5	id3
51	14,8	17,1
52	16,7	18,2
53	61,9	18,7
54	124,6	18,9
55	262,9	19,6
56	264,7	20
57	268,7	19,5
58	266,1	19,6
59	268,4	19,7
60	299	20,7
D110402	id5	id3
51	74,3	20,4
52	74,5	20,6
53	107,2	20,7
54	184,9	21,4
55	344,9	21,2
56	476,1	23,5
57	473,1	23,6
58	504,3	24,1
59	636,9	22,6
60	672,5	23

D120402	id5	id3
51	0,1	20,6
52	35,2	22,5
53	35,2	22,3
54	41,9	23,8
55	61,9	24,2
56	61,6	24,3
57	141,9	24,2
58	145,5	25,2
59	182,9	23,8
60	186,5	23,2

D130402	id5	id3
51	8	20
52	8,5	20
53	197,8	27,4
54	199,6	27,4
55	291,8	20,2
56	592,6	26,3
57	674,5	26,7
58	699	26,2
59	739,1	25,2
60	748,3	24,7

D140402	id5	id3
51	37,7	26,8
52	51,1	27,1
53	53,3	27,2
54	53,7	28,4
55	54,1	27,3
56	103,3	27,6
57	103,9	28,2
58	155,3	30,5
59	158,1	32,4
60	157,8	32,2

D150402	id5	id3
51	1,4	25,4
52	1,8	25,4
53	95,8	25,6
54	97,6	25,8
55	98,6	26,3
56	100,3	28,1
57	112,4	30,5
58	339,3	29,6
59	349,2	29,5
60	364,3	29,5

D160402	id5	id3
51	46,1	30,7
52	47,5	31,3
53	48,9	31,8
54	49,4	31,9
55	50,3	31,7
56	50	32,3
57	89,9	34,5
58	132,8	37,4
59	150	37,6
60	150,5	37,7

D170402	id5	id3
51	124,1	40,5
52	138,5	37,1
53	159,9	37,6
54	161,7	38
55	161,6	38,1
56	243,7	41
57	287,6	41
58	289	41
59	557,4	42,7
60	595,4	45,1

D180402	id5	id3
51	0,6	38,9
52	41,8	41,8
53	115,2	41,6
54	127,1	42,4
55	126,4	42,5
56	173	43,2
57	175,7	43,5
58	191,7	43,2
59	194,1	47,2
60	409	46,8

D190402	id5	id3
51	0,9	42,2
52	57,9	46,2
53	252,7	50,3
54	325,7	47
55	433,2	51
56	702,1	49,9
57	786,7	50,4
58	982,4	51,4
59	1055,9	51,9
60	1135,9	50,8

D200402	id5	id3
51	46,2	51
52	70,3	51,3
53	71,5	51,6
54	125,9	48,1
55	159,4	52,1
56	177,7	55,5
57	241,7	58,7
58	242,8	59,3
59	308,8	56,4
60	318,4	56,9

D020402	id5	id3
61	0	1
62	0,1	1,1
63	0,1	0,4
64	0	0,6
65	0,1	0,7
66	0,2	0,8
67	0,4	0,8
68	0,3	0,7
69	0,4	0,6
70	0,7	0,8

D030402	id5	id3
61	0,1	2,8
62	0,1	2,7
63	0,7	3
64	1,1	2,9
65	1,2	3
66	1,3	2,9
67	1,7	3
68	2,1	3,3
69	2,1	3,2
70	2,7	2,9

D040402	id5	id3
61	0	5,7
62	35,9	5,6
63	91,9	6,1
64	93,8	5,6
65	92,9	6,1
66	93	6
67	91,6	6
68	91,8	5,9
69	95,1	6
70	102,6	6,8

D050402	id5	id3
61	1	8,3
62	46,1	8,8
63	97,3	9,1
64	95,9	8,6
65	99,7	9,2
66	144,2	9,5
67	138,6	9,1
68	133,5	8,7
69	133,9	8,9
70	146,3	8,6

D060402	id5	id3
61	57,4	9,5
62	194,8	8,6
63	256,1	8,7
64	311,5	8,4
65	501,7	10,3
66	575,3	11,4
67	616,3	11,4
68	768,7	9,3
69	786,5	8,5
70	835,4	12,1

D070402	id5	id3
61	1,4	10,7
62	91,4	11,8
63	97,3	12,5
64	98,3	12,6
65	118,3	13,9
66	120	14,4
67	121,9	14,6
68	121,6	14,8
69	122,8	14,7
70	128,9	14,9

D080402	id5	id3
61	0,2	13,2
62	0,6	13,6
63	39,5	14,1
64	91,9	15
65	91,6	15,1
66	92,9	15,1
67	93,6	15,6
68	161,4	16,3
69	163,1	16,2
70	184	15,9

D090402	id5	id3
61	12,1	18,3
62	12,2	18,6
63	162,8	18,5
64	208,7	19,7
65	255,7	19,9
66	335,7	19,3
67	338,5	19,2
68	390,2	18,7
69	426,6	19,8
70	434,1	19,7

D100402	id5	id3
61	84	19,9
62	109,9	20,5
63	108,9	20,6
64	140,1	20,8
65	178,7	21
66	286,1	23
67	352,1	23,2
68	353,2	22,9
69	421,7	24,1
70	490	24,5

D110402	id5	id3
61	65,5	22,7
62	83,3	23,9
63	117,2	23,4
64	137,9	24,2
65	173,9	24
66	174,5	24,8
67	239,5	24,8
68	254,7	25,5
69	265,4	27,3
70	328,2	28,4

D120402	id5	id3
61	0,5	23,5
62	20,4	23,3
63	21,8	23,3
64	74,2	27,5
65	162,7	30,5
66	163,2	30,4
67	187,4	30,9
68	188,9	30,9
69	190	31,6
70	271,7	30,1

D130402	id5	id3
61	0,4	24,8
62	1,2	25
63	1,9	25
64	277	32,5
65	315	32,5
66	407,6	32,3
67	407,8	32,9
68	717,1	31,6
69	1028,2	29,8
70	1139,4	30,7

D140402	id5	id3
61	0,4	32,7
62	1	32,5
63	46,2	34,9
64	47,2	35
65	47	35,5
66	58	35,5
67	118,7	37,6
68	212,2	37,2
69	239,8	37,7
70	240,4	37,9

D150402	id5	id3
61	19,3	30
62	71,2	32,5
63	72,5	32,5
64	138,7	33,1
65	138,8	33,3
66	140	33,4
67	141,6	33,4
68	141,3	33,9
69	142,7	34,1
70	145,5	36,3

D160402	id5	id3
61	0,8	38,5
62	1,4	38,5
63	14,5	41,8
64	16,6	44,4
65	96	44,3
66	127,1	45
67	207,3	45,4
68	207,1	45,4
69	243,8	45,5
70	312,3	45,2

D170402	id5	id3
61	242	44,3
62	304,9	44,1
63	486	45
64	517,6	45,1
65	518,5	45,3
66	566	46,3
67	593,6	46,4
68	646,4	46,6
69	647,2	47,1
70	646,4	47,4

D180402	id5	id3
61	62,4	46,9
62	63,3	46,8
63	98,3	47,8
64	120,7	48,1
65	163,9	51
66	166,2	51,7
67	166,8	51,8
68	254,9	51,6
69	275,5	51,7
70	287,4	52

D190402	id5	id3
61	79,9	51,1
62	135,4	52,2
63	138,4	56
64	399,9	55
65	681,5	57,3
66	922,9	58,4
67	946,1	59,1
68	1031,9	61,6
69	1049,2	62,3
70	1081,7	65,3

D020402	id5	id3
71	0,1	0,8
72	0,1	1
73	0,2	0,6
74	0	0,8
75	0,2	1,1
76	0,2	1,3
77	0,3	0,8
78	0,5	0,6
79	0,2	0,9
80	0,6	1

D030402	id5	id3
71	0,2	3
72	0,2	3,3
73	0,5	3
74	0,6	3
75	0,4	3
76	0,6	3,3
77	2	3,3
78	2	3,6
79	2	3,5
80	1,8	3,7

D040402	id5	id3
71	0,1	6,6
72	11,2	6,2
73	26,4	6,6
74	26,2	6,2
75	26,7	6,8
76	27	6,6
77	28,9	7,4
78	52,7	7,8
79	51,6	7,3
80	74,7	8

D050402	id5	id3
71	1,4	9
72	43,4	9,3
73	258,1	10,4
74	259,7	10,8
75	263	10,8
76	275,1	10,9
77	348,9	11,5
78	444,9	11,1
79	446,5	11,5
80	512,2	11,5

D060402	id5	id3
71	59,3	12,6
72	139,5	11,8
73	138,7	12,7
74	152,7	11,8
75	153,8	11,9
76	167,6	12,3
77	170,5	13,1
78	258,6	12,8
79	513	15,1
80	585,2	13,3

D070402	id5	id3
71	1,1	15,8
72	249,4	13,3
73	314,6	14,5
74	401,5	16,4
75	429,3	16,3
76	467,1	16,4
77	605	15,5
78	710	16,1
79	749	16,4
80	850,2	16,5

D080402	id5	id3
71	0,4	16,9
72	7,2	17,5
73	7,4	17,9
74	7,4	17,2
75	122,6	16,4
76	126,1	18,8
77	229,5	18,1
78	248,5	18,9
79	299,5	19,1
80	298,3	18,7

D090402	id5	id3
71	0,2	20,5
72	0,8	19,9
73	143,4	22
74	143	22,8
75	241,4	21,3
76	274,2	22,1
77	360,6	22,3
78	442	23,4
79	594,1	22,8
80	724,5	24,1

D100402	id5	id3
71	22,2	25
72	30,9	25,8
73	32,1	25,9
74	47,3	25,8
75	48,1	25,6
76	48,6	25,8
77	101,8	25,6
78	178,8	27
79	228,4	27,2
80	276,1	28,3

D110402	id5	id3
71	0,1	29,3
72	10,8	29,1
73	10,8	29
74	17,9	31,3
75	19,3	30,6
76	19,2	31,1
77	19,9	30,2
78	73,4	32,1
79	86,9	31,6
80	171,3	31

D120402	id5	id3
71	126,8	30,2
72	153,4	28,1
73	169,4	30,2
74	199,7	31,9
75	398,2	34,8
76	432	36,8
77	475,9	36,7
78	813,8	33,7
79	1049,3	35
80	1049,5	36,2

D130402	id5	id3
71	51,8	30,2
72	162,6	31,6
73	205,5	30,8
74	250,3	31,3
75	321	35,5
76	336,5	35,4
77	327,2	34,8
78	327,9	35,3
79	353,9	35,3
80	357	35,8

D140402	id5	id3
71	12,5	37,6
72	295,6	34,1
73	599,3	35,8
74	708,1	37,9
75	881,8	41,7
76	916,4	42
77	935	42,3
78	1079,6	42,6
79	1126,5	42,8
80	1254,4	42,9

D150402	id5	id3
71	15,3	38,8
72	117,4	38,5
73	220,2	40,5
74	292,4	43,6
75	312,2	43,8
76	312,5	43,9
77	327,8	43,9
78	391,7	42,8
79	468,2	47
80	548,7	45,3

D160402	id5	id3
71	0,7	44,8
72	54,5	46,2
73	55,4	46
74	56	47,2
75	69,1	47,3
76	69,5	47,7
77	70,7	48,1
78	71,4	48,1
79	72,4	48,8
80	99,1	50,5

D170402	id5	id3
71	78,3	47
72	93,9	50,4
73	94,4	50,9
74	95,6	50,4
75	108,8	50,8
76	120,1	54,2
77	388,5	61,3
78	415,5	61,9
79	478,1	58,7
80	493,2	59,1

D180402	id5	id3
71	17,9	52
72	84	52,7
73	293,8	62,9
74	361,1	56,5
75	418,3	60,2
76	471,4	60,3
77	690,8	62,3
78	778,3	62,7
79	958,6	64,5
80	988	65

D190402	id5	id3
71	48,1	67
72	57,3	66,6
73	150,4	66,4
74	150,5	66,6
75	249,1	67,1
76	273,1	67,3
77	274,5	68
78	334,4	68
79	335	72,1
80	678,7	72,9

D200402	id5	id3
71	198,2	76,7
72	266,6	73,6
73	276,2	76,8
74	313,5	77,7
75	332,6	78
76	342,4	78,6
77	343,3	78,3
78	363,3	79,4
79	373,4	79,7
80	417,6	79,8

D020402	id5	id3
81	0,1	1
82	0,1	1,1
83	0,2	0,9
84	0,2	0,7
85	0,2	1
86	0,3	0,6
87	0,1	1,1
88	0,1	1,3
89	0,3	1
90	0,7	1,2
D030402	id5	id3
81	0,1	3,8
82	0,3	3,6
83	0,2	3,7
84	1,7	3,9
85	1,5	4,1
86	2,1	4,2
87	2,2	4,1
88	2,3	3,8
89	28,1	4,4
90	29,5	4,5
D040402	id5	id3
81	0,1	8
82	53,8	8,6
83	108,5	9,2
84	108,4	9,3
85	108,8	9,4
86	108,3	9,1
87	109,5	9,9
88	116,8	10,2
89	126,6	10,6
90	126,3	10,6
D050402	id5	id3
81	18,6	11,7
82	54,3	12,1
83	72,1	12,2
84	81,8	12
85	82,3	12,3
86	143,1	12,5
87	160	13,1
88	178,3	13,2
89	200,5	13,4
90	199,7	13,3
D060402	id5	id3
81	120,2	15,2
82	342,6	11,1
83	427	11,6
84	549,7	14,9
85	611,4	14,9
86	732	12,4
87	977	14,8
88	1125,2	12,5
89	1210,2	12,9
90	1233,5	13,6

D070402	id5	id3
81	0,3	16,5
82	9,2	17,4
83	98,3	16,3
84	175,7	16,8
85	240,4	16,9
86	239,4	16,7
87	261	17
88	266,5	17,6
89	291,1	18
90	321,8	17,9
D080402	id5	id3
81	15,7	18,6
82	108,5	19,6
83	110,5	19,7
84	178,2	19,8
85	213,2	19,9
86	293,6	20,8
87	308,4	21
88	373,6	21,8
89	417,6	22,4
90	432,2	23,3
D090402	id5	id3
81	113,6	24
82	170,6	24,7
83	587,4	27,9
84	587,7	28,4
85	626,6	28,1
86	690,7	28,6
87	703,5	28,3
88	850,3	26,9
89	1114,8	25,4
90	1239,9	28,8
D100402	id5	id3
81	24,1	28,4
82	24,9	28,9
83	25,1	28,6
84	39,8	30
85	126,6	30,5
86	127,4	30,7
87	211,7	31,4
88	212,9	31,5
89	235,3	32,9
90	510,7	30,7
D110402	id5	id3
81	12,5	31,3
82	27,1	31,6
83	27,7	31,7
84	27,7	31,5
85	28,5	31,5
86	30	33,1
87	36,5	34,5
88	99,3	34,5
89	469,8	36,4
90	476,3	37,9

D120402	id5	id3
81	26	34,7
82	48,1	34,7
83	58,1	36,1
84	136,3	38,5
85	136,5	38,6
86	136,9	39,1
87	156,9	39,1
88	157,7	39,1
89	182,3	39,5
90	182,4	39,2

D130402	id5	id3
81	0,6	35,5
82	1	35,4
83	2,1	35,5
84	112,5	35,7
85	113,9	36
86	114,3	36,1
87	142,1	36,6
88	142,7	36,8
89	404,6	42,6
90	449,4	41,3

D140402	id5	id3
81	1	43,2
82	15,9	43,2
83	17,9	45,5
84	18	45,7
85	19,2	45,5
86	31,3	45,9
87	62,7	48
88	87,2	48,4
89	87,4	48,4
90	93,7	50,1

D150402	id5	id3
81	23,6	45,5
82	26,5	47,4
83	57,1	48,2
84	57,8	48,9
85	122,1	51,1
86	122,3	51,5
87	149	51,8
88	167,3	51,8
89	169,2	52
90	246,1	52,4

D160402	id5	id3
81	0,7	50,8
82	22,9	51,5
83	42,5	51,4
84	64,8	51,6
85	86,9	52,1
86	89,3	52
87	89,2	52,4
88	90,7	52,8
89	107,7	53,2
90	117,1	55,6

D170402	id5	id3
81	35	62
82	108,8	62,3
83	134,4	64,7
84	463,8	63,5
85	496,1	66,2
86	563,6	63,1
87	1061,4	69,8
88	1142,8	70,2
89	1205,1	71,5
90	1282,4	69,1

D180402	id5	id3
81	387,4	71
82	474,4	73
83	533,9	73,3
84	830,6	72,2
85	942,2	77,5
86	1045,5	76,9
87	1306,9	70,4
88	1448,8	75,4
89	1615,8	75,3
90	1686,8	76,5

D190402	id5	id3
81	122,2	78,2
82	121,6	79,1
83	134,8	80,8
84	271,7	73,9
85	352,3	73,9
86	375	73,7
87	860,1	78,8
88	1217,7	83,5
89	1283,8	77,5
90	1377,1	85,9

D200402	id5	id3
81	324,2	81,1
82	409,2	81,8
83	596,4	85,8
84	745,3	88
85	837,7	86,5
86	933,2	92
87	1056,2	90,4
88	1182,8	87,2
89	1326,1	93,6
90	1418,3	98,3

D020402	id5	id3
91	0,2	1,2
92	0,1	1,3
93	0,2	1,4
94	0,2	1,1
95	0,4	1
96	0,5	1
97	0,4	1,3
98	0,5	0,9
99	0,5	1
100	0,6	0,8

D030402	id5	id3
91	0,1	4,4
92	0,4	4,3
93	0,3	4,2
94	0,6	4,5
95	0,7	4,7
96	0,7	4,5
97	0,6	4,4
98	0,5	4,2
99	1	4,4
100	1,3	4,7

D040402	id5	id3
91	7,7	10,8
92	25,6	11,1
93	26,2	11,4
94	26,4	11,5
95	26,7	11,7
96	27,2	11,9
97	38,6	11,8
98	40,3	12
99	84,3	12,1
100	114	11,8

D050402	id5	id3
91	17,8	13,3
92	86,7	13,5
93	88,5	13,4
94	152,2	14
95	151,4	14,5
96	449,7	14
97	505,5	14,5
98	554,5	14,6
99	599,1	14,8
100	648,7	15,5

D060402	id5	id3
91	16,7	13,4
92	37,1	13,3
93	433,8	17,5
94	499,9	18,6
95	541,7	18,5
96	718,7	15,4
97	742,1	15,3
98	824,7	15,1
99	825,3	15,8
100	903,4	16,1

D070402	id5	id3
91	0,3	18,1
92	25	18,1
93	31,4	19
94	53,7	18,8
95	54,2	18,8
96	55	19
97	56,1	19,9
98	57,2	20,3
99	146,8	20,6
100	171,9	21,1

D080402	id5	id3
91	104,5	23,4
92	117,5	24
93	135,4	24
94	136,2	24
95	136,6	24
96	179,2	25,2
97	207,8	26,1
98	306	25,7
99	333,1	26,5
100	347,5	26,3

D090402	id5	id3
91	97,5	29
92	174,6	30,1
93	193,6	30,6
94	193,7	30,4
95	194,2	31,1
96	245,3	31,6
97	625,8	28,4
98	711,7	32
99	867,2	28,5
100	976,6	28,8

D100402	id5	id3
91	13,6	32,2
92	84,7	32,2
93	87,4	33,1
94	187,9	34,5
95	188,2	34,1
96	194,5	34,2
97	201,9	34,9
98	271,8	33,5
99	272,5	34,8
100	284,4	35,1

D110402	id5	id3
91	47,7	38
92	136,1	39,4
93	137,6	39,7
94	306,5	39,2
95	407,9	39,9
96	569,3	41,8
97	625,3	42,5
98	791,4	39,3
99	937,1	40,7
100	996	40,9

D120402	id5	id3
91	0,5	39,9
92	0,8	41,6
93	1,8	41
94	49,9	41,7
95	89,8	40
96	175,9	43,7
97	177,6	44,1
98	180,6	44,8
99	240,2	44,4
100	247	45,9

D130402	id5	id3
91	132,8	43
92	133,8	43,2
93	133,3	43,6
94	145,8	45,3
95	181	45,1
96	271,8	45,5
97	473,5	46,4
98	542,7	46,9
99	547,9	48
100	607,3	45,5

D140402	id5	id3
91	31,8	53,2
92	32,6	54,4
93	55,9	54
94	58,1	56,5
95	79,7	55,9
96	78,6	56,3
97	107,7	57,7
98	119	60,8
99	119,1	61,1
100	120,2	62,2

D150402	id5	id3
91	66,8	52,3
92	136,1	55,3
93	176,7	53
94	208,3	56,6
95	214,5	57,8
96	241,4	58,4
97	265,2	58,4
98	290,8	58,6
99	288,5	58,6
100	318,8	62,1

D160402	id5	id3
91	0,9	57,4
92	1,5	56,8
93	2,4	56,7
94	3,1	57,3
95	3,8	57,3
96	69	57,9
97	70,1	57,7
98	81,5	58,4
99	95,6	60,5
100	109,6	61

D170402	id5	id3
91	116,7	71,8
92	566,2	72,5
93	657,4	72,8
94	682,4	73,1
95	780,4	76,9
96	904,4	78,4
97	1278,7	73,5
98	1321,7	74
99	1422,6	76,7
100	1692,1	76,7

D180402	id5	id3
91	85,6	85,1
92	149,7	85
93	257,3	75,7
94	390,8	75,5
95	391,9	75,9
96	463	76,7
97	576,7	79,3
98	621,5	78,5
99	619	78,8
100	630,8	78,7

D190402	id5	id3
91	102,4	84,5
92	207,5	87,6
93	315,2	84,1
94	332,3	84,8
95	364,1	84,9
96	395,6	85,1
97	449	85,4
98	462	85,6
99	552,1	86,5
100	554,3	86,9

D200402	id5	id3
91	113,3	98,7
92	152,3	101,6
93	186,8	102,4
94	261,4	103,4
95	533,2	90,4
96	609,6	90,5
97	755,7	91,2
98	915,5	103,3
99	1044,9	109,8
100	1153,5	108,5

APÊNDICE C - Código fonte do algoritmo ID5 utilizado na análise experimental

```
//=============================================
//                    ID5.H
//=============================================
// ESTRUTURA DE DADOS E PROTÓTIPOS DE FUNÇÃO
// PARA O ID5
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <sys/times.h>

#define MAX_STRING_NOME  20
#define MAX_NOME_ABREV   10
#define QTDEMAXVALOR     8
#define QTDEMAXEXEM      1000
#define QTDEMAXATRIB     21
#define QTDEMAXVLRATRIB  168
#define QTDEMAXCLASS     21
#define QTDEMAXREGRAS    1000
#define TAMMAXLISTAPAR   76
#define PARAMETRO1       1
#define PARAMETRO2       2

/*----------EXEMPLO---------- estrutura de dados
        static char *nome_tabex;
        static int qtde_classes;
                static int qtde_exemplos;
                static int qtde_exemplos_real;
                static int qtde_atributos;
        static int qtde_valor_tot;
                static int qtde_atrib_val;
                static int contra_exemplo;

//      TABELA DE ATRIBUTOS
        static struct atributos {
            int cod;
            int custo;
            unsigned tipo;
            int qtde_valores;
            int existe_buraco;
            int valores[QTDEMAXVLRATRIB];
            char nome[MAX_STRING_NOME];
            char nome_abrev[MAX_NOME_ABREV];
            };
         static atributos lista_atrib[QTDEMAXATRIB];

//      TABELA DE CLASSES
         static struct tab_classes {
                int cod;
                int freq_cl;
                char nome[MAX_STRING_NOME];
                };
                static tab_classes
         lista_classes[QTDEMAXCLASS];

//      TABELA DE VALORES
         static struct tab_valor {
                int cod;
                char nome[MAX_STRING_NOME];
                };
                static tab_valor
         lista_valor[QTDEMAXVALOR];

//      TABELA DE EXEMPLO
static int tabex[QTDEMAXEXEM][QTDEMAXATRIB];
static int freq_exemplo[QTDEMAXEXEM];
static int flag_exemplo[QTDEMAXEXEM];

//   ESTRUTURA DE CONTROLE
                static struct par_atrib_val {
                        int atrib;
                        int valor;
                };
                static par_atrib_val
lista_par[TAMMAXLISTAPAR];
                static int
matriz_par[QTDEMAXATRIB][QTDEMAXATRIB];
                static int
matriz_classe_par[TAMMAXLISTAPAR][QTDEMAXCLASS];

//   ESTRUTURA DE DADOS PARA INCREMENTAIS
FILE *f_arv;
FILE *f_imp;
FILE *fpr;
FILE *fpt;
int qtd_noh=0;
static struct Noh_m {int tipo_noh;
        int atributo;
        int qtd_filhos;
        int contador
[QTDEMAXATRIB][QTDEMAXVALOR][QTDEMAXCLASS]
;
        int valor           [QTDEMAXVALOR];
        struct Noh_m *end_filho [QTDEMAXVALOR];
        struct Noh_m *end_pai;
        int qtd_exem_lista;
        struct lista_ex *end_ex[100];  // 500 exemplos por folha
       }noh_mem;
static struct lista_ex { int num_ex;
          int qtd_atr_lista;
          int classe;
          int cod_atrib   [QTDEMAXATRIB];
          int valor_atrib [QTDEMAXATRIB];
          };
static struct Noh_m *p_filho;
static struct Noh_m *p_pai;
static struct Noh_m *p_raiz;
static struct Noh_d {  int tipo_noh;
        int atributo;
        int qtd_filhos;
        int contador
[QTDEMAXATRIB][QTDEMAXVALOR][QTDEMAXCLASS]
;
        int valor     [QTDEMAXVALOR];
        long end_filho [QTDEMAXVALOR];
        long end_pai  ;
        int qtd_exem_lista;
        struct lista_ex  lista_exemplo[100];
      } noh_d,d_filho;

static struct lote {float inicio; /*num primeiro exemplo do lote */
                    float fim;   /* num ultimo exemplo do lote */
                   }lote;

static char nome_dominio[8];
static char nome_alg[20];
static char nome_arq_arv[20];
static int ind_caminho[QTDEMAXATRIB];
static int val_caminho[QTDEMAXATRIB];
static int ind_caminho_sobe[QTDEMAXATRIB];
static int val_caminho_sobe[QTDEMAXATRIB];
static int ind_caminho_volta[QTDEMAXATRIB];
static int val_caminho_volta[QTDEMAXATRIB];
```

```
typedef struct regra {int at;
                      int gen;
                      int vlr;
                      int classe;
                      int nexmap;
                      }regra;
static regra
TabRegras[QTDEMAXREGRAS][QTDEMAXATRIB];
 static int r;  // indice de regras
 static int p;  // indice de premissas

/*----------exemplo------------ prototipo funcao EXEMPLO-*/
int carrega(char * nome_tex);
/*----------alg---------------------- prototipo funcao ALG--*/
void GravaRegras();
/*----------incre ------------------- prototipo funcao INCRE--*/
int ler_arq_arvore();
int ler_filhos_arq_arvore(struct Noh_m *p_pai);
struct Noh_m *construir_noh_memoria(struct Noh_m *p_pai);
int mostra_noh_disco(struct Noh_d noh, long end_disco);
int mostra_controle();

int gravar_arvore();
int grava_de_lote();
int gravar_filhos_na_arvore(struct Noh_m *p_pai,long
end_disco_pai);
 int formata_d_filho(long end_disco_pai);

int gerar_regra_incre();
int percorre_arvore(struct Noh_m *p_pai);
int mostrar_regra(int num_regra);
int mostrar_todas_regras();
void inicializa_tab_regras();
void  inicializar_regra(int num_regra);

int seta_parametros();
int mostra_arvore_memoria(struct Noh_m *p);
int mostra_filhos_memoria(struct Noh_m *p_pai);
int mostra_noh_memoria(struct Noh_m *p);

int formata_nome_arq_arv();
int abre_arquivo_arv_leitura();
int abre_arquivo_arv_gravacao();
int fecha_arquivo_arv();

int abre_arquivo_imp();
int fecha_arquivo_imp();
int ler_para_lote();
int ler_para_d_filho();
int ler_para_d_pai();
int ler_para_noh_d();
int grava_de_lote();
int grava_de_noh_d();
/*----------id5  ------------------------ prototipo funcaoos ---*/
int processa_ID5(int num_ex, struct  Noh_m  *p);
int incorpore_ex_na_arvore  (int num_ex, struct  Noh_m  *p);
void classifique_ex(int num_ex, struct  Noh_m  *p);
struct  Noh_m  *adiciona_novo_noh_classe(int num_ex,struct
Noh_m  *p);
int escolhe_melhor_atrib_lista(struct Noh_m *p);
float calcular_entropia(int atributo,struct Noh_m *p);
float entropia_valor(int at, int valor, struct Noh_m *p);
int adiciona_novo_noh_atributo(int atr,struct Noh_m
*ultimo_noh,int opcao);
int move_ex_lista_velha_para_nova(struct lista_ex
*ex_lista_velha,struct lista_ex *ex_lista_nova,int j);
struct lista_ex *guardar_at_val_na_lista(int num_ex,struct  Noh_m
*p,int qtd_ex_list,int qtd_atr_lista);
int atualizar_contadores(int num_ex, struct  Noh_m  *p);
void mostra_contadores(struct  Noh_m  *p);
void inicializa_endereco_valor(struct  Noh_m  *p);
void inicializa_contadores(struct  Noh_m  *p);
int confere_contadores(struct Noh_m *p);
int freq_at(int at,struct Noh_m *p);
int freq_at_val(int at,int valor,struct Noh_m *p);
int freq_at_val_classe(int at,int valor,int classe, struct Noh_m  *p);
int mostra_soma_atributos_caminho();
int mostra_soma_atributos_caminho_sobe();
int mostra_caminho_volta();
int soma_atributos_caminho();
void mostra_lista(struct Noh_m *p);
void atualiza_cont_noh_folha(int atr,int val, struct Noh_m *p, int
opcao);
void processa_primeiro_ex(int num_ex, struct Noh_m *p);
void  reestruture_ID5(int num_ex, struct Noh_m *p);
int  escolhe_melhor_atr_do_noh(struct Noh_m *p);
void inicializa_caminho();
void coloca_atr_no_caminho(int num_ex, int at);
void inicializa_caminho_volta();
void coloca_atr_no_caminho_sobe(int val , int at);
void inicializa_caminho_sobe();
void atualizar_caminho(struct Noh_m *p);
void atualizar_caminho_volta(struct Noh_m *p);
void move_caminho_para_caminho_sobe();

struct  Noh_m  *sobe_melhor_atributo(int atr, struct Noh_m *p);
struct Noh_m *cria_arvore_para_filho(int indicefilho, struct Noh_m
*p);
void copia_contadores(struct Noh_m *origem,struct Noh_m
*destino);
void copia_dados_noh(struct Noh_m *origem,struct Noh_m
*destino);
void copia_dados_lista(struct Noh_m *origem,struct Noh_m
*destino);
void copia_dados_filhos(struct Noh_m *origem,struct Noh_m
*destino);
void troca_atr_pai_filho(struct Noh_m *pai);
void mergear_arvore(struct Noh_m *x,struct Noh_m *y);
void soma_contadores(struct Noh_m *x,struct Noh_m *y);
void testa_poda(struct Noh_m *p);
void poda(struct Noh_m *p);
void incorpora_lista_do_filho(int i,struct Noh_m *p);
int  mostra_defines();
void mostra_atributos();
void mostra_valores();
char *nome_atributo(int cod_at);
char *nome_valor(int cod_val);
char *nome_classe(int cod_classe);
int recebe_fim_lote();
int mostrar_todas_regras();
int mostrar_regra(int num_regra);


//=============================================================
//                          ID5.C
//=============================================================
// Esta versao do algoritmo Id5 contém métodos utilizados na
// depuração do  algoritmo  bem como métodos utilizados no
// processo de apuração dos  tempos de execução do mesmo
//=============================================================
#include  "id5.h"

//=============================================================MAIN
void main()
{float  tempo_inicio, tempo_fim;
 struct tms buffer;
 FILE *ftb;   // arquivo.tab (padrao ambiente A4)
 FILE *ftbc; //arquivo.tabc (padrao ambiente A4)
 FILE *fid5; //arquivo.id5 (contem arvore gerada pelo ID5)
 FILE *f1;    // temp.arq (contem tipo de dominio a processar)
 FILE *fpt;    //estatistica.arq (contem resultados da apuração de
tempos de processamento)
abre_arquivo_imp();
if (( f1=fopen("temp.arq", "r"))==NULL)
    {printf("temp.arq nao pode ser aberto");
     return ;
    };
  char atrib  [3];
  char val    [3];
  char classe [3];
  char exem   [4];
  fscanf(f1,"%2s%2s%2s%2s%3s",atrib,val,classe,exem);
  strcpy(nome_dominio,"D");
  strcat(nome_dominio,atrib);
  strcat(nome_dominio,val);
  strcat(nome_dominio,classe);
  strcat(nome_dominio,exem);
  fclose(f1);

if (( fpt=fopen("estatistica.arq", "a"))==NULL)
    {printf("estatistica.arq nao pode ser aberto");
     return ;
```

```
    };

char nome_arq_tab[13];
strcpy(nome_arq_tab,nome_dominio);
strcat(nome_arq_tab,".TAB");
if ((ftb=fopen(nome_arq_tab, "r"))==NULL)
        {printf("\narq exemplos %s nao pode ser
aberto\n",nome_arq_tab);
         return;
        };
char nome_arq_tabc[13];
strcpy(nome_arq_tabc,nome_dominio);
strcat(nome_arq_tabc,".TABC");

if ((ftbc=fopen(nome_arq_tabc, "r"))==NULL)
        {printf("\narq tbc %s nao pode ser aberto\n",nome_arq_tabc);
         return ;
        };
char nome_arq_id5[13];
strcpy(nome_arq_id5,nome_dominio);
strcat(nome_arq_id5,".ID5");
if ((fid5=fopen(nome_arq_id5, "r"))==NULL)
        {printf("\narq arvore %s nao pode ser
aberto\n",nome_arq_id5);
         return ;};
carrega(nome_arq_tab);


if (contra_exemplo != 0)
     {printf("Este dominio tem contra exemplos -> ID5 nao
processa");
      return;
     };
if (mostra_defines()== 1)
     {printf("\n OS DEFINES ESTAO SUBDIMENSIONADOS");
      return;
     };
sprintf(nome_arq_arv,"%s",nome_arq_id5);
ler_arq_arvore();


int inicio=lote.fim;
int fim =qtde_exemplos_real;   // para processar ate ultimo exemplo
if (fim == -1)
     {return;
     };

//------------------------------------------------------------ ID5
float tempo_medio=0;
float total_vezes=1;
for (int k=0;k<total_vezes;k++)
{printf(" \nvezs=%d ",k);
 for ( int gg=inicio; gg< fim; gg++)
     {ult_noh=0;
      inicializa_caminho();
      inicializa_caminho_sobe();
      if (gg==10)
        { if ( times(&buffer) == -1 )
            {printf( "\n Falha na funcao times");
             return;
            }
          else
            {tempo_inicio = buffer.tms_utime;
            };
        };
      processa_ID5(gg, p_raiz);
      lote.inicio=inicio+1;
      lote.fim=gg+1;
   };

if ( times(&buffer) == -1 )
   {printf( "\n Falha na funcao times");
   return;
   }
else
   {tempo_fim = buffer.tms_utime;
   printf( "\n tempo inicio=%04.0f",tempo_inicio);
   tempo_inicio = tempo_fim - tempo_inicio;
   };
mostra_controle();
gravar_arvore();
inicializa_tab_regras();
gerar_regra_incre();
mostrar_todas_regras();
GravaRegras();
fprintf(fpt, "%sA%03.0fA",nome_dominio ,tempo_inicio);
fclose(fid5);
strcpy(nome_arq_id5,nome_dominio);
strcat(nome_arq_id5,".ID5");
if ((fid5=fopen(nome_arq_id5, "w"))==NULL)
        {printf("\n %s nao pode ser aberto para
gracvacao\n",nome_arq_id5);
         return ;};
fprintf(fid5,"00000000");
fclose(fid5);
};  //final do for total_vezes


fclose(ftb);
fclose(ftbc);
fclose(fid5);

fclose(fpr);
fclose(fpt);
return;
}
//================================== recebe_fim_lote
int recebe_fim_lote()
{int fim =0;
 int erro=1;
 while (erro==1)
 {erro=0;
  printf("\n QUANT_EX_EXISTENTES NA
TABELA=%d",qtde_exemplos_real);
  printf("\n primeiro exemplo deste processamento serah o
%f",lote.fim+1);
if (lote.fim >= qtde_exemplos_real)
    {printf("\nTODOS OS EXEMPLOS JA FORAM
PROCESSADOS");
     return -1;
    };
 printf("\n digite numero do ultimo exemplo a processar=>");
 scanf("%d",&fim);
 printf("\n o numero do ultimo exemplo recebido foi = %d",fim);
if (fim < lote.fim +1)
    { printf("\n ULTIMO EXEMPLO DEVE SER MAIOR QUE
%f",lote.fim);
      erro=1;
    };
if (fim > qtde_exemplos_real)
    { printf("\n ULTIMO EXEMPLO DEVE SER MENOR QUE
%d",qtde_exemplos_real+1);
      erro=1;
    };
 };
return fim;
}


// ================================== mostra defines
int  mostra_defines()
{
if (qtde_atributos > QTDEMAXATRIB)
   {printf("\n quantidade de atributos subdimensionada" );
   return 1;
   };
if (qtde_valor_tot > QTDEMAXVALOR)
   {printf("\n quantidade de valores  subdimensionada" );
   return 1;
   };
if (qtde_exemplos_real > QTDEMAXEXEM )
   {printf("\n quantidade de exemplos  subdimensionada" );
   return 1;
   };
if (qtde_atrib_val > QTDEMAXVLRATRIB)
   {printf("\n quantidade de val_atrib subdimensionada" );
   return 1;
   };
if (qtde_classes > QTDEMAXCLASS)
   {printf("\n quantidade de classes  subdimensionada" );
   return 1;
   };
```

```
printf("\n mQTDEMAXREGRAS =%d ", QTDEMAXREGRAS);
printf("\n mTAMMAXLISTAPAR=%d ",TAMMAXLISTAPAR
);
return 0;
}

//=========================================== mostra atributos
void mostra_atributos()
{ for (int i=0;i<qtde_atributos;i++) {
    printf("\n%2d",lista_atrib[i].cod);
    printf("%30s %11s",lista_atrib[i].nome,
lista_atrib[i].nome_abrev);
            }
return;
}

//=========================================== mostra  valores
void mostra_valores()
{ for (int i=0;i<qtde_valor_tot;i++) {
    printf("\n%03d %30s",lista_valor[i].cod,lista_valor[i].nome);
            }
return;
}
//=========================================== mostra_todas regras
int mostrar_todas_regras()
{int cont=0;
 if (p_raiz->tipo_noh==1) // caso da arvore soh com raiz
   {printf("\nNao ha regras ainda, arvore soh tem o raiz");
    return 1;
   };

 for (int i=0; i<r; i++)
    {printf("\nregra %d =>",i);
     for (int j=0;  TabRegras[i][j].classe == -1  ;j++)
       {printf("%s = "  ,lista_atrib[TabRegras[i][j].at].nome_abrev);
        printf("%s & ",  lista_valor[TabRegras[i][j].vlr].nome);
       };
     printf("%s = "      ,lista_atrib[TabRegras[i][j].at].nome_abrev);
     printf("%s "        ,lista_valor[TabRegras[i][j].vlr].nome);
     printf("entao %s", lista_classes[ TabRegras[i][j].classe].nome);
     printf("\n  map=%d ",  TabRegras[i][j].nexmap);
     cont += TabRegras[i][j].nexmap;
    };
printf ("\nSoma dos exemplos mapeados = %d",cont);
printf ("\nTotal de regras = %d",r);
return 1;
}
//=========================================== atualizar_caminho_volta
void atualizar_caminho_volta(struct Noh_m *p)
{if (p==p_raiz)
   {return;
   }
 else
   {ind_caminho_volta[p->end_pai->atributo]=1;
    for (int i=0;i<p->end_pai->qtd_filhos;i++)
       { if (p->end_pai->end_filho[i]==p)
           {val_caminho_volta[p->end_pai->atributo]=p->end_pai-
>valor[i];
            atualizar_caminho_volta(p->end_pai);
            };
       };
   };
return;
}

//=========================================== inicializa_caminho_volta
void inicializa_caminho_volta()
{for (int at=0; at< qtde_atributos; at++)
    {ind_caminho_volta[at]=0;
     val_caminho_volta[at]=0;
    };
 return;
}

// ========================================== processa_ID5
int processa_ID5(int num_ex, struct  Noh_m  *p)
{incorpore_ex_na_arvore( num_ex, p);
 inicializa_caminho();
 inicializa_caminho_sobe();
 reestruture_ID5(num_ex, p_raiz);
return 1;
}
//=========================================== reestruture id5
void reestruture_ID5(int num_ex, struct Noh_m *p)
{
int at;
if (p->tipo_noh==1)
  { return;
  }
else
  {at=escolhe_melhor_atr_do_noh(p);
   coloca_atr_no_caminho(num_ex,at);
   if (at != p->atributo)
     {move_caminho_para_caminho_sobe();
      sobe_melhor_atributo(at,p);
     };
   for (int i=0; i< p->qtd_filhos; i++)
     {if (p->valor[i] == tabex[num_ex][p->atributo+1])
        {reestruture_ID5(num_ex, p->end_filho[i]);
        };
     };
  };
return;
}
//=========================================== sobe_melhor_atributo
struct Noh_m *sobe_melhor_atributo(int at, struct Noh_m *p)
{struct Noh_m *end_arv[QTDEMAXVALOR];
for (int i=0;i<p->qtd_filhos; i++)
  {if (p->end_filho[i]->tipo_noh==1)
     {coloca_atr_no_caminho_sobe(p->valor[i],p->atributo);
      adiciona_novo_noh_atributo(at,p->end_filho[i],2);
     }
   else
     {if (p->end_filho[i]->atributo != at)
        {coloca_atr_no_caminho_sobe(p->valor[i],p->atributo);
         sobe_melhor_atributo(at,p->end_filho[i]);
        };
     };
  };
for (int j=0; j<p->qtd_filhos; j++)
 {end_arv[j]=cria_arvore_para_filho(j,p);
  troca_atr_pai_filho(end_arv[j]);
 };
if (p->qtd_filhos>1)
  {for (int k=1; k<p->qtd_filhos; k++)
     {mergear_arvore(end_arv[0],end_arv[k]);
      free((char *)end_arv[k]);
      desalocado++;
     };
  };
testa_poda(end_arv[0]);
copia_dados_noh(end_arv[0],p);
return end_arv[0];
}
//=========================================confere_contadores
int confere_contadores(struct Noh_m *p)
{int
tot_cont_filhos[QTDEMAXATRIB][QTDEMAXVALOR][QTDE
MAXCLASS];
int i,j,k,f=0;
printf("\n CONfERINDO CONTADORES DO NOH=%p",p);
if (p->tipo_noh == 0)
 { for (i=0; i<QTDEMAXATRIB; i++)
     { for (j=0; j<QTDEMAXVALOR; j++)
         {for (k=0; k<QTDEMAXCLASS; k++)
            {tot_cont_filhos[i][j][k]=0;
            };
         };
     };
     printf("\ zerou CONTADORES DO NOH=%p",p);

   for (f=0; f<p->qtd_filhos; f++)
      {for (i=0; i<qtde_atributos; i++)
        {for (j=0; j<lista_atrib[i].qtde_valores; j++)
          {for (k=0; k<qtde_classes; k++)
             {tot_cont_filhos[i][j][k]+=p->end_filho[f]-
>contador[i][j][k];
              };
          };
        };
   for (i=0; i<qtde_atributos; i++)
```

```
      {for (j=0; j<lista_atrib[i].qtde_valores; j++)
         {for (k=0; k<qtde_classes; k++)
            {if (tot_cont_filhos[i][j][k]!=p->contador[i][j][k])
              {printf("\n CONT NAO OK NOH=%p i=%d j=%d
k=%d",p,i,j,k);
              };
            };
         };
      };

   for (f=0; f<p->qtd_filhos; f++)
      {confere_contadores(p->end_filho[f]);
      };
 }
else
 {return 1;
 };
return 1;
}
//=========================================== percorre arvore
int percorre_arvore(struct Noh_m *p_pai)
{
 if (p_pai->tipo_noh == 0)    /* se noh eh GALHO, ou tem filho */
   {TabRegras[r][p].at =p_pai->atributo;
for (int k=0; k<p_pai->qtd_filhos; k++)
       {TabRegras[r][p].vlr=p_pai->valor[k];
        p++;
       percorre_arvore(p_pai->end_filho[k]);
       };
   p--;
   }
 else
   {TabRegras[r][p-1].classe=p_pai->atributo;
   TabRegras[r][p-1].nexmap=p_pai->qtd_exem_lista;
   inicializar_regra(r+1);
   r++;
   p--;
   return 1;
  };
return 1;
}

//=========================================== GravaRegras
void GravaRegras ( )
{
int cont=0;
fpr = fopen ("regras.arq", "a");
fprintf (fpr,"dominio =%s ",nome_dominio);
for ( int i=0; i < r; i++ )
 {fprintf (fpr, "\n%3d Se ", i+1);
  for ( int j=0; j < QTDEMAXATRIB; j++ )
   {if ( TabRegras[i][j].classe == -1 )
     {fprintf (fpr,"at%d = %d  &
",TabRegras[i][j].at,TabRegras[i][j].vlr);
     }
    else
     {fprintf (fpr,"at%d = %d
",TabRegras[i][j].at,TabRegras[i][j].vlr);
      fprintf (fpr,"===> classe=%d
%d",TabRegras[i][j].classe,TabRegras[i][j].nexmap);
      cont += TabRegras[i][j].nexmap;
                          break;
     };
   };
 };
fprintf (fpr,"\nSoma dos exemplos mapeados = %d\n\n",cont);
}

//=========================================== inicializar_regra
void  inicializar_regra(int r)
{for (int m=0; m<QTDEMAXATRIB ;m++)
   {TabRegras[r][m].at   =-1;
    TabRegras[r][m].vlr  =-1;
    TabRegras[r][m].classe=-1;
    TabRegras[r][m].gen  =-1;
    TabRegras[r][m].nexmap=0;
    };
 for (int a=0; a<QTDEMAXATRIB ;a++)
   {TabRegras[r][a].at   =TabRegras[r-1][a].at;
    TabRegras[r][a].vlr  =TabRegras[r-1][a].vlr;
    TabRegras[r][a].gen  =TabRegras[r-1][a].gen;
    TabRegras[r][a].classe=-1;
    TabRegras[r][a].nexmap=0;
    };
return ;
}

//=========================================== inicializa_tab_regras
void inicializa_tab_regras()
{for (int i=0;i<QTDEMAXREGRAS; i++)
   {for (int j=0;j<QTDEMAXATRIB; j++)
      { TabRegras[i][j].at   = -1;
        TabRegras[i][j].vlr  = -1;
        TabRegras[i][j].classe= -1;
        TabRegras[i][j].gen  = -1;
        TabRegras[i][j].nexmap= 0;
      };
   };
 r=p=0;
 return;
}
//=========================================== gerar_regra
int gerar_regra_incre()
{ r=0;p=0;
 p_pai=p_raiz;
 percorre_arvore(p_pai);
 return 1;
}
//=========================================== testa_poda
void testa_poda(struct Noh_m *p)
{int ind_poda=1;
int achou_folha=0;
int classe=999;
 if (p->tipo_noh==1)
   {return;
   }
 else
   {ind_poda=1;
    achou_folha=0;
    for (int i=0;i<p->qtd_filhos;i++)
      {if (p->end_filho[i]->tipo_noh==0)
         {ind_poda=0;
         testa_poda(p->end_filho[i]);
         }
       else
         {if (achou_folha==0)
            {classe=p->end_filho[i]->atributo;
             achou_folha=1;
            };
          if (p->end_filho[i]->atributo !=classe)
            {ind_poda=0;
            };
         };
      };
   };
if (ind_poda==1)
  {poda(p);
   return;
  };
return;
}
//=========================================== poda
void poda(struct Noh_m *p)
{
p->tipo_noh=1;
p->qtd_exem_lista=0;
for (int i=0;i<p->qtd_filhos;i++)
   {incorpora_lista_do_filho(i,p);
   };
p->atributo=p->end_filho[0]->atributo;
for (int j=0;j<p->qtd_filhos;j++)
   {free((char *)p->end_filho[j]);
    desalocado_poda++;
   };
p->qtd_filhos=0;
return;
}
//=========================================== incorpora_lista_do_filho
void incorpora_lista_do_filho(int f, struct Noh_m *p)
{int x;
for (int k=0; k<p->end_filho[f]->qtd_exem_lista; k++)
```

```
  {p->end_ex[p->qtd_exem_lista]=(struct lista_ex
*)malloc(sizeof(struct lista_ex));
  if (!p->end_ex[p->qtd_exem_lista])
    {printf ("\n FALTOU MEMORIA PARA ALOCACAO EM
INCORPORA_LISTA_DO_FILHO");
     return;
    };
  p->end_ex[p->qtd_exem_lista]->num_ex      = p->end_filho[f]-
>end_ex[k]->num_ex;
  p->end_ex[p->qtd_exem_lista]->classe      = p->end_filho[f]-
>end_ex[k]->classe;
  p->end_ex[p->qtd_exem_lista]->qtd_atr_lista = p->end_filho[f]-
>end_ex[k]->qtd_atr_lista;
  for (int j=0;j<p->end_filho[f]->end_ex[0]->qtd_atr_lista;j++)
    {p->end_ex[p->qtd_exem_lista]->cod_atrib[j] = p-
>end_filho[f]->end_ex[k]->cod_atrib[j];
     p->end_ex[p->qtd_exem_lista]->valor_atrib[j] = p-
>end_filho[f]->end_ex[k]->valor_atrib[j];
    };
  x = p->end_ex[p->qtd_exem_lista]->qtd_atr_lista;
  p->end_ex[p->qtd_exem_lista]->cod_atrib[x]  = p->atributo;
  p->end_ex[p->qtd_exem_lista]->valor_atrib[x] = p->valor[f];
  p->end_ex[p->qtd_exem_lista]->qtd_atr_lista++;
  p->qtd_exem_lista++;
};
return;
}
//================================== copia_dados_noh
void copia_dados_noh(struct Noh_m *origem,struct Noh_m
*destino)
{destino->tipo_noh     =origem->tipo_noh;
 destino->atributo     =origem->atributo;
 copia_contadores(origem,destino);
 copia_dados_lista(origem,destino);
 copia_dados_filhos(origem,destino);
 for (int i=0;i< origem->qtd_filhos;i++)
   {origem->end_filho[i]->end_pai=destino;
   };
return;
}
//================================== copia_lista
void copia_dados_lista(struct Noh_m *origem,struct Noh_m
*destino)
{destino->qtd_exem_lista =origem->qtd_exem_lista;
 for (int i=0;i<origem->qtd_exem_lista;i++)
    {destino->end_ex[i] = origem->end_ex[i];
    };
return;
}

//================================== copia_dados_filhos
void copia_dados_filhos(struct Noh_m *origem,struct Noh_m
*destino)
{ destino->qtd_filhos    =origem->qtd_filhos;
 for (int i=0;i<origem->qtd_filhos;i++)
    {destino->valor[i]   =origem->valor[i];
     destino->end_filho[i]=origem->end_filho[i];
    };
return;
}
//================================== escolhe_melhor_atr_do_noh
int escolhe_melhor_atr_do_noh(struct Noh_m *p)
{int melhor_atributo;
 float menor_entropia=9999;
 float entropia[QTDEMAXATRIB];
 for (int atr=0; atr < qtde_atributos; atr++)
   { if (ind_caminho[atr]==0)
       { entropia[atr]=calcular_entropia(atr,p);
         if (entropia[atr] < menor_entropia)
           {menor_entropia=entropia[atr];
            melhor_atributo=atr;
           };
       };
   };
return melhor_atributo;
}
//================================== coloca_atr_no_caminho
void coloca_atr_no_caminho(int num_ex, int at)
{ind_caminho[at]=1;
 val_caminho[at]=tabex[num_ex][at];
 return;
}
//================================== coloca_atr_no_caminho_sobe
void coloca_atr_no_caminho_sobe(int val, int at)
{ind_caminho_sobe[at]=1;
 val_caminho_sobe[at]= val;
 return;
}
//================================== inicializa_caminho
void inicializa_caminho()
{for (int at=0; at< qtde_atributos; at++)
   {ind_caminho[at]=0;
    val_caminho[at]=0;
   };
 return;
}
//================================== inicializa_caminho sobe
void inicializa_caminho_sobe()
{for (int at=0; at< qtde_atributos; at++)
   {ind_caminho_sobe[at]=0;
   val_caminho_sobe[at]=0;
   };
 return;
}
//==========================move_caminho_para_caminho_sobe
void move_caminho_para_caminho_sobe()
{for (int at=0;at<qtde_atributos; at++)
   {ind_caminho_sobe[at]=ind_caminho[at];
    val_caminho_sobe[at]=val_caminho[at];
   };
return;
}
//================================== troca_atr_pai_filho
void troca_atr_pai_filho(struct Noh_m *pai)
{static struct Noh_m *novo[QTDEMAXVALOR];
 static struct Noh_m *filho;
filho=pai->end_filho[0];
for (int i=0; i<filho->qtd_filhos; i++)
  {novo[i]=(struct Noh_m *)malloc(sizeof(struct Noh_m));
   if (!novo[i])
     {printf ("\n FALTA MEMORIA PARA ALOCACAO EM
TROCA_ATR_PAI_FILHO");
      return;
     };
   inicializa_endereco_valor(novo[i]);
   inicializa_contadores(novo[i]);
   copia_contadores(filho->end_filho[i] , novo[i]);
   novo[i]->tipo_noh      = 0;
   novo[i]->atributo      = pai->atributo;
   novo[i]->qtd_filhos    = 1;
   novo[i]->end_filho[0]  = filho->end_filho[i];
   novo[i]->valor[0]      = pai->valor[0];
   novo[i]->end_pai       = pai;
   novo[i]->qtd_exem_lista = 0;
   filho->end_filho[i]->end_pai=novo[i];
  };

pai->atributo=filho->atributo;
for (int ii=0; ii< filho->qtd_filhos; ii++)
  { pai->qtd_filhos   = filho->qtd_filhos;
    pai->valor[ii]    = filho->valor[ii];
    pai->end_filho[ii] = novo[ii];
  };
return;
}
//================================== mergear_arvore
void mergear_arvore(struct Noh_m *a, struct Noh_m *b)
{int encontrou_igual=0;
 int m;
 int t;
for (int j=0;j<b->qtd_filhos; j++)
 { encontrou_igual=0;
   for (int i=0;i<a->qtd_filhos;i++)
     {if (b->valor[j] == a->valor[i])
        {m=i;
         encontrou_igual=1;
        };
     };
   if (encontrou_igual==1)
     {a->end_filho[m]->qtd_filhos++;
      t = a->end_filho[m]->qtd_filhos -1;
```

```
      a->end_filho[m]->valor[t]          = b->end_filho[j]-
>valor[0];
      a->end_filho[m]->end_filho[t]      = b->end_filho[j]-
>end_filho[0];
      a->end_filho[m]->end_filho[t]->end_pai = a->end_filho[m];
      soma_contadores(a->end_filho[m], a->end_filho[m]-
>end_filho[t]);
      soma_contadores(a,          a->end_filho[m]->end_filho[t]);
      free((char *)b->end_filho[j]);
      desalocado++;
     }
   else
     {a->qtd_filhos++;
     a->end_filho[a->qtd_filhos-1]      = b->end_filho[j];
     a->valor   [a->qtd_filhos-1]       = b->valor   [j];
     a->end_filho[a->qtd_filhos-1]->end_pai = a;
     soma_contadores(a, a->end_filho[a->qtd_filhos-1]);
     };
 };
return;
}
//================================== soma_contadores
void soma_contadores(struct Noh_m *x,struct Noh_m *y)
{int i,j,k=0;
 for (i=0; i<QTDEMAXATRIB; i++)
   { for (j=0; j<QTDEMAXVALOR; j++)
       { for (k=0; k<QTDEMAXCLASS; k++)
           {x->contador[i][j][k]=x->contador[i][j][k]+y-
>contador[i][j][k];
           };
       };
   };
return;
}
// ================================ cria arvore para filho
struct Noh_m *cria_arvore_para_filho(int j,struct Noh_m *p)
{static struct Noh_m *novo;
novo=(struct Noh_m *)malloc(sizeof(struct Noh_m));
if (!novo)
  {printf ("\n FALTA MEMORIA PARA ALOCACAO EM
CRIA_ARVORE_PARA_FILHO");
      return NULL;
     };
inicializa_endereco_valor(novo);
inicializa_contadores(novo);
copia_contadores(p->end_filho[j], novo);
novo->atributo     =p->atributo;
novo->tipo_noh     =p->tipo_noh;
novo->qtd_exem_lista=0;
novo->end_pai      =p->end_pai;
novo->qtd_filhos   =1;
novo->end_filho[0] =p->end_filho[j];
novo->valor[0]     =p->valor[j];
return novo;
}
//================================== copia_contadores
void copia_contadores(struct Noh_m *origem,struct Noh_m
*destino)
{int i,j,k=0;
 for (i=0; i<QTDEMAXATRIB; i++)
   { for (j=0; j<QTDEMAXVALOR; j++)
       { for (k=0; k<QTDEMAXCLASS; k++)
           {destino->contador[i][j][k]=origem->contador[i][j][k];
           };
       };
   };
return;
}

// ============================== incorpore_ex_na_arvore
int incorpore_ex_na_arvore  (int num_ex, struct  Noh_m  *p)
{int qtd_atr_caminho=0;
struct Noh_m *ultimo_noh;
inicializa_caminho();
classifique_ex(num_ex, p);
ultimo_noh=ult_noh;
qtd_atr_caminho= soma_atributos_caminho();
int atr_lista= qtde_atributos-qtd_atr_caminho;
ultimo_noh->end_ex[ultimo_noh-
>qtd_exem_lista]=guardar_at_val_na_lista(
num_ex,ultimo_noh,ultimo_noh->qtd_exem_lista,atr_lista);
ultimo_noh->qtd_exem_lista++;
return 1;
}
// ======================== adiciona_novo_noh_atributo
int adiciona_novo_noh_atributo(int atr,struct Noh_m
*ultimo_noh,int opcao)
{int ult_ex_lista;
int atualizou_lista=0;
struct Noh_m *novo;
ultimo_noh->tipo_noh=0;
ultimo_noh->atributo=atr;
for (int i=0; i < ultimo_noh->qtd_exem_lista; i++)
 {for (int j=0; j < ultimo_noh->end_ex[i]->qtd_atr_lista; j++)
   {if (ultimo_noh->end_ex[i]->cod_atrib[j]==atr)
     {atualizou_lista=0;
     for (int k=1;k < ultimo_noh->qtd_filhos+1;k++)
      {if (ultimo_noh->valor[k-1]==ultimo_noh->end_ex[i]-
>valor_atrib[j])
         { ult_ex_lista=ultimo_noh->end_filho[k-1]-
>qtd_exem_lista;
          ultimo_noh->end_filho[k-1]-
>end_ex[ult_ex_lista]=(struct lista_ex *)malloc(sizeof(struct
lista_ex));
          if (!ultimo_noh->end_filho[k-1]->end_ex[ult_ex_lista])
            {printf ("\n FALTA MEMORIA PARA ALOCACAO
EM adiciona_novo_noh_atributo");
             return 0;
            };
          move_ex_lista_velha_para_nova(ultimo_noh-
>end_ex[i],ultimo_noh->end_filho[k-1]->end_ex[ult_ex_lista],j);
          ultimo_noh->end_filho[k-1]->qtd_exem_lista++;
          atualiza_cont_noh_folha(atr,ultimo_noh->valor[k-
1],ultimo_noh->end_filho[k-1],opcao);
          atualizou_lista=1;
         };
       };
     if (atualizou_lista==0)
       {novo=(struct Noh_m *)malloc(sizeof(struct Noh_m)); /*-
-aloca area-*/
        if (!novo)
          {printf ("\n FALTA MEMORIA PARA ALOCACAO
EM adiciona_novo_noh_atributo");
           return 0;
          };
        ultimo_noh->end_filho[ultimo_noh->qtd_filhos]=novo;
        ultimo_noh->valor   [ultimo_noh-
>qtd_filhos]=ultimo_noh->end_ex[i]->valor_atrib[j];
        ultimo_noh->qtd_filhos++;
        novo->tipo_noh=1;
        inicializa_endereco_valor(novo);
        novo->atributo=ultimo_noh->end_ex[i]->classe;
        novo->qtd_filhos=0;
        novo->end_pai=ultimo_noh;
          novo->end_ex[0]=(struct lista_ex *)malloc(sizeof(struct
lista_ex));
          if (!novo->end_ex[0])
            {printf ("\n FALTA MEMORIA PARA
ALOCACAO EM adiciona_novo_noh_atributo");
             return 0;
            };
          novo->qtd_exem_lista=1;
          move_ex_lista_velha_para_nova(ultimo_noh-
>end_ex[i],novo->end_ex[0],j);
         inicializa_contadores(novo);
        atualiza_cont_noh_folha(atr, ultimo_noh-
>valor[ultimo_noh->qtd_filhos -1],novo,opcao);
       };
     };
   };
 };
ultimo_noh->qtd_exem_lista=0;
return 1;
}
// ============================ atualiza contadores do noh
folha
void atualiza_cont_noh_folha(int atr, int valor,struct Noh_m *p,int
opcao)
{int at;
 int val;
 int classe;
classe=p->end_ex[0]->classe;
```

```
if (opcao==1)     // caso do caminho de classificacao
 {
  for (int j=0; j< p->end_ex[0]->qtd_atr_lista ; j++)
       {at =p->end_ex[p->qtd_exem_lista-1]->cod_atrib[j];
        val=p->end_ex[p->qtd_exem_lista-1]->valor_atrib[j];
        for (int m=0;m< lista_atrib[at].qtde_valores;m++)
           {if (lista_atrib[at].valores[m] == val)
              { val=m;
              };
           };
        p->contador[at][val][classe]++;
       };
  for (int k=0; k< qtde_atributos ; k++)
     {if (ind_caminho[k]==1)
        {at=k;
         val=val_caminho[k];
         for (int n=0;n< lista_atrib[at].qtde_valores;n++)
            {if (lista_atrib[at].valores[n] == val)
               { val=n;
                 p->contador[at][val][classe]++;
               };
            };
        };
     };
   for (int x=0;x< lista_atrib[atr].qtde_valores;x++)
    {if (lista_atrib[atr].valores[x] == valor)
       { val=x;
         p->contador[atr][val][classe]++;
       };
    };
  };

if (opcao==2)
{
 int i;
 inicializa_contadores(p);
 classe=p->end_ex[0]->classe;

 for (i=0; i< p->qtd_exem_lista; i++)
    {for (int j=0; j< p->end_ex[0]->qtd_atr_lista ; j++)
       {at =p->end_ex[i]->cod_atrib[j];
        val=p->end_ex[i]->valor_atrib[j];
        for (int m=0;m< lista_atrib[at].qtde_valores;m++)
           {if (lista_atrib[at].valores[m] == val)
              { val=m;
              };
           };
        p->contador[at][val][classe]++;
       };
    };
 inicializa_caminho_volta();     // 11/01/95
atualizar_caminho_volta(p);      // 11/01/95
for (int k=0; k< qtde_atributos ; k++)
   {if (ind_caminho_volta[k]==1)
      {at=k;
       val=val_caminho_volta[k];
       for (int n=0;n< lista_atrib[at].qtde_valores;n++)
          {if (lista_atrib[at].valores[n] == val)
             { val=n;
               for (i=0;i<p->qtd_exem_lista;i++)
                  {p->contador[at][val][classe]++;
                  };
             };
          };
       };
   };
};
return;
}
// ========================= mostra_soma_atributos_caminho
int mostra_soma_atributos_caminho()
{int qtd_atr_caminho=0;
for (int k=0; k<qtde_atributos; k++)
   {fprintf(f_imp,"\nATR=%d VAL=%d
IND=%d",k,val_caminho[k],ind_caminho[k]);
    if (ind_caminho[k]==1)
      {qtd_atr_caminho++;};
   };
return qtd_atr_caminho;
}
// ============================== mostra_caminho_volta
int mostra_caminho_volta()
{int qtd_atr_caminho=0;
printf("\nCAM_VOLTA at-val-ind==>");
for (int k=0; k<qtde_atributos; k++)
   {
    printf(" %d-
%d=%d",k,val_caminho_volta[k],ind_caminho_volta[k]);
    if (ind_caminho_volta[k]==1)
      {qtd_atr_caminho++;};
   };
return qtd_atr_caminho;
}
//
=====================================soma_atributos_caminh
o
int soma_atributos_caminho()
{int qtd_atr_caminho=0;
for (int k=0; k<qtde_atributos; k++)
   {if (ind_caminho[k]==1)
      {qtd_atr_caminho++;};
   };
return qtd_atr_caminho;
}
//
=========================mostra_soma_atributos_caminho_sob
e
int mostra_soma_atributos_caminho_sobe()
{int qtd_atr_caminho=0;
for (int k=0; k<qtde_atributos; k++)
   {fprintf(f_imp,"\nATR=%d VAL=%d
IND=%d",k,val_caminho_sobe[k],ind_caminho_sobe[k]);
    if (ind_caminho_sobe[k]==1)
      {qtd_atr_caminho++;};
   };
fprintf(f_imp,"\n qtd_atr_caminho_sobe=%d",qtd_atr_caminho);
return qtd_atr_caminho;
}

// ================================ escolhe_melhor_atrib_lista
int escolhe_melhor_atrib_lista(struct Noh_m *p)
{int melhor_atributo;
 float menor_entropia=9999;
 float entropia[QTDEMAXATRIB];
 soma_atributos_caminho();
 int atr;
 for (int i=0; i < p->end_ex[0]->qtd_atr_lista; i++)
   { atr=p->end_ex[0]->cod_atrib[i];
     entropia[atr]=calcular_entropia(atr,p);
     if (entropia[atr] < menor_entropia)
       {menor_entropia= entropia[atr];
        melhor_atributo=atr;
       };
   };
return melhor_atributo;
}
//=========================================== calcular
entropia
float calcular_entropia(int at, struct Noh_m *p)
{float HA=0;
float temp=0;
for (int i=0; i<lista_atrib[at].qtde_valores;i++)
   {float HAi= entropia_valor(at, i, p);
   temp= (float) (freq_at_val(at,i,p))/ freq_at(at,p);
   HA=HA + temp *HAi;
};
if (HA < 0)
  { return 0;};
return HA;
}
//=========================================== entropia
valor
float entropia_valor(int at, int valor, struct Noh_m *p)
{float PHi;
float HAi=0;
if (freq_at_val( at, valor, p) > 0)
  {for (int j=0; j< qtde_classes ; j++)
     {PHi=((float)freq_at_val_classe( at, valor , j , p))/freq_at_val(
at, valor, p);
      if (PHi >0 )
        {HAi=HAi+(-PHi * (log10 (PHi)/log10 (2)));
        };
```

```
    };
  };
return HAi;
}

//=============================================
frequencia_at
int freq_at(int at, struct Noh_m *p)
{int acum=0;
 for (int j=0; j< lista_atrib[at].qtde_valores; j++)
   {for (int k=0; k< qtde_classes; k++)
      {acum = acum + p->contador[at][j][k];
      };
   };
return acum;
}
//=========================================== frequencia_at_val
int freq_at_val(int at, int valor, struct Noh_m *p)
{int acum=0;
 for (int k=0; k< qtde_classes; k++)
   {acum = acum + p->contador[at][valor][k];
   };
return acum;
}
//================================= frequencia_at_val_classe
int freq_at_val_classe(int at, int valor, int classe, struct Noh_m *p)
{int acum;
acum=p->contador[at][valor][classe];
return acum;
}
//========================= move_ex_lista_velha_para_nova
int move_ex_lista_velha_para_nova(struct lista_ex
*ex_lista_velha,struct lista_ex *ex_lista_nova,int j)
{ex_lista_nova->num_ex      =ex_lista_velha->num_ex;
 ex_lista_nova->qtd_atr_lista=ex_lista_velha->qtd_atr_lista-1;
 ex_lista_nova->classe      =ex_lista_velha->classe;
 for (int m=0; m<ex_lista_velha->qtd_atr_lista; m++)
   {if ( m < j )
      {ex_lista_nova->cod_atrib [m]= ex_lista_velha->cod_atrib
[m];
       ex_lista_nova->valor_atrib[m]= ex_lista_velha-
>valor_atrib[m];
      };
    if ( m > j )
      {ex_lista_nova->cod_atrib [m-1]= ex_lista_velha->cod_atrib
[m];
       ex_lista_nova->valor_atrib[m-1]= ex_lista_velha-
>valor_atrib[m];
      };
   };
return 1;
}
//============================================= classifique_ex
void classifique_ex(int num_ex, struct Noh_m *p)
{ if (p != ult_noh)
   {atualizar_contadores(num_ex,p);
   };
ult_noh=p;
int melhor_atrib;
if (lote.inicio==0 && lote.fim ==0)
  {processa_primeiro_ex(num_ex,p);
  return;
  };
if (p->tipo_noh==1)
  {if (p->atributo != tabex[num_ex][0])
     {if (p->end_ex[0]->qtd_atr_lista > 0)
        {melhor_atrib=escolhe_melhor_atrib_lista(p);
         adiciona_novo_noh_atributo(melhor_atrib,p,1);
         classifique_ex(num_ex, p);
        }
      else
        {return;
        };
     };
  return;
  };
 if (p->tipo_noh==0)
   { int kkk;
     kkk=p->qtd_filhos;
     for (int iii=0; iii<kkk ; iii++)
       {if (p->valor[iii] == tabex[num_ex][p->atributo+1] )
          {kkk=0;
           ind_caminho[p->atributo]=1;
           val_caminho[p->atributo]=p->valor[iii];
           classifique_ex(num_ex, p->end_filho[iii]);
          };
       };
    if (kkk !=0)
       {ind_caminho[p->atributo]=1;
        val_caminho[p->atributo]=p->valor[iii];
        ult_noh= adiciona_novo_noh_classe(num_ex, p);
        atualizar_contadores(num_ex,ult_noh);
        return;
       };
  };
return;
}
//================================= processa_primeiro_ex
void processa_primeiro_ex(int num_ex,struct Noh_m *p)
{p->tipo_noh=1;
p->atributo=tabex[num_ex][0];
p->qtd_filhos=0;
p->qtd_exem_lista=0;
return ;
}
//================================= atualizar
contadores
int atualizar_contadores(int num_ex, struct Noh_m *p)
{for (int i=0; i<qtde_atributos; i++)
   {for (int k=0; k<lista_atrib[i].qtde_valores; k++)
      {if (lista_atrib[i].valores[k] == tabex[num_ex][i+1])
        {p->contador[i] [k] [tabex[num_ex][0]] ++;
        };
      };
  };
return 1;
}
//=========================================mostra
contadores
void mostra_contadores(struct Noh_m *p)
{ printf(" contadores");
 for (int i=0; i< qtde_atributos; i++)
    {for (int j=0; j<lista_atrib[i].qtde_valores; j++)
      {for (int k=0; k<qtde_classes; k++)
        {printf(" %d" ,p->contador[i][j][k]);
        };
       };
    printf(".");
    };
return;
}

//=================================
guardar_at_val_na_lista
struct lista_ex *guardar_at_val_na_lista(int num_ex, struct Noh_m
*p,int qtd_ex_list,int atr_lista)
{
p->end_ex[qtd_ex_list]=(struct lista_ex *)malloc(sizeof(struct
lista_ex));
if (!p->end_ex[qtd_ex_list])
  { printf ("\n FALTA MEMORIA PARA ALOCACAO EM
GUARDAR_AT_NA_LISTA");
   return NULL;
  };
p->end_ex[qtd_ex_list]->num_ex      = num_ex;
p->end_ex[qtd_ex_list]->classe      = tabex[num_ex][0];
p->end_ex[qtd_ex_list]->qtd_atr_lista= atr_lista;
int ind_atr=0;
for (int k=0; k<qtde_atributos; k++)
   {if (ind_caminho[k]==0)
     {p->end_ex[qtd_ex_list]->cod_atrib [ind_atr]=k;
      p->end_ex[qtd_ex_list]-
>valor_atrib[ind_atr]=tabex[num_ex][k+1];
      ind_atr++;
     };
   };
return p->end_ex[qtd_ex_list];
}
// ============================= adiciona_novo_noh_classe
struct Noh_m *adiciona_novo_noh_classe(int num_ex, struct
Noh_m *p)
{struct Noh_m *novo;
```

```
novo=(struct Noh_m *)malloc(sizeof(struct Noh_m));
if (!novo)
  {printf ("\n FALTA MEMORIA PARA ALOCACAO EM
ADICIONA_NOVO_NOH_CLASSE");
   return NULL;
  };
inicializa_endereco_valor(novo);
inicializa_contadores(novo);
 novo->atributo   = tabex[num_ex][0];
 novo->tipo_noh   = 1;
 novo->qtd_filhos = 0;
 novo->end_pai    = p;
 p->valor   [p->qtd_filhos]= tabex[num_ex][p->atributo+1];
 p->end_filho[p->qtd_filhos]=novo;
 p->qtd_filhos++;
 novo->qtd_exem_lista=0;
return  novo;
}

//=================================inicializa endereco valor
void inicializa_endereco_valor(struct  Noh_m  *p)
{ for (int i=1; i<QTDEMAXATRIB; i++)
    { p->end_filho[i] = NULL;
     p->valor    [i] = 0;
    };
return;
}
//=================================inicializa contadores
void inicializa_contadores(struct  Noh_m  *p)
{int i,j,k=0;
 for (i=0; i<QTDEMAXATRIB; i++)
   { for (j=0; j<QTDEMAXVALOR; j++)
       { for (k=0; k<QTDEMAXCLASS; k++)
           {p->contador[i][j][k]=0;
           };
       };
   };
return;
}
//=============================================carreg
a
int carrega(char * nome_tex)
{ FILE *fp;
            char nome_arq_temp[MAX_STRING_NOME];
            nome_tabex = nome_tex;
            (void) sprintf(nome_arq_temp,"%sC",nome_tex);

            fprintf(f_imp,"NOME DO ARQUIVO
%s\n",nome_arq_temp);
            fp=fopen(nome_arq_temp, "r");
            if (fp == NULL)        {
                        return 1;
            }

// le header de controle com quantidades
            fscanf(fp,"%4d%2d%2d%4d%4d%4d%1d",&qtde_exe
mplos,&qtde_atributos,

            &qtde_classes,&qtde_valor_tot,&qtde_atrib_val,&qtde_
exemplos_real,&contra_exemplo);

// le tabela de atributos
            for (int i=0;i<qtde_atributos;i++) {

            fscanf(fp,"%2d%1d%4d%1d%2d",&lista_atrib[i].cod,

&lista_atrib[i].tipo,&lista_atrib[i].custo,&lista_atrib[i].existe_burac
o,&lista_atrib[i].qtde_valores);
                        for (int
k=0;k<lista_atrib[i].qtde_valores;k++)

            fscanf(fp,"%3d",&lista_atrib[i].valores[k]);
                        fscanf(fp,"%30s%12s",lista_atrib[i].nome,
lista_atrib[i].nome_abrev);
            }

// le tabela de classes
            for (i=0;i<qtde_classes;i++)

            fscanf(fp,"%2d%04d%30s",&lista_classes[i].cod,&lista
_classes[i].freq_cl,lista_classes[i].nome);

// le tabela de valores
            for (i=0;i<qtde_valor_tot;i++)

            fscanf(fp,"%3d%30s",&lista_valor[i].cod,lista_valor[i].
nome);

// le estrutura de controle
            for (i=1;i<qtde_atrib_val+1;i++) {
                        for (int k=1;k<qtde_atrib_val+1;k++)

            fscanf(fp,"%04d",&matriz_par[i][k]);
            }
// le pares atributo valor
            for (i=1;i<qtde_atrib_val+1;i++) {

            fscanf(fp,"%2d%3d",&lista_par[i].atrib,&lista_par[i].va
lor);
                        for (int j=0;j<qtde_classes;j++)

            fscanf(fp,"%04d",&matriz_classe_par[i][j]);
            }
            fclose(fp);
// le tabela de exemplos
            fp=fopen(nome_tex,"r");

            if (fp == NULL)        {
                        return 1;
      }
            for (i=0;i<qtde_exemplos;i++) {
                        fscanf(fp,"%02d",&tabex[i][0]);
                        for (int k=1;k<qtde_atributos+1;k++) {
                                    fscanf(fp,"%03d",&tabex[i][k]);
                                    }

            fscanf(fp,"%04d%01d",&freq_exemplo[i],&flag_exemp
lo[i]);
            }
            fclose(fp);
            return 0;

}

//=================================== mostra noh do disco
int mostra_noh_disco(struct Noh_d n, long d)
{ fprintf(f_imp,"\n NOH=%ld TIPO=%d", d,n.tipo_noh);
 fprintf(f_imp," atr=%d "          ,n.atributo);
 fprintf(f_imp,"              PAI=%ld ", n.end_pai);
 fprintf(f_imp,"\n    contadores");
 for (int i=0; i<ex->qtde_atributos; i++)
    {for (int j=0; j<QTDEMAXVALOR; j++)
       {for (int k=0; k<ex->qtde_classes; k++)
          {fprintf(f_imp," %d"  ,n.contador[i][j][k]);
          };
       };
    };
 fprintf(f_imp,"\n QTD_FILHOS= %d",n.qtd_filhos);
 for (int k=0; k<n.qtd_filhos; k++)
    {fprintf(f_imp,"val[%d]=%d end[%d]%ld",
k,n.valor[k],k,n.end_filho[k]);
    };
fprintf(f_imp,"\n qtd_ex_lista=%d",n.qtd_exem_lista);  // valor
?????????
 for (int m=0; m<n.qtd_exem_lista; m++)
    {fprintf(f_imp,"\nnum_ex=%d",
n.lista_exemplo[m].num_ex);

fprintf(f_imp,"qtd_atr_lista=%d",n.lista_exemplo[m].qtd_atr_lista);
     fprintf(f_imp,"classe=%d",      n.lista_exemplo[m].classe);
     for (int kk=0; kk<n.lista_exemplo[m].qtd_atr_lista;kk++)
        {fprintf(f_imp,"atr[%d]=%d ",kk,
n.lista_exemplo[m].cod_atrib[kk]);
         fprintf(f_imp,"valor[%d]=%d",kk,
n.lista_exemplo[m].valor_atrib[kk]);
        };
    };
return 1;
}

//=================================== gravar arvore
int gravar_arvore()
{ long s;
```

```
 abre_arquivo_arv_gravacao();
 grava_de_lote();
 mostra_controle();
 lote.inicio=0;
 lote.fim  =0;
 p_filho=p_raiz;
 formata_d_filho(999);
 s=ftell(f_arv);
 qtd_noh++;
 mostra_noh_disco(noh_d,s);
 grava_de_noh_d();
 gravar_filhos_na_arvore(p_raiz,s);
 fecha_arquivo_arv();
 return 1;
}
//================================= gravar filhos na
arvore
int gravar_filhos_na_arvore(struct Noh_m *p_pai, long
end_disco_pai)
{long d;
 if (p_pai->tipo_noh == 0)
   {for (int k=0; k<p_pai->qtd_filhos; k++)
      {p_filho=p_pai->end_filho[k];
       formata_d_filho(end_disco_pai);
       d=ftell(f_arv);
       qtd_noh++;
       mostra_noh_disco(noh_d,d);
       grava_de_noh_d();
       gravar_filhos_na_arvore(p_pai->end_filho[k],d);
      };
   }
 else
   {return 1;};
return 1;
}
//=================================== formata filho
disco
int formata_d_filho(long end_disco_pai)
{       noh_d.tipo_noh=p_filho->tipo_noh;
        noh_d.atributo=p_filho->atributo;
        noh_d.qtd_filhos=p_filho->qtd_filhos;
        for (int j=0; j<p_filho->qtd_filhos; j++)
           {noh_d.valor[j]=p_filho->valor[j];
           };

        for (int m=0;m<QTDEMAXATRIB; m++)
          {for (int n=0;n<QTDEMAXVALOR; n++)
            {for (int s=0;s<QTDEMAXCLASS; s++)
              {noh_d.contador[m][n][s]=p_filho->contador[m][n][s];
              };
            };
          };
        for (int k=0; k<QTDEMAXVALOR; k++)
          {noh_d.end_filho [k]=0;
          };
noh_d.qtd_exem_lista=p_filho->qtd_exem_lista;
if (noh_d.qtd_exem_lista>0 )
  {for (int r=0;r<noh_d.qtd_exem_lista;r++)
    {noh_d.lista_exemplo[r].num_ex      =p_filho->end_ex[r]-
>num_ex;
     noh_d.lista_exemplo[r].qtd_atr_lista=p_filho->end_ex[r]-
>qtd_atr_lista;
     noh_d.lista_exemplo[r].classe      =p_filho->end_ex[r]->classe;
     noh_d.lista_exemplo[r].classe      =p_filho->end_ex[r]->classe;
     for (int q=0;q<p_filho->end_ex[r]->qtd_atr_lista;q++)
      {noh_d.lista_exemplo[r].cod_atrib[q]=p_filho->end_ex[r]-
>cod_atrib[q];
       noh_d.lista_exemplo[r].valor_atrib[q]=p_filho->end_ex[r]-
>valor_atrib[q];
      };
    };
  };
noh_d.end_pai=end_disco_pai;
 return 1;
}
//============================ ler arq arvore alocando memoria
int ler_arq_arvore()
{ abre_arquivo_arv_leitura();
 ler_para_lote();
 ler_para_d_filho();
 p_raiz=construir_noh_memoria(0);
 ler_filhos_arq_arvore(p_raiz);
 fecha_arquivo_arv();
 return 1;
}
//=====================================ler filhos do arq arvore
int ler_filhos_arq_arvore(struct Noh_m *p_pai)
{if (p_pai->tipo_noh == 0)
   {for (int k=0; k<p_pai->qtd_filhos; k++)
      {ler_para_d_filho();
       p_filho=construir_noh_memoria(p_pai);
       p_pai->end_filho[k]=p_filho;
       ler_filhos_arq_arvore(p_filho);
      };
   }
 else
   {return 1;
   };
return 1;
}

//=================================== constroi noh memoria
struct Noh_m *construir_noh_memoria(struct Noh_m *p_pai)
{p_filho=(struct Noh_m *)malloc(sizeof(struct Noh_m));
 if (!p_filho)
   {printf ("\n FALTA MEMORIA PARA ALOCACAO EM
CONSTRUIR NOH MEMORIA");
    return NULL;
   };

 int i,j,k=0;
 for (i=1; i<QTDEMAXATRIB; i++)
    { p_filho->end_filho[i] = NULL;
      p_filho->valor    [i] = 0;
    };
 for (i=0; i<QTDEMAXATRIB; i++)
    { for (j=0; j<QTDEMAXVALOR; j++)
        { for (k=0; k<QTDEMAXCLASS; k++)
            {p_filho->contador[i][j][k]=d_filho.contador[i][j][k];
            };
        };
    };
 p_filho->atributo      = d_filho.atributo;
 p_filho->tipo_noh      = d_filho.tipo_noh;
 p_filho->qtd_filhos    = d_filho.qtd_filhos;
 for (int m=0;m<d_filho.qtd_filhos;m++)
    { p_filho->valor[m]=d_filho.valor[m];
    };
p_filho->qtd_exem_lista = d_filho.qtd_exem_lista;
if (d_filho.qtd_exem_lista>0 )
  {for (int r=0;r<d_filho.qtd_exem_lista;r++)
    {p_filho->end_ex[r]=(struct lista_ex *)malloc(sizeof(struct
lista_ex));
     if (!p_filho->end_ex[r])
      {printf ("\n FALTA MEMORIA ALOCACAO EX %p
lista",p_filho->end_ex[r]);
       return NULL;
       };

     p_filho->end_ex[r]->num_ex
=d_filho.lista_exemplo[r].num_ex;
     p_filho->end_ex[r]-
>qtd_atr_lista=d_filho.lista_exemplo[r].qtd_atr_lista;
     p_filho->end_ex[r]->classe
=d_filho.lista_exemplo[r].classe;
     p_filho->end_ex[r]->classe
=d_filho.lista_exemplo[r].classe;
     for (int q=0;q<d_filho.lista_exemplo[r].qtd_atr_lista;q++)
      {p_filho->end_ex[r]-
>cod_atrib[q]=d_filho.lista_exemplo[r].cod_atrib[q];
     p_filho->end_ex[r]-
>valor_atrib[q]=d_filho.lista_exemplo[r].valor_atrib[q];
      };
    };
  };
 p_filho->end_pai   = p_pai;
return  p_filho;
}
//=================================== mostra
controle
int mostra_controle()
{ fprintf(f_imp,"\n numero primeiro exemplo=%f ", lote.inicio);
```

```
 fprintf(f_imp,"\n numero ultimo   exemplo=%f ", lote.fim);
return 1;
}
//=================================== mostra arvore
memoria
int mostra_arvore_memoria(struct Noh_m *p)
{ mostra_noh_memoria(p);
 mostra_filhos_memoria(p);
 return 1;
}
//=================================== mostra filhos
memoria
int mostra_filhos_memoria(struct Noh_m *p_pai)
{if (p_pai->tipo_noh == 0)
   {for (int k=0; k<p_pai->qtd_filhos; k++)
      {mostra_noh_memoria(p_pai->end_filho[k]);
       mostra_filhos_memoria(p_pai->end_filho[k]);
      };
   }
 else
   {return 1;};
return 1;
}
//=================================== mostra noh da
memoria
int mostra_noh_memoria(struct Noh_m *p)
{printf("\n%p ",p);
 if (p->tipo_noh==0)
   {printf(" atr=%d "        ,p->atributo);
   };
 if (p->tipo_noh==1)
   {printf(" cla=%d "        ,p->atributo);
   };
 printf("   PAI=%p ", p->end_pai);
 mostra_contadores(p);

if (p->qtd_filhos >0)
   {printf("\n    %d FILHOS val_end==>",p->qtd_filhos);
    for (int m=0; m<p->qtd_filhos; m++)
      {printf(" %d--%p", p->valor[m],p->end_filho[m]);
      };
   };

if (p->qtd_exem_lista>0)
   { mostra_lista(p);
   };

return 1;
}

//=================================== mostra_lista
void mostra_lista(struct Noh_m *p)
{printf(" \n     %d EXEMPLOS",p->qtd_exem_lista);
for (int n=0; n<p->qtd_exem_lista; n++)
  {printf("\n     EX %d",         p->end_ex[n]->num_ex);
   printf(" CL=%d",          p->end_ex[n]->classe);
   printf(" qtd_atr_lista=%d atr_val=>",p->end_ex[n]-
>qtd_atr_lista);
   for (int kk=0; kk<p->end_ex[n]->qtd_atr_lista;kk++)
     {printf(" %d-%d", p->end_ex[n]->cod_atrib[kk],p->end_ex[n]-
>valor_atrib[kk]);
     };
 };
return;
}
//=================================== abre arquivo arv gravacao
int abre_arquivo_arv_gravacao()
{fprintf(f_imp,"\n>>>>>>>>>>>>>>>>>NOME FORMATADO DO
ARQUIVO = %s ",nome_arq_arv);
if (( f_arv=fopen(nome_arq_arv, "w"))==NULL)
    {fprintf(f_imp,"arq saida arv %s nao pode ser
aberto",nome_arq_arv);
     return 0;
    };
 fprintf(f_imp,"\n===abriu arq %s para gravacao",nome_arq_arv);
return 1;
}
//=================================== formata nome arq
arv
int formata_nome_arq_arv()
{ char nome_temp[20];
  strcpy(nome_temp,"QQ");
  strcat(nome_temp,".");
  sprintf(nome_temp,"%s%s",nome_temp,nome_alg);
  strcpy(nome_arq_arv,nome_temp);
return 1;
}

//=================================== abre arquivo arv leitura
int abre_arquivo_arv_leitura()
{if((f_arv=fopen(nome_arq_arv, "r"))==NULL)
        {printf("arq arv entrada %s nao pode ser
aberto",nome_arq_arv);
         return 0;
        };
return 1;
}
//=================================== abre arquivo
impressao
int abre_arquivo_imp()
{ if((f_imp=fopen("ALG19.sai", "w"))==NULL)
   {printf("arquivo impressao nao pode ser aberto");
    return(0);};
return 1;
}
//=================================== fecha arquivo arv
int fecha_arquivo_arv()
{fclose(f_arv);
 return 1;
}
//=================================== fecha arquivo
imp
int fecha_arquivo_imp()
{fclose(f_imp);
return 1;
}

//=================================== ler para lote
int ler_para_lote()
{fscanf(f_arv,"%4f%4f",&lote.inicio,&lote.fim);
return 1;
}
//=================================== ler para
d_filho
int ler_para_d_filho()
{fscanf(f_arv,"%1d%2d%2d",&d_filho.tipo_noh,&d_filho.atributo,
&d_filho.qtd_filhos);
 for (int i=0; i<qtde_atributos; i++)
    {for (int j=0; j<lista_atrib[i].qtde_valores; j++)
       {for (int k=0; k<qtde_classes; k++)
          {fscanf(f_arv,"%2d",&d_filho.contador[i][j][k]);
          };
       };
    };
 for (int m=0; m<d_filho.qtd_filhos; m++)

{fscanf(f_arv,"%3d%2d",&d_filho.valor[m],&d_filho.end_filho[m]
);
   };
fscanf(f_arv,"%2d",&d_filho.qtd_exem_lista);
for (int q=0; q<d_filho.qtd_exem_lista; q++)
  {fscanf(f_arv,"%3d",&d_filho.lista_exemplo[q].num_ex);
   fscanf(f_arv,"%2d",&d_filho.lista_exemplo[q].qtd_atr_lista);
   fscanf(f_arv,"%2d",&d_filho.lista_exemplo[q].classe);
   for (int r=0; r<d_filho.lista_exemplo[q].qtd_atr_lista; r++)
     {fscanf(f_arv,"%2d",&d_filho.lista_exemplo[q].cod_atrib [r]);
      fscanf(f_arv,"%3d",&d_filho.lista_exemplo[q].valor_atrib[r]);
     };
  };
return 1;
}
//=================================== grava de
lote
int grava_de_lote()
{fprintf(f_arv,"%04.0f%04.0f\n",lote.inicio,lote.fim);
return 1;
}
//=================================== grava de
noh_d
int grava_de_noh_d()
{fprintf(f_arv,"%01d%02d%02d",noh_d.tipo_noh,noh_d.atributo,n
oh_d.qtd_filhos);
```

```
for (int i=0; i<qtde_atributos; i++)
    {for (int j=0; j<lista_atrib[i].qtde_valores; j++)
       {for (int k=0; k<qtde_classes; k++)
          {fprintf(f_arv,"%02d",noh_d.contador[i][j][k]);
          };
       };
    };
for (int m=0; m<noh_d.qtd_filhos; m++)

{fprintf(f_arv,"%03d%02d",noh_d.valor[m],noh_d.end_filho[m]);
   };
fprintf(f_arv,"\n");
fprintf(f_arv,"%02d",noh_d.qtd_exem_lista);
for (int q=0; q<noh_d.qtd_exem_lista; q++)
 {fprintf(f_arv,"%03d",noh_d.lista_exemplo[q].num_ex);
  fprintf(f_arv,"%02d",noh_d.lista_exemplo[q].qtd_atr_lista);
  fprintf(f_arv,"%02d",noh_d.lista_exemplo[q].classe);
  for (int r=0; r<noh_d.lista_exemplo[q].qtd_atr_lista; r++)
    {fprintf(f_arv,"%02d",noh_d.lista_exemplo[q].cod_atrib [r]);
     fprintf(f_arv,"%03d",noh_d.lista_exemplo[q].valor_atrib[r]);
    };
 };
 fprintf(f_arv,"\n");
 return 1;
}
```

ABSTRACT

The critical phase of the construction process of expert systems is the one responsible for the generation of the knowledge base because it is expensive, difficult and great influence upon the final quality of the referred systems.

The induction originated from examples generates knowledge base as decision trees or rules through algorithms called inductive algorithms.

Inductive algorithms can be of non-incremental or incremental types.

Non-incremental algorithms receive, as input, a set de examples and generate, as output, a knowledge base.

Incremental algorithms receive, as input, a knowledge base and a set of new examples and generate, as output, an up to date knowledge base.

We present in this paper a comparative analysis of costs between these two families of algorithms generators of decision tree.

We show, through experimental and theoretical analysis that, against the conclusion of previous authors about incremental algorithms, the referred algorithms are more efficient only in rare situations of domains and quantity of examples, not reaching the ends they aimed at.

Referências Bibliográficas

[Alexandre 94] Alexandre, C.R.

Aquisição indutiva de conhecimento contemplando os aspectos sintáticos, semânticos, de generalização e de custo. Dissertação de Mestrado, em Ciências da Computação. Centro de Ciências e Tecnologia, Campina Grande, Universidade Federal da Paraíba, 120p, 1994.

[Bezerra 94a] Bezerra, H.C.F., Mongiovi, G., Alexandre, C.R.

Uma análise comparativa entre algoritmos incrementais e não-incrementais utilizados para geração de árvores de decisão no processo de aquisição de conhecimento. Anais do II Congreso e Exposición International de Informática - Mendonza/Argentina - junho/1994.

[Bezerra 94b] Bezerra, H.C.F., Mongiovi, G., Silva, H.M.

Incremental Inductive Algorithms: When Should they be Used?. Procedings of the XI Brazilian Symposium on Artificial Intelligence - Fortaleza - outubro /1994.

[Bock 85] Bock, P.

The emergence of Artificial Intelligence: Learning to Learn. The AI Magazine, Fall, 1985.

[Boden 77] Boden, M.A.

Artificial Intelligence and Natural Man. Basic Books, Inc New York, NY, 1977.

[Boose 84] Boose, J.H.

Personal Construct Theory and the Transfer of Human Expertise. Anais do 3o AAAI, 1984.

[Boose 87] Boose, J.H.
Expertise Transfer and Complex Problems: Usign AQUINAS as a Knowledge. Aquisition Workbench for Knowledge-Based Systems, II Man-Machine Studies, Academic Press, 1987.

[Boose 90] Boose, J.H.
Knowledge Acquisition for Knowledge-Based Sistems. IOS Press, Tokio, 1990.

[Boy 87] Boy, G; Faller, B. e Sallantin, J.
Acquisition et Ratification de Connaissances. Relatório Técnico. CRIM, Montpellier, França, 1987.

[Breiman 84] Breiman, L., Friedman, J.H., Oslen, R.A. e Stone, C.J.
Classification and Regression Trees. Wadsworth International Group, 1984.

[Carbonell 89] Carbonell, J. G.
Introduction: Paradigms for Machine Learning. Artificial Intelligence, vol.40, 1989.

[Cirne 91] Cirne Filho, W.C.
Aquisição de conhecimento através de Aprendizado Automático. Anais do 2o Simpósio de Inteligencia Artificial y Robótica.Luján, Argentina, 1991.

[Cirne 92] Cirne Filho, W.C.
Uso de semântica na melhoria dos métodos indutivos de aquisição de conhecimento. Dissertação de mestrado. Curso de pós-graduação em Informática, Centro de Ciências e Tecnologia UFPB, Campina Grande-PB, 1992.

[Charniak 85] Charniak,E. e McDermott,D.
Introduction to Artificial Intelligence. Addison-Wesley Publishing Company, Reading, MA, 1985.

[Diederich 88] Diederich, J., Ruhmann, I. e May, M.
KRITON: A Knowledge Aquisition Tool for Expert Systems. Academic Press, vol.2 - 1988.

[Donato 94] Donato Junior, E.T.

Uso de conhecimeto preliminar na melhoria do aprendizado em um Modelo Simbólico-Conexionista. Campina Grande, UFPB. Dissertação (Mestrado em Ciências da Computação) Centro de Ciências e Tecnologia-Universidade Federal da Paraíba, 136p, 1993.

[Eshelman 88] Eshelman, L.

MOLE: A Knowledge Acquisition Tools that Burries Certainty Factors. Knowledge-Based Systems, vol. 4, Academic Press, 1988.

[Feigenbaum 81] Feigeibaum, E.A.

Expert Systems in 1980's. In: A. Bond (Ed.). The state of the art report on machine intelligence. Oxford: Pergamon-Infotech, 1981.

[Firebaugh 89] Firebaugh, M.W.

Artificial Intelligence - A knowledge-based approach. Boston: PWS-Kent Publishing Company, 1989. 740p.

[Goldberg 89] Goldberg, D. E.

Genetic Algorithms in Search, Optimization and Machine Learning - Reading Masachussets: Addison Wesley Publishing, 1989.

[Jackson 86] Jackson, P.

Introduction to expert systems. Addison-Weslwy Publishing, 1986.

[Klir 88] Klir, G.J., e Folger T.A.

Fuzzy Sets , Uncertainty and Information. Prentice-Hall, 1988.

[MacGraw 89] McGraw, K.L., e Harbison-Briggs, K.

Knowledge Acquisition: principles and guidelines. Prentice-Hall International Editions, 1989.

[Marcus 87] Marcus, S.

Taking Backtracking with a Grain of SALT. Knowledge-Based Systems, vol. 2, Academic Presss, 1987.

[Michaelsen 85] Michaelsen, R.H., Michie, D. e Boulanger, A.

The tecnology of expert systems. Byte, v.10, n.4, p.310, 1985.

[Michalski 69] Michalski, R.S.

On the Quasi-Minimal Solution of the General Covering Problem. *Procedings of the V International Symposium on Information Processing (FCIP 69),* Vol. A3 (Switching Circuits), (p.125-128), Bled, Yuguslavia, October 8-11, 1969.

[Michalski 75] Michalski, R.S. e Larson, J.

AQVAL/1 (AQ 7) User's Guide and Program Description. *Report No. 731*, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1975.

[Michalski 78] Michalski, R.S. e Larson, J.

Selection of the Most Representative Training Examples and Incremental Generation of VL_1 Hipoteses: the underlying methodology and the descrition of programs ESEL and AQ11. *Report No. 867*, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1978.

[Michalski 83] Michalski, R.S. e Larson, J.

Incremental Generation of VL_1 Hypóteses: the underlying methodology and description of program AQ11. ISG 83-5, UIUCDCS-F-83-905. Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1983.

[Michalski 85] Michalski, R.S.

Knowledge Repair Mechanisms: Evolution Versus Revolution. *Proceedings of the Third International Machine Learning Workshop* (p.14-16). Rutgers University.

[Michalski 86 a] Michalski, R.S.

Understanding the nature of learning: Issues and research directions. In: Jaime G. Carbonell and Tom M. Mitchell (Ed.), *Machine Learning - An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann Publisher Inc., 1986. v.2, p.3-25.

[Michalski 86 b] Michalski, R.S., Hong, J. e Mozetic, I.

AQ15: Incremental Learning of Attribute-Based Descritions from Examples, the Method and User's Guide. Report ISG 86-5, UIUCDCS-F-86-949, Dept. of Computer Science, University of Illinois at Urbana-Champain, 1986.

[Mongiovi 95] Mongiovi, G.

Uso de relevância semântica na melhoria da qualidade dos resultados gerados pelos métodos indutivos de aquisição de conhecimento a partir de exemplos. Campina Grande, UFPB, 1995, 232p. Dissertação (Doutorado em Engenharia Elétrica) Centro de Ciências e Tecnologia-Universidade Federal da Paraíba, 1995.

[Mongiovi 93] Mongiovi, G.

Aquisição automática de conhecimento a partir e exemplos: uma abordagem pragmática. Campina Grande, UFPB, 1993, 108p. **Tese** (Concurso Publico para prof. Titular) Departamento de Sistemas e computação, Universidade Federal da Paraíba, 1993.

[Nguifo 93] Nguifo, E. M.

Consevoir une Abstraction a Partir de Ressemblances - Tese de doutorado, Université Montpellier II, Montpellier, França. 1993

[Quinlan 83] Quinlan, J.R.

Learning efficient classification procedures and their application to chess end-games. In: Jaime G. Carbonell, Tom M. Mitchell (Ed). *Machine Learning - An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann Publisher Inc., 1983. P.468-482.

[Quinlan 86] Quinlan, J.R.

Induction of decision Tree. Machine Learning Journal I.Kluver Academic Publisher,1986.

[Quinlan 87] Quinlan, J.R.

Generating Production Rules from Decision Trees. Anais do IJCAI. Milão, Itália, 1987.

[Schlimmer 86] Schlimmer, J.C. e Fisher, D.

A case study of incremental concept induction. In: *Fifth National Conference on Artificial Intelligence*, 1986. Proceedings. San Mateo CA: Morgan Kaufmann, 1986. p.496-501.

[Sousek 89] Soucek, B. E Sousek, M.

Neural and Massive Parallel Computers, the Sixth Generation - John Willey and Sons,1989.

[Shaw 88] Shaw, L.G. e Gaines, B.R.
KITTEN: Knowledge Initiation and Transfer Tools for Experts and Novices. Knowledge-Based Systems, vol. 2, Academic Press, 1988.

[Shapiro 87] Shapiro, A.D.
Strutured Induction in Expert Systems. Adison Wesley, 1987.

[Utgoff 88] Utgoff, P.E.
ID5: An incremental ID3. In: J. Laird (Ed.) *Fifth International Conference on Machine Learning*, 1988. Proceedings. San Mateo CA: Morgan Kaufmann, 1988. p.107-120.

[Utgoff 89] Utgoff, P.E.
Incremental induction of decision trees. *Machine Learning*, Boston, v.4, p.161-186, 1989.

[Van de Velde 89] Van de Velde, W.
IDL or taming the multiplexer. In: Katharina Morik (Ed.) *Fourth European Working Session on Learning (EWSL)*, 1989, Montpellier. Proceedings. London: Pitman Publishing, 1990. p.211-225.

[Vasco 93] Vasco, J.J.F.
Um ambiente de apoio à aquisição automática de conhecimento - Dissertação de mestrado, Curso de Pós-Graduação em Informática da UFPB, Campina Grande PB, Junho /1994.

[Winston 84] Winston, P.H.
Artificial Intelligence. Segunda edição - Addison-Wesley Publishing Company, Reading, MA, 1984.