Juliano de Almeida Monte-Mor

Paralelização de um Método de Aprendizado Indutivo de Máquina Baseado na Teoria de Conjuntos Aproximados

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Jones Oliveira de Albuquerque

Co-Orientador
Prof. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais - Brasil
2002

Juliano de Almeida Monte-Mor

Paralelização de um Método de Aprendizado Indutivo de Máquina Baseado na Teoria de Conjuntos Aproximados

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *26 de março de 2002*

Profa. Olinda Nogueira Paes Cardoso

Prof. Jones Oliveira de Albuquerque
(Orientador)

Prof. Joaquim Quinteiro Uchôa
(Co-Orientador)

Lavras
Minas Gerais - Brasil

A Deus de onde provém
toda a sabedoria, e com
muito carinho aos meus
pais Divino e Maria José,
que me deram condições
de chegar até aqui.

Agradecimentos

A minha amada Gláucia pela paciência e apoio.
Aos Profs. Jones e Joaquim, pelo conhecimento repassado.
A Profa. Olinda e ao colega Marcos Aurélio, pelo auxílio no trabalho.
Aos meus familiares e amigos, pela força e incentivo
nos momentos difícies.

Resumo

Este trabalho apresenta e avalia a paralelização de um método de Aprendizado de Máquina baseado na Teoria de Conjuntos Aproximados, chamado de RS1+. Esta teoria é um importante formalismo para a representação de conhecimento e tratamento de incerteza em Sistemas Baseados em Conhecimento ou Sistemas Especialistas. É apresentado o algoritmo paralelo que implementa os conceitos da Teoria de Conjuntos Aproximados na geração de regras a partir de uma tabela de exemplos, chamado de PRS1+.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Uma das mais importantes áreas de atuação da Inteligência Artificial é o desenvolvimento de Sistemas Baseados em Conhecimento (SBCs). Em linhas gerais, SBCs são definidos como programas de computador que resolvem problemas utilizando conhecimento representado explicitamente e que, não fosse essa representação, exigiriam um especialista humano no domínio do problema para a sua solução [34].

Estes sistemas devem ser capazes de representar, manipular e comunicar informações, que muitas vezes podem ser consideradas imperfeitas, isto é, informações imprecisas, inconsistentes, parcialmente ignoradas e mesmo incompletas.

Problemas relacionados com incerteza podem acontecer em todo SBC, entretanto, muitas vezes o tratamento de incerteza é realizado sem nenhuma fundamentação teórica, apesar de existirem vários formalismos disponíveis, permitindo a obtenção de resultados melhores e mais seguros. Dentre estes formalismos podemos citar:

- Regra de Bayes [11, 25];
- Fatores de Certeza [32];
- Teoria de Dempster-Shafer [31];
- Teoria de Conjuntos *Fuzzy* [39];
- Raciocínio *Default* [29];
- Teoria de *Endorsements* [9];

- Teoria de Conjuntos Aproximados [23].

A Teoria de Conjuntos Aproximados (TCA) é um modelo matemático que se destaca em relação aos outros formalismos por não necessitar, a priori, de qualquer informação adicional a respeito dos dados, como grau de pertinência, distribuição de probabilidade ou atribuição de crenças.

Outro importante ponto a ser observado no desenvolvimento de SBCs é o processo de construção da base de conhecimento do SBC, pois nem sempre é possível extrair dados de uma forma clara e completa, o que pode introduzir mais incerteza ao SBC.

Geralmente essas bases são montadas a partir de entrevistas com especialistas, sendo este um processo com custo elevado. Por isso, foram desenvolvidas várias técnicas de Aprendizado de Máquina (AM), que além do fato de automatizarem o processo da aquisição do conhecimento, essas técnicas podem permitir que o SBC “aprenda” através de seus erros, melhorando o seu desenvolvimento diante de problemas não previstos inicialmente, como apontado em [34].

Em [34], pôde ser verificado que a TCA pode ser utilizada com sucesso para a implementação de métodos de representação de conhecimento incerto, bem como um formalismo subsidiando AM.

Um algoritmo básico de aprendizado, subsidiado pela TCA, que pode ser utilizado na construção de SBCs é o algoritmo de aprendizado indutivo de máquina baseado em índice discriminante de atributos. Este algoritmo, conhecido como RS1, é descrito em [37]. Em [34], Uchôa descreve e apresenta o pseudocódigo de uma versão "melhorada"desse algoritmo, denominada RS1+.

Para que um SBC possa refletir melhor a atuação de um especialista humano na área, em certas situações, faz-se necessário a utilização de uma grande base de dados, o que acarreta um alto custo de processamento. Dessa forma, para melhorar o processamento de SBCs pode-se dividir alguns problemas, como o método de aprendizado do SBC, em diversas tarefas que podem ser realizadas em paralelo.

A computação paralela utiliza simultaneamente um conjunto de processadores interligados na resolução de um problema a fim de reduzir consideravelmente o tempo de execução [2]. Sendo que esta é utilizada na resolução de problemas onde o volume de dados e cálculos é grande e há a necessidade de um custo menor de processamento na obtenção da resposta.

Dessa forma, se a TCA for aplicada em conjunto com a programação paralela no tratamento de incerteza e representação de conhecimento em SBCs que possuem grandes bases de dados, pode ser obtido um alto desempenho nesses sistemas.

Logo o método de aprendizado baseado na TCA, que envolve processos considerados problemas *NP-Hard* [36], terá grandes ganhos com sua paralelização quando esta for utilizada na construção de SBCs com grandes bases de dados. Sendo, então, o objetivo desse trabalho, a análise e paralelização do método de AM baseado na TCA, conhecido como RS1.

Este trabalho foi organizado da forma como se segue. O Capítulo 2 apresenta os principais conceitos de sistemas paralelos, necessários para se entender o funcionamento de um algoritmo paralelo. O Capítulo 3 expõe os conceitos básicos da TCA. No Capítulo 4 são descritas as principais características dos SBCs, destacando-se a área de AM. No Capítulo 5 é realizada uma descrição do funcionamento do algoritmo seqüencial. O Capítulo 6 apresenta o algoritmo paralelo e analisa e discute os resultados encontrados nos testes práticos e que levaram às conclusões apontadas no Capítulo 7.

Capítulo 2

Computação Paralela

2.1 Introdução

Existem várias razões que justificam a necessidade do uso de processamento paralelo. A mais importante delas é o aumento de desempenho, pois este tipo de processamento pode proporcionar uma redução considerável no tempo de resolução de um problema [2].

Outra razão que justifica a implementação paralela é que alguns problemas são por natureza paralelos e sua implementação seqüencial não produz resultados satisfatórios.

Além disso, com o avanço da computação, cada vez mais as aplicações estão necessitando de maior poder computacional por causa do aumento do conjunto de dados a ser processado. E o uso de computadores paralelos se torna uma solução mais viável que o uso de computadores seqüenciais (baseados no modelo de von Neumann, com somente um processador) com mesma capacidade de processamento.

2.2 Principais conceitos

Programas e processos são conceitos distintos que ocasionalmente são confundidos. Os programas referem-se ao código fonte e possuem caráter estático. Já os processos são os programas em execução, com caráter dinâmico, possuindo um fluxo de controle e de dados.

Um programa pode disparar vários processos durante a sua execução e quando o mesmo programa é executado por dois usuários distintos são gerados dois

processos independentes.

Processamento paralelo é o processamento da informação que enfatiza a manipulação concorrente de dados pertencentes a um ou mais processos de um mesmo problema [27].

Segundo Ducan, em [10], uma arquitetura paralela fornece uma estrutura explícita e de alto nível para o desenvolvimento de soluções utilizando o processamento paralelo, através da existência de múltiplos processadores, estes simples ou complexos, que cooperam para resolver problemas através de execução concorrente.

Em um computador seqüencial há a concorrência lógica entre processos, isto é, o sistema operacional compartilha o tempo de processamento entre os processos. Somente um processo é executado por vez pelo processador, como ilustrado na Figura 2.1. Isso fornece ao usuário uma falsa impressão de que os processos estão sendo executados em paralelo, mas na verdade o sistema operacional é que controla qual processo está ativo no momento, sendo isso transparente ao usuário.

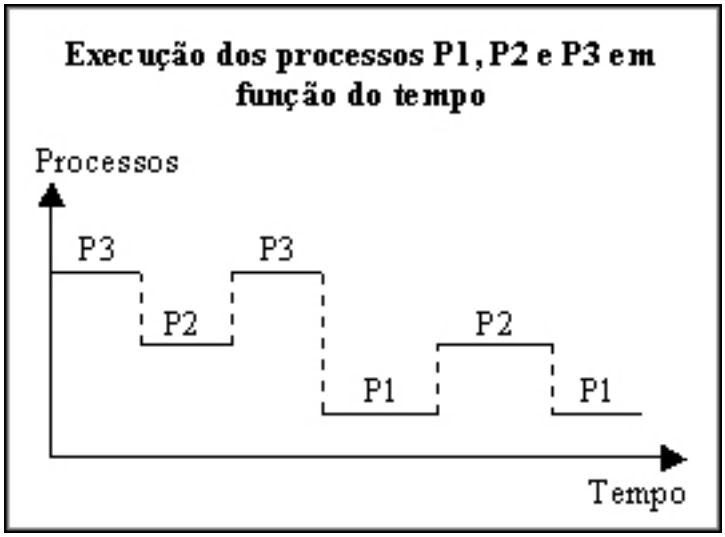

Figura 2.1: Paralelismo lógico (pseudoparalelismo)

Por outro lado, em um computador paralelo existe a concorrência real entre os processos, visto que simultaneamente cada processador está executando um processo diferente, como pode ser visto na Figura 2.2.

Dessa forma as tarefas de um problema podem ser distribuídas entre os pro-

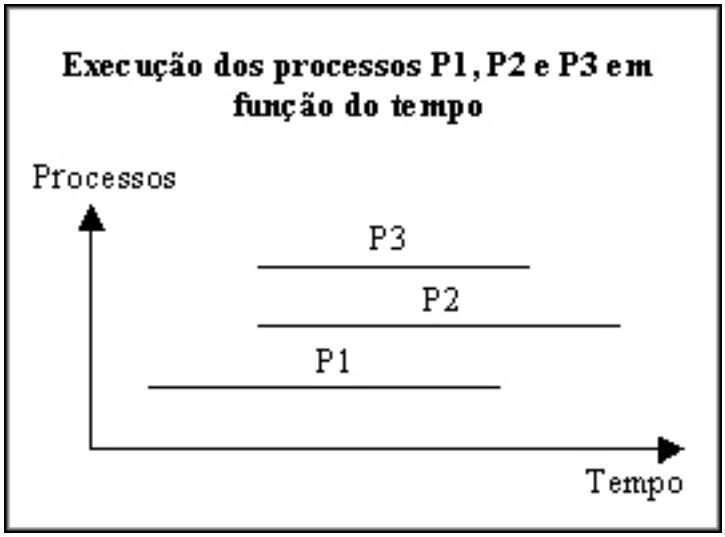

Figura 2.2: Paralelismo físico

cessadores acarretando um tempo final de processamento menor. Sendo de responsabilidade do programador a divisão dessas tarefas entre os processadores.

2.3 Tipos de paralelismo

A computação paralela é classificada em dois estilos de paralelismo: o paralelismo de dados e o paralelismo de controle.

O paralelismo de dados consiste do uso de unidades funcionais múltiplas para aplicar a mesma operação simultaneamente aos elementos de uma base de dados. Teoricamente k-unidades produzem um aumento de vazão (número de resultados produzidos por unidade de tempo) de k-vezes no sistema [27]. Neste tipo de paralelismo os processadores executam as mesmas instruções sobre dados diferentes. Sendo este estilo de paralelismo aplicado, por exemplo, na resolução de sistemas lineares e multiplicação de matrizes.

Em contraste com o paralelismo de dados, onde uma simples operação é aplicada sobre uma base de dados, o paralelismo de controle consiste em aplicar diferentes operações sobre diferentes elementos de dados simultaneamente. O fluxo de dados sobre estes processos pode ser arbitrariamente complexo [27]. No para-

lelismo de controle a computação é dividida em passos, chamados de segmentos ou estágios, que são distribuídos entre os processadores. Cada segmento trabalha numa parte da computação e, em alguns casos, a saída de um segmento é a entrada do próximo segmento (princípio utilizado em linhas de produção).

A Figura 2.3 faz uma comparação entre os tipos de paralelismos. Em (a) tem-se um algoritmo seqüencial executando uma computação C que produz um resultado R_i a cada três unidades de tempo. Neste caso, o resultado R_2 só começa a ser produzido após o resultado R_1 já estar terminado.

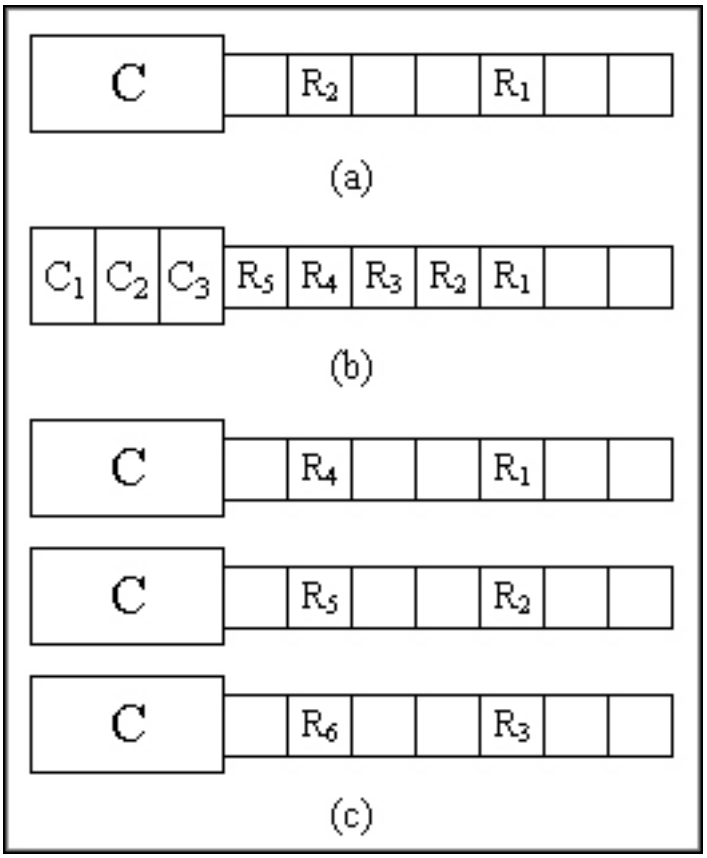

Figura 2.3: Comparação entre os tipos de paralelismo

Em (b) tem-se um exemplo de paralelismo de controle, onde a computação C é dividida nos passos C_1, C_2 e C_3, que são executados em paralelo. O primeiro

resultado é produzido após três unidades de tempo, e em seguida um resultado é obtido a cada unidade de tempo. Isso é possível pois o próximo resultado não precisa esperar o término do anterior para iniciar sua produção (como em uma linha de produção). Dessa forma, quando o resultado R_1 segue para o passo C_2, o resultado R_2 já inicia sua produção no passo C_1. De forma semelhante, quando o R_1 chega ao passo C_3, R_2 vai para o passo C_2 e R_3 inicia o passo C_1, e assim sucessivamente. Este processo também é conhecido como *Pipeline*.

Já em (c) ocorre o paralelismo de dados, onde três unidades funcionais executam as mesmas operações sobre os dados. De forma que a cada três unidades de tempo são produzidos três resultados.

No desenvolvimento de algoritmos paralelos, deve-se procurar escolher o estilo de paralelismo que mais se adapte a natureza do problema. Muitos problemas reais também podem explorar ambos os tipos de paralelismo.

2.4 Escalabilidade

Um algoritmo é escalável se o nível de paralelismo cresce pelo menos linearmente com o tamanho do problema. E uma arquitetura é escalável se ela continua rendendo o mesmo desempenho por processador quando o número de processadores aumenta [27].

As escalabilidades algorítmica e arquitetural são importantes, porque elas permitem a resolução de grandes problemas na mesma quantidade de tempo ao se usar um computador paralelo com mais processadores.

Algoritmos paralelos em dados são mais escaláveis que algoritmos paralelos em controle, pois o nível de paralelismo de controle é geralmente uma constante, independente do tamanho do problema, enquanto que o nível de paralelismo de dados é uma função crescente do tamanho do problema [27].

2.5 Medidas de qualidade

Duas importantes medidas de qualidade de algoritmos paralelos são ganho e eficiência.

O ganho obtido com a paralelização, chamado de *Speedup*, é a razão entre o tempo em que um computador paralelo executando o algoritmo seqüencial mais eficiente realiza uma computação (com apenas um processador) e o tempo em que o mesmo computador paralelo executando o correspondente algoritmo paralelo

usando p processadores realiza esta mesma computação [27]. Ou seja,

$$Speedup = \frac{tempo\ seqüencial(1\ processador)}{tempo\ paralelo(p\ processadores)}$$

A eficiência de um algoritmo paralelo ao ser executado em uma máquina paralela com p processadores corresponde à razão entre o *Speedup* alcançado por este algoritmo e o número de processadores utilizados [27]. Ou seja,

$$Eficiência = \frac{Speedup}{p}$$

O *Speedup* ideal a ser alcançado, isto é, o máximo ganho obtido com a paralelização, deveria assintotar ao número de processadores utilizados, entretanto na prática isto não acontece.

O *Speedup* não é diretamente proporcional ao número de processadores utilizados. Contudo, afirma-se que um *Speedup Superlinear* (maior que linear) é possível desde que a escolha do algoritmo seja feita antes da instância do problema e que o algoritmo paralelo trate alguns casos específicos que o seqüencial trata de modo genérico [27].

A Figura 2.4 ilustra a comparação entre *Speedup* ideal (linear) e o *Speedup* obtido na prática em função do número de processadores para um exemplo extraído de [27].

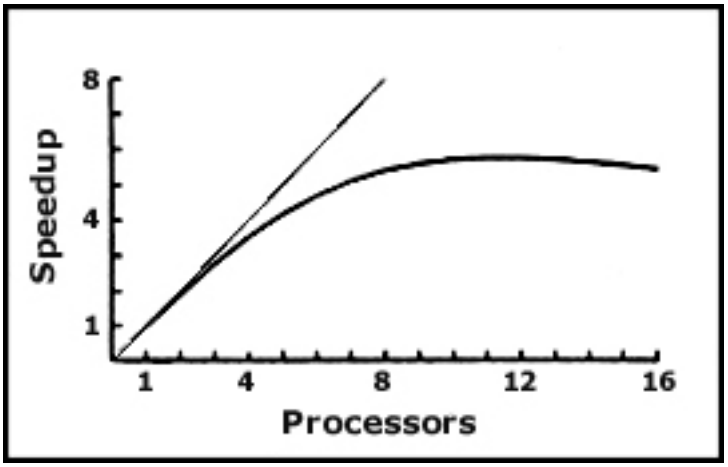

Figura 2.4: Comparação entre *Speedup* ideal e o *Speedup* teórico

A sobrecarga de tempo de comunicação entre os processadores, e o número de tarefas inerentemente seqüenciais são fatores que causam queda de *Speedup*. À medida que são adicionados processadores à execução de um problema, ocorre uma diminuição do tempo de computação do problema, entretanto, o tempo gasto em comunicação entre os processadores aumenta, influindo no tempo total de execução. A Figura 2.5 ilustra essa situação para o exemplo abordado na Figura 2.4 [27].

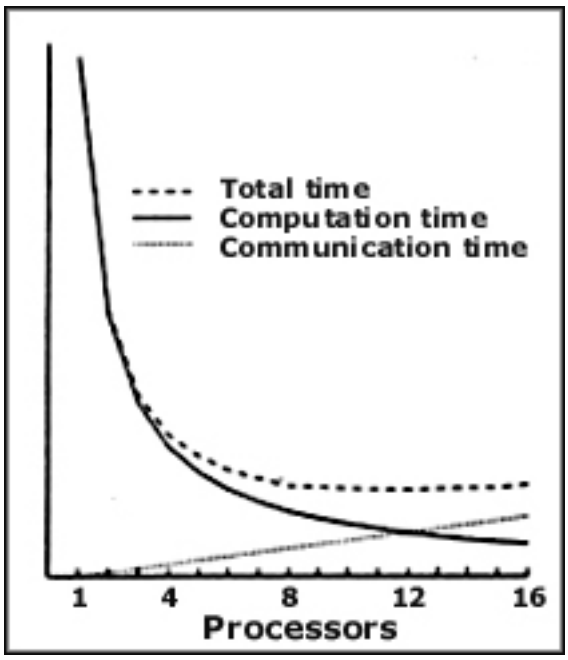

Figura 2.5: Tempo total de execução

O tempo de computação é inversamente proporcional ao número de processadores utilizados, enquanto que o tempo de comunicação aumenta linearmente com o número de processadores usados. Depois de um limite (*Speedup* máximo alcançado), o aumento no tempo de comunicação é maior que a diminuição no tempo de computação, e o tempo total de execução começa a aumentar [27].

Como exemplo de operações estritamente seqüenciais que causam perda de Speedup temos a incorporação de dispositivos de *Input/Output* (I/O) a uma arquitetura paralela, como ilustrado na Figura 2.6.

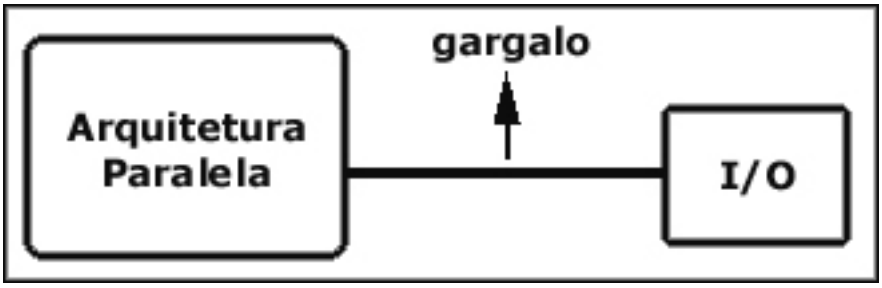

Figura 2.6: Incorporação de um dispositivo de I/O a uma arquitetura paralela

Como a saída para o dispositivo de I/O deve ser realizada seqüencialmente, esta funciona como um regulador para o *Speedup* obtido na paralelização. A saída é similar a um gargalo para o algoritmo paralelo.

A Lei de Amdahl [3] é uma maneira de expressar o *Speedup* máximo como uma função da quantidade de paralelismo e da fração da computação que é inerentemente seqüencial.

Esta lei tem o seguinte enunciado: seja f a fração de operações numa computação que deve ser executada seqüencialmente, onde $0 \leq f \leq 1$. O *Speedup* máximo (S) alcançado por um computador paralelo com p processadores executando a computação é

$$S \leq \frac{1}{f + \frac{(1-f)}{p}}$$

Como conseqüência dessa lei tem-se que a fração de operações seqüenciais pode significantemente limitar o *Speedup* alcançado por um computador paralelo.

A Lei de Amdahl objetiva reduzir o tempo no qual problemas com tamanho fixo podem ser resolvidos. Em muitas situações, contudo, o processamento paralelo pode ser usado para aumentar o tamanho de problemas que podem ser resolvidos em um tempo fixo. Além disso, todo programa paralelo possui sobrecargas fixas, como a criação e destruição de processos, que são independentes do tamanho do problema, e sobrecargas, como I/O e sincronização de processos, que podem aumentar com o tamanho do problema, mas em uma taxa bem menor do que o nível de paralelismo.

Sob essas circunstâncias a Lei de Amdahl não é um bom indicador de *Speedup*, pois ela assume que a fração seqüencial é fixa, o que não é verdade. E como pode ser visto na Figura 2.7, em geral, quando o tamanho do problema cresce a

fração f de operações seqüenciais decresce, tornando o problema mais viável à paralelização. Este fenômeno é chamado de Efeito Amdahl [12]. Isto pode ser confirmado na Figura 2.8, que exibe o *Speedup* obtido com o aumento do tamanho de um problema. As Figuras 2.7 e 2.8 são de um exemplo extraído de [27].

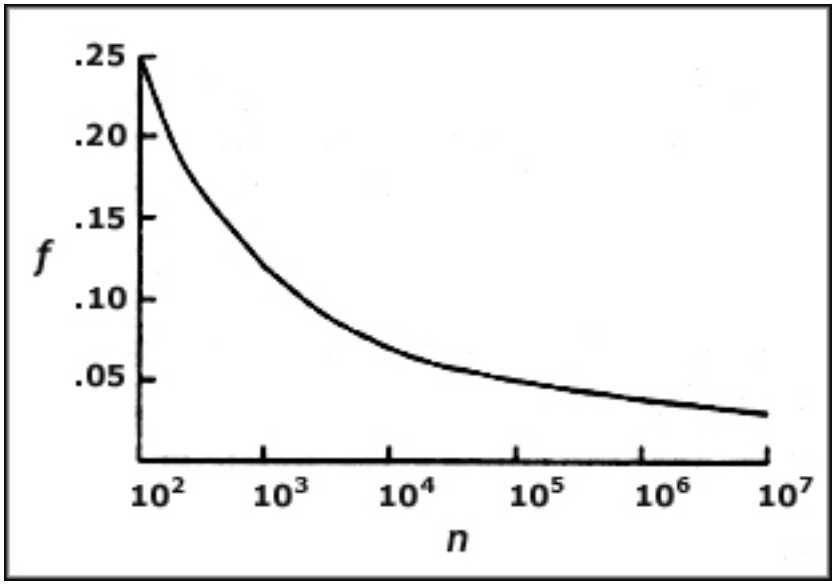

Figura 2.7: Fração de operações seqüenciais em função do tamanho do problema

Outro aspecto a ser observado é que a definição dada de *Speedup* não é muito realista. Para medir o *Speedup*, o algoritmo seqüencial deve ser executado em apenas um processador, o que significa que os dados do problema devem ser colocados na memória desse processador.

Os problemas de grande porte geralmente são resolvidos por uma enorme quantidade de processadores, estando sua imensa base de dados distribuída entre as memórias dos mesmos. Com isso, se torna praticamente impossível que os dados desses problemas sejam alocados na memória de um único processador. Logo estes problemas não podem ser usados para determinar o *Speedup* alcançado pelo computador paralelo.

Em razão dessa restrição, faz-se necessário a adaptação do conceito de *Speedup*, definindo-se seu variante, chamado de *Scaled Speedup*, como a razão entre o tempo que o algoritmo seqüencial ótimo realiza uma computação em um computa-

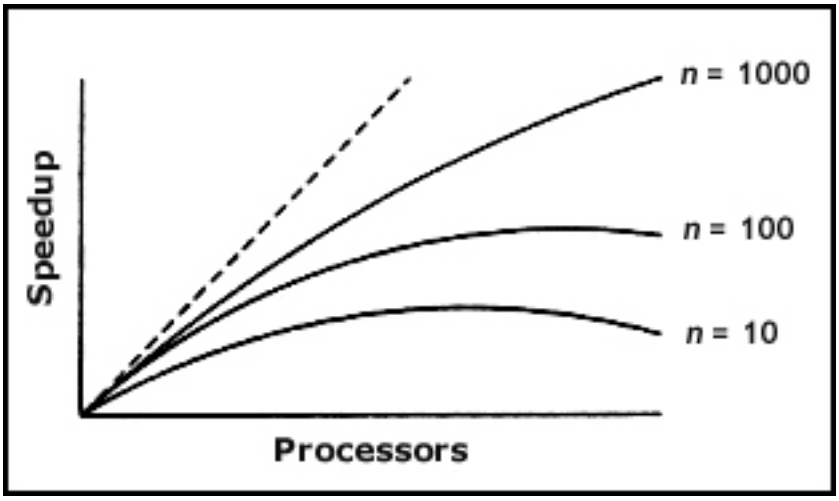

Figura 2.8: Speedup alcançado para diferentes tamanhos de um problema

dor paralelo com apenas um processador, desde que o problema possa ser alocado por completo na memória deste, e o tempo que o algoritmo paralelo executa o mesmo problema no mesmo computador paralelo utilizando p processadores [13, 14]. Ou seja,

$$Scaled\ Speedup = \frac{*tempo\ seqüencial(1\ processador)}{tempo\ paralelo(p\ processadores)}$$

* *considerando-se que o problema caiba na memória do processador*.

E de forma semelhante, tem-se a variante de eficiência, chamada de *Scaled Efficiency*, definida como a razão entre o *Scaled Speedup* e o número p de processadores utilizados [27]:

$$Scaled\ Efficiency = \frac{Scaled\ Speedup}{p}$$

Por refletir melhor a realidade, o *Scaled Speedup* se torna superior ao *Speedup* como uma medida da qualidade de algoritmos paralelos.

2.6 Modelo PRAM

O modelo de computação paralela PRAM [27] (*Parallel Random Access Machine*) permite aos desenvolvedores de algoritmos paralelos tratarem poder de processamento como recurso ilimitado. Este é o modelo mais conhecido de computação paralela, contudo ele não é realista, pois ignora a comunicação entre processadores.

Como a complexidade de comunicação não é considerada, ao se desenvolver algoritmos PRAM a preocupação está apenas no paralelismo inerente à computação.

O PRAM consiste de uma unidade de controle, memória global e um conjunto ilimitado de processadores, cada um com sua própria memória local, como ilustrado na Figura 2.9.

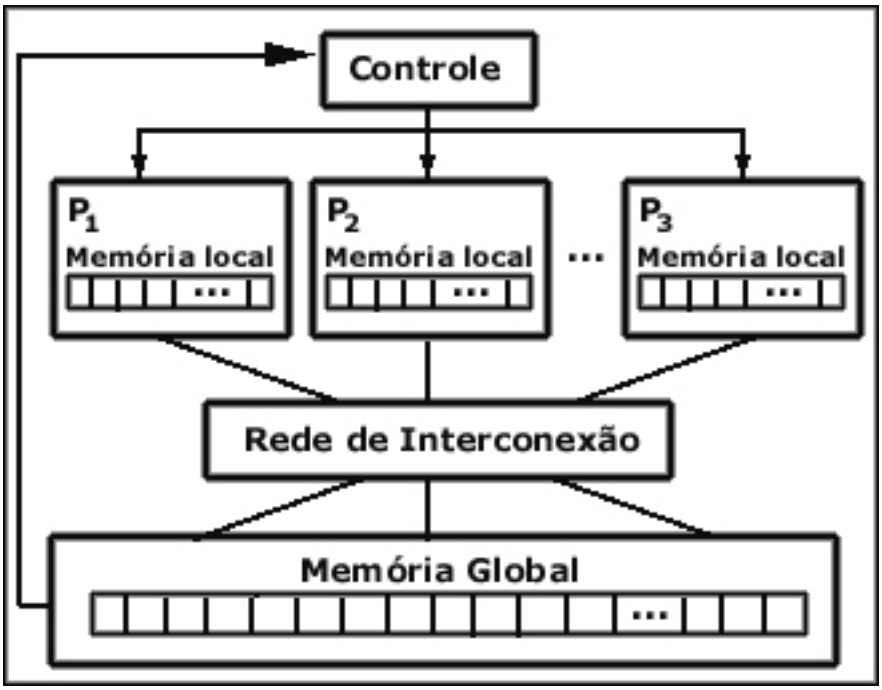

Figura 2.9: Modelo PRAM

Neste modelo os processadores ativos executam instruções idênticas, e cada processador tem um único índice. Este índice pode ser utilizado para habilitar ou

desabilitar o processador ou influenciar no acesso à memória.

Uma computação começa com a entrada armazenada em memória global e um único processador ativo. A cada passo de computação um processador ativo pode ler um valor de sua memória local ou global, executar uma operação simples e escrever o resultado na memória local ou global. Durante a computação, um processador pode ativar um outro processador. Todos os processadores ativos devem executar a mesma instrução, embora em posições de memória diferentes. A computação termina com o fim do processamento do último processador.

Como um algoritmo PRAM inicia com um único processador, os algoritmos têm duas fases. Na primeira fase ocorre a ativação de um número suficiente de processadores e na segunda fase a computação é realizada pelos processadores ativos.

O custo de uma computação PRAM é o produto da complexidade de tempo paralela e o número de processadores usados [27]. Por exemplo, um algoritmo PRAM que tenha complexidade de $\Theta(log\ p)$ usando p processadores tem custo $\Theta(p\ log\ p)$.

Um algoritmo paralelo com custo ótimo é um algoritmo para o qual o custo está na mesma classe de complexidade do algoritmo seqüencial ótimo [27]. Um algoritmo PRAM de custo ótimo é um dos principais candidatos para servir como base para eficientes algoritmos em computadores paralelos reais.

2.7 Ambiente paralelo em sistemas distribuídos

Em razão do alto custo de arquiteturas maciçamente paralelas, se torna uma alternativa interessante e altamente viável o uso de sistemas distribuídos para computação paralela. A idéia básica é ter um grupo de computadores interligados, funcionando como os elementos de processamento de uma máquina paralela.

Para a realização da computação paralela sobre sistemas distribuídos são utilizados ambientes de troca de mensagens (ou interfaces de troca de mensagens). Esses ambientes têm sido aperfeiçoados nos últimos anos para serem utilizados por uma grande quantidade de equipamentos diferentes, ganhando com isso popularidade e aceitação, visto que proporcionam o desenvolvimento de aplicações paralelas a um custo relativamente baixo em relação às máquinas paralelas [4].

Nos ambientes de troca de mensagem (*Message Passing*), cada computador é visto como um processador da arquitetura paralela com sua memória local, e para a interação entre estes computadores é utilizado um método de comunicação que é baseado no envio e recebimento de mensagens através da rede, seguindo as regras

do protocolo de comunicação da rede.

A comunicação é feita basicamente através de rotinas de uma biblioteca de comunicação que atua como uma extensão das linguagens seqüenciais (como C e Fortran).

O modelo computacional de troca de mensagem não inclui sintaxe de linguagem e nem de biblioteca e é completamente independente do hardware.

Exemplos de ambientes de passagem de mensagem para redes de computadores heterogêneos são: *Parallel Virtual Machine* (PVM) [4] e *Message Passing Interface* (MPI) [16]. Por causa de suas características, apresentadas na Seção 2.8, o ambiente MPI foi utilizado neste trabalho.

Ao se realizar a computação em sistemas distribuídos heterogêneos, pode ocorrer vários tipos de incompatibilidades, como por exemplo [4]: arquiteturas, formato de dados, potência computacional, carga de trabalho em cada máquina e carga de trabalho na rede. Há, portanto, a necessidade dos ambientes de passagem de mensagem tratarem essas diferenças.

Apesar dessas diferenças, a computação paralela sobre sistemas distribuídos heterogêneos oferece várias vantagens, entre elas pode-se citar:

- custo reduzido, pois utiliza hardware já existente;
- alto desempenho, pois cada tarefa pode ser atribuída para a arquitetura mais apropriada;
- utilização de editores e compiladores, entre outros aplicativos, já disponíveis para a arquitetura, que auxiliem no desenvolvimento (existem muitas versões gratuitas);
- recurso mais acessível a maioria dos desenvolvedores de programas paralelos.

2.8 MPI

MPI é um ambiente de passagem de mensagem desenvolvido para arquiteturas de memória distribuída, máquinas paralelas massivas, NOWs (*network of workstations*) e redes heterogêneas.

Ele define um conjunto de rotinas que facilitam a comunicação (troca de dados e sincronização) entre os processos paralelos. Sua biblioteca é portável para qualquer arquitetura, e tem aproximadamente 125 funções para programação além de

ferramentas para análise do desempenho da computação paralela. Sua biblioteca pode ser usada para programas em C, C++ e Fortran.

O MPI assume a existência de comunicações confiáveis. Ele não é um ambiente completo para computação paralela, pois não implementa: I/O paralelo, depuração de programas concorrentes, canais virtuais para comunicação e outras características próprias de tais ambientes [35].

No ambiente MPI, se os processadores executarem códigos diferentes sobre os dados está sendo utilizado o paradigma MPMD (*Multiple Program Multiple Data*), por outro lado, quando eles executam o mesmo código é utilizado o paradigma SPMD (*Simple Program Multiple Data*) [35].

Como vantagens da utilização do ambiente MPI na implementação de algoritmos paralelos, podem ser citadas dentre outras:

- *eficiência*: foi cuidadosamente projetado para executar eficientemente em máquinas heterogêneas;
- *fácil implementação*: linguagem similar à linguagem de algoritmos PRAM, possuindo muita flexibilidade;
- *portabilidade*: é compatível a sistemas de diferentes arquiteturas;
- *padronização*: especifica somente o funcionamento lógico das operações, deixando em aberto a implementação.
- *otimização*: os desenvolvedores otimizam o código usando características específicas de cada máquina;
- *transparência*: permite que um programa seja executado em sistemas heterogêneos sem mudanças significativas;
- *segurança*: provê uma interface de comunicação confiável, com isso o desenvolvedor não precisa se preocupar com falhas na comunicação;
- *escalabilidade*: suporta escalabilidade sob diversas formas. O uso de operações de comunicação coletiva, por exemplo, melhora o alcance dos processos.

As principais implementações do padrão MPI são:

- IBM MPI: Implementação IBM para SP e clusters;

- MPICH: Argonne National Laboratory;
- LAM: Ohio Supercomputer Center;
- UNIFY: Mississipi State University;
- PMPIO: NASA;
- CHIMP: Edinburgh Parallel Computing Center.

Para este trabalho foi utilizada a implementação LAM (*Local Area Multicomputer*). Esta implementação é portável à maioria das máquinas UNIX e inclui suporte padrão para SUN (SunOS and Solaris), SGI, IRIX, IBM AIX, DEC OSF/1, HP-UX e Linux.

LAM implementa todo o padrão MPI-1, e muitas características do padrão MPI-2. Possui tolerância a falha, e rápida comunicação. Além disso, esta implementação também facilita o trabalho dos desenvolvedores de algoritmos paralelos, pois pode ser instalada em uma máquina seqüencial com o sistema operacional Linux, onde pode ser simulado o funcionamento de algoritmos paralelos. A implementação LAM/MPI é gratuita, sob licença BSD, e está disponível em [16].

Capítulo 3

Teoria de Conjuntos Aproximados

3.1 Introdução

A Teoria de Conjuntos Aproximados, considerada uma extensão da teoria de conjuntos, é um modelo matemático utilizado no tratamento de incerteza e imprecisão, representação de conhecimento e classificação aproximada.

A principal vantagem da utilização da TCA é que esta não necessita de qualquer informação adicional ou preliminar a respeito dos dados, como por exemplo, distribuição de probabilidade, atribuição de crenças, grau de pertinência ou possibilidade. Como aplicações dessa teoria em Inteligência Artificial e Ciências Cognitivas, podem ser citadas:

- representação de conhecimento [38];
- aquisição de conhecimento [15];
- tratamento de incerteza [33];
- descoberta de conhecimento em base de dados [19];
- sistemas de suporte à decisão [1].

Esta teoria tem incentivado pesquisas que buscam o desenvolvimento de sistemas lógicos e métodos dedutivos para a representação, manipulação e raciocínio na presença de informações incompletas.

Os conceitos e definições apresentados neste capítulo foram em sua maioria extraídos de [34], quando isto não ocorrer será citada a referência adequada.

3.2 Conceitos básicos

Seja U um conjunto universo, finito e não-vazio de objetos. Neste conjunto U são definidos subconjuntos através de uma relação de equivalência R, chamada de *relação de indiscernibilidade*. A relação R induz uma partição (e conseqüentemente, classificação) dos objetos de U. Objetos pertencentes a uma mesma classe de equivalência de R, isto é, a uma mesma partição de U, não são distinguíveis.

Dessa forma um espaço aproximado pode ser definido como um par ordenado $A = (U, R)$, onde dados $x, y \in U$, se xRy então x e y são indiscerníveis em A. A classe de equivalência definida por x é a mesma que é definida por y, ou seja, $[x]_R$ = $[y]_R$.

As classes de equivalência induzidas por R em U são denominadas *conjuntos elementares*. Uma partição de U por R, notada por U/R, pode ser vista como o conjunto $\tilde{R} = U/R = \{E_1, E_2, ..., E_n\}$, onde cada E_i, com $1 \leq i \leq n$, é um conjunto elementar de A. Assume-se que o conjunto vazio $\emptyset$ é um conjunto elementar para todo espaço aproximado A. O espaço aproximado $A = (U, R)$ pode ser alternativamente notado por $A = (U, \tilde{R})$.

Seja E uma classe de equivalência em U, $des(E)$ denota a *descrição desse conjunto elementar*. Essa descrição é função do conjunto de atributos que define R. Dados $x, y \in E$, onde E é um conjunto elementar no espaço A, neste espaço não se consegue distinguir x de y, pois $des(x) = des(y) = des(E)$.

Um *conjunto definível* em A é qualquer união finita de conjuntos elementares. A família de todos os conjuntos definíveis em A será denotada por $def(A)$.

Exemplo: Seja $A = (U, R)$ um espaço aproximado, onde $U = \{a, b, c, d, e\}$ e $U/R = \{\{a, e\}, \{c\}, \{b, d\}\}$.

Os conjuntos elementares em A são:
$$E_1 = \{a, e\},\ E_2 = \{c\},\ E_3 = \{b, d\},\ E_4 = \emptyset.$$

Os conjuntos definíveis em A são:
$$D_1 = \{a, e\},\ D_2 = \{c\},\ D_3 = \{b, d\},\ D_4 = \emptyset,\ D_5 = \{a, b, d, e\},$$
$$D_6 = \{a, c, e\},\ D_7 = \{b, c, d\},\ D_8 = \{a, b, c, d, e\} = U.$$

3.3 Aproximações de um conjunto

Dado um espaço aproximado $A = (U, R)$, seja $X \subseteq U$ um subconjunto qualquer de objetos de U. Com o objetivo de verificar o quão bem X é representado pelos conjuntos elementares de A, são definidos os seguintes conceitos:

- *aproximação inferior* de X em A, formada pela união de todos os conjuntos elementares de A que estão totalmente contidos em X, ou seja, é o maior conjunto definível em A contido em X:

$$A_{A-inf}(X) = \{x \in U \mid [x]_R \subseteq X\}.$$

- *aproximação superior* de X em A, formada pela união de todos os conjuntos elementares de A que possuem intersecção não-vazia com X, ou seja, é o menor conjunto definível em A contendo X:

$$A_{A-sup}(X) = \{x \in U \mid [x]_R \cap X \neq \emptyset\}.$$

Nas notações utilizadas neste trabalho, quando o espaço aproximado for conhecido e não houver risco de confusão, a referência ao espaço será abolida. As aproximações superiores e inferiores, por exemplo, podem ser simbolizadas respectivamente como $A_{sup}(X)$ e $A_{inf}(X)$.

Exemplo: Dado o espaço aproximado $A = (U, R)$, induzido pela relação de indiscernibilidade R em U e dado X um conjunto tal que $X \subseteq U$, como ilustrado na Figura 3.1. As aproximações inferior e superior de X encontram-se ilustradas nas Figuras 3.2 e 3.3, respectivamente.

Exemplo: Seja $A = (U, R)$ um espaço aproximado, onde $U = \{a, b, c, d, e\}$ e $U/R = \{\{a, e\}, \{c\}, \{b, d\}\}$. Dado um conjunto qualquer $X = \{a, c\}$, tem-se que as aproximações desse conjunto são:

$$A_{inf}(X) = \{c\},\ A_{sup}(X) = \{a, c, e\}.$$

Em muitos casos, o conjunto X pode ser uma união finita de conjuntos elementares, o que caracteriza X como um conjunto definível em A. Isto implica que:

$$A_{sup}(X) = A_{inf}(X) = X.$$

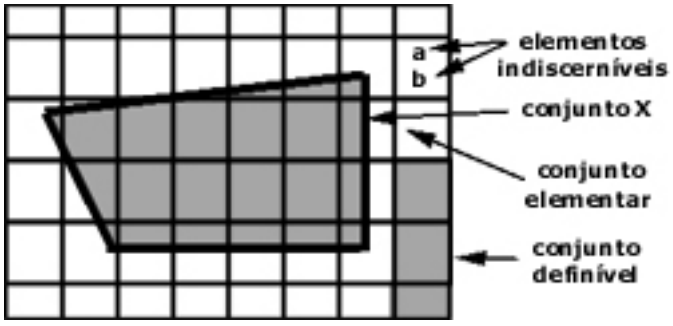

Figura 3.1: Espaço aproximado $A = (U, R)$ e o conjunto $X \subseteq U$

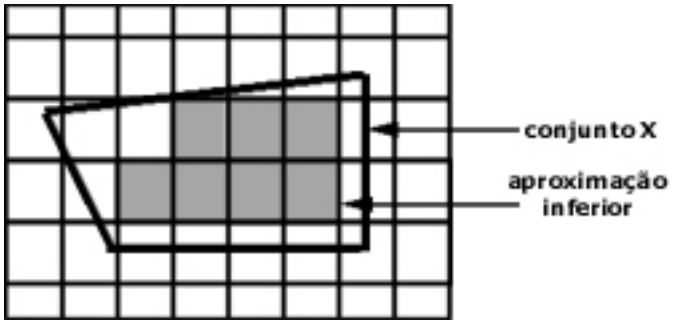

Figura 3.2: Aproximação inferior de X em A

3.4 Regiões de um espaço aproximado

Em um espaço aproximado $A = (U, R)$, com base na classificação aproximada de um conjunto $X \subseteq U$, é possível se identificar as seguintes regiões:

- *região positiva* de X em A, formada pela união de todos os conjuntos elementares de U contidos integramente em X, ou seja:

$$pos_A(X) = A_{A-inf}(X).$$

- *região negativa* de X em A, formada pelos conjuntos elementares de U que não possuem nenhum elemento em X, ou seja:

$$neg_A(X) = U - A_{A-sup}(X).$$

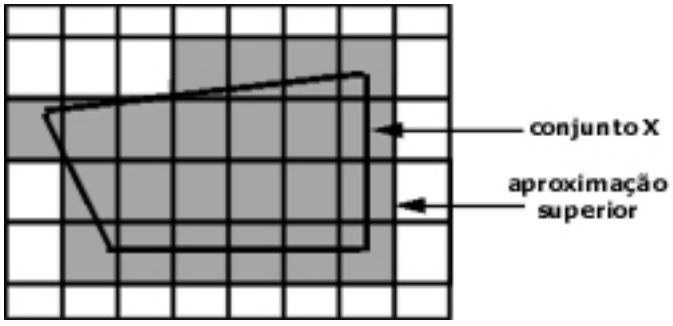

Figura 3.3: Aproximação superior de X em A

- *região duvidosa* de X em A, também chamada de fronteira de X, formada pelos conjuntos elementares de U que pertencem à aproximação superior mas não pertencem à aproximação inferior. A pertinência de um elemento dessa região ao conjunto X é incerta com base apenas nas classes de equivalência de A. Ou seja:

$$duv_A(X) = A_{A-sup}(X) - A_{A-inf}(X).$$

A Figura 3.4 ilustra as possíveis regiões para um conjunto $X \subseteq U$ em um espaço aproximado $A = (U, R)$.

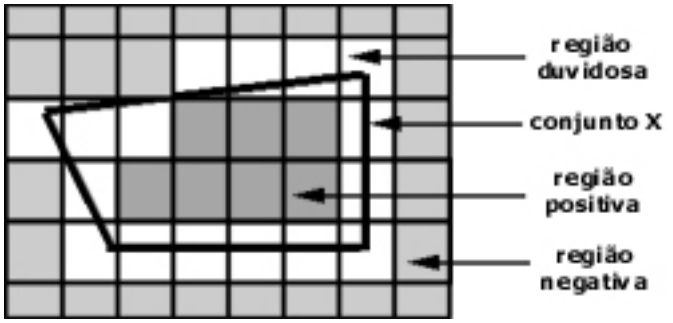

Figura 3.4: Regiões de um espaço aproximado $A = (U, R)$ para um conjunto $X \subseteq U$

Exemplo: Considere o espaço aproximado $A = (U, R)$, onde $U = \{a, b, c, d, e\}$ e $U/R = \{\{a, e\}, \{c\}, \{b, d\}\}$. Seja $X = \{a, c\}$ um conjunto tal $X \subseteq U$. Tem-se

então que:

$$A_{inf}(X) = \{c\},\ A_{sup}(X) = \{a, c, e\},\ pos(X) = \{c\},\ neg(X) = \{b, d\},\ duv(X) = \{a, e\}.$$

3.5 Igualdade aproximada de conjuntos

Dado um espaço aproximado $A = (U, R)$, o conjunto $X \subseteq U$ pode ou não ter sua fronteira claramente definida em função das descrições dos conjuntos elementares de A. Dessa forma, conjuntos aproximados são considerados conjuntos com fronteiras nebulosas, ou seja, conjuntos que não podem ser caracterizados precisamente utilizando-se do conjunto de atributos disponíveis.

Segundo Pawlak, em [22], um conjunto aproximado de $X \subseteq U$ no espaço aproximado $A = (U, R)$ é a família de todos os subconjuntos de U que possuem a mesma aproximação inferior e a mesma aproximação superior com relação a X em A.

Estas considerações levam ao conceito de igualdade aproximada: dois ou mais conjuntos são aproximadamente iguais se e se somente se possuem a mesma região positiva, negativa e duvidosa, isto é, definem o mesmo conjunto aproximado. Formalmente, num espaço aproximado $A = (U, R)$, com $X \subseteq U$, o conceito de igualdade aproximada é abordado da seguinte forma:

- X é *aproximadamente inf-igual* a Y, $X \,\underline{\tilde{=}}_A\, Y$, se somente se $A_{inf}(X) = A_{inf}(Y)$;
- X é *aproximadamente sup-igual* a Y, $X \,\tilde{=}_A\, Y$, se somente se $A_{sup}(X) = A_{sup}(Y)$;
- X é *aproximadamente igual* a Y, $X \approx_A Y$, se somente se $A_{inf}(X) = A_{inf}(Y)$ e $A_{sup}(X) = A_{sup}(Y)$, ou seja, $X \,\underline{\tilde{=}}_A\, Y$ e $X \,\tilde{=}_A\, Y$.

Exemplo: Os conjuntos $X, Y \subseteq U$ mostrados na Figura 3.5 são aproximadamente iguais. Nesse espaço aproximado o conjunto Y pode representar o conjunto X.

A partir da abordagem de igualdade aproximada, alguns casos especiais de conjuntos aproximados podem ser observados:

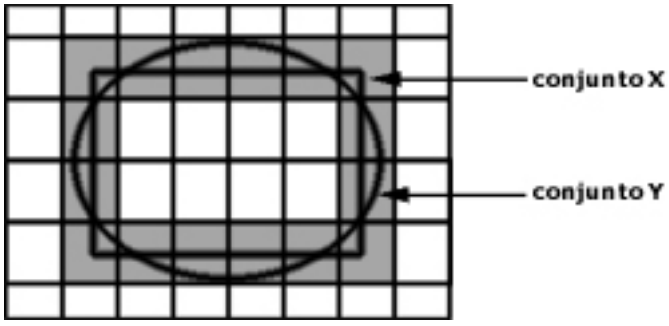

Figura 3.5: Conjuntos aproximadamente iguais

- *conjunto denso*, se $X \cong U$;
- *conjunto co-denso*, se $X \tilde{\cong} \emptyset$;
- *conjunto disperso*, se X for ao mesmo tempo denso e co-denso, ou seja, $X \cong U$ e $X \tilde{\cong} \emptyset$.

Exemplo: Seja $A = (U, R)$ um espaço aproximado e $X \subseteq U$, conforme mostra a Figura 3.6. Tem-se que $X \cong U$ e $X \tilde{\cong} \emptyset$, portanto X é um conjunto disperso.

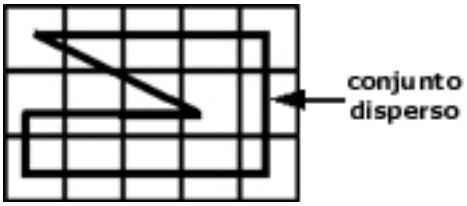

Figura 3.6: Conjunto disperso

3.6 Medidas da TCA

Algumas medidas são definidas com o propósito de verificar o quão bem um conjunto $X \subseteq U$ pode ser representado em um espaço aproximado $A = (U, R)$:

- *medida interna* de X em A:

$$\varpi_{A-inf}(X) =\mid A_{A-inf}(X) \mid .$$

- *medida externa* de X em A:

$$\varpi_{A-sup}(X) =\mid A_{A-sup}(X) \mid .$$

- *qualidade da aproximação inferior* de X em A:

$$\gamma_{A-inf}(X) = \frac{\varpi_{A-inf}(X)}{\mid U \mid} = \frac{\mid A_{A-inf}(X) \mid}{\mid U \mid}.$$

- *qualidade da aproximação superior* de X em A:

$$\gamma_{A-sup}(X) = \frac{\varpi_{A-sup}(X)}{\mid U \mid} = \frac{\mid A_{A-sup}(X) \mid}{\mid U \mid}.$$

- *acuracidade* de X em A:

$$\varpi_A(X) = \frac{\gamma_{A-inf}(X)}{\gamma_{A-sup}(X)} = \frac{\varpi_{A-inf}(X)}{\varpi_{A-sup}(X)} = \frac{\mid A_{A-inf}(X) \mid}{\mid A_{A-sup}(X) \mid}.$$

A medida de acuracidade indica o quanto X pode ser representado em A, onde $0 \leq \varpi_A(X) \leq 1$. Se $\varpi_A(X) = 1$, significa que $A_{A-inf}(X) = A_{A-sup}(X)$, logo X é definível em A. Se $\varpi_A(X) < 1$, então X é indefinível em A.

Com o propósito de verificar a pertinência de um elemento do universo U a qualquer das regiões definidas por um conjunto $X \subseteq U$, em [24] é proposta uma *função de pertinência aproximada*. A pertinência de um elemento x a um conjunto X pode ser definida como:

$$\mu_X^A(x) = \frac{\mid [x]_R \cap X \mid}{[x]_R}.$$

Nesta equação, $[x]_R$ denota a classe de equivalência induzida pela relação de equivalência R que contém o elemento x, ou seja, o conjunto elementar ao qual x pertence.

O número total de objetos em U que podem, com certeza, ser classificados em dois subconjuntos disjuntos, X e $U - X$, é dado por:

$$\mid U - duv_A(X) \mid =\mid U - (A_{A-sup}(X) - A_{A-inf}(X)) \mid$$

ou,

$$| U | - | A_{A-sup}(X) - A_{A-inf}(X) | .$$

A partir da consideração anterior é possível fornecer uma medida do grau de certeza na determinação da pertinência de um elemento de U no conjunto X, chamada de *índice discriminante* de A em relação a X, que é definido como:

$$\alpha_A(X) = \frac{| U | - | A_{A-sup}(X) - A_{A-inf}(X) |}{| U |}.$$

Sabendo-se que $0 \leq \alpha_A(X) \leq 1$, é possível se classificar a ***definibilidade*** de um conjunto $X \subseteq U$ em um espaço aproximado $A = (U, R)$ como:

- *totalmente A-definível*, se $\alpha_A(X) = 1$. Isto ocorre quando $A_{A-sup}(X) = A_{A-inf}(X)$, ou seja, a a pertinência do conjunto X pode ser especificada pelas descrições dos conjuntos elementares induzidos por R;

- *parcialmente A-definível*, se $0 < \alpha_A(X) < 1$. Isto ocorre se e somente se $A_{A-sup}(X) \neq A_{A-inf}(X)$ e $A_{A-sup}(X) \neq U$ ou $A_{A-sup}(X) \neq \emptyset$, ou seja, nem todos os objetos de U podem ser classificados com certeza em um dos conjuntos X ou $U - X$, uma vez que $duv(X) \neq \emptyset$. Um conjunto X *parcialmente A-definível* ainda pode ser classificado como:

  - *externamente A-indefinível*, se $A_{A-inf}(X) \neq \emptyset$ e $A_{A-sup}(X) = U$. O que implica em ser impossível excluir qualquer elemento de U de ser também elemento de X, ou seja, todo elemento de U é também elemento de X;
  - *internamente A-indefinível*, se $A_{A-inf}(X) = \emptyset$ e $A_{A-sup}(X) \neq U$. O que implica em ser impossível garantir a pertinência de qualquer elemento de U a X.

- *totalmente A-indefinível*, se $\alpha_A(X) = 0$. Isto ocorre se e somente se $A_{A-inf}(X) = \emptyset$ e $A_{A-sup}(X) = U$, ou seja, X é disperso. Neste caso é totalmente impossível a especificação da pertinência de objetos ao conjunto X.

Capítulo 4

Sistemas Baseados em Conhecimento

4.1 Introdução

Os Sistemas Baseados em Conhecimento (SBCs) são definidos como programas de computador que resolvem problemas utilizando conhecimento representado explicitamente e que, não fosse essa representação, exigiriam um especialista humano no domínio do problema para a sua solução [34]. Dentre as características deste conhecimento pode-se ressaltar que este é volumoso, dinâmico, que envolve relacionamento entre objetos e que pode ser representado sob diversas formas [28].

Os SBCs são muitas vezes referenciados na literatura como Sistemas Especialistas (SEs). Contudo os SEs correspondem a uma importante subclasse dos SBCs. Os SEs são concebidos para reproduzir o comportamento de especialistas humanos na resolução de problemas do mundo real, mas o domínio destes problemas é altamente restrito [5]. Em outras palavras, um SE pode ser definido como um SBC que resolve problemas bem específicos do mundo real, problemas esses que requerem considerável habilidade, conhecimento e heurísticas para sua resolução [34].

A maior parte dos SBCs se apresentam sob a forma de Sistemas Baseados em Regras (SBRs) ou Sistemas de Produção (SPs). Os SBRs são todos os sistemas baseados em regras de produção (ou regras de decisão), isto é, pares de expressões consistindo em uma condição e uma ação [5]. Dessa forma, esses SBCs utilizam o formato dessas regras como método de representação do conhecimento. Uma regra de produção possui a forma "antecedente $\rightarrow$ conseqüente", onde o antecendente da regra corresponde a uma condição, que se satisfeita leva à ação expressa

no conseqüente da mesma. Uma regra pode ser lida como “se o antecendente é verdade, logo o conseqüente também é verdade”.

As principais vantagens dos sistemas de produção como método de representação de conhecimento são: modularidade, uniformidade e naturalidade. As principais desvantagens são: ineficiência em tempo de execução e complexidade do fluxo de controle que leva à solução dos problemas. Estas vantagens e desvantagens determinam as características que devem ter os domínios que se adaptam ao desenvolvimento de SBCs baseados em sistemas de produção: ser descrito por um conhecimento consistindo em um conjunto muito grande de fatos parcialmente independentes, dispor de métodos de solução consistindo de ações independentes, e apresentar uma nítida separação entre conhecimento e ação [5].

4.2 Arquitetura Básica

Na Figura 4.1 pode ser visualizada a arquitetura básica de um SBC, sendo que esta é constituída de três módulos principais: Base de Conhecimento, Motor de Inferência e Interface com o Usuário.

Figura 4.1: Arquitetura de um Sistema Baseado em Conhecimento

A Interface com o Usuário é o módulo responsável pela comunicação entre o usuário e o sistema. Ela deve fornecer também justificativas e explicações referentes às conclusões obtidas, bem como do raciocínio utilizado [34].

O Motor de Inferência é o mecanismo responsável pelo processamento do conhecimento da Base de Conhecimento, utilizando-se de alguma linha de raciocínio. Implementa as estratégias de inferência e controle do SBC. Em um SBC que utiliza regras de produção como representação do conhecimento, o Motor de Inferência corresponde ao interpretador de regras. É ele que avalia e aplica as regras quando necessário [34].

Em último lugar, a Base de Conhecimento contém o conhecimento específico do domínio da aplicação. É composto de fatos sobre o domínio, regras que descrevem relações no domínio e métodos e heurísticas para resolução de problemas no

domínio.

Em seu estado inicial, o conhecimento de um SBR se encontra em um Sistema de Representação de Conhecimento (SRC). Geralmente um SRC corresponde a uma tabela de dados ou tabela de decisão, onde cada linha equivale a descrição de um objeto e cada coluna a um atributo de descrição ou classificação. A Tabela 4.1 é um exemplo de uma tabela de decisão onde o último atributo poderia ser utilizado como atributo de classificação. Neste exemplo, a partir da especificação de algumas vitaminas, o SBC poderia fornecer ao usuário o(s) alimento(s) onde estas vitaminas são encontradas.

Tabela 4.1: Exemplo de SRC

$vitA$	$vitB1$	$vitB2$	$vitB5$	$vitC$	$Alimento$
S	S	S	S	S	Agrião
N	S	N	N	N	Milho
S	S	S	N	S	Pera
N	S	S	S	S	Cajú
N	N	N	N	S	Laranja
S	S	S	S	S	Tomate
S	S	S	S	N	Cenoura

Um SRC é formalmente definido como a 4-upla $S = (U, Q, V, p)$ [21], onde:

- U: *universo* finito de S. Os elementos de U são chamados *objetos*, que são caracterizados por um conjunto de *atributos* e seus respectivos *valores*. Quando SRCs são usados no contexto de aquisição de conhecimento, os elementos de U são também denominados *exemplos*;
- Q: conjunto finito de atributos. Se o SRC é expresso como uma tabela de decisão, o conjunto Q pode ser expresso como $C \cup \{\delta\}$, onde o conjunto C representa os atributos de descrição (ou condição) e o atributo δ representa o atributo de classificação (ou decisão);
- $V = \bigcup_{q \in Q} V_q$ é o conjunto dos valores de atributo. V_q é chamado de *domínio* do atributo q;
- p: U x $Q \rightarrow V$ é uma *função de descrição* tal que $p(x, q) \in V_q$ para $x \in U$ e $q \in Q$. Um par (q, v), $q \in Q$, $v \in V_q$, é denominado *descritor* em S. A função de descrição é muitas vezes notada por p_x: $Q \rightarrow V$, tal que $p_x(q) = p(x, q) \in V_q$, $x \in U$, $q \in Q$.

O SBR necessita extrair de um SRC as regras de produção que constituirão a Base de Conhecimento. Este pré-processamento é realizado através de algum processo de aquisição de conhecimento.

4.3 Aquisição de Conhecimento

Aquisição de conhecimento corresponde ao processo de transferência do conhecimento - informações e/ou formas de condução do raciocínio - do especialista humano (ou de qualquer outra fonte) à Base de Conhecimento do SBC. Este processo é a parte mais difícil no desenvolvimento de SBCs, exigindo um grande investimento em tempo e esforço [34].

Além da extração do conhecimento necessário, tal cohecimento deve ser traduzido para o esquema formal de representação usado pelo SBC e deve ser repetidamente refinado, até que o sistema atinja um grau de desempenho próximo ao de um especialista humano, quando da resolução do problema [34]. Segundo Boose, em [6], os métodos de aquisição de conhecimento podem ser classificados como:

- *manuais*, que tipicamente consistem de entrevistas e análise de protocolos;
- *baseados em computador*, que podem ser divididos em:
  - *interativos*: que englobam, principalmente, ferramentas que entrevistam o especialista, que fazem análise textual, que extraem e analisam conhecimento de múltiplas fontes separadamente e as combinam para uso;
  - *baseados em aprendizado*: os quais, via de regra, generalizam situações específicas em conceitos.

Como observado por Nicolleti, em [20], por ter o conhecimento natureza inerentemente dinâmica, o processo de aquisição de conhecimento nunca termina. Para a maioria dos domínios não existe um estado final de conhecimento completo. Dessa forma, o processo de aquisição de conhecimento não deve se restringir apenas a adição de novos elementos de conhecimento a Base de Conhecimento, ele deve viabilizar revisões, reestruturações e reorganizações da Base de Conhecimento, de maneira a possibilitar mudanças.

Muito embora quase sempre contextualizada como uma das maneiras de automaticamente adquirir conhecimento para alimentar um SBC, a área de Aprendizado de Máquina (AM) é bem mais ampla, no sentido que a pesquisa em AM se

dedica também à investigação de formas de condução do raciocínio humano, de formas de organização do conhecimento existente, de diferentes estratégias de raciocínio e suas articulações, assim como à análise teórica e à exploração do espaço de possíveis métodos. Além de SBCs, AM já tem sido utilizada, entre outros, em sistemas de planejamento, sistema tutores inteligentes, sistemas sensoriais, robôs autônomos e descoberta de conhecimento em base de dados [20]. De acordo com Uchôa, em [34], uma possível taxonomia para AM seria:

- *Aprendizado Simbólico*: aquisição de conceitos expressos em símbolos, através de conjuntos de exemplos;
- *Aprendizado Baseado em Instância*: métodos que simplesmente armazenam as instâncias de treinamento;
- *Aprendizado Baseado em Algoritmos Genéticos*: que inclui algoritmos genéticos, que induzem hipóteses descritas usando cadeias de bits, e programação genética, que induz hipóteses descritas como programas;
- *Aprendizado Conexionista*: buscam modelar o processo de funcionamento dos neurônios e/ou áreas do cérebro humano;
- *Aprendizado Analítico*: aprendizado baseado em explicações e certas formas de métodos de aprendizado analógicos e baseados em casos.

Dentre todos os modelos citados acima, apenas o Aprendizado Analógico é de natureza dedutiva, sendo todos os demais indutivos.

Em um Aprendizado Simbólico, conhecido como aprendizado indutivo baseado em exemplos, a partir de um conjunto de exemplos, expressões para tarefas classificatórias podem ser aprendidas (induzidas) como, por exemplo, diagnóstico de doenças, previsão meteorológica, predição do comportamento de novos compostos químicos, predição de propriedades mecânicas de metais com base em algumas de suas propriedades químicas, etc [34].

Um ponto importante na aquisição de conhecimento é o tratamento de incoerências. Dependendo da forma como o conhecimento é adquirido, pode haver erros de aquisição. Estes erros podem resultar da própria natureza do conhecimento, como em dados obtidos através de sensores sujeitos a ruído, ou podem ser gerados pela interface humana existente entre o mundo real e o sistema de representação.

Técnicas foram desenvolvidas para evitar erros de aquisição, como por exemplo, a especificação de regras de aquisição em que o tipo de conhecimento esperado

é definido. Estas técnicas são comuns aos SRCs e aos sistemas de gerenciamento de bancos de dados.

Por outro lado, uma base de conhecimento pode ser examinada periodicamente com a finalidade de detectar incoerências eventualmente introduzidas no processo de aquisição. Este método é limitado pelo fato de que linguagens de representação razoavelmente expressivas não contam com procedimentos completos de verificação conhecidos. Finalmente, deve-se observar que a adequação do formalismo de representação ao tipo de conhecimento do mundo real a ser representado é fundamental para a eficiência do processo de aquisição [5].

O tratamento de incerteza é uma ativa área de pesquisa de SBCs, pois os domínios adequados à implementação de SBCs se caracterizam exatamente por não serem modelados por nenhuma teoria geral, o que implica descrições incompletas, inexatas ou incertas. Diversos métodos foram propostos para tratar este problema, por exemplo, Método Bayesiano, Fatores de Certeza, Teoria de Dempster-Shafer, Teoria dos Conjuntos *Fuzzy*, Teoria de Probabilidades Subjetivas e Teoria de Possibilidades. De maneira geral, estes métodos atribuem aos fatos e regras uma medida numérica que represente de alguma forma a "confiança" do especialista. Os métodos utilizados não são necessariamente coerentes uns com os outros e cada método adapta-se melhor a determinados tipos de problemas [5].

O método baseado na Teoria de Conjuntos Aproximados, por outro lado, além de não necessitar de uma medida númerica adicional sobre os fatos ou regras, possui mecanismos para expressão de, entre outros, um tipo fundamental de incerteza: a indiscernibilidade. A indiscernibilidade surge quando não é possível distinguir objetos de um mesmo conjunto, ou seja, todos objetos de um conjunto parecem ser o mesmo objeto.

Entre os algoritmos mais conhecidos de aprendizado indutivo podem ser citados o ID3 [26], o AQ [18] e o CN2 [7]. Como foi observado por Pawlak, em [22], a TCA pode ser utilizada como formalismo para AM, além de ser utilizada no tratamento de incertezas. Um algoritmo de AM baseado na TCA, conhecido por RS1, é descrito por Wong em [37]. Sob certas condições, o algoritmo RS1 pode ser considerado um caso especial do algoritmo ID3 [34].

Capítulo 5

Algoritmo Seqüencial

5.1 Introdução

Os conceitos da TCA podem ser muito bem utilizados como formalismo para o estabelecimento de métodos de aprendizado indutivo de máquina. Na literatura são encontrados dois algoritmos de AM que utilizam esta teoria. O primeiro, descrito em [23], realiza análise de dependência entre os atributos para a geração das regras. O segundo, conhecido como RS1 [37], é baseado no conceito de índice discriminante de atributos. Os dois algoritmos possuem algumas semelhanças, entretanto o RS1 possue melhor desempenho que o primeiro algoritmo por evitar alguns casos de complexidade.

Os dois algoritmos, a partir de uma tabela de decisão, geram regras de produção com a seguinte forma:

$$(a_1 = v_{a1}) \,\&\, (a_2 = v_{a2}) \,\&\, \ldots \,\&\, (a_n = v_{an}) \Rightarrow (b = v_b).$$

Nesta regra, o símbolo $\&$ representa o "e" lógico. Ela pode ser lida como: "se o atributo a_1 possue o valor v_{a1}, e o atributo a_2 possue valor v_{a2}, e assim sucessivamente até a_n, então o atributo de decisão b possue valor v_b".

As regras de decisão podem ser consideradas como consistentes ou inconsistentes. Uma regra é qualificada como consistente se não existe uma outra regra com mesmo antecendente. Caso contrário, a regra é considerada inconsistente, pois existe mais de uma regra com mesmo antecendente e, no entanto, com diferentes conseqüentes. Em outras palavras, existem condições iguais levando a decisões diferentes.

Em [34], Uchôa apresenta uma versão "melhorada" do algoritmo RS1, denominada RS1+. Esta nova versão do algoritmo RS1 foi utilizada como base para a criação do algoritmo paralelo.

O pseudocódigo dos principais conceitos da TCA e do algoritmo RS1+ são descritos em [34]. Nesta trabalho somente será descrito o algoritmo RS1 básico e sua adaptação para o algoritmo paralelo. As melhorias introduzidas na versão RS1+ não alteram a idéia do algoritmo paralelo, elas apenas adicionam algumas características que melhoram o desempenho do algoritmo.

5.2 Descrição do algoritmo

O algoritmo de aprendizado indutivo de máquina baseado em índice indiscriminante de atributos, denominado RS1, é descrito em [37]. Para uma dada tabela de decisão $S = (U, C \cup \{\delta\}, V, p)$, a idéia básica do algoritmo consiste em encontrar um conjunto mínimo de regras de decisão que sejam capazes de classificar todos as situações descritas nesta tabela.

A seguir são descritos os principais passos do algoritmo RS1:

---

1. calcular $Class_S(\delta) = \{X_1, X_2, \ldots, X_n\}$, a família de conjuntos elementares do espaço aproximado induzido por $\{\delta\}$
2. fazer $j = 1$
3. fazer $U' = U$, $C' = C$, $B = \emptyset$, $X = X_j$ e $S' = (U', C \cup \{\delta\}, V, p)$
4. calcular o conjunto de índices discriminantes $\{\alpha_{B'}(X) | B' = B \cup \{c\}, \forall c \cup C'\}$ em S'
5. selecionar o conjunto de atributos $B' = B \cup \{c\}$ com o maior valor $\alpha_{B'}(X)$
6. fazer $B = B'$
7. se $A_{B-inf}(X) = \emptyset$, ir para o passo 10
8. identificar os conjuntos elementares $E = \{E_1, E_2, \ldots, E_r\}$ do espaço aproximado induzido por B que estão contidos em $A_{B-inf}(X) = \emptyset$

9. para cada elemento $E_k \in E$, gerar uma regra de decisão determinística (consistente). Considerando que B possui m atributos, então cada regra tem a forma $(a_1 = v_{a1})$ & $(a_2 = v_{a2})$ & ... & $(a_m = v_{am}) \Rightarrow (b = v_b)$, onde $a_1, a_2, \ldots, a_m \in B$. Cada elemento $(a_k = v_{ak})$ do antecedente é construído da seguinte forma: a_k recebe o nome do k-ésimo atributo de B e v_ak recebe o valor atribudo a E_k por esse atributo. O conseqüente $(b = v_b)$ é construído de forma semelhante: b recebe o nome do atributo de decisão δ e v_b recebe o valor atribuído a E_k por esse atributo

10. fazer $U' = U' - [(U' - A_{B-sup}(X)) \cup A_{B-inf}(X)]$, $X = X - A_{B-inf}(X)$ e $S' = (U', C \cup \{\delta\}, V, p)$

11. se $U' = \emptyset$ ir para o passo 16

12. fazer $C' = C' - B$

13. se $C' \neq \emptyset$, voltar ao passo 4

14. identificar todos conjuntos elementares $E = \{E_1, E_2, \ldots, E_r\}$ do espaço aproximado induzido por B

15. para cada elemento $E_k \in E$, gerar uma regra de decisão não-determinística (inconsistente) de forma semelhante à descrita no passo 9

16. fazer $j = j + 1$

17. se $j \leq n$ voltar ao passo 3

18. devolver todas as regras encontradas nos passos 9 e 15

Descrição do funcionamento do algoritmo RS1: inicialmente são agrupados em conjuntos (chamados de X_j) os objetos que possuem a mesma classificação (mesmo valor) para o atributo de decisão. Em seguida, para cada um desses conjuntos X_j, o algoritmo procura um conjunto mínimo de atributos que sejam suficientes para classificar o conjunto X_j em questão. Para isso o algoritmo inicialmente verifica se apenas um dos atributos não é suficiente para classificar o conjunto X_j em questão (isto significa afirmar, por exemplo, que sempre que o atributo a for igual a v_a o valor do atributo de decisão b é v_b). Isto é feito através do cálculo

do índice discriminante para cada um dos atributos. Sendo que este valor indica o quanto cada atributo melhor classifica o conjunto X_j em questão. Se nenhum dos atributos for capaz de classificar com exatidão o conjunto de objetos em questão, é armazenado o atributo que melhor classificaria este conjunto (ou seja, atributo com maior índice discriminante). No próximo passo o algoritmo verifica se o atributo armazenado, em conjunto com um dos outros atributos, não classificam o conjunto X_j. E assim, sucessivamente, o algoritmo vai adicionando, a cada passo, novos atributos até encontrar um conjunto de atributos que classificam o conjunto X_j ou então até acabarem os atributos de descrição. Caso seja encontrado um conjunto mínimo de atributos que classificam o conjunto de objetos em questão, é gerada com esses atributos uma regra de decisão consistente. Caso contrário, quando são utilizados todos os atributos de descrição, é gerada uma regra considerada inconsistente. Isto ocorre pois existem objetos com mesma descrição que entretanto possuem classificações distintas. Dessa forma o algoritmo tenta encontrar regras de classificação para cada um dos conjuntos X_j.

No algoritmo RS1+, quando for gerada uma regra consistente é atribuído a ela um fator de credibilidade 1. Por outro lado, quando a regra gerada for inconsistente, o fator de credibilidade dessa regra é determinado através da função de pertinência aproximada.

Para exemplificar o funcionamento do algoritmo RS1, seja dada tabela de decisão $S = (U, C \cup \{\delta\}, V, p)$, onde $U = \{e_1, e_2, \ldots, e_6\}$, $C = \{a, b, c\}$, $\delta = d$ e V conforme mostrado na Tabela 5.1.

Tabela 5.1: Tabela de Decisão onde $C = \{a, b, c\}$ e $\delta = d$

	a	b	c	d
e_1	0	1	1	0
e_2	0	2	1	0
e_3	1	2	1	2
e_4	1	0	1	1
e_5	1	2	3	2
e_6	1	0	1	3

A execução passo a passo do algoritmo RS1 para este exemplo é mostrada a seguir:

1. $Class_S(\delta) = \{\{e_1, e_2\}, \{e_3, e_5\}, \{e_4\}, \{e_6\}\}$

2. j = 1

3. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $C' = \{a, b, c\}$, $B = \emptyset$ e $X = \{e_1, e_2\}$

4. Conjuntos elementares:

$\{a\}$: $E = \{\{e_1, e_2\}, \{e_3, e_4, e_5, e_6\}\}$
$\{b\}$: $E = \{\{e_1\}, \{e_2, e_3, e_5\}, \{e_4, e_6\}\}$
$\{c\}$: $E = \{\{e_1, e_2, e_3, e_4, e_6\}, \{e_5\}\}$

A Tabela 5.2 exibe o valor do índice discriminante para $X = \{e_1, e_2\}$.

Tabela 5.2: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_1, e_2\}$ e $B = \emptyset$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{a\}$	$\frac{6-0}{6} = \frac{6}{6} = 1$
$\{b\}$	$\frac{6-3}{6} = \frac{3}{6} = 0,5$
$\{c\}$	$\frac{6-5}{6} = \frac{1}{6} \cong 0,17$

5. $B' = B \cup \{a\}$

6. $B = B' = \{a\}$

7. $A_{B-inf}(X) = \{e_1, e_2\} \neq \emptyset$

8. $E = A_{B-inf}(X) = \{e_1, e_2\}$

9. $(a = 0) \Rightarrow (d = 0)$ (consistente)

10. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\} - [(\{e_1, e_2, e_3, e_4, e_5, e_6\} - \{e_1, e_2\}) \cup \{e_1, e_2\}] = \emptyset$, $X = \{e_1, e_2\} - \{e_1, e_2\} = \emptyset$

11. $U = \emptyset$, ir para o passo 16

16. $j = 1 + 1 = 2$

17. $j \leq n$ $(2 \leq 4)$, voltar para o passo 3

3. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $C' = \{a, b, c\}$, $B = \emptyset$ e $X = \{e_3, e_5\}$

4. Conjuntos elementares:

$\{a\}$: $E = \{\{e_1, e_2\}, \{e_3, e_4, e_5, e_6\}\}$
$\{b\}$: $E = \{\{e_1\}, \{e_2, e_3, e_5\}, \{e_4, e_6\}\}$
$\{c\}$: $E = \{\{e_1, e_2, e_3, e_4, e_6\}, \{e_5\}\}$

A Tabela 5.3 exibe o valor do índice discriminante para $X = \{e_3, e_5\}$.

Tabela 5.3: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_3, e_5\}$ e $B = \emptyset$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{a\}$	$\frac{6-4}{6} = \frac{2}{6} \cong 0,34$
$\{b\}$	$\frac{6-3}{6} = \frac{3}{6} = 0,5$
$\{c\}$	$\frac{6-5}{6} = \frac{1}{6} \cong 0,17$

5. $B' = B \cup \{b\}$

6. $B = B' = \{b\}$

7. $A_{B-inf}(X) = \emptyset$, ir para o passo 10

10. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\} - [(\{e_1, e_2, e_3, e_4, e_5, e_6\} - \{e_2, e_3, e_5\}) \cup \emptyset] = \{e_2, e_3, e_5\}$, $X = \{e_3, e_5\} - \emptyset = \{e_3, e_5\}$

11. $U \neq \emptyset$

12. $C' = \{a, b, c\} - \{b\} = \{a, c\}$

13. $C' \neq \emptyset$, voltar para o passo 4

4. Conjuntos elementares:

$\{b, a\}$: $E = \{\{e_2\}, \{e_3, e_5\}\}$
$\{b, c\}$: $E = \{\{e_2, e_3\}, \{e_5\}\}$

A Tabela 5.4 exibe o valor do índice discriminante para $X = \{e_3, e_5\}$.

Tabela 5.4: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_3, e_5\}$ e $B = \{b\}$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{b, a\}$	$\frac{3-0}{3} = \frac{3}{3} = 1$
$\{b, c\}$	$\frac{3-2}{3} = \frac{1}{3} \cong 0,34$

5. $B' = B \cup \{a\} = \{b, a\}$

6. $B = B' = \{b, a\}$

7. $A_{B-inf}(X) = \{e_3, e_5\} \neq \emptyset$

8. $E = A_{B-inf}(X) = \{e_3, e_5\}$

9. $(b = 2)$ & $(a = 1) \Rightarrow (d = 2)$ (consistente)

10. $U' = \{e_2, e_3, e_5\} - [(\{e_2, e_3, e_5\} - \{e_3, e_5\}) \cup \{e_3, e_5\}] = \emptyset$, $X = \{e_3, e_5\} - \{e_3, e_5\} = \emptyset$

11. $U = \emptyset$, ir para o passo 16

16. $j = 2 + 1 = 3$

17. $j \leq n$ $(3 \leq 4)$, voltar para o passo 3

3. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $C' = \{a, b, c\}$, $B = \emptyset$ e $X = \{e_4\}$

4. Conjuntos elementares:

$\{a\}$: $E = \{\{e_1, e_2\}, \{e_3, e_4, e_5, e_6\}\}$
$\{b\}$: $E = \{\{e_1\}, \{e_2, e_3, e_5\}, \{e_4, e_6\}\}$
$\{c\}$: $E = \{\{e_1, e_2, e_3, e_4, e_6\}, \{e_5\}\}$

A Tabela 5.5 exibe o valor do índice discriminante para $X = \{e_4\}$.

5. $B' = B \cup \{b\}$

6. $B = B' = \{b\}$

7. $A_{B-inf}(X) = \emptyset$, ir para o passo 10

10. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\} - [(\{e_1, e_2, e_3, e_4, e_5, e_6\} - \{e_4, e_6\}) \cup \emptyset] = \{e_4, e_6\}$, $X = \{e_4\} - \emptyset = \{e_4\}$

11. $U \neq \emptyset$

Tabela 5.5: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_4\}$ e $B = \emptyset$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{a\}$	$\frac{6-4}{6} = \frac{2}{6} \cong 0,34$
$\{b\}$	$\frac{6-2}{6} = \frac{4}{6} \cong 0,67$
$\{c\}$	$\frac{6-5}{6} = \frac{1}{6} \cong 0,17$

12. $C' = \{a, b, c\} - \{b\} = \{a, c\}$

13. $C' \neq \emptyset$, voltar para o passo 4

4. Conjuntos elementares:

$\{b, a\}$: $E = \{\{e_4, e_6\}\}$
$\{b, c\}$: $E = \{\{e_4, e_6\}\}$

A Tabela 5.6 exibe o valor do índice discriminante para $X = \{e_4\}$.

5. $B' = B \cup \{a\} = \{b, a\}$

6. $B = B' = \{b, a\}$

7. $A_{B-inf}(X) = \emptyset$, ir para o passo 10

10. $U' = \{e_4, e_6\} - [(\{e_4, e_6\} - \{e_4, e_6\}) \cup \emptyset] = \{e_4, e_6\}$, $X = \{e_4\} - \emptyset = \{e_4\}$

11. $U \neq \emptyset$

Tabela 5.6: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_4\}$ e $B = \{b\}$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{b, a\}$	$\frac{2-2}{2} = \frac{0}{2} = 0$
$\{b, c\}$	$\frac{2-2}{2} = \frac{0}{2} = 0$

12. $C' = \{a, c\} - \{b, a\} = \{c\}$

13. $C' \neq \emptyset$, voltar para o passo 4

4. Conjuntos elementares:

$\{b, a, c\}$: $E = \{\{e_4, e_6\}\}$

A Tabela 5.7 exibe o valor do índice discriminante para $X = \{e_4\}$.

Tabela 5.7: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_4\}$ e $B = \{b, a, c\}$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{b, a, c\}$	$\frac{2-2}{2} = \frac{0}{2} = 0$

5. $B' = B \cup \{c\} = \{b, a, c\}$

6. $B = B' = \{b, a, c\}$

7. $A_{B-inf}(X) = \emptyset$, ir para o passo 10

10. $U' = \{e_4, e_6\} - [(\{e_4, e_6\} - \{e_4, e_6\}) \cup \emptyset] = \{e_4, e_6\}$, $X = \{e_4\} - \emptyset = \{e_4\}$

11. $U \neq \emptyset$

12. $C' = \{c\} - \{b, a, c\} = \emptyset$

13. $C' = \emptyset$

14. $E = \{e_4\}$

15. $(b = 0)$ & $(a = 1)$ & $(c = 1) \Rightarrow (d = 1)$ (inconsistente)

16. $j = 3 + 1 = 4$

17. $j \leq n$ $(4 \leq 4)$, voltar para o passo 3

3. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $C' = \{a, b, c\}$, $B = \emptyset$ e $X = \{e_6\}$

4. Conjuntos elementares:

$\{a\}$: $E = \{\{e_1, e_2\}, \{e_3, e_4, e_5, e_6\}\}$
$\{b\}$: $E = \{\{e_1\}, \{e_2, e_3, e_5\}, \{e_4, e_6\}\}$
$\{c\}$: $E = \{\{e_1, e_2, e_3, e_4, e_6\}, \{e_5\}\}$

A Tabela 5.8 exibe o valor do índice discriminante para $X = \{e_6\}$.

5. $B' = B \cup \{b\}$

6. $B = B' = \{b\}$

7. $A_{B-inf}(X) = \emptyset$, ir para o passo 10

10. $U' = \{e_1, e_2, e_3, e_4, e_5, e_6\} - [(\{e_1, e_2, e_3, e_4, e_5, e_6\} - \{e_4, e_6\}) \cup \emptyset] = \{e_4, e_6\}$, $X = \{e_6\} - \emptyset = \{e_6\}$

Tabela 5.8: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_6\}$ e $B = \emptyset$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{a\}$	$\frac{6-4}{6} = \frac{2}{6} \cong 0,34$
$\{b\}$	$\frac{6-2}{6} = \frac{4}{6} \cong 0,67$
$\{c\}$	$\frac{6-5}{6} = \frac{1}{6} \cong 0,17$

11. $U \neq \emptyset$

12. $C' = \{a, b, c\} - \{b\} = \{a, c\}$

13. $C' \neq \emptyset$, voltar para o passo 4

4. Conjuntos elementares:

$\{b, a\}$: $E = \{\{e_4, e_6\}\}$
$\{b, c\}$: $E = \{\{e_4, e_6\}\}$

A Tabela 5.9 exibe o valor do índice discriminante para $X = \{e_6\}$.

5. $B' = B \cup \{a\} = \{b, a\}$

6. $B = B' = \{b, a\}$

7. $A_{B-inf}(X) = \emptyset$, ir para o passo 10

10. $U' = \{e_4, e_6\} - [(\{e_4, e_6\} - \{e_4, e_6\}) \cup \emptyset] = \{e_4, e_6\}$, $X = \{e_6\} - \emptyset = \{e_6\}$

Tabela 5.9: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_6\}$ e $B = \{b\}$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{b, a\}$	$\frac{2-2}{2} = \frac{0}{2} = 0$
$\{b, c\}$	$\frac{2-2}{2} = \frac{0}{2} = 0$

11. $U \neq \emptyset$

12. $C' = \{a, c\} - \{b, a\} = \{c\}$

13. $C' \neq \emptyset$, voltar para o passo 4

4. Conjuntos elementares:

$\{b, a, c\}$: $E = \{\{e_4, e_6\}\}$

A Tabela 5.10 exibe o valor do índice discriminante para $X = \{e_6\}$.

Tabela 5.10: Cálculo de $\alpha_{B'}(X)$ para $X = \{e_6\}$ e $B = \{b, a\}$

$B \cup \{c\}$	$\alpha_{B'}(X)$
$\{b, a, c\}$	$\frac{2-2}{2} = \frac{0}{2} = 0$

5. $B' = B \cup \{c\} = \{b, a, c\}$

6. $B = B' = \{b, a, c\}$

7. $A_{B-inf}(X) = \emptyset$, ir para o passo 10

10. $U' = \{e_4, e_6\} - [(\{e_4, e_6\} - \{e_4, e_6\}) \cup \emptyset] = \{e_4, e_6\}$, $X = \{e_6\} - \emptyset = \{e_6\}$

11. $U \neq \emptyset$

12. $C' = \{c\} - \{b, a, c\} = \emptyset$

13. $C' = \emptyset$

14. $E = \{e_6\}$

15. $(b = 0)$ & $(a = 1)$ & $(c = 1) \Rightarrow (d = 3)$ (inconsistente)

16. $j = 4 + 1 = 5$

17. $j > n$ $(5 > 4)$

18. Regras geradas pelo algoritmo RS1:

- $(a = 0) \Rightarrow (d = 0)$ (consistente)
- $(b = 2)$ & $(a = 1) \Rightarrow (d = 2)$ (consistente)
- $(b = 0)$ & $(a = 1)$ & $(c = 1) \Rightarrow (d = 1)$ (inconsistente)
- $(b = 0)$ & $(a = 1)$ & $(c = 1) \Rightarrow (d = 3)$ (inconsistente)

Conforme pôde ser observado no exemplo anterior, durante a geração das regras o algoritmo RS1 permite ordenar os elementos do antecedente da regra de acordo com o valor do índice discriminante. Com isso os atributos que forem mais significativos para a determinação da classificação serão checados primeiro.

No algoritmo RS1+ cada regra consistente gerada possue um fator de credibilidade igual a 1. Já o fator de credibilidade das regras inconsistentes é dado pela função de pertinência aproximada. No exemplo anterior, as regras inconsistentes

geradas receberiam um fator de credibilidade igual a 0,5. Sendo este fator de credibilidade de regras um importante mecanismo de avaliação das regras produzidas.

Na geração de regras inconsistentes, conforme observado no exemplo anterior, o algoritmo continua adicionando atributos ao conjunto B', mesmo quando não ocorre uma maior discriminação do conjunto U. Em outras palavras, o algoritmo continua adicionando atributos mesmo quando o conjunto U não sofre mais redução. Quando é encontrado um índice discriminante igual a 0 significa que não existe mais nenhum atributo que poderia discenir entre as regras em questão (por serem inconsistentes). No algoritmo RS1+, quando não ocorre uma redução no conjunto U, o algoritmo automaticamente classifica e gera a regra como inconsistente. Dessa forma o algoritmo não realiza passos desnecessários, o que melhora o desempenho do algoritmo.

Na execução do algoritmo muitas vezes as regras consistentes podem ser geradas com atributos desnecessários. Por esta razão, foi introduzido no algoritmo RS1+, no passo 9, um método para simplificação dessas regras. Isto é feito da seguinte forma: para todo elemento $(a_k = v_{ak})$ do antecedente da regra, deve-se verificar se sua eliminação não aumenta o número de regras inconsistentes. Caso isto não ocorra, então deve-se eliminar $(a_k = v_{ak})$ do antecedente e continuar o processo de eliminação de atributos desnecessários.

O pseudocódigo do algoritmo RS1+ e maiores detalhes destas melhorias sugeridas por Uchôa podem ser encontrados em [34].

Capítulo 6

Algoritmo Paralelo

6.1 Introdução

Como já observado por Uchôa, em [34], o processo de geração de regras do algoritmo RS1 é facilmente paralelizável. Isto se deve ao fato de ser possível se fazer chamadas independentes para cada elemento de $Class(\delta)$, onde δ corresponde ao atributo de decisão.

Em uma análise do algoritmo é possível perceber que este é o ponto mais crítico do problema de geração de regras. O algoritmo começa sua execução gerando as regras para o primeiro conjunto de $Class(\delta)$. Após isso o algoritmo reinicia o mesmo processo para o próximo conjunto de $Class(\delta)$, e assim sucessivamente para todos os conjuntos. Dessa forma, cada conjunto de $Class(\delta)$ pode ser atribuído a um processador diferente. Todos os processadores irão executar simultaneamente os mesmos passos sobre um conjunto diferente de $Class(\delta)$, produzindo as regras em um tempo bem menor.

A geração de regras de decisão a partir de uma tabela de decisão é um problema considerado *NP-Hard* conforme descrito em [36].

6.2 Descrição do algoritmo

O algoritmo paralelo criado neste trabalho será denominado PRS1+ (*Parallel RS1+*), por ser este baseado no algoritmo RS1+. Para uma dada tabela de decisão $S = (U, C \cup \{\delta\}, V, p)$, a seguir são descritos os principais passos do algoritmo paralelo PRS1+:

Calcular $Class_S(\delta) = \{X_1, X_2, \ldots, X_m\}$, a família de conjuntos elementares do espaço aproximado induzido por $\{\delta\}$. Sendo $Class_w(\delta) = \{X_1, X_2, \ldots, X_n\}$, onde $1 \leq n \leq m$ e t igual ao número máximo de processadores, tem-se que $Class_S(\delta) = \bigcup_{1 \leq w \leq t} Class_w(\delta)$.

Para todo P_k, onde $1 \leq k \leq t$, faça

1. fazer $j = 1$
2. fazer $U' = U$, $C' = C$, $B = \emptyset$, $S' = (U', C \cup \{\delta\}, V, p)$ e $X = X_j$, onde $X_j \in Class_w(\delta)$
3. calcular o conjunto de índices discriminantes $\{\alpha_{B'}(X) | B' = B \cup \{c\}, \forall c \cup C'\}$ em S'
4. selecionar o conjunto de atributos $B' = B \cup \{c\}$ com o maior valor $\alpha_{B'}(X)$
5. fazer $B = B'$
6. se $A_{B-inf}(X) = \emptyset$, ir para o passo 9
7. identificar os conjuntos elementares $E = \{E_1, E_2, \ldots, E_r\}$ do espaço aproximado induzido por B que estão contidos em $A_{B-inf}(X) = \emptyset$
8. para cada elemento $E_k \in E$, gerar uma regra de decisão determinística (consistente). Considerando que B possui m atributos, então cada regra tem a forma $(a_1 = v_{a1})$ & $(a_2 = v_{a2})$ & $\ldots$ & $(a_m = v_{am}) \Rightarrow (b = v_b)$, onde $a_1, a_2, \ldots, a_m \in B$. Cada elemento $(a_k = v_{ak})$ do antecedente é construído da seguinte forma: a_k recebe o nome do k-ésimo atributo de B e v_ak recebe o valor atribuido a E_k por esse atributo. O conseqüente $(b = v_b)$ é construído de forma semelhante: b recebe o nome do atributo de decisão δ e v_b recebe o valor atribuído a E_k por esse atributo
9. fazer $U' = U' - [(U' - A_{B-sup}(X)) \cup A_{B-inf}(X)]$, $X = X - A_{B-inf}(X)$ e $S' = (U', C \cup \{\delta\}, V, p)$
10. se $U' = \emptyset$ ir para o passo 15

11. fazer $C' = C' - B$
12. se $C' \neq \emptyset$, voltar ao passo 3
13. identificar todos conjuntos elementares $E = \{E_1, E_2, \ldots, E_r\}$ do espaço aproximado induzido por B
14. para cada elemento $E_k \in E$, gerar uma regra de decisão não-determinística (inconsistente) de forma semelhante à descrita no passo 8
15. fazer $j = j + 1$
16. se $j \leq n$ voltar ao passo 2

Fim do para

Devolver todas as regras encontradas nos passos 8 e 14

6.3 Implementação do algoritmo

O trabalho foi implementado no ambiente LAM/MPI utilizando-se a linguagem C++, estando disponível para o sistema operacional Linux.

Foi utilizado na implementação o paradigma mestre/escravo, onde um processador (chamado de mestre) é o responsável por calcular o conjunto $Class_S(\delta)$ e distribuir seus elementos e outros dados necessários para os outros processadores (chamados de escravos). Cada escravo por sua vez gera suas regras de decisão a partir de seu conjunto $Class_w(\delta)$ local e as retorna para o mestre.

O liguagem MPI é bastante simples, sendo que com apenas algumas funções básicas é possível escrever um vasto número de programas paralelos. No desenvolvimento do algoritmo PRS1+ foram utilizadas basicamente as funções:

- *MPI::Init*: para inicialização de processos;
- *MPI::Finalize*: para finalização de processos;
- *MPI::Get_rank*: usada na identificação do processo;
- *MPI::Get_size*: que retorna o número de processos que foram inicializados;

- *MPI::Send*: para envio de mensagens na comunicação entre dois processos;
- *MPI::Recv*: para recebimento de mensagens na comunicação entre dois processos;
- *MPI::Bcast*: para *broadcast* (difusão) de mensagens, ou seja, para o envio de mensagens para todos os processadores;
- *MPI::Pack*: utilizada no empacotamento de dados no envio de mensagens;
- *MPI::Unpack*: para desempacotamento de dados no recebimento de mensagens empacotadas.

Além dessas, existem várias funções avançadas que podem ser utilizadas, quando for necessário, na criação de programas paralelos em MPI. Dentre estas destacam-se as funções *MPI::Scatter* e *MPI::Gather*, que são utilizadas na distribuição e coleta de dados entre os processadores. A função *MPI::Scatter* pode ser utilizada, por exemplo, para distribuir uma base de dados existente em um processador entre todos processadores de um grupo. A função *MPI::Gather*, por outro lado, pode ser utilizada para coletar e agrupar dados distribuídos entre processadores de um grupo em um único processador.

6.4 Análise de Desempenho

A análise de desempenho do algoritmo PRS1+ foi feita a partir dos resultados encontrados na realização de alguns testes executados sobre o ambiente MPI. Os testes foram realizados na rede de computadores do Departamento de Ciência da Computação da Universidade Federal de Lavras (UFLA - MG). Foram utilizados 17 computadores heterogêneos para os testes, os quais possuem a seguinte descrição:

- 7 computadores Celeron 300 MHz, 128 Mb RAM, 7 Gb HD;
- 10 computadores Pentium 500 MHz, 128 Mb RAM, 4 Gb HD, com placa de rede *on-board*.

Nos testes foram usados dois arquivos de dados de domínio público, disponíveis em um respositório de bases de dados para aprendizado de máquina [17]. Segue abaixo a descrição de cada um desses arquivos:

- *Mushroom*: consiste de um conjunto de dados com descrições hipotéticas de 23 espécies de cogumelos das famílias *Agaricus* e *Lepiota*. A descrição de cada instância utiliza-se de 22 atributos de descrição, podendo os cogumelos serem classificados em duas classes: "coméstivel" ou "venenoso". Foram utilizados nos testes um conjunto de 5.936 elementos, sendo que o conjunto original possui 8.124 elementos. Este subconjunto de elementos foi escolhido pois os elementos restantes possuem atributos com valores nulos, que não são suportados pelo algoritmo e que podem ser tratados através de pré-processamento;

- *Letter-recognition*: consiste de um banco de dados real para reconhecimento de caracteres por *scanners*. Os 20.000 elementos dessa base de dados são descritos através de 16 atributos, e podem ser classificados em uma das 26 letras do alfabeto. Na geração dessa base de dados foram utilizados 20 diferentes estilos de fontes.

6.5 Resultado dos testes

Neste trabalho não serão analisadas as regras geradas nos testes, pois em [34] já encontram-se avaliadas as regras produzidas pelo algoritmo RS1+. As regras geradas pelo algoritmo PRS1+ foram aferidas com o resultado encontrado pelo algoritmo RS1+, sendo que neste trabalho o interesse recai somente sobre o desempenho do algoritmo paralelo PRS1+.

Para a análise de desempenho inicialmente foi executado o algoritmo seqüencial RS1+ em uma das máquinas do ambiente MPI. A Tabela 6.1 mostra o resultado alcançado por este algoritmo.

Tabela 6.1: Resultado da execução do algoritmo RS1+

Base de Dados	Regras Geradas	Tempo (segundos)
$Mushroom$	6	41
$Letter - Recognition$	8161	1131

O primeiro teste do algoritmo PRS1+ foi executado utilizando-se 1 processador e a cada novo teste foi dobrado o número de processadores até ter sido atingido o máximo de 16 processadores. Além disso, cada teste possuia um processador adicional que correspondia ao processador mestre.

A Tabela 6.2 e Figura 6.1 exibem os resultados encontrados para a base de dados *Mushroom*.

Tabela 6.2: Resultado da execução do algoritmo PRS1+ para a base de dados *Mushroom*

Número de Processadores	Regras Geradas	Tempo (segundos)
1	6	204
2	6	198
4	6	199
8	6	200
16	6	202

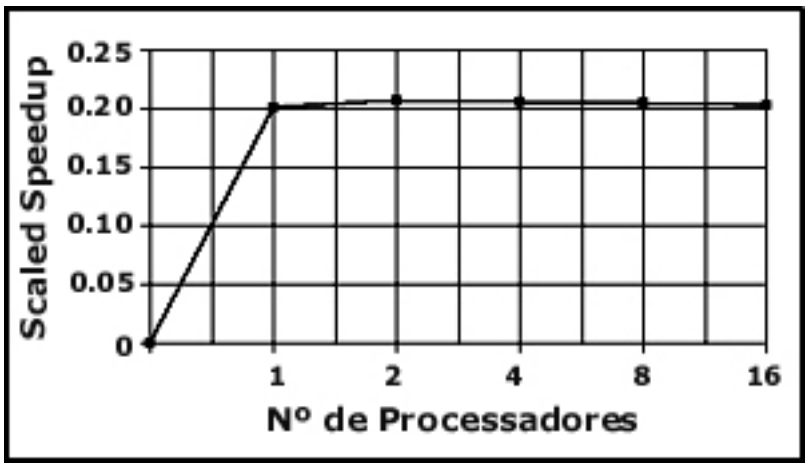

Figura 6.1: *Scaled Speedup* alcançado para a base de dados *Mushroom*

O máximo *Speedup* do algoritmo paralelo para a base de dados *Mushroom* foi alcançado com 2 processadores. Entretanto, em todos os testes o algoritmo seqüencial foi mais rápido que o algoritmo paralelo. Isso ocorreu por causa da base *Mushroom* ser "pequena", o que faz com que o algoritmo paralelo não seja viável neste caso.

Nos testes com a base de dados *Mushroom*, pode-se observar que ocorreu queda de desempenho do algoritmo paralelo após 2 processadores. A base de dados *Mushroom* possui apenas duas possíveis classificações: "comestível" ou "venenoso", ou seja, o conjunto $Class_S(\delta)$ para esta base possui apenas dois subconjuntos. No algoritmo paralelo estes dois subconjuntos são distribuídos para no

máximo dois processadores. Isto significa que o acréscimo de mais processadores diminui a eficiência do algoritmo paralelo, sendo isto justificado pelas razões que causam queda de *Speedup*, como discutido na Seção 2.5. Quando são utilizados mais do que 2 processadores, tem-se que apenas 2 processadores estão realmente executando a computação. Com isso mais tempo é adicionado para se inicializar e finalizar os processos nos processadores ociosos, além da comunicação necessária para sincronização desses processos.

Na Tabela 6.3 e Figura 6.2 podem ser visualizados os resultados obtidos nos testes para a base de dados *Letter-Recognition*.

Tabela 6.3: Resultado da execução do algoritmo PRS1+ para a base de dados *Letter-Recognition*

Número de Processadores	Regras Geradas	Tempo (segundos)
1	8161	4147
2	8161	2096
4	8161	1181
8	8161	692
16	8161	615

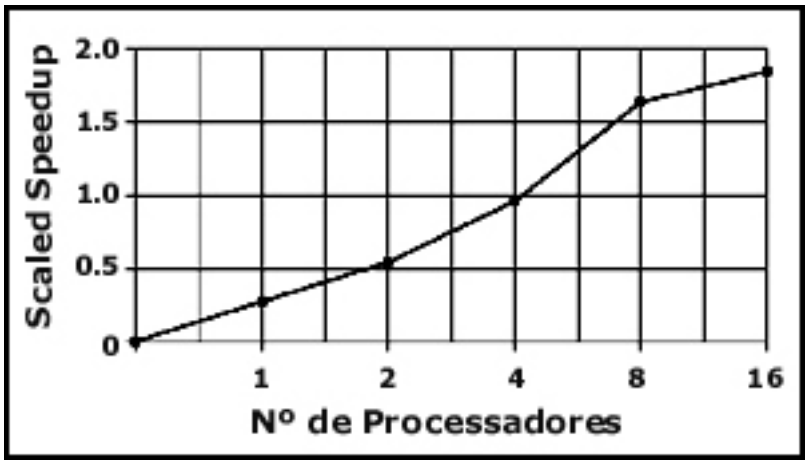

Figura 6.2: *Scaled Speedup* alcançado para a base de dados *Letter-Recognition*

O máximo *Speedup* para a base *Letter-Recognition*, foi alcançado utilizando-se o número máximo de processadores. Contudo pode-se perceber que o desempenho do algoritmo paralelo começou a diminuir ao se utilizar 16 processadores. Até 08

processadores, enquanto o número de processadores era aumentado a uma taxa de 2, o tempo de execução era reduzido a uma taxa superior a 1,7. Após isso, o tempo de execução do algoritmo paralelo foi reduzido a uma taxa de aproximadamente 1,125. Isso ocorre devido ao aumento excessivo de comunicação entre os processadores, o que compromete o tempo total de execução do algoritmo paralelo, como discutido na Seção 2.5.

Capítulo 7

Conclusões

A Teoria de Conjuntos Aproximados é um poderoso formalismo matemático para tratamento da indiscernibilidade. Ela supera outros métodos de tratamento de incerteza por não necessitar de nenhuma informação adicional dos dados.

O algoritmo PRS1+ se mostrou bastante eficiente na geração de regras de produção a partir de uma tabela de decisão. Entretanto para o bom desempenho dos métodos baseados em TCA é necessário que os dados da tabela de decisão não sejam muito dispersos. Caso os dados de origem sejam muito distintos entre si, o método de aprendizado de máquina não terá bom desempenho pois seu mecanismo de funcionamento baseia-se na semelhança entre objetos. Em seu pior caso o algoritmo irá gerar uma regra de produção para cada objeto da tabela de decisão. O algoritmo se comportará assim por não conseguir agrupar objetos com valores de atributos semelhantes.

Como a geração de regras a partir de uma tabela de decisão é um problema considerado *NP-Hard*, não é viável a utilização do algoritmo paralelo PRS1+ em problemas com pequenas bases de dados, pois seu desempenho pode se tornar inferior ao desempenho do algoritmo seqüencial. O algoritmo paralelo obteve um execelente ganho ao ser utilizado em um problema com base de dados possuíndo 20.000 elementos. Por outro lado, para um problema com base de dados com 5.936 elementos, o algoritmo seqüencial se mostrou mais rápido. O resultado desses testes são descritos na Seção 6.5 .

É aconselhável que o número de processadores utilizados na execução do algoritmo seja menor ou igual ao número de classificações da base de dados, para que o algoritmo não tenha seu desempenho comprometido. Ao se utilizar muitos processadores em máquina paralela sobre ambientes distribuídos também pode

ocorrer queda de *Speedup* por causa da quantidade excessiva de comunicação entre as máquinas.

O ambiente MPI rodando sobre uma rede heterogênea de computadores teve resultados satisfatórios na execução de algoritmos paralelos. A linguagem MPI é bastante simples e pode ser utilizada em conjunto com as linguagens C, C++ e Fortran. Algoritmos PRAM são facilmente implementados utilizando-se a linguagem MPI.

Como trabalhos futuros sugere-se:

- fazer uma análise de complexidade do algoritmo PRS1+ através de análise estatística da distribuição dos dados, pois o algoritmo possui comportamento dependente da natureza dos dados;
- utilizar funções mais avançadas da linguagem MPI na implementação do algoritmo PRS1+ como as funções *MPI::Scatter* e *MPI::Gather* citadas na Seção 6.5, o que teoricamente poderia melhorar o desempenho do algoritmo;
- implementar do algoritmo PRS1+ utilizando-se outras liguagens paralelas para ambientes distribuídos, principalmente linguagens que aceitem a transferência de objetos na comunicação entre processos, o que teoricamente também poderia melhorar o desempenho do algoritmo;
- paralelizar outros pontos críticos do algoritmo RS1+, como por exemplo, a geração dos conjuntos elementares do espaço aproximado.

Referências Bibliográficas

[1] AASHEIM, O.T.; SOLHEIM, H.G. *Rough Set as a Framework for Data Mining*. Project Report of Knowledge Systems Group - Faculty of Computer Systems and Telematics. Trondheim, Norwegian University of Science and Technology, 1996.

[2] ALMASI, G.S.; GOTTLIEB A. *Highly Parallel Computing*, 2. ed. The Benjamin Cummings Publishing Company, 1994.

[3] AMDAHL, G. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In: *AFIPS Conference Proceedings*, Thompson Books, v.30, April, 1967, p.483-485.

[4] BEGUELIN, A. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1994.

[5] BITTENCOURT, G. *Inteligência Artificial: ferramentas e teorias*. Florianópolis: Ed. da UFSC, 1998, 362p.

[6] BOOSE, J.H. A knowledge acquisition: techniques and tools. In: *Knowledge Aquisition*, (1):3-37, 1989.

[7] CLARK, P.; NIBLETT, T.; The CN2 induction algorithm. *Machine Learning Journal*, 3(4):261-283, 1989.

[8] CENAPAD NE. *Apostila do Workshop MPI*. 72p. Apostila. fevereiro, 1998.

[9] COHEN, P.R. *Heuristic reasoning about uncertainty: an Al approach*. Boston: Pitman, 1985.

[10] DUNCAN, R., A Survey of Parallel Computer Architectures. *IEEE Computer*, p.5-16, Fevereiro, 1990.

[11] DUDA, R.O.; HART, P.E; NILSSON, N.J. Subjective bayesian methods for rule-based inference systems. In: *AFIPS Conference Proceedings*. New York, June, 1976, p.1075-1082.

[12] GOODMAN, S.E.; HEDETNIEMI, S.T. *Introduction to the Design and Analysis of Algorithms*. McGraw-Hill, 1977, 265p.

[13] GUSTAFSON, J.L. *Reevaluating Amdahl's Law. Communications of the ACM*, v.21, n.5, May, 1988, p.532-533.

[14] GUSTAFSON, J.L.; MONTRY, G.R; BENNE, R.E. *Development of parallel Methods for a 1024-processor Hypercube. SIAM Journal on Scientific and Statistical Computing*, v.9, n.4, July, 1988, p.609-638.

[15] HU, X.; CERCONE, N.; HAN, J. An Attribute-oriented Rough Set Aproarch for Knowledge Discovery in Databases. In: Ziarko, W.P. (Ed.): *Rough Sets, Fuzzy Sets and Knowledge Discovery*. Springer-Verlag, p.90-99, 1994.

[16] LUMSDAINE, A.; SQUYRES, J.; BARRETT, B. *LAM/MPI Parallel Computing*. Bloomington, Indiana University, Laboratory for Scientific Computing (LSC), 1996. [http://www.lam-mpi.org/].

[17] MERZ, C.J; MURPHY, P.M. *UCI repository of Machine Learning databases*. Irvine, University of California, Department of Information and Computer Science, 1998. [http://www.ics.uci.edu/ mlearn/MLRepository.html].

[18] MICHALSKI, R.S. A theory and methodology of inductive learning. In: MICHALSKI, R.S.; CARBONELL, J. G.; MITCHELL, T.M. *Machine learning: an Artificial Intelligence approach*. Palo Alto: Tioga, 1983. p.83-134.

[19] MRÓZEK, A. *A New Method for Discovering Rules from Examples in Expert Systems*. Int. Journal of Man-Machine Studies 36, p.127-143, 1992.

[20] NICOLETTI, M.C. *Ampliando os limites do aprendizado indutivo de máquina através das abordagens construtiva e relacional*. São Carlos: USP, 1994. (Tese de Doutorado).

[21] PAWLAK, Z. Information systems. *Information system: theoretical foundations*, 6(3):205-218, 1981.

[22] PAWLAK, Z. Rough sets. *International Journal of Computer and Information Sciences*, 11(5):341-356, 1982.

[23] PAWLAK, Z. *Rough sets: theoretical aspects of reasoning about data*. London: Kluwer, 1991.

[24] PAWLAK, Z. Hard and soft sets. In: Ziarko, W.P. (Ed.). *Rough sets, fuzzy sets and knowledge discovery*. London: Springer-Verlag, p.130-135, 1994.

[25] PEARL, J. Reverend Bayes on inference engines: a distributed hierarchical approach. In: *Proceedings of the Second National Conference on Artificial Intelligence*. Pittsburgh, PA, 1982, p.133-136.

[26] QUINLAN, J.R. Induction of decision trees. *Machine learning*, (1):81-106, 1986.

[27] QUINN, M.J. *Parallel Computing: Theory and Practice*. McGraw Hill, 1994.

[28] RICH, E.; KNIGHT, K. *Artificial Inteligence*. New York: McGraw-Hill, 1991.

[29] REITER, R. A Logic for Default Reasoning. *Artificial Intelligence*, (13):81-132, 1980.

[30] SANTANA, R. H. C.; SOUSA, M. A. de; PIEKARSKI, A. E. T. et al. *Computação Paralela*. São Carlos: USP, 1997. 61p. Apostila.

[31] SHAFER, G. *A mathemathical theory of evidence*. Princeton: Princeton University Press, 1976.

[32] SHORTLIFFE, E.H.; BUCHANAN, B.G. A Model of inexact reasoning in Medicine. *Math. Biosci*, v.23, p.351-379, 1975.

[33] SZLADOW, A.; ZIARKO, W. *Rough Sets: Working with Imperfect Data*. AI Expert, July, p.36-41, 1993.

[34] UCHÔA, J.Q. *Representação e Indução de Conhecimento Usando Teoria de Conjuntos Aproximados*. São Carlos: UFSC, 1998, 237p. (Dissertação - Mestrado em Ciência da Computação).

[35] WALKER, D. W. The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, v.20, p.657-673, 1994.

[36] WANG, S.K.M.; ZIARKO, W. On optimal decision rules in decision tables. *Bulletin of Polish Academy os Sciences,* 33(11-12):693-696, 1985.

[37] WONG, S.K.M.; ZIARKO, W.; YE, R. L. Comparison of rough-set and statistical methods in inductive learning. *Internacional Journal of Man-Machine Studies*, (24):53-72, 1986.

[38] ZIARKO, W. The Discovery, Analysis, and Representation of Data Dependencies in Databases. In: Piatestsky-Shapiro, G.; Frawley, W. (Eds.): *Knowledge Discovery in Databases*. AAI Press/MIT Press, p.195-209, 1991.

[39] ZADEH, L.A. Fuzzy Sets as a Basis for a Theory of Possibility. *Fuzzy Sets and Systems*, (1):3-28, 1978.