Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Curso de Pós-Graduação em Informática

Reconhecimento de Caracteres Manuscritos Utilizando Regras de Associação

Juliano Varella de Carvalho

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba – Campus II, como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Marcus Costa Sampaio
Orientador

Giuseppe Mongiovi
Orientador

Campina Grande – Paraíba - Brasil
Agosto – 2000

C331R

CARVALHO, Juliano Varella de

Reconhecimento de Caracteres Manuscritos Utilizando Regras de Associação.

Dissertação de Mestrado, Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande - Pb, Novembro de 2000.

117 p. Il.

Orientadores: - Marcus Costa Sampaio
Giuseppe Mongiovi

Palavras Chave:

1. Processamento de Imagens
2. Reconhecimento de Caracteres Manuscritos
3. Data Mining
4. Regras de Associação

CDU - 681.332.3

"RECONHECIMENTO DE CARACTERES MANUSCRITOS UTILIZANDO REGRAS DE ASSOCIAÇÃO"

JULIANO VARELLA DE CARVALHO

DISSERTAÇÃO APROVADA EM 30.08.2000

PROF. MARCUS COSTA SAMPAIO, Dr.
Orientador

PROF. GIUSEPPE MONGIOVI , D.Sc
Orientador

PROF. JOÃO MARQUES DE CARVALHO, D.Sc
Examinador

PROF. ALEX ALVES FREITAS, Ph.D
Examinador

CAMPINA GRANDE – PB

Dedicatória

Dedico este trabalho às pessoas que fazem a minha vida ter sentido.

Aos meus pais, Tabajara e Edite, por moldarem o amor
e o respeito de uma maneira tão simples e fazer destes
dois ingredientes o leme de nossas vidas.

Aos meus irmãos, Rodrigo e André Luiz, pelo
amor e união que nutrimos em nossos corações.

À minha esposa, Ligiane, por ter encontrado nela
o mais sublime amor, e saber que ela fará parte de
todos os momentos da minha vida.

Agradecimentos

Agradeço ao meu orientador, prof. Marcus Costa Sampaio, pela amizade que construímos durante estes dois anos e meio e por ter feito parte deste trabalho de maneira tão intensa, sempre auxiliando-me com novas idéias, incentivos e críticas. Agradeço também por todo seu apoio prestado durante a minha permanência em Campina Grande.

Agradeço ao meu outro orientador, prof. Giuseppe Mongiovi, pelas suas grandes contribuições ao trabalho, suas dicas e idéias tendo sido imprescindíveis à realização do mesmo.

Ao prof. João Marques de Carvalho e à doutoranda Luciana Veloso, do Departamento de Engenharia Elétrica da Universidade Federal da Paraíba, pela inestimável ajuda, sem a qual este trabalho demoraria muito para ficar pronto.

Aos grandes amigos que fiz durante o mestrado, tais como Adriano Barata, Angel, Carlinhos, Marinaldo, Gilene, André Rocha, Júlio Gomes, entre outros.

Aos amigos que deixo na Paraíba, mas que sempre estarão sendo lembrados por mim: Rogério, Jorge, Rodrigo Pereira, Rodrigo Souza, Carlos Pedro, etc.

À CAPES, por financiar parcialmente este trabalho.

Resumo

Apesar da crescente difusão das tecnologias de informação baseadas em mídias eletrônicas, ainda pode-se constatar a existência de uma quantidade incomensurável de informações 'em papel', como formulários, memorandos, cartas, requerimentos, etc. O processo de conversão destas formas de armazenamento para a mídia eletrônica tem sido pouco produtivo e custoso, pois envolve a participação de pessoas muitas vezes despreparadas. Esta dissertação apresenta os resultados de uma aplicação pioneira da tecnologia de "*data mining"* conhecida como *regras de associação*, para o reconhecimento automático de caracteres manuscritos.

Além de analisarmos os resultados obtidos com esta técnica, fazemos também uma comparação do nosso sistema de reconhecimento de caracteres manuscritos com sistemas que utilizam outras tecnologias, como análise sintática e redes neurais.

Abstract

In spite of the growing spread of information technologies based on electronic media, hand-printed information, such as forms, memoranda, letters, requirements, etc. is still abound. The conversion process of these stored forms to electronic media has been onerous and unproductive, mainly due to the involvement of unprepared personnel. This paper presents the results of a pioneer application of the data mining technology known as association rules for the automatic recognition of hand-printed characters.

Besides analysing the results obtained with this technique, we also compare our system for automatic recognition of hand-printed characters with systems that use other tecnologies, as syntactic analysis and neural nets.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo Um

Introdução

Apesar da crescente difusão das tecnologias de informação baseadas em mídias eletrônicas, ainda pode-se constatar a existência de uma quantidade incomensurável de informações 'em papel', como formulários, memorandos, cartas, requerimentos, etc. O processo de conversão destas formas de armazenamento para a mídia eletrônica tem sido pouco produtivo e custoso, pois envolve a participação de pessoas, muitas vezes pouco preparadas para o uso de software especializado, como os editores/processadores de texto. Uma solução para este problema seria dotar os computadores da capacidade de 'ler' documentos, desta forma tornando mais produtivo e menos oneroso o processo de alimentação dos bancos de dados eletrônicos.

O interesse pela área de reconhecimento de manuscritos não é recente, vindo desde a origem dos computadores [Veloso1998]. Atualmente, devido à evolução tecnológica, novas tecnologias com este fim estão surgindo. Congressos científicos, conferências, simpósios, "workshops", livros e outros meios de divulgação estão dando muita atenção ao processamento de manuscritos. Centros de pesquisa, tais como o CENPARMI (*Center for Pattern Recognition and Machine Intelligence*), CEDAR (*Center of Excellence for Document Analysis and Recognition*) e BELL (*AT&T Bell Laboratories*) vêm se aprofundando no estudo de novas tecnologias de reconhecimento de manuscritos.

Vários estudos têm sido realizados na área de reconhecimento de manuscritos, tais como reconhecimento de caracteres manuscritos chineses, arábicos, japoneses, ocidentais, entre outros ([Hanmandlu1999], [Yuccer1993]).

Além do reconhecimento de caracteres isolados, muitos trabalhos também procuram explorar o reconhecimento de palavras e frases manuscritas, as quais oferecem obstáculos ainda maiores para a obtenção de um reconhecimento eficaz [Grandidier1999].

O processamento de textos manuscritos não é uma tarefa trivial, e isto devido principalmente à diversidade de maneiras e formas com que os caracteres podem ser escritos: *pessoas diferentes escrevem diferente!* Para uma mesma pessoa, a diversidade da escrita também é um fato. Entre os fatores dessa diversidade, destacamos: o tipo de caneta e papel utilizado, habilidade, estilo, nível social, educação, origem e o estado psicológico da pessoa.

Por exemplo, uma pessoa com nível de escolaridade alto, ao escrever números, provavelmente o fará de maneira próxima da forma padrão destes dígitos. Ao contrário, uma pessoa com nível de escolaridade muito baixo, poderá ter dificuldade ao escrever números, criando dígitos bem diferentes das formas que conhecemos. O nível escolar é um fator que influencia a diversidade na escrita, a qual impõe dificuldades imensas aos sistemas de reconhecimento de caracteres manuscritos.

Nos últimos anos houve um avanço considerável nos experimentos com reconhecimento de caracteres manuscritos. O investimento nesta área tornou-se possível devido à combinação de vários fatores dentre os quais podemos citar: computadores mais baratos e mais poderosos, o desenvolvimento de novos algoritmos e a disponibilidade de bancos de caracteres manuscritos digitalizados, pois estes últimos até bem pouco tempo eram muito escassos [LeCun1995].

Existem duas principais áreas de pesquisa em reconhecimento de caracteres: “offline handwriting” e “online handwriting” [Guyon1996].

“Offline handwriting” é a área que estuda aqueles caracteres que são concebidos por um indivíduo, por meio de um lápis ou caneta sobre um papel, os quais posteriormente são transformados num formato digital através de um “scanner”. Os conjuntos de dados geralmente armazenam caracteres isolados que são escritos por um grupo escolhido de pessoas, formando os conjuntos de treinamento e teste.

“Online handwriting” é a área que procura fazer o reconhecimento dos caracteres à medida que eles vão sendo escritos, através de mesas de digitalização, por exemplo.

Além do reconhecimento de caracteres isolados, objeto de estudo neste trabalho, muitos centros de pesquisa aprofundam seus estudos também em reconhecimento de palavras e frases, que impõem mais obstáculos para um bom reconhecimento, pois nestes casos é preciso utilizar técnicas de segmentação, ou seja,

algoritmos que possam delimitar as sentenças, as palavras e os caracteres analisados [Guyon1996].

O nosso trabalho de reconhecimento de caracteres numéricos manuscritos é "offline handwriting", utilizando a base de dados do Cenparmi (doravante, base do Cenparmi), utilizada também em outras pesquisas ([Veloso1998], [Guyon1996]). A base contém numerais isolados e digitalizados de códigos postais de cartas.

1.1 Objetivos da dissertação

O objetivo principal de nossa pesquisa é a utilização de uma técnica de "*data mining", regras de associação*, para o desenvolvimento de um sistema de reconhecimento de caracteres manuscritos. Como estudo de caso, utilizamos a base de dados do Cenparmi, a qual possui cerca de 17.000 dígitos. Inicialmente, extraímos desta base um conjunto de dígitos (conjunto de treinamento), do qual são geradas regras de associação. Essas regras de associação definem padrões para cada tipo de dígito. Com estes padrões definidos, o restante da base (conjunto de teste) é utilizado para avaliar a capacidade do sistema de reconhecer dígitos.

O outro objetivo desta dissertação consiste na comparação quantitativa dos resultados (índices de acertos, rejeições e erros) da técnica que utilizamos com os resultados obtidos com outras técnicas de reconhecimento, mais precisamente algoritmos sintáticos e redes neurais.

1.2 Relevância da dissertação

Os computadores tornaram-se presentes no cotidiano das empresas e no dia a dia das pessoas, atuando efetivamente em diversas áreas do conhecimento humano, promovendo uma diminuição no tempo de realização das tarefas, bem como um aumento do desempenho das mesmas.

Um fator essencial à vulgarização dos computadores é a simplicidade da interação homem-máquina. A comunicação mais fácil e amigável entre o homem e a máquina é um assunto que vem sendo estudado há algum tempo, e devido ao avanço destes estudos foram desenvolvidos o mouse, canetas óticas, telas sensíveis, entre outros equipamentos. Como a escrita manual é uma das formas mais naturais de comunicação entre as pessoas, constata-se a geração de uma quantidade de dados em papel muito volumosa. Muitas vezes é necessário processar os dados contidos neste papéis por

máquinas. É extremamente desejável, então, permitir que os computadores tenham capacidade de 'ler' e interpretar documentos em papel.

O reconhecimento de caracteres manuscritos tem sido uma preocupação da comunidade científica. As aplicações para um sistema que faça tal tipo de reconhecimento são muitas, podendo citar leitoras automáticas de cheques bancários, máquinas automáticas de processamento de códigos postais para auxiliar as agências de correio, máquinas automáticas para processar todo e qualquer tipo de formulário preenchido manualmente, etc.

Embora já existam muitas pesquisas na área, melhores resultados precisam ser alcançados para tornar o reconhecimento de caracteres manuscritos viável. Além disso, as tecnologias atuais ainda apresentam alguns problemas na tarefa de reconhecimento, como veremos no decorrer desta dissertação. Portanto, ainda é satisfatório e justificável que novas tecnologias sejam investigadas com o propósito de acrescentar melhorias no trabalho de reconhecer caracteres manuscritos.

A relevância de nosso trabalho reside na exploração do potencial da técnica de data mining, conhecida como regras de associação, para o reconhecimento de caracteres manuscritos. Embora tenhamos utilizado uma base de dados de caracteres numéricos manuscritos em nosso trabalho, é importante salientar que o nosso sistema está capacitado a fazer o reconhecimento de caracteres manuscritos não numéricos sem qualquer alteração em sua estrutura.

Nesse sentido, o nosso trabalho trata-se de uma tarefa original, pois não encontramos, na literatura, este tipo de aplicação para regras de associação.

1.3 Metodologia de Trabalho

Durante o nosso trabalho, identificamos várias atividades. Dentre as principais, podemos citar:

✓ Atividade 1: pesquisa bibliográfica, compreendendo livros, artigos, revistas, anais de congressos e simpósios, internet e outros meios de comunicação, relacionados, essencialmente, com os seguintes temas:

- Reconhecimento de caracteres manuscritos;
- Preparação das imagens dos caracteres manuscritos;

- *Descoberta de conhecimento* e data mining;
- Regras de associação e regras de classificação;
- Algoritmos de data mining*;*
- Redes neurais;
- Utilização de redes neurais para o reconhecimento de manuscritos;
- Utilização de algoritmos sintáticos para o reconhecimento de caracteres manuscritos.

✓ Atividade 2: implementação de alguns algoritmos para preparar as imagens digitalizadas da base do Cenparmi, a fim de adaptá-los ao problema do reconhecimento de caracteres manuscritos;

✓ Atividade 3: estudo e utilização de alguns algoritmos de pré-processamento (preparação das imagens digitalizadas) existentes;

✓ Atividade 4: estudo e implementação de algoritmos de regras de associação, e suas adaptações ao problema do reconhecimento de caracteres manuscritos.

✓ Atividade 5: compreensão do funcionamento de sistemas de reconhecimento de manuscritos que se utilizam de algoritmos sintáticos e de redes neurais artificiais;

✓ Atividade 6: análise comparativa dos resultados do nosso trabalho com outros trabalhos (redes neurais e algoritmos sintáticos);

✓ Atividade 7: discussão sobre os pontos positivos e negativos do nosso sistema de reconhecimento de caracteres manuscritos.

1.4 Estrutura da dissertação

A dissertação está estruturada como segue. O capítulo 1 é esta introdução, a qual discorre sobre o reconhecimento de caracteres manuscritos, delimita e expõe os objetivos e a relevância da dissertação, bem como a metodologia empregada para a preparação da mesma.

O capítulo 2 trata de outros trabalhos relacionados com o reconhecimento de caracteres manuscritos, baseados em redes neurais e algoritmos sintáticos.

O capítulo 3 concentra-se na discussão dos conceitos de descoberta de conhecimento e data mining, com ênfase especial à técnica de regras de associação, utilizada em nosso sistema de reconhecimento de caracteres manuscritos.

O capítulo 4 detalha o funcionamento do nosso sistema de reconhecimento de caracteres manuscritos.

O capítulo 5 descreve os experimentos realizados, interpretando os resultados. Inclui comparações com os resultados das técnicas de redes neurais e análise sintática.

O capítulo 6 apresenta as conclusões e as perspectivas do trabalho.

Completam o documento a bibliografia e três anexos.

Capítulo Dois

Trabalhos Relacionados

Existem muitos trabalhos que exploram diversas tecnologias com o propósito de fazer o reconhecimento de caracteres manuscritos. Dentre estas tecnologias, podemos destacar algoritmos de análise sintática, redes neurais multicamadas Perceptron, redes neurais de Hopfield, redes baseadas em modelos de Markov, entre outras ([Subrahmonia1996], [Hanmandlu1999]). Restringir-nos-emos às duas primeiras tecnologias, as quais foram objeto de comparação com a técnica de regras de associação, proposta nesta dissertação.

2.1 Análise Sintática

Esta técnica exige um prévio e fastidioso trabalho manual de examinar coleções de imagens digitalizadas de caracteres como letras e/ou dígitos, a fim de determinar os padrões ou atributos sintáticos que possam identificar cada um dos caracteres [Lee1999]. Estes atributos dizem respeito à topologia e ao desenho das imagens dos caracteres. Podemos citar: o número de cavidades centrais, à direita e à esquerda; a posição da cavidade central, superior ou inferior; a distribuição de *pixels* na imagem, entre outros ([Veloso1998], [Lee1999]).

Na Fig 2.1, ilustramos um dos caracteres numéricos (matriz binária) da base do Cenparmi, com alguns atributos sintáticos identificados.

A seqüência de cruzamento vertical e a seqüência de cruzamento horizontal dizem respeito à quantidade de blocos com pontos pretos consecutivos nas colunas e linhas dos caracteres, respectivamente.

Cavidades são as regiões dos caracteres delimitadas pelo seu traçado e pelas bordas do menor retângulo que enquadra o caractere. As cavidades centrais são aquelas cujos pontos pretos não pertencem às bordas da matriz do numeral. As cavidades à direita são formadas pelos pixels entre a última coluna da matriz e o traçado mais a

direita do caractere. As cavidades à esquerda, ao contrário, são formadas pelos pixels entre a primeira coluna da matriz e o traçado mais a esquerda do caractere.

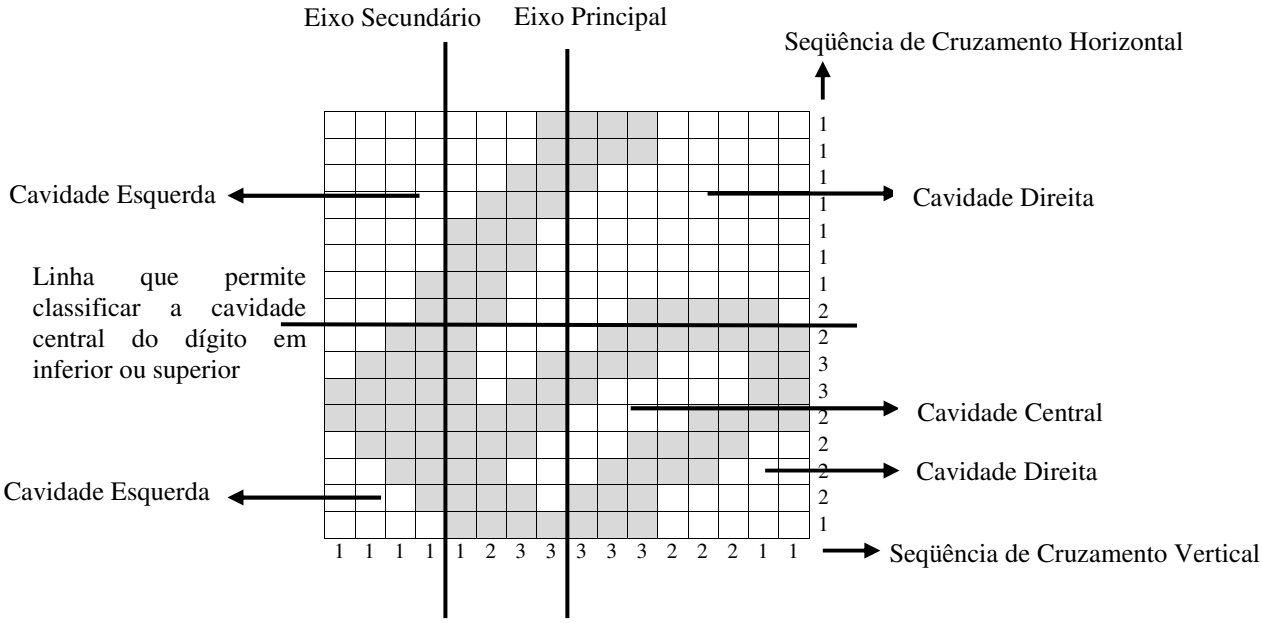

Figura 2.1 – Numeral 6 e seus atributos sintáticos

O eixo principal de um dígito, como o da Fig 2.1, é definido como o segmento de reta vertical que passa através do centro geométrico do mesmo, enquanto o eixo secundário é a linha vertical que fica eqüidistante do eixo principal e da primeira coluna da matriz de pixels [Lee1999].

Uma outra característica das imagens é a *distribuição pictorial*. Ela denota a quantidade de pixels pretos existentes na matriz ou a disposição física dos mesmos de acordo com algum critério [Veloso1998]. Seria possível, por exemplo, baseado na distribuição pictorial, formar critérios de decisão e diferenciar os caracteres manuscritos considerando o número total de pixels pretos de um numeral, ou o número de pixels pretos centrais da matriz, ou o número de pixels pretos dispostos à esquerda e/ou à direita da matriz, entre outros critérios.

Com relação a dígitos, a posição da cavidade central na matriz também é uma característica importante a fim de conseguir uma maior singularidade entre as diversas classes (de 0 à 9) de dígitos. A posição da cavidade pode ser inferior ou superior em relação à linha horizontal que divide o numeral manuscrito em duas regiões de mesmo tamanho.

De posse destas características, é possível construir um conjunto de regras para cada um dos caracteres, e a partir daí elaborar um algoritmo que seja capaz de reconhecer qualquer um dos caracteres estudados.

Dada a imagem digitalizada de um número, por exemplo, o primeiro critério de reconhecimento poderia ser o de saber quantas cavidades centrais a imagem possui. Se a mesma apresentasse duas cavidades centrais seria reconhecida como o numeral 2, ou o numeral 8. Para decidir entre os dois numerais, o algoritmo se utilizaria de outros atributos sintáticos, como a distribuição de pixels na imagem.

Em [Gomes1994], é proposto um algoritmo sintático para o reconhecimento de caracteres numéricos manuscritos que foram extraídos de cheques bancários. A Fig 2.2 mostra um trecho desse algoritmo.

O algoritmo começa dividindo as classes de dígitos em três grupos, aqueles que possuem caracteres com duas cavidades centrais, os que têm apenas uma cavidade e aqueles que não possuem nenhuma cavidade ou contêm mais de duas.

Se a imagem digitalizada possui duas cavidades centrais, ela pode ser um ‘2’ ou um ‘8’. A decisão acontece avaliando a seqüência de cruzamento: caso a matriz binária de entrada tenha uma seqüência de cruzamento mais parecida com a seqüência de cruzamento do dígito ‘2’ (extraída durante a inspeção das características dos caracteres), ela é classificada como sendo o numeral ‘2’, caso contrário é reconhecida como o ‘8’.

No caso da imagem ter apenas uma cavidade central, ela é reconhecida como um numeral ‘0’, ‘4’, ‘6’, ‘9’ ou numeral desconhecido (R). A conclusão final sobre à qual classe a imagem de entrada pertence é feita, em primeiro lugar, com base na posição em que está localizada a cavidade central na imagem (superior, inferior ou ambas). Caso o dígito de entrada possua sua cavidade central localizada tanto na região superior quanto na inferior ele é classificado como um ‘0’.

Se a cavidade estiver na posição inferior, é verificada a distribuição pictorial do dígito. Nesse caso, se não houver a presença de pixels pretos do lado direito do eixo principal, entre a primeira e a segunda interseção deste eixo com os pixels pretos que compõem o traçado do dígito, ele é conhecido como ‘6’, caso contrário não pode ser reconhecido (R) [Veloso1998].

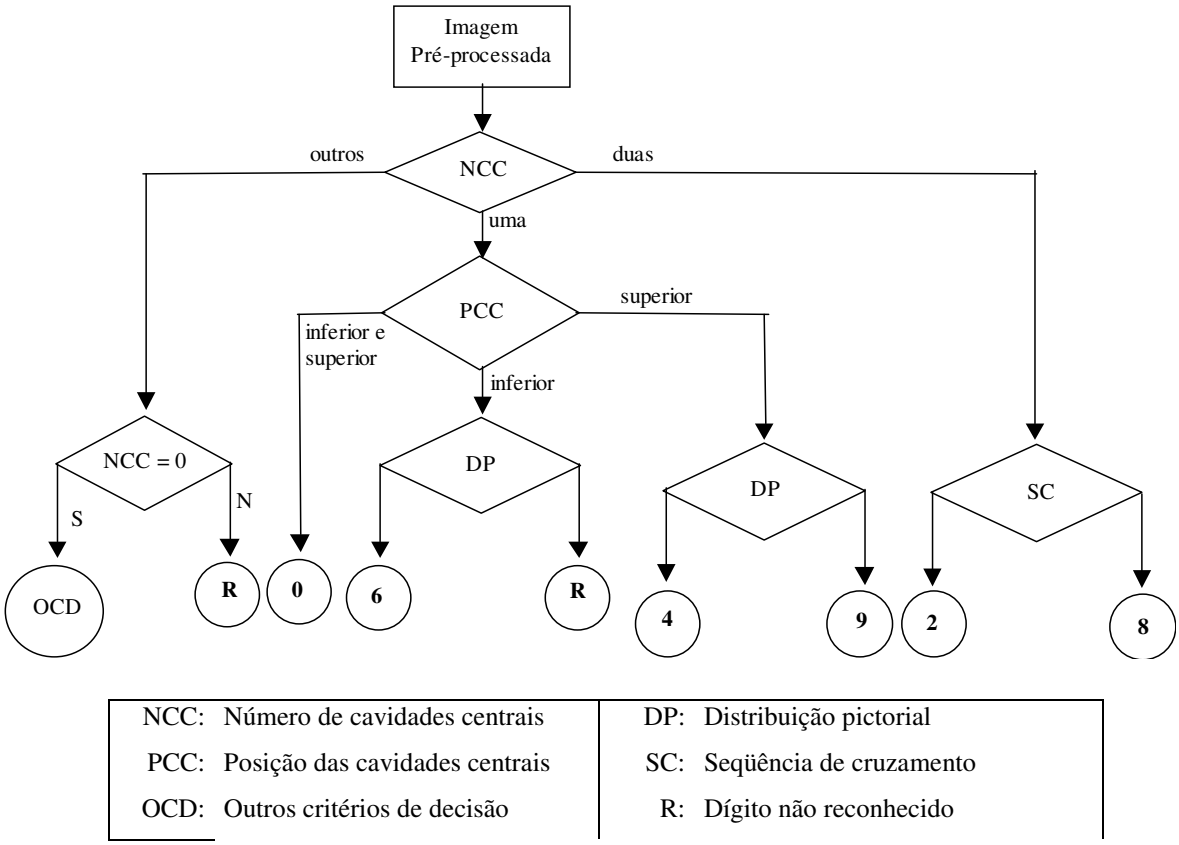

Figura 2.2 – Árvore de decisão para classificação de dígitos

Caso a cavidade central esteja na parte superior do dígito, ele pode ser reconhecido como um ‘4’ ou um ‘9’. Para tomar essa decisão utiliza-se também outras várias regras de distribuição pictorial.

Se a imagem de entrada possui mais de duas cavidades centrais ela não pode ser reconhecida (R). Caso ela não contenha nenhuma cavidade central então, de acordo este algoritmo, muitas outras regras topológicas são usadas para realizar a classificação. Dentre essas regras temos o número de interseções entre os pixels pretos e o eixo principal, distribuição pictorial, número de cavidades à esquerda, entre outras ([Gomes1994], [Veloso1998]).

Este algoritmo foi implementado por [Gomes1994] e permitiu que se obtivesse uma avaliação de sua acurácia a partir de um conjunto de exemplos de matrizes de dígitos que foram utilizados no teste. Para esse teste, foram utilizados caracteres provenientes de cheques bancários brasileiros.

Entretanto, ao mudar a base de dados de avaliação, ou seja, testar o algoritmo com outros caracteres, que não provindos de cheques bancários, o sistema mostrou-se

bastante ineficiente. Em [Veloso1998], há uma descrição de testes realizados com a base de dados do Cenparmi. Os resultados desses testes mostraram-se insatisfatórios, tendo um acerto médio de apenas 30,45%. Este fraco reconhecimento pode ser explicado pelo fato de dígitos presentes em cheques bancários serem normalmente escritos com maior atenção e conseqüentemente maior padronização. A base de dados utilizada por [Veloso1998], base do Cenparmi, provém de dígitos inseridos no código postal de cartas do correio norte americano, o que geralmente não exige tanto cuidado e acaba gerando muitos dígitos desiguais, vários deles sendo incompreensíveis até pelo próprio olho humano.

Um outro problema é que existem diferenças entre a escrita de numerais no Brasil e nos Estados Unidos da América (EUA). Por exemplo, nos EUA, a maioria das pessoas tende a não colocar uma linha horizontal cruzando o dígito '7', o que não acontece com os brasileiros.

Para solucionar esses maus resultados, em [Veloso1998] é feita uma remodelagem em todo o algoritmo sintático projetado por [Gomes1994], através de uma inspeção minuciosa nos caracteres da base de dados do Cenparmi, com o propósito de fazer uma reorganização dos critérios (atributos sintáticos) existentes, obtendo assim melhor desempenho no reconhecimento.

Isto demonstra uma deficiência dos algoritmos sintáticos: *eles são extremamente sensíveis a diferentes tipos de bases de dados*, fazendo com que o mesmo algoritmo não possa ser utilizado para fazer o reconhecimento de caracteres de uma base de dados diferente daquela para a qual foi elaborado. Logo, se quisermos utilizar um algoritmo sintático para reconhecer caracteres de uma nova base de dados, é necessário um novo estudo sobre as matrizes binárias do banco de dados, para extrair das mesmas suas características topológicas e montar um novo algoritmo.

O algoritmo proposto em [Veloso1998] pode encontrar muitas dificuldades de fazer o reconhecimento em uma base de dados que tenha uma outra origem. Para exemplificar, na base de dados dos códigos postais, os dígitos '7' não são cortados com uma barra horizontal no seu centro, conforme escrevemos no Brasil. Ao utilizar uma base de dados com '7's' cortados no centro, certamente teriam que ser implementadas novas regras no algoritmo sintático.

Um outro problema desses algoritmos está na responsabilidade total do ser humano extrair, sutilmente, as diferenças entre os diversos caracteres e montar as

regras, procurando eliminar através das características topológicas encontradas nos dígitos quaisquer confusões no momento do reconhecimento.

O algoritmo apresentado em [Veloso1998] conseguiu obter melhores resultados em relação ao seu similar [Gomes1994]. Permanece, no entanto, uma grande barreira a ultrapassar para melhorar a eficácia dos algoritmos sintáticos, qual seja, a extrema dificuldade de se obter padrões sintáticos genéricos e confiáveis.

2.2 Redes Neurais

Existem muitos trabalhos e pesquisas em redes neurais. Sua aplicação no reconhecimento de padrões visuais tem demonstrado grande eficácia [Tafner1995].

Além da área de reconhecimento de padrões, diversos campos da ciência têm demonstrado grande interesse nos modelos de redes neurais artificiais: áreas como a psicologia têm investigado as semelhanças estruturais destes modelos em relação à mente humana. Pesquisadores de inteligência artificial buscam nestes modelos soluções para um dos grandes problemas da inteligência artificial, o aprendizado de máquina [Hertz1990].

As redes neurais artificiais são um conjunto de modelos de neurônios artificiais, que através de sua estrutura e suas interligações procuram fazer com que as máquinas venham a ter um comportamento 'quase' humano. Isto é, essas redes permitem que as máquinas 'aprendam' e retenham informações [Tafner1995].

De acordo com [Cabena1997], as redes neurais artificiais são conjuntos de nós (neurônios) conectados. Os neurônios são ligados por conexões, cada uma com um peso (W) associado, que corresponde à influência do neurônio no processamento do sinal de saída. Pesos positivos correspondem a fatores de reforço do sinal de entrada, e pesos negativos correspondem a fatores de inibição [Hertz1990].

Para [Yuceer1993], as redes neurais artificiais são modelos computacionais com inspiração na estrutura do cérebro humano. Uma rede neural é constituída por diversas redes que atuam paralelamente, onde em cada uma delas existem unidades de processamento simples denominadas neurônios artificiais.

O funcionamento de um neurônio é mostrado na Fig 2.3. As entradas dos neurônios são valores provindos do ambiente externo ou da saída de outros neurônios existentes; associado a cada valor de entrada há um outro valor: o peso da entrada. É realizado um somatório da multiplicação das entradas pelos seus pesos correspondentes

e o resultado é inserido em uma função de ativação, a qual determinará o valor de saída do neurônio ([Kovacs1986], [Yuccer1993]). Dentre as funções de ativação, F(Yi), mais utilizadas temos a linear, a semi-linear, lógica, tangente hiperbólica, sino e sigmóide [Cavalcanti1998].

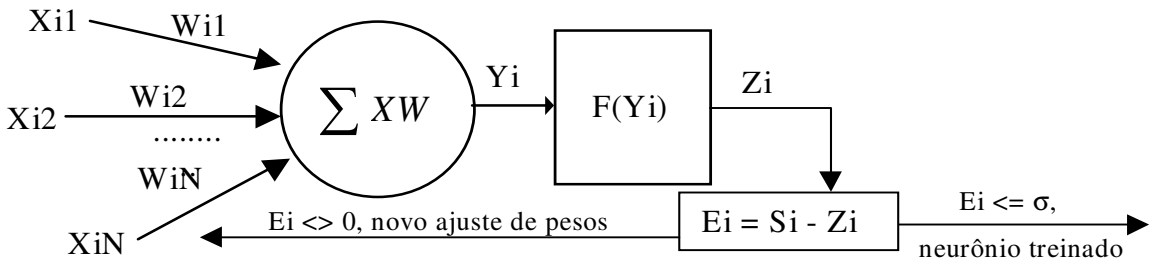

Figura 2.3 – Modelo de um neurônio artificial

As redes neurais funcionam basicamente em duas etapas, a fase de treinamento e a de teste. Na primeira fase são fornecidas a cada neurônio 'i' as entradas 'Xi's' e os pesos 'Wi's' correspondentes a cada uma destas. Nesta fase é sabido o valor desejado de saída do neurônio (Si).

Ao calcular Yi (somatório do produto das entradas Xi's pelos respectivos pesos Wi's) e Zi (saída do neurônio, calculada através de uma função de ativação F(Yi)), é feita uma verificação se Zi é igual a Si (o valor desejado). Na verdade, calcula-se o erro existente na saída do neurônio: Ei = Si – Zi. A existência de erro significa que o neurônio ainda não produziu a saída (Zi) que se espera (Si). Assim, é realizada uma modificação no valor dos pesos de cada entrada, de acordo com alguma equação, e consequentemente recalculam-se os valores de Yi e Zi e faz-se nova verificação de Ei. O processo terminará quando Ei for menor ou igual a um limiar previamente estabelecido (σ). Caso Ei continue maior que esse limiar, o processo de ajuste dos pesos continuará ([Kovacs1986], [Cavalcanti1998], [Hertz1990]).

Depois que os pesos dos neurônios estiverem ajustados (fase de treinamento), podemos dizer que a rede neural reteve um conhecimento sobre os elementos de entrada. A partir de então, pode-se iniciar a fase de teste, a qual consiste no processo de inserir diferentes elementos nas entradas Xij da rede neural e esperar que a rede calcule a saída correta.

Como dito previamente, uma rede neural engloba vários neurônios que estão dispostos conforme uma determinada arquitetura. Dentre as principais arquiteturas de redes neurais existentes, podemos destacar as redes neurais multicamadas com propagação direta e as redes recorrentes ([Kovacs1986], [Tafner1995], [Krose1995]).

Uma rede neural multicamada, ilustrada na Fig 2.4, possui, entre as camadas de entrada e saída, uma ou mais camadas denominadas ocultas. Os neurônios que se situam nessas camadas são chamados de neurônios ocultos. Nesse modelo de rede, um neurônio de uma camada está conectado a todos os neurônios da próxima camada, e o fluxo de dados é num único sentido (propagação direta) [Cavalcanti1998].

Especificamente, as redes neurais multicamadas com propagação direta têm demonstrado uma grande eficácia no reconhecimento de padrões visuais e no controle de processos, e são também conhecidas como redes neurais artificiais de Humelhart.

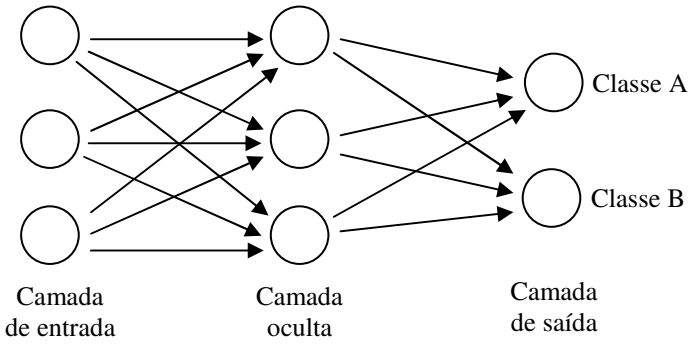

Figura 2.4 – Rede neural multicamada com propagação direta

Diz-se que a rede é de propagação direta porque cada neurônio ‘ï’ de uma camada transmite seu valor de saída (Zi) para uma entrada da camada posterior.

A rede da Fig 2.4 é totalmente conectada, pois todos os neurônios de uma camada comunicam-se com todos os neurônios da camada adjacente posterior. Existem também as redes parcialmente conectadas onde isto não acontece.

As redes neurais recorrentes, também chamadas de redes neurais de Hopfield, apresentam comportamento dinâmico e fluxo de dados multidirecional devido à integração total dos neurônios, desaparecendo assim a idéia das camadas bem distintas. Com isso, seu funcionamento é mais complexo, havendo certas complicações, seja na fase de aprendizado quanto na fase de testes. Seu uso é direcionado a problemas de

minimização e otimização, como por exemplo, descobrir o melhor percurso para um veículo chegar ao seu destino [Krose1995]. Tais modelos de redes são também utilizados no reconhecimento de padrões visuais, bem como em problemas NP completos [Cavalcanti1998]. A Fig 2.5 ilustra uma rede neural recorrente.

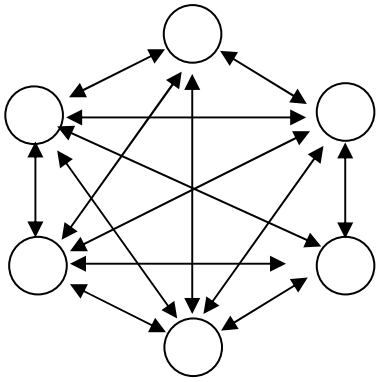

Figura 2.5 – Rede neural recorrente totalmente conectada

Para realizar o treinamento das redes neurais multicamadas com propagação direta, o algoritmo mais utilizado é o de "*back propagation*" ([Cabena1997], [Rummelhart1986]). Este algoritmo ilustra a maneira pela qual os erros são transmitidos da camada de saída até a camada de entrada da rede, a fim de ajustar os pesos. Vários trabalhos o utilizam para o reconhecimento de padrões, dentre eles temos [Yuccer1993], [LeCun1995], [Veloso1998]. O maior problema desse algoritmo é o longo processo de treinamento. Ele pode também resultar em uma taxa de generalização ruim, pois não há garantias de que os mínimos globais de erro (Ei <= σ) sejam encontrados. Em outras palavras, algumas vezes, dependendo dos pesos iniciais, da curva de aprendizado e dos dados de entrada, a rede neural artificial pode até não aprender [Kroer1995].

2.2.1 Aprendizado de redes neurais

Um algoritmo de aprendizado é um conjunto de regras bem definidas para a solução de um problema de aprendizado. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais e eles diferem entre si principalmente pelo modo como os pesos são modificados [Tafner1995].

No trabalho de [Yuceer1993], é utilizada uma rede multicamadas com propagação direta para fazer o reconhecimento de caracteres alfabéticos manuscritos. A camada de entrada da rede é constituída de um número de nós igual ao número de pixels

da imagem digitalizada do símbolo a ser reconhecido. O número de nós na camada de saída é igual ao número de caracteres alfabéticos existentes, neste caso 26 (alfabeto inglês, de A à Z).

Em [Veloso1998], todas as imagens de caracteres numéricos manuscritos a serem reconhecidas são matrizes de tamanho 16x16 (256 pixels). O tamanho da matriz foi padronizado porque a área em que a rede neural deve atuar, durante a fase de treinamento, é a região onde a matriz se encontra, pois caso existam muitos pixels fora dessas regiões, o desempenho da rede neural pode ser muito prejudicado.

Esse número para normalização, 256 pixels (16x16), foi escolhido de forma a não prejudicar o desempenho durante o treinamento da rede neural, caso fosse escolhido um número muito grande de pixels, pois, desta forma, existiriam muitas entradas na rede neural, ocasionando uma demora para ajustar os pesos da mesma. Em contrapartida, se o número de pixels para normalização fosse muito pequeno haveria uma perda muito grande de informação, pois transformar um caractere com, por exemplo, 20 linhas e 22 colunas (20*22=440 pixels), em um caractere com 6 linhas e 6 colunas (6*6=36 pixels) acarretaria muitas distorções topológicas nos caracteres.

Após essa normalização, em [Veloso1998], cada matriz 16x16 foi transformada em cinco matrizes de tamanho 4x4 (16 pixels), aplicando-se os gradientes de Kirsch [Suen1992]. Essas máscaras foram utilizadas a fim de capturar, para cada *pixel*, a presença de segmentos de linhas em uma dada direção (vertical, horizontal, diagonal à direita e diagonal à esquerda). Então, usando as máscaras de Kirsch, foram geradas, para cada caractere, 4 matrizes 4x4 (16 pixels) contendo características direcionais. A quinta matriz 4x4 usada foi extraída através da normalização a partir da imagem 16x16 original, a fim de contar com características globais do caractere. E são esses 80 (5 x 16) pixels que constituem as entradas dos 80 neurônios da camada de entrada da rede neural, que juntamente com os 80 neurônios da camada oculta terão os pesos de suas interconexões ajustados a fim de classificar uma determinada matriz binária em uma das 10 classes (0 a 9) de numerais existentes. Portanto, a camada de saída ainda contém 10 neurônios, os quais representam as 10 classes possíveis [Veloso1998].

Nos testes realizados em [Veloso1998] foram criados dois subconjuntos do conjunto de imagens digitalizadas de dígitos, denominados conjunto de treinamento e conjunto de testes, respectivamente. O conjunto de treinamento foi utilizado para

realizar o treinamento da rede neural artificial. Ou seja, cada dígito (imagem) foi inserido na camada de entrada da rede, e após algumas interações os pesos das ligações da rede foram ajustados de forma a gerarem na camada de saída o resultado correto, isto é, a classificação correta do dígito. O algoritmo de [Rummelhart1986] treinou a rede (isto é, ajustou os seus pesos). Uma vez a rede treinada, o conjunto de testes lhe foi submetido para testar a sua capacidade de reconhecer caracteres numéricos manuscritos.

2.2.2 Avaliação das redes neurais

Embora as redes neurais apresentem excelentes resultados quando aplicadas no reconhecimento de caracteres manuscritos, conseguindo em muitos trabalhos taxas de acerto maiores que 90%, alguns problemas persistem. O tempo de treinamento de uma rede neural é extremamente longo para a maioria das aplicações de uso prático, como o é o reconhecimento de caracteres. Em [LeCun1995], existem treinamentos realizados com 60.000 caracteres manuscritos, em uma máquina SPARC 10, utilizando-se classificadores baseados em redes neurais que chegam a demorar 35 dias.

Há também certos tipos de padrões, como os envolvidos em séries temporais, que demandam uma grande necessidade de pré-processamento, dada a complexidade da análise e classificação dos padrões em questão [Kovacs1986].

Também é necessária uma boa escolha dos valores iniciais dos pesos da rede para diminuir o tempo necessário de treinamento [Kovacs1986]. A rigor, tal tempo pode tender ao infinito!

Alguns modelos de redes neurais falham em convergir, ou seja, não conseguem gerar um nível estável dos pesos para realizar a predição correta dos dados. Este problema decorre do fato de haver, por exemplo, muito ruído nos dados de entrada. No caso do reconhecimento de caracteres manuscritos, esses ruídos seriam caracteres totalmente diferentes do convencional, caracteres ‘bizarros’, os quais certamente estariam presentes em bases de dados reais, exigindo, portanto, uma eliminação (limpeza, filtragem) prévia desses caracteres [Cabena1997].

Não há regras gerais para se determinar o volume de dados de entrada para o treinamento, quantas camadas ocultas devem ser utilizadas, a melhor estratégia de treinamento, que percentagem de dados deve ser destinada ao treinamento e ao teste da rede; esses parâmetros só podem ser estabelecidos através de bom-senso, experiência com redes neurais e com o processo de tentativa-e-erro ([Hertz1990], [Kovacs1986]).

2.3 Sistemas híbridos

Uma outra abordagem para se implementar sistemas que fazem reconhecimento de caracteres manuscritos é a utilização de sistemas híbridos (redes neurais e algoritmos sintáticos trabalhando em cooperação), pois segundo [Lixin1999], a combinação de diferentes classificadores permite que se capture as diferentes informações que cada um dos classificadores conseguem extrair. [Lixin1999] implementa um classificador que engloba outros quatro classificadores (três deles baseados nas características topológicas de seus caracteres e um baseado em redes neurais), conseguindo resultados excelentes, com acertos de 96%.

Seguindo esta abordagem híbrida, em [Lee1999] é apresentado um sistema que, num primeiro momento, procura fazer o reconhecimento dos caracteres a partir de suas características topológicas e da distribuição de pixels na imagem. Quando um dos diversos caracteres de teste não consegue ser reconhecido através destas características, devido às deformações encontradas no numeral, o processo de classificação é desempenhado por uma rede neural de Hopfield ([Cavalcanti1998], [Tafner1995], [Kovacs1986]). Desta forma, o sistema híbrido extrai tanto as características dos algoritmos sintáticos quanto das redes neurais para fazer o reconhecimento. Nesse trabalho, [Lee1999] conseguiu atingir uma taxa de reconhecimento de caracteres manuscritos em torno de 92%.

Nos trabalhos apresentados em [Lixin1999] e [Lee1999], foi necessário usar duas fases de treinamento, uma para extrair características topológicas dos caracteres, permitindo a elaboração de um algoritmo sintático, e a outra fase utilizando uma rede neural.

No capítulo 5, serão apresentadas taxas de reconhecimento de caracteres manuscritos utilizando algoritmos sintáticos e redes neurais, para efeito de comparação com o nosso sistema de reconhecimento de caracteres manuscritos, baseado em regras de associação.

Capítulo Três

Descoberta de Conhecimento com Data Mining

É extraordinária a quantidade de dados que dispomos nos dias atuais. Esses dados têm sido armazenados em bases de dados gigantescas, as quais vêm crescendo assustadoramente, pois novas tecnologias têm permitido tal expansão. Dentre estas tecnologias, podemos destacar a leitura do código de barras de diversos produtos, facilitando e acelerando a inserção de uma gama de dados nas bases atuais; a *Internet*, cuja dinâmica e ubiqüidade têm permitido, às diversas organizações, armazenar dados sobre clientes, produtos, compras, vendas, etc, de maneira extremamente prática, através dos "*web browsers*"; a tecnologia de "*data warehousing",* permitindo integrar diferentes bancos de dados; a evolução do "*hardware",* propiciando discos rígidos cada vez mais velozes e maiores, e computadores com alto desempenho.

Todavia, como processar esta quantidade incomensurável de dados, transformando-os em informações valiosas? Embora o armazenamento de dados tenha crescido assustadoramente, não temos testemunhado técnicas computacionais que nos ajudem a analisar esses dados acumulados. É inconcebível que um gerente tenha em suas mãos somente planilhas e consultas *ad-hoc*, a fim de extrair informação 'proveitosa' de um grande volume de dados [Fayaad1998].

Assim, embora a tecnologia de armazenagem de dados tenha evoluído significativamente mais que a tecnologia de recuperação de informações, é inevitável o aparecimento de técnicas e ferramentas de recuperação muito mais eficazes, a fim de atender às necessidades de usuários como gerentes de negócio, "*decision makers*" e executivos em geral, ou de qualquer outro usuário que precise de uma informação mais apurada.

É este o foco de uma nova tecnologia de informática, a *Descoberta de Conhecimento em Base de Dados* ("*Knowledge Discovery in Databases"* – KDD).

KDD é uma das áreas que recebe mais atenção dos pesquisadores, com o propósito de extrair informações gerenciais de, possivelmente, diversas bases de dados [Feldens1997a].

3.1 Descoberta de Conhecimento em Bases de Dados: Conceitos

Descobrir conhecimento significa extrair, de grandes bases de dados, sem nenhuma formulação prévia de hipóteses, informações *desconhecidas*, *válidas* e *acionáveis*, que podem ser utilizadas para a tomada de decisão [Cabena1997].

Informação desconhecida significa geração de um conhecimento que não é intuído, um conhecimento totalmente novo, provocando, às vezes, surpresa. Podemos citar, para ilustrar, o famoso exemplo de uma rede de supermercados, na qual se descobriu que havia um aumento considerável nas vendas de fraldas e cervejas nas sextas-feiras à noite (conhecimento este que faz uma associação entre os itens de venda). Esta informação é muito mais valiosa do que descobrir que os carros de luxo são comprados por pessoas ricas e idosas, já que tal informação é de conhecimento geral [Mongiovi1998].

A busca por informações válidas está intimamente ligada ao conhecimento do negócio. A extração de conhecimento em grandes bases de dados tende a gerar um número muito grande de associações entre os seus itens. Logo, muita informação espúria também é produzida por um sistema KDD. Se uma cadeia de lojas resolve buscar associações entre os itens que ela vende, certamente haverá associações entre itens que farão sentido, e outras que não farão [Mongiovi1998]. Portanto, é necessário, após a fase final de um processo KDD, uma compreensão da informação gerada, a fim de extrair as associações realmente relevantes para a empresa [Cabena1997].

Por último, as associações descobertas, ou outros tipos de conhecimento, precisam ser acionáveis. Em outras palavras, é necessário que possam ser realizadas ações simples para que o conhecimento gerado seja traduzido em vantagem aos negócios da empresa. O conhecimento da associação entre fraldas e cervejas poderia fazer com que os produtos fossem colocados em prateleiras próximas, ou quem sabe, gerar um desconto para só um desses produtos, mantendo o preço do outro um pouco mais elevado.

A fim de obter conhecimento útil, compreensível, relevante, válido e acionável, deve ser utilizado um processo de descoberta de conhecimento, o qual será explicado na próxima seção.

3.2 O Processo de Descoberta de Conhecimento

O processo de KDD é interativo, pois permite que os usuários tomem decisões no sistema, e iterativo, porque envolve numerosas etapas, podendo haver retorno a etapas anteriores [Feldens1997a]. A Fig 3.1 mostra a seqüência de etapas de um ciclo do processo de KDD [Cabena1997].

A primeira etapa do processo é a *determinação dos objetivos do negócio*, as três próximas etapas, *seleção*, *pré-processamento* e *transformação* compõem o que se chama *preparação de dados*. Data mining é o coração do processo de KDD, situado entre a preparação de dados e a *análise e assimilação* dos resultados. A seguir, descreveremos cada uma das etapas de um ciclo deste processo.

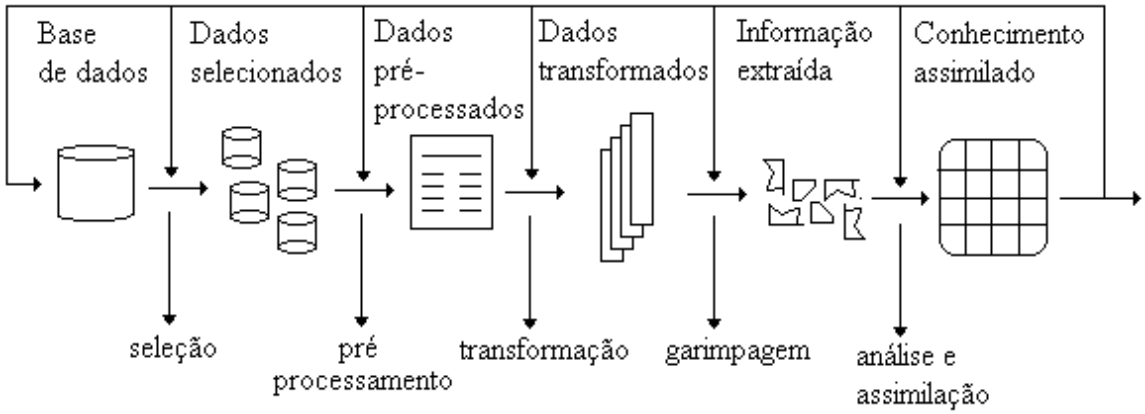

Figura 3.1 – Etapas de um processo de KDD

3.2.1 Determinação dos Objetivos do Negócio

Como ocorre em todo projeto que almeje o sucesso, é necessário, em primeira instância, analisar e definir os objetivos do negócio. Para isso, é estritamente necessária a presença de analistas que conheçam profundamente o negócio. Definidos os objetivos, um analista de dados pode, em primeiro lugar, julgar se a tecnologia de KDD os atingirá, para que depois possa traduzi-los em uma aplicação. É estritamente importante

responder à seguinte pergunta: será que a descoberta de conhecimento é a tecnologia adequada para as intenções do negócio?

Nessa fase do processo de KDD, é preciso também analisar a relação custo-benefício para se aplicar essa tecnologia, já que é muito difícil quantificar os benefícios de KDD devido à sua característica de exploração em busca de informação desconhecida.

Analisadas essas questões e julgando-se KDD a solução para o problema, parte-se então para a próxima fase do processo: a preparação dos dados.

3.2.2 Preparação de Dados

Inseridas nesta etapa estão três atividades: a seleção, o pré-processamento e a transformação de dados.

- Seleção de Dados: nesta etapa do processo de KDD, e consoante com os objetivos do processo, são identificadas as bases de dados, e quais variáveis e exemplos de dados devem ser extraídos para a fase de data mining (próxima fase). Por exemplo, ao extrair associações entre compras de clientes, alguns dados poderiam ser descartados, por não terem representatividade ao alvo de interesse, tais como endereço, telefone e e-mail, entre outros ([Fayyad1996], [Cabena1997]).

- Pré-processamento de Dados: em geral, as bases de dados existentes não estão adaptadas para a aplicação de algoritmos de data mining sobre elas, pois na maioria das vezes, quando foram concebidas não se pensava em executar esta tarefa sobre os dados, ou seja, eles não foram capturados e modelados para os propósitos desta tecnologia.

  Além disso, como os dados podem vir de várias bases, não necessariamente consistentes, impõe-se remover as inconsistências e integrá-los, tendo para isso que resolver vários conflitos típicos de integração de dados [Mongiovi1998].

  Portanto, nesta fase, procura-se identificar e, se possível, eliminar campos da base de dados contendo ruídos e ausência de valores. Dentre alguns exemplos de campos com problemas temos: idades com valores absurdos (idade = 899 anos), renda negativa (salário = -3899,00), número de filhos negativo (numfilhos = -3), nome da cidade ausente, etc. Geralmente os valores desses campos surgem devido às

falhas no momento de digitação dos dados, ou porque essas informações não eram conhecidas no momento da entrada dos dados ([Fayyad1996], [Cabena1997]).

É importante salientar que o resultado desta etapa é, em geral, um arquivo completamente distinto das bases de dados originais.

- Transformação de Dados: existem diversos algoritmos de data mining, cada um necessitando de uma entrada específica. A finalidade desta etapa, então, é transformar os dados que foram pré-processados na etapa anterior, de modo a torná-los compatíveis com as entradas desses algoritmos.

Dentre essas transformações podemos citar conversões de dados, criação de novas variáveis, categorização de variáveis contínuas, entre outras tarefas ([Cabena1997], [Damsels1996]). Por exemplo, vamos supor que a entrada de um algoritmo de data mining receba as idades dos clientes de uma empresa. Entretanto, a base de dados de estudo só tem as datas de nascimento dos clientes, portanto, faz-se necessária a realização de uma transformação sobre esses dados, adequando-os a entrada do algoritmo.

3.2.3 Etapa de data mining

Existem diversas tecnologias de data mining. Este passo do processo caracteriza-se pela escolha e aplicação do algoritmo ou algoritmos de data mining, de acordo com o método adequado ao problema que está se resolvendo. Entre os principais métodos existentes pode-se citar *regras de associação*, *regras de classificação*, "*clustering"* (agrupamento), entre outros.

Para regras de associação tem-se a seguinte família de algoritmos: *LargeItemSets* [Agrawal1993], *Apriori* e *AprioriTid* [Agrawal1994], *Partition* [Savasere1995], *Regras de Múltiplos Níveis* [Han1995], entre outros ([Srikant1996], [Yen1996]). Para regras de classificação destacam-se *ID3*, *PRISM*, *ADEX*, *IDRT*, *RPRISM* ([Mongiovi1995], [Mongiovi1992]), *C4.5* [Quinlan1993], *C5* [See5_1999], *Sipina* [SipinaPro1998]. E finalmente para algoritmos de "clustering", temos *Métodos baseados em Partições* e *Métodos Hierárquicos* [Mongiovi1998].

Nesta pesquisa nos detivemos nas regras de classificação e nas regras de associação, que serão explicadas nas seções 3.3.1 e 3.3.2, respectivamente.

3.2.4 Análise e Assimilação dos Resultados

Nesta etapa, é de suma importância a presença de um conhecedor do domínio onde se está aplicando o KDD, pois a seguinte questão deve ser respondida: o conhecimento gerado é útil, válido, relevante e acionável? Se a resposta não for satisfatória, então poderá ser necessário repetir todo ou parte do processo de KDD.

As etapas do processo de KDD consomem esforços diferentes. A fase de preparação de dados é a maior consumidora de esforço computacional e tempo [Cabena1997]. A Fig. 3.2 ilustra a diferença entre os esforços das etapas do processo.

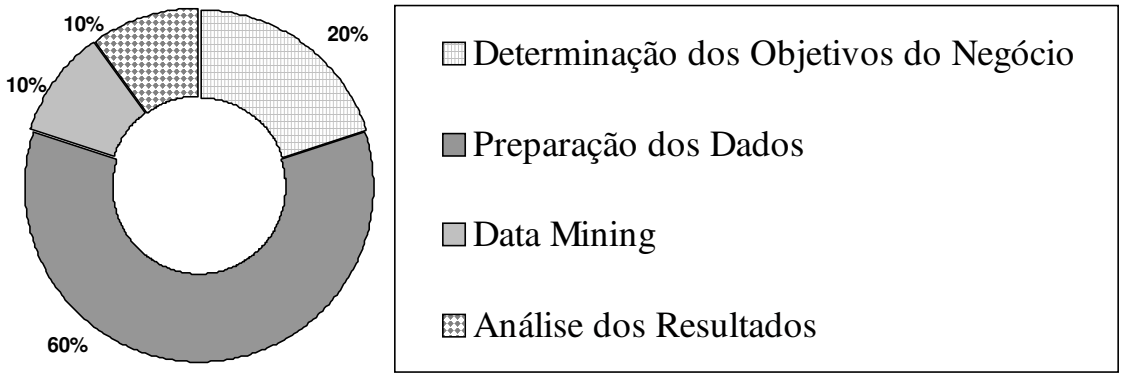

Figura 3.2 – Percentagem de esforço para cada etapa do processo de KDD

Na seqüência concentraremos nossa atenção na etapa de data mining.

3.3 Data mining

Como comentamos na seção anterior, data mining, ou *mineração de dados*, ou ainda *garimpagem de dados*, é apenas uma etapa de todo o processo de descoberta do conhecimento.

Data mining integra diversas áreas, principalmente estatística, inteligência artificial e banco de dados. Esta afirmação é coerente, pois ao analisarmos os algoritmos de data mining, verificamos que os mesmos utilizam-se de diversas medidas estatísticas a fim de fazer, por exemplo, classificações e/ou relacionamentos entre os itens de uma base de dados. Eles fazem uso também de métodos e algoritmos de indução estudados na Inteligência Artificial e são aplicados, geralmente, em um grande conjunto de dados

armazenados. A Fig 3.3 ilustra a interdisciplinaridade da tecnologia de data mining [Freitas1998b].

Qualquer algoritmo que gere um padrão a partir de dados é um algoritmo de data mining [Fayaad1998].

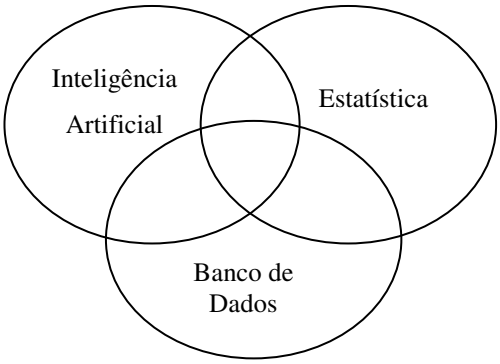

Figura 3.3 – Data mining e sua interdisciplinaridade

Até pouco tempo atrás, as ferramentas que se utilizavam de algoritmos de data mining eram de difícil utilização porque não tinham uma interface amigável com o usuário, exigindo do mesmo, conhecimentos em áreas como estatística. Uma ferramenta de data mining deve oferecer ao usuário, basicamente, uma escolha dentre os diversos algoritmos de data mining existentes, capacidade de *crescimento* ("scalability") e um bom desempenho, além de proporcionar uma facilidade na visualização dos resultados, auxiliando assim a interpretação dos mesmos ([Cabena1997], [SipinaPro1998], [See5_1999]).

Um exemplo destas ferramentas é a "Advanced Scout", desenvolvida pela IBM, para auxiliar os treinadores de basquete. O treinador do "Toronto Raptors", time de basquete da Liga Nacional Americana (NBA) já a utiliza intensamente fazendo consultas inteligentes aos dados sem a necessidade de grandes conhecimentos em estatística ou banco de dados. Ela permite ao treinador decidir onde colocar seus jogadores para obter o máximo de eficiência em um jogo, decifrar as forças e fraquezas do time adversário, conhecer os melhores arremessadores e defensores do seu time sob determinadas condições, entre outras informações válidas, desconhecidas e acionáveis. A "Advanced Scout" permite ao treinador, por exemplo, que ele entre com uma consulta em linguagem natural, tal como: qual é nosso melhor arremessador no terceiro quarto? [Baltazar2000].

Esta ferramenta roda localmente, mas foi projetada para possibilitar que ela combine os dados privados do treinador com estatísticas provindas de "*web sites*", de outros times e treinadores da NBA, facilitando, desta forma, a formulação de estratégias para os próximos jogos [Baltazar2000].

Sistemas que se aproveitam do potencial de data mining para extrair conhecimento de grandes bases de dados têm sido utilizados em vários ramos do mercado. Podemos citar soluções de aplicação de data mining em conjunto com "*Data Warehouse"* ([Kimball1998], [Korth1999]), que vem sendo implementado por grandes empresas [Scott1998]. Data mining tem estado presente também em aplicações de detecção de fraudes, aplicações de "marketing" (otimizando a relação de uma empresa com seus clientes) e empreendimentos para análise de investimentos e emissão de créditos [Indurkhia1998].

Na área de saúde esta tecnologia é usada para extrair informações sobre os procedimentos médicos, perfis de paciente e sobre os processos hospitalares utilizados a fim de possibilitar a descoberta de relacionamentos difíceis de se perceber, problemas com a qualidade de serviço, além de fraudes [Feldens1997b].

Em aplicações bancárias, data mining também tem sido objeto de estudo, permitindo aos bancos uma melhor compreensão do perfil financeiro de seus clientes [Scott1998]. Por exemplo, o Banco de Montreal tem extraído o comportamento de seus clientes e também tem tomado decisões estratégicas de negócio através da garimpagem dos seus 'terabytes' de dados [Montreal1999].

Existem diversos problemas onde KDD se aplica, dentre os quais podemos citar: *detecção de desvio*, *classificação*, *regras de associação*, *clustering*, *regressão*, *visualização*, entre outros [Indurkhia1998]. Para cada um dos problemas citados anteriormente, foram projetados diversos algoritmos de data mining. Nesta seção, descreveremos a classificação e nos aprofundaremos no algoritmo apriori, utilizado para extração de regras de associação.

3.3.1 Classificação

Dado um conjunto de classes $C=\{C_1, C_2,..., C_n\}$ e um banco de dados contendo um conjunto de exemplos $E=\{E_1, E_2,..., E_m\}$, onde cada Ej contém 'k' atributos, a técnica de classificação procura mapear um exemplo $Ep \notin E$ em uma das classes de C.

Basicamente, esta técnica consiste em criar um modelo de classificação, denominado *classificador*, a partir de um conjunto de registros existentes (exemplos), para que ele possa ser utilizado posteriormente, para classificar outros exemplos de classe desconhecida. Um método eficiente para produzir classificadores é gerar *árvores de decisão ou regras de classificação* [Fayyad1996].

Em problemas de classificação, devemos ter a presença de dois conjuntos de dados: o de *treinamento* e o de *teste*. O primeiro contém vários exemplos, onde se conhece a que classe pertence cada um deles. O algoritmo de classificação então, procurará extrair, a partir do conjunto de treinamento, uma árvore de decisão para permitir que um exemplo de teste possa ser classificado como uma determinada classe. Estabelecida a árvore, todo o conjunto de teste será utilizado para verificar a acurácia do sistema.

Um algoritmo já clássico para gerar árvores de decisão é o ID3 [Mongiovi1998]. Ele 'varre' todos os dados (exemplos) de treinamento e começa a construir a árvore a partir de sua raiz. Inicialmente, é escolhido um atributo '*a*', com a melhor função de avaliação (função esta que deve gerar um subconjunto de dados com a maioria dos exemplos pertencentes à mesma classe), para particionar esses dados. Para cada valor '*i*' do atributo '*a*', um ramo '*r*' é criado junto com o correspondente subconjunto de dados que possuem o valor de '*a*' igual a 'i'. Assim, para cada ramo '*r*', é criado um nó '*n*' na árvore que poderá ser uma classe (caso todos os exemplos analisados tenham '*a*' igual a '*i*' e representem a mesma classe) ou um outro atributo '*a*' (se algum exemplo de '*a*' igual a '*i*' representar classes diferentes) para se fazer nova partição. O algoritmo é executado recursivamente em todas aqueles nós que não representarem uma classe.

Exemplificando, imaginemos que uma instituição financeira tenha interesse em descobrir classes de clientes 'confiáveis' (clientes cujos empréstimos apresentam um baixo risco de prejuízo) e clientes 'não confiáveis' (aqueles cujos empréstimos seriam altamente arriscados), isto é:

confiáveis $\Rightarrow$ Clientes que pagarão seu empréstimo

não confiáveis $\Rightarrow$ Clientes que não pagarão seu empréstimo

Para montar estas duas classes, é necessário que exista no conjunto de treinamento um histórico sobre os empréstimos dos clientes, bem como a situação dos mesmos: ‘pagos’ ou ‘não pagos'. O algoritmo de classificação faz uma varredura no banco de dados, relacionando atributos (escolhidos, normalmente, pelo especialista) que, de acordo com os seus valores, gerem as tais classes. Dentre estes atributos, poderíamos citar a renda anual do cliente e o seu grau de escolaridade. Supondo que esses atributos possam receber os seguintes valores:

Grau de escolaridade: 1º grau; 2º grau; 3º grau; Mestrado; Doutorado
Renda anual (*Ra*): Ra <= R$ 10.000,00 (Ra<=10)
R$ 10.000,00 < Ra <= R$ 50.000,00 (10<Ra<=50)
Ra > R$ 50.000,00 (Ra>50)

Um subconjunto do histórico de empréstimos de clientes é ilustrado na Fig 3.4.:

Exemplo	Grau de escolaridade	Renda anual	Resultado
1	Mestrado	18.000,00	Pago
2	Mestrado	6.000,00	Não pago
...	...	...	...
n-1	Mestrado	25.000,00	Pago
n	3º grau	5.000,00	Não Pago

Figura 3.4 – Subconjunto do histórico de empréstimos de clientes

O algoritmo de classificação, inicialmente, irá escolher um atributo a fim de particionar o conjunto de clientes, e este atributo (atributo mais informativo) escolhido deve ser aquele que produza uma árvore mínima, tanto em sua altura, quanto em sua largura. Os clientes então serão divididos em grupos baseados nos valores do atributo escolhido. A partir daí, para cada grupo, é realizada uma nova divisão dos clientes com base nos atributos restantes [Korth1999].

Vamos supor que seja escolhido, em nosso exemplo, como atributo mais informativo, o grau de escolaridade. Após a escolha, é feita uma partição dos dados a fim de montar os grupos (nós da árvore). Por exemplo, serão criados cinco grupos para

as pessoas com 1º grau; 2º grau; 3º grau; Mestrado e Doutorado, respectivamente. Finalizado este primeiro passo, escolhe-se um outro atributo para se fazer nova divisão. Em nosso exemplo, para cada grau de escolaridade vamos dividir o grupo para cada faixa de renda anual existente. A Fig 3.5 mostra um exemplo de árvore de decisão gerada para o nosso exemplo.

Uma análise da árvore da Fig. 3.5, gerada a partir do histórico dos empréstimos, nos informa que uma pessoa que possui mestrado e ganha acima de dez mil reais por ano é uma pessoa que cumpre (paga) com os seus empréstimos, portanto, uma pessoa confiável. Ainda de acordo com a figura, neste contexto, uma pessoa com o título de mestrado e com uma renda anual menor que 10 mil reais não é uma pessoa confiável para se conceder um empréstimo. Tal conhecimento extraído permite ao gerente tomar a decisão de fazer novos empréstimos com uma maior segurança.

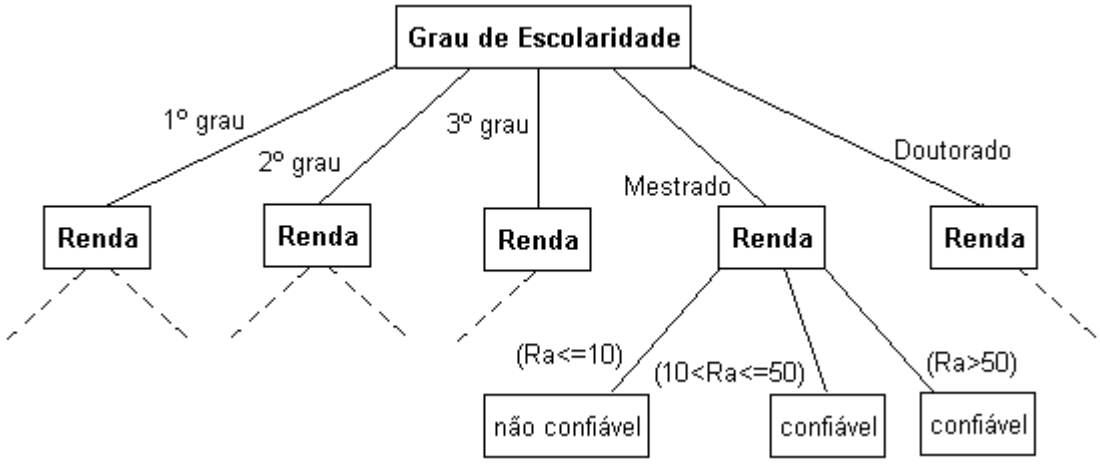

Figura 3.5 – Árvore de decisão gerada a partir dos valores no Banco de Dados

Note que cada nó da árvore separa os dados em grupos com base nos valores dos atributos e, além disso, é importante salientar que os ramos da árvore podem crescer de maneira diferente [Korth1999]. Este último fato ocorre porque, por exemplo, para um determinado grau de escolaridade como o de 1º grau pode ocorrer de nem existir empréstimos realizados a clientes com tal atributo. Na realidade, isso significa que o domínio não foi modelado adequadamente.

As árvores de decisão apresentam dois grandes problemas: o *problema sintático* e o *problema semântico*.

O problema sintático das árvores de decisão está relacionado ao seguinte fato: todas as regras geradas a partir de uma árvore terão que conter o atributo raiz em seu antecedente [Mongiovi1992]. Em nosso exemplo, já que grau de escolaridade foi o atributo raiz escolhido, não há como se ter uma regra como *Se Ra > 50 então confiável*. Além disso, caso o atributo raiz seja mal escolhido, ele influenciará as demais regras geradas.

O problema semântico diz respeito ao fato de as regras extraídas sobre o conjunto de treinamento não levar em conta nenhum conhecimento semântico sobre os atributos, ou seja, como a base de dados ilustra apenas um subconjunto do domínio em que se está trabalhando, casos raros podem existir neste universo de dados, provocando a geração de regras irrelevantes e, consequentemente, inúteis [Mongiovi1995]. Em [Gomes88], ao criar um sistema especialista para ginecologia, aplicando-se o algoritmo ID3 sobre os atributos existentes de um fichário médico, foi encontrada a seguinte regra: *Se idade=criança então vaginite*. Isto aconteceu porque haviam poucas crianças no conjunto de treinamento e, por coincidência, todas estavam com vaginite, portanto temos um caso raro que gera uma regra irrealista.

Existem algoritmos de classificação que, ao invés de montarem uma árvore de decisão, expressam o conhecimento extraído através de regras do tipo *Se...então...* ou *X, Y* $\Rightarrow$ *Z*. Para exemplificarmos tais regras, a Fig. 3.6 mostra as regras geradas a partir da árvore da Fig 3.5.

...Se (Grau de Escolaridade = Mestrado) e (Ra <= 10) $\Rightarrow$ não confiável

Se (Grau de Escolaridade = Mestrado) e (10 < Ra <= 50) $\Rightarrow$ confiável

Se (Grau de Escolaridade = Mestrado) e (Ra > 50) $\Rightarrow$ confiável...

Figura 3.6 – Regras extraídas a partir de um conjunto de exemplos

Dentre os algoritmos de classificação mais conhecidos, podemos citar os algoritmos da família TDIDT (ID3, e suas evoluções: ID3X, ADEX, IDRT, C4.5, C5), e os da família não TDIDT (PRISM, RPRISM e ISREG) ([Quinlan1993], [Mongiovi1995], [See5_1999]). Alguns destes ainda possuem os problemas sintáticos

e/ou semânticos descritos acima, e outros tais como o ISREG e o RPRISM resolvem estes problemas.

A técnica de classificação é uma das mais usadas na fase de data mining. Além de criar classificadores com árvores de decisão e regras de classificação, há também classificadores gerados a partir de redes neurais (já discutidas no capítulo 2), sendo esses últimos muito utilizados no reconhecimento de caracteres manuscritos.

3.3.2 Regras de Associação

Uma regra de associação é definida como: *Se X então Y* ou $X \Rightarrow Y$, onde X e Y são conjuntos de itens e $X \cap Y = \varnothing$. Diz-se que X é o antecedente da regra, enquanto Y é o seu conseqüente.

Um algoritmo baseado em regras de associação consiste em descobrir regras desse tipo entre os dados preparados para a garimpagem. Medidas estatísticas revelam a freqüência de uma regra no universo dos dados garimpados. Por exemplo, encontrar a seguinte associação {mercúrio, gaze, esparadrapo} $\Rightarrow$ {algodão} (0,78), significa dizer que 78% dos clientes que compram mercúrio, gaze, esparadrapo também compram algodão. Assim, o gerente de uma farmácia pode veicular campanhas publicitárias utilizando estes produtos, dispô-los em lugares próximos na prateleira, entender o porquê de uma possível queda nas vendas de alguns dos produtos, entre outras conclusões [Carvalho1999b].

Muitos algoritmos foram desenvolvidos com o objetivo de descobrir regras de associação entre itens de dados. Citamos: *LargeKItemSets* [Agrawal1993], *Apriori* [Agrawal1994], *AprioriTid* [Agrawal1994], *Partition* [Savasere1995] e *Multiple Level* (ML-T2L1) [Srikant1996]. Desses, o mais utilizado é o apriori, sendo que os demais, ou são extensões dele, ou o utilizam. Concentrar-nos-emos no algoritmo apriori.

Algoritmo Apriori

Vamos considerar um fragmento dos dados de uma farmácia, preparados para a fase de data mining (Tab. 3.1).

O arquivo é em forma de tabela, com três colunas representando itens de venda ou produtos. Cada linha da tabela representa uma transação de venda a um cliente, a primeira coluna da tabela contendo os identificadores das transações. Dada

uma transação de um cliente, o cliente terá comprado esparadrapo, por exemplo, se o valor da coluna Esparadrapo (E) para a determinada transação for 1, caso contrário ela estará marcada com 0.

Por exemplo, ao olharmos a Tab. 3.1 verificamos que na transação 2 o cliente comprou esparadrapo e algodão e não comprou mercúrio, já que as colunas que representam estes itens estão marcadas com '1', '1' e '0', respectivamente.

Tabela 3.1 – Tabela de transações de venda a clientes

Transação	Algodão (A)	Esparadrapo (E)	Mercúrio (M)
1	1	1	0
2	1	1	0
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	0	1	1
8	1	1	1
9	1	0	1
10	1	1	1

O algoritmo Apriori, apresentado detalhadamente no Anexo I, trabalha basicamente fazendo uma varredura sobre o arquivo, a fim de gerar todos os conjuntos de combinações de valores de colunas que aparecem no arquivo. Posteriormente, são considerados apenas aqueles conjuntos que aparecem no arquivo com uma freqüência não menor que um valor mínimo pré-fixado, gerando o denominado *conjunto G de grandes conjuntos*. A medida da freqüência de um elemento do conjunto G é chamada de *suporte*, assim definido:

$$Suporte(X) = \frac{\text{Nº de registros do arquivo que contêm os elementos do conjunto X}}{\text{Nº total de registros do arquivo}} \quad (3.1)$$

Quando procura-se descobrir regras de associação, é necessário definir um suporte mínimo (*supmin*[1]) para as regras. Referente à Tab. 3.1, considerando supmin igual a 0,6, {E} é um elemento do conjunto G de grandes conjuntos porque seu suporte é 0,9 (em dez linhas, nove têm valor 1), maior que o mínimo, portanto.

O conjunto { AE } também é um elemento do grande conjunto G porque estes dois itens foram comprados juntos em 8 transações das 10 existentes. Para supmin igual a 0,6, os demais grandes conjuntos que formam o conjunto G são mostrados na Fig 3.7 (a) (b) (c).

Coluna	Suporte
A	0,9
E	0,9
M	0,8

(a)

Coluna	Suporte
AE	0,8
AM	0,7
EM	0,7

(b)

Coluna	Suporte
AEM	0,6

(c)

Figura 3.7 – Elementos do conjunto G de grandes conjuntos, com um, dois e três itens

Após a geração do conjunto G dos grandes conjuntos, é necessário extrair as regras de associação sobre cada elemento de G, com um fator de confiança mínimo (C_{min}):

$\forall Y \subset X$, *se suporte*$(X \cup Y)$ */ suporte* $(X - Y) \geq C_{min}$, *então gera a regra* $X - Y \Rightarrow Y$,

onde o fator de confiança de uma regra R: X $\Rightarrow$ Y, é definido como:

$$Confiança(R) = \frac{\text{Nº de registros com X e Y}}{\text{Nº de registros com X}} \qquad (3.2)$$

O fator de confiança de uma regra denota o quanto o antecedente e o conseqüente desta estão relacionados (mais precisamente, o quanto X-Y e Y estão relacionados). Supondo um fator de confiança mínimo de 0,8 para o nosso exemplo, teríamos como válidas apenas as regras sombreadas da Fig 3.8.

A regra '{E} $\Rightarrow$ A' tem um fator de confiança maior que o mínimo (0,8), porque o suporte de {EA} é 0,8, que dividido pelo suporte de {E}, 0,9, resulta em 0,88.

Corolário: Seja a regra R dada por X $\Rightarrow$ Y: se |Y| = 1 e todas as transações tiverem o mesmo valor para o item Y, então Confiança(R) é igual a 1 [Carvalho1999a].

[1] supmin – menor suporte aceitável de um elemento do conjunto G.

Este corolário denota que se o conseqüente tiver apenas um item e este estiver presente em todas as transações do banco de dados, quaisquer outros itens que estiverem no antecedente da regra formarão uma regra com confiança igual a 1, ou 100%. Isto acontece porque o número de transações onde aparecem o antecedente e o conseqüente juntos será igual ao número de transações do antecedente.

Regra	Fator de confiança
{E} ⇒ A	0,88
{A} ⇒ E	0,88
{E} ⇒ M	0,77
{M} ⇒ E	0,87
{A} ⇒ M	0,77
{M} ⇒ A	0,87
{EA} ⇒ M	0,75
{EM} ⇒ A	0,85
{AM} ⇒ E	0,85

Figura 3.8 - Regras com fator de confiança além do mínimo

Um problema do algoritmo apriori é que se o conjunto de itens e a base de transações forem muito grandes, existe uma grande possibilidade de se gerar um número muito grande de combinações entre os itens para extrair os elementos do conjunto G de grandes conjuntos, o que impede, muitas vezes, uma única leitura no banco de transações [Mongiovi1998].

Dentre as principais diferenças entre regras de associação (RA) e classificação (C), citamos [Freitas1998b]:

- Todos os itens em (RA) podem aparecer tanto no antecedente quanto no conseqüente, já em (C), o conseqüente é a classe predita;
- Em (RA), a definição do problema é claro, encontrar todas as regras baseadas nos valores de 'suporte' e 'fator de confiança'. Em (C), não é tão claro que regras devem ser descobertas pelo sistema;
- Através das definições de 'suporte' e 'fator de confiança', em (RA), o usuário pode controlar a qualidade das regras descobertas, o que é muito difícil de se fazer em (C).

- Em (C) é criado um classificador, como por exemplo, uma árvore de decisão que enquadrará um exemplo dentro de uma das classes existentes. Em (RA) não é realizado nenhum tipo de classificação , nesta técnica são extraídas as associações existentes entre os itens de uma base de dados.

Vimos que os algoritmos de regras de associação extraem, de um conjunto de exemplos, associações entre os diversos itens da base de dados. As associações geradas são aquelas contendo os itens muito freqüentes (suporte) na base de dados, com um grau de relação grande entre eles (fator de confiança). O nosso interesse por algoritmos de regras de associação, tais como o apriori, é utilizá-los para resolver o problema do reconhecimento de caracteres manuscritos, gerando regras de associação para cada uma das classes de caracteres manuscritos treinados. Tais regras servirão, em nosso sistema, para montar os padrões de cada uma das classes treinadas, criando, desta forma, um classificador baseado em regras de associação. E, a partir da definição do suporte, controlaremos a qualidade dos padrões das classes de cada um dos caracteres treinados.

Portanto, o trabalho corrente investigará a viabilidade do emprego de um classificador baseado em regras de associação para o reconhecimento de caracteres manuscritos.

No próximo capítulo, explicaremos como foram gerados esses padrões para os caracteres manuscritos através do algoritmo apriori. Para mais detalhes do algoritmo apriori, recomendamos a leitura do Anexo I.

Capítulo Quatro

Treinamento do Sistema para Reconhecimento de Caracteres Manuscritos utilizando Regras de Associação

Este capítulo é uma explanação sobre o funcionamento do nosso sistema de reconhecimento de caracteres manuscritos, utilizando regras de associação. Descrevemos, detalhadamente, todas as etapas do processo, relatando o porquê delas e como foram implementadas.

4.1 Visão Geral do Sistema

O nosso sistema de reconhecimento de caracteres tem como propósito estar apto a classificar corretamente um caractere que lhe seja submetido. Por exemplo, se um dígito '9' for apresentado ao sistema, ele deverá concluir que o dígito realmente é um '9'. De um modo genérico, podemos identificar três fases neste processo: a *fase de preparação dos caracteres*, a qual procura eliminar distorções nos caracteres existentes, a *fase de treinamento*, com o objetivo de extrair regras de associação (através da aplicação do algoritmo apriori) a fim de formar padrões para cada um dos caracteres e a *fase de testes*, na qual é verificada a acurácia do sistema. Como estudo de caso utilizamos a base de dados do Cenparmi, a qual contém 17771 caracteres numéricos (de 0 a 9) manuscritos. Obtivemos esta base junto ao Prof. João Marques de Carvalho do Departamento de Engenharia Elétrica da Universidade Federal da Paraíba (Campus Campina Grande).

A fase de preparação dos caracteres tem o objetivo de eliminar os 'ruídos', caracteres 'bizarros', que muitas vezes encontram-se na base de dados de treinamento.

Segue-se a extração dos padrões para cada um dos caracteres, isto é, um padrão para o dígito ‘0’, outro padrão para o dígito ‘1’, e assim sucessivamente até o dígito ‘9’, e isto foi realizado através da geração de regras de associação, por meio do algoritmo apriori.

Construídos os padrões para cada um dos caracteres existentes, o sistema pode classificar um caractere de entrada. Explanando melhor, o caractere que entrar no sistema conterá um determinado padrão, o qual deverá se assemelhar a um dos padrões criados durante a etapa de treinamento, permitindo assim que ele seja classificado. Ou seja, o nosso sistema utiliza regras de associação para construir um classificador.

Com a entrada de vários caracteres no sistema (fase de testes), e o conseqüente erro ou acerto na classificação dos mesmos, podemos argüir a respeito da eficácia de nosso sistema, em comparação com outros existentes. A fase de testes será tratada no capítulo 5.

4.2 A Fase de Preparação dos Dados

No processo de descoberta de conhecimento, esta é uma das fases que mais consome tempo [Cabena1997], e em nosso trabalho não foi diferente.

Como já comentado, a base de dados na qual trabalhamos foi obtida do Cenparmi. Ela contém 17.771 caracteres digitalizados, coletados de envelopes de cartas ‘mortas’ (cartas que não chegaram ao seu destino) pelo “*U. S. Postal Service”*, de diferentes regiões dos Estados Unidos [Guyon1996]. Esta base está distribuída entre os dez dígitos existentes de acordo com a Tab 4.1.

Tabela 4.1 – Distribuição dos dígitos na base de dados do Cenparmi

Dígito	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Qtd. de Caracteres	2792	2697	1701	2213	1509	817	1372	1741	1213	716

A fase de preparação dos dados consumiu bastante esforço e tempo. A razão principal para tal esforço é que os caracteres do Cenparmi são geometricamente e topologicamente muito irregulares, caracteres extremamente deformados, além de estarem em diversos tamanhos. Por outro lado, o algoritmo apriori, em nosso escopo, é extremamente sensível a mudanças de geometria e de topologia, pois ele extrai os

padrões dos caracteres a partir das posições de ‘1’s’ e ‘0’s’ existentes na representação desses caracteres. A Fig. 4.1 ilustra três exemplos desses caracteres.

Ao visualizarmos a Fig 4.1, notamos que as matrizes na base de dados possuem largura e altura distintas, ou seja, algumas são enormes, Fig 4.1 (b), outras muito pequenas, Fig 4.1 (c). Além desta irregularidade, encontramos também na base de dados alguns dígitos que estão ‘em pé’, outros inclinados para a direita ou para a esquerda.

000000000000000000 000000000000000000 000000000000010000 000000000000111011 000000000000111111 000000000000011111 000000000000011110 000000000000111110 000000000001111110 000000000011111110 000000000111111110 000000001111111110 000000011111100110 000000011111001110 000000111110001110 000001111100001110 000011111000001110 000011110000001110 000111110000001110 000111100000011100 001111100000011100 001111000000111000 001111000000111000 001111000000110000 001110000001100000 011110000011100000 011110001111000000 001111111111000000 001111111100000000 000111111000000000 000001000000000000	00000000000000000000000000000000000 00000000000000000000111110000000000 00000000000000000000111111000000000 00000000000000000000111111100000000 00000000000000000000111111110000000 00000000000000000000001111111000000 00000000000000000000000001111100000 00000000000000000000000001111110000 00000000000000000000000000111110000 00000000000000000000000001111110000 00000000000000000000000001111110000 00000000000000000000000011111100000 00000000000000000000000111111000000 00000000000000000000001111111000000 00000000000000001111111111111000000 00000000000000011111111111100000000 00000000000000111111111111000000000 00000000000001111111111110000000000 00000000000011111111111111000000000 00000000000011111111111111100000000 00000000000001111111001111111000000 00000000000000111000000011111100000 00000000000000000000000001111110000 00000000000000000000000000111110000 00000000000000000000000000001111100 00000000000000000000000000001111100 00000000000000000000000000001111110 00000000000000000000000000000111110 00000000000000000000000000000111110 00000000000000000000000000000111111 00000000000000000000000000000011111 01110000000000000000000000000111111 01111000000000000000000000000111111 00111110000000000000000000011111111 00011111110000000000000011111111110 00001111111111111001111111111111110 00000011111111111111111111111111100 00000001111111111111111111111110000 00000000011111111111111111111000000	000000000000000 000000000000000 000111111100000 011111000000000 001110000000011 000110000000110 000111000011100 000011101110000 000001111100000 000000011100000 000000111100000 000001111110000 000001101111000 000011100111000 000011000011000 000011000011100 000011100001100 000011110001110 000001111111110 000000111111100 000000001000000
(a) Digito ‘0’	(b) Dígito ‘3’	(c) Digito ‘8’

Figura 4.1 (a) (b) (c) –Dígitos irregulares contidos na base de dados original

Para se conseguir um bom desempenho é necessário descartar, inicialmente, muitos destes dígitos deformados [Lee1999]. Entretanto, antes de partir para uma eliminação de alguns caracteres, aplicamos sobre os mesmos, diversos algoritmos de

processamento de imagens, com o intuito de tornar as matrizes mais próximas dos seus respectivos padrões.

Esta fase de preparação dos caracteres tem como propósito melhorar a *informação pictorial*, visando aumentar assim, o desempenho dos algoritmos de reconhecimento [Veloso1998].

Assim, foi necessário todo um trabalho de regularização destes caracteres, a fim de se obter numerais mais semelhantes aos padrões que conhecemos e consequentemente conseguir resultados mais significativos no reconhecimento de caracteres manuscritos. O trabalho de regularização destes caracteres consistiu das seguintes tarefas: rotação, normalização, esqueletização (afinamento), dilatação, suavização e centralização.

Com o objetivo de se obter um resultado melhor na imagem final das matrizes de caracteres, as tarefas para a regularização dos mesmos foram efetuadas algumas dezenas de vezes e em ordens diferentes. Dentre estas ordens diferentes pode-se citar: aplicação de apenas normalização; apenas normalização e rotação; rotação, normalização e esqueletização; normalização, rotação e esqueletização; entre outras ordens.

A seguir serão explanadas todas as tarefas realizadas durante a preparação de dados seguindo a ordem na qual os caracteres obtiveram um resultado melhor (melhor qualidade visual) na imagem final.

4.2.1 Rotação

Na base de dados do Cenparmi existem dígitos com inclinações mais ou menos para a direita, outros inclinados mais ou menos para a esquerda, alguns dígitos estão ‘em pé’ e outros estão ‘deitados’.

A fim de deixar os caracteres em uma mesma orientação, fez-se necessário um algoritmo que girasse todos os dígitos inclinados ou ‘deitados’, colocando-os ‘em pé’, ou seja, fazendo uma rotação.

Giramos todos os dígitos da base através de dois algoritmos de rotação. Um deles é o *R-Block* [Yuccer1993], que se mostrou bastante eficiente na rotação do dígito 1. Para os demais dígitos, foi utilizado um algoritmo melhorado do *R-Block,*

desenvolvido por [Veloso1998]. Ambos algoritmos foram implementados por [Veloso1998].

A Fig 4.2 mostra um dígito em sua posição original (a) e o mesmo dígito após a rotação (b).

Verificamos que alguns caracteres que encontravam-se muito *deitados*, ou seja, bastante inclinados, não conseguiram ser rotacionados corretamente pelos algoritmos que utilizamos. Estes caracteres acabaram sendo rotacionados ao contrário, ou seja, ao invés de ficarem em pé, acabaram ficando de cabeça para baixo, prejudicando, futuramente, o seu reconhecimento.

00000000000000000001111111000000000000000000 00000000000000000011111111000000000000000000 00000000000000000111111111000000000000000000 00000000000000011110001111111000000000000000 00000000000001111100001111111000000000000000 00000000000011111000011111110000000000000000 00000000000111100000111111100000000000000000 00000000001111011111101111000000000000000000 00000000001111111000011110000000000000000000 00000000000111110000111100000000000000000000 00000000000000000000111100000000000000000000 00000000000000000001111000000000000000000000 00000000000000000011110000000000000000000000 00000000000000000111100000000000000000000000 00000000000000001111000000000000000000000000 00000000000000011110000000000000000000000000 00000000000000111100000000000000000000000000 00000000000001111000000000000000000000000000 00000000000011110000000000000000000000000000 00000000000111100000000000000000000000000000 00000000001111000000000000000000000000000000 00000000011110000000000000000000000000000000	00000000000000000001111100000000000000000000 00000000000000000011111000000000000000000000 00000000000000000111111011100000000000000000 00000000000000011111111111000000000000000000 00000000000000011100111111000000000000000000 00000000000000111000111111000000000000000000 00000000000001110001111110000000000000000000 00000000000011100001111110000000000000000000 00000000000111000011011100000000000000000000 00000000000111001110011100000000000000000000 00000000000111011000011100000000000000000000 00000000001111111000011100000000000000000000 00000000001111100001110000000000000000000000 00000000000110000011100000000000000000000000 00000000000000000011100000000000000000000000 00000000000000000111000000000000000000000000 00000000000000001110000000000000000000000000 00000000000000011100000000000000000000000000 00000000000000111000000000000000000000000000 00000000000001110000000000000000000000000000 00000000000011100000000000000000000000000000 00000000000111000000000000000000000000000000
(a)	(b)

Figura 4.2 – Matriz do dígito ‘9’ original (a) e rotacionada (b)

4.2.2 Normalização

Após o trabalho de rotação, foi necessário deixar os dígitos com um mesmo tamanho. Para concretizar esta tarefa fez-se a normalização. A normalização é o processo de fazer com que as matrizes de 0’s e 1’s, de tamanhos variados, ficassem com um tamanho comum. Como veremos adiante, é importante que tenhamos as posições dos pixels de todos os caracteres de treinamento limitados por um quadrado de mesmo tamanho.

Para efeito de comparação com [Veloso1998], todos os caracteres foram normalizados em matrizes de tamanho 16 x 16.

O algoritmo utilizado para gerar os caracteres normalizados foi proposto por [Yuccer1993] e obtido a partir da implementação dele feita por [Veloso1998].

A Fig. 4.3 (a) ilustra uma matriz com 21 colunas e 20 linhas, antes do processo de normalização e a Fig. 4.3 (b) demonstra a matriz normalizada, contando com 16 linhas e 16 colunas. Note que a Fig 4.3 (b) também está rotacionada.

000000000000000000000 000000000000000001110 000000000000000011110 000000000000011111110 000000000001111111111 000000000111111101111 000000001111110000111 000000011111100000111 000000111111000000111 000001111100000000111 000011110000000001111 000111110000000011110 001111100000000111110 001111000000001111000 001110000000111110000 011110000111111000000 011111111111110000000 001111111111000000000 000111111100000000110 000000100000000000000	0000000000000000 0000011111100000 0000011111111100 0000011001111110 0000111000000111 0001111000000111 0001111000000011 0001111000000011 0001111000000111 0001110000000111 0001111110001111 0000111000001110 0000011110011100 0000111111111100 0000001111110000 0000000000000110
(a)	(b)

Figura 4.3 – Matriz do dígito ‘0’ (a) e matriz do mesmo dígito (b) normalizado

4.2.3 Esqueletização

O processo de esqueletização procura obter uma réplica *estruturada* da imagem, preservando as características estruturais da imagem original [Yuccer1993]. Um algoritmo de esqueletização precisa preservar a topologia do caractere [Lee1999].

O objetivo da esqueletização é a geração de caracteres com uma largura de apenas um *pixel*. Durante a fase de aquisição da imagem, devido a diferentes tipos de lápis, papel e pressão de escrita, o contorno dos caracteres fica com diferentes larguras, o que pode prejudicar no instante do reconhecimento. Desta forma, a aplicação da esqueletização obtém o esqueleto de cada um dos caracteres existentes, eliminando diferentes larguras existentes nas bordas deles.

0000000000000000 0000011111100000 0000011111111100 0000011001111110 0000111000000111 0001111000000111 0001111000000011 0001111000000011 0001111000000111 0001110000000111 0001111110001111 0000111000001110 0000011110011100 0000111111111100 0000001111110000 0000000000000110	0000000000000000 0000000000000000 0000000100000000 0000000011110000 0000000010011100 0000000010000110 0000000100000010 0000001000000001 0000001000000001 0000001000000001 0000010000000001 0000010000000001 0000001000000010 0000000100011110 0000000011100000 0000000000000000
(a)	(b)

Figura 4.4 – Dígito ‘0’ (a) e o mesmo dígito (b) após o processo de esqueletização

A Fig 4.4 ilustra uma matriz de um caractere (a) e o mesmo caractere após o processo de esqueletização. O algoritmo de esqueletização utilizado em nosso pré-processamento foi proposto por [Gonzales1992] e implementado por [Veloso1998].

4.2.4 Dilatação

Um número pequeno de pixels pretos na imagem, isto é, marcados com ‘1’, afeta o sucesso do reconhecimento, pois diminui bastante a quantidade de informação (contorno dos caracteres) existente o que dificulta a geração de padrões [Lee1999]. Além disso, a esqueletização nem sempre deixa os contornos dos dígitos com um número uniforme de pixels. Para resolver estes dois problemas, empregamos a técnica de *dilatação*.

0000000000000000 0000000000000000 0000000100000000 0000000011110000 0000000010011100 0000000010000110 0000000100000010 0000001000000001 0000001000000001 0000001000000001 0000010000000001 0000010000000001 0000001000000010 0000000100011110 0000000011100000 0000000000000000	0000000000000000 0000000000000000 0000000000000000 0000011100000000 0000001111110000 0000001111111100 0000001110011110 0000011100001110 0000111000000111 0000111000000111 0000111000000111 0001110000000111 0001110000000111 0000111000001110 0000011101111110 0000001111100000
(a)	(b)

Figura 4.5 – Matriz antes (a) e após (b) o processo de dilatação com espessura de três pixels

Portanto, a fim de contornar estes empecilhos, foi processado sobre as matrizes de caracteres um algoritmo de dilatação, o qual deixa todos os caracteres com uma mesma largura de três *pixels* pretos.

A dilatação também tem como propósito tornar as imagens de um mesmo dígito ainda mais semelhantes, contribuindo para um melhor reconhecimento.

O algoritmo utilizado para implementar a dilatação foi o de [Gomes1994], também implementado por [Veloso1998]. A Fig 4.5 mostra o resultado da aplicação deste algoritmo de dilatação sobre o esqueleto (largura de 1 pixel) de um caractere.

4.2.5 Suavização

Numa imagem digitalizada, as posições marcadas com 1's denotam o contorno do dígito. Todavia, muitas imagens, além dos 1's formando o contorno do dígito, apresentam também alguns 1's em posições bem distintas às do contorno (posições que deveriam estar marcadas com 0 – ver primeira linha da Fig 4.6 (a)). Este problema surge durante a fase de digitalização dos caracteres e durante a aplicação dos diversos algoritmos de preparação dos dados vistos anteriormente.

1001000000000000 0000000000101100 0100000000000000 0000011100000000 0000001111110000 0100001111111100 0000001110011110 1000011100001110 0000111000000111 1000111000000111 0000111000000111 0001110000000111 0001110000000111 0000111000001110 0000011101111110 0000001111100000	0000000000000000 0000000000000000 0000000000000000 0000011100000000 0000001111110000 0000001111111100 0000001110011110 0000011100001110 0000111000000111 0000111000000111 0000111000000111 0001110000000111 0001110000000111 0000111000001110 0000011101111110 0000001111100000
(a)	(b)

Figura 4.6 – Matriz do dígito '0' (a) e matriz do mesmo dígito suavizado (b)

O processo de suavização tem como objetivo retirar da imagem original os pontos isolados (ruído) e reduzir os picos e buracos existentes nos contornos dos caracteres [Veloso1998].

Portanto, a suavização procura eliminar os 1's fora do contorno, e também faz a remoção de buracos existentes no contorno dos caracteres. O algoritmo empregado para a suavização foi o de [Suen1980]. A Fig 4.6 exemplifica o processamento do algoritmo de suavização sobre uma imagem com pontos isolados (ruído).

Vale salientar que todos esses algoritmos de processamento de imagem vistos até aqui foram implementados e utilizados em [Veloso1998].

4.2.6 Centralização

Vamos considerar um caractere 'C' delimitado por um quadrado 'Q', com 'n' colunas e 'n' linhas. Definiremos:

- 'ps', como o pixel do caractere 'C' que se encontra mais próximo da borda superior de 'Q';
- 'pi', o pixel do caractere 'C' que se encontra mais próximo da borda inferior de 'Q';
- 'pe', o pixel do caractere 'C' que se encontra mais próximo da borda esquerda de 'Q';
- 'pd', o pixel do caractere 'C' que se encontra mais próximo à borda direita de 'Q'.

Dizemos que um caractere *'C' está centralizado em 'Q'*, se a distância entre 'ps' e a primeira linha de 'Q' for igual à distância entre 'pi' e a última linha de 'Q', e a distância entre 'pe' e a primeira coluna de 'Q' também for igual à distância entre 'pd' e a última coluna de 'Q'.

Após todo o pré-processamento realizado até aqui, muitos dígitos da base do Cenparmi ficaram mais próximos da extremidade esquerda (coluna 1), outros mais próximos da extrema direita (coluna 16), alguns muito perto da linha superior (linha 1) ou da linha inferior (linha 16), ou seja, quase todos os dígitos não ficaram centralizados.

Para resolver este problema, implementamos, utilizando-se o ambiente de programação Delphi 4.0, um algoritmo que procurou ajustar os pixels pretos (contornos dos dígitos) para que ficassem localizados no centro da matriz 16x16.

A matriz de um caractere é varrida pelo algoritmo de modo que seja descoberto a distância (em número de linhas) entre ‘ps’ e o limite superior do caractere e entre ‘pi’ e a última linha do caractere de entrada, bem como a distância (em número de colunas) entre ‘pe’ e a primeira coluna do caractere e entre ‘pd’ e a última coluna. De posse desses valores o algoritmo calcula a quantidade de linhas e/ou colunas que o caractere se deslocará (para baixo ou para cima e/ou para a esquerda ou para a direita), por fim o caractere é deslocado de maneira que fique centralizado na matriz 16x16.

0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000001111110000 0000001111111100 0000001110011110 0000011100001110 0000111000000111 0000111000000111 0000111000000111 0001110000000111 0001110000000111 0000111000001110 0000011101111110 0000001111100000	0000000000000000 0000000000000000 0000111111000000 0000111111110000 0000111001111000 0001110000111000 0011100000011100 0011100000011100 0011100000011100 0111000000011100 0111000000011100 0011100000111000 0001110111111000 0000111110000000 0000000000000000 0000000000000000
(a)	(b)

Figura 4.7 – Matriz do dígito ‘0’ (a) e matriz do mesmo dígito após a centralização (b)

Observando a Fig 4.7 (a), notamos que a distância entre ‘ps' e a primeira linha do quadrado que envolve o caractere é igual a 4 unidades (ou 4 linhas), e a distância entre ‘pi’ e a última linha desse quadrado é igual a 0, isto é, o caractere está deslocado para baixo.

Ainda observando a Fig 4.7 (a), verificamos também que a distância entre ‘pe’ e a primeira coluna do quadrado que envolve o caractere é igual a 3 unidades (ou 3 colunas), enquanto a distância entre ‘pd’ e a última coluna é igual a 0 unidades (ou 0 colunas). Portanto, o caractere encontra-se deslocado para a direita.

Então, para centralizarmos o dígito da Fig 4.7 (a) aplicamos o nosso algoritmo de centralização. Como vemos na Fig 4.7 (b), a distância entre ‘ps' e a primeira linha do quadrado que envolve o caractere é 2, assim como a distância entre ‘pi’ e a última linha. Portanto temos o caractere centralizado verticalmente.

Todavia, vale notar que o caractere da Fig 4.7 (b) não encontra-se centralizado horizontalmente, ou seja, a distância entre ‘pe’ e a primeira coluna é igual a 1, enquanto a distância entre ‘pd’ e a última coluna é igual a 2. Isto acontece porque o número de colunas existentes entre ‘pe’ e a primeira coluna, adicionado ao número de colunas presentes entre ‘pd’ e a última coluna é igual a um número ímpar (no caso da Fig 4.7 (a), igual a 3). Nesse caso é impossível centralizar o caractere horizontalmente. Quando é impossível centralizar as matrizes da base de dados do Cenparmi, adotamos o seguinte critério para todos os caracteres: ‘os caracteres ficam deslocados mais à esquerda e/ou mais para cima’. Isto é, a distância entre ‘ps' e a primeira linha é uma unidade (linha) menor que a distância entre ‘pi’ e a última linha. Horizontalmente, teremos a distância entre ‘pe’ e a primeira coluna, uma unidade (coluna) menor que a distância entre ‘pd’ e a última coluna do quadrado que envolve a matriz.

4.2.7 Limpeza de caracteres espúrios

Após todas essas etapas do processamento das imagens do banco de dados do Cenparmi, encontramos na base de dados, matrizes com muito ‘ruído’, ou seja, muitos caracteres distorcidos de sua imagem original, caracteres estes que acabam, muitas vezes, não sendo reconhecidos mesmo pelo olho humano.

0000000000000000	0000000000000000	0000000000000000	0000000000000000
0111110000000000	0000000000000000	0000001100000000	0000000000000000
0111111000001110	0111111111111000	0000001110000000	0000000001110000
0111111100001110	0000000000011100	0000001110000000	0000000011100000
0111011110011110	0000000000011100	0000001110000000	0000001111000000
0111001111111000	0000000000011100	0000000111000000	0000111100000000
0000000011100000	0000001111111100	0000000111100000	0001111000000000
0000000111000000	0000011110001110	0000000011111100	0001110000000000
0000000111000000	0000011100001110	0000000001110000	0001110000000000
0000000111000000	0001111000001110	0000001111110000	0001110000000000
0000000111000000	0001110000001110	0000111110111000	0011100000000000
0000001110000000	0001110000111100	0111111000111000	0011100000000000
0000111110000000	0000111111111000	0000000000111000	0011100011100000
0000111110000000	0000000000000000	0000000000011000	0011111111000000
0000111110000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
(a)	(b)	(c)	(d)

Figura 4.8 – Caracteres numéricos espúrios

A Fig 4.8 ilustra alguns exemplos de dígitos onde o reconhecimento é praticamente impossível (2, 3, 4 e 6, correspondendo respectivamente a (a),(b),(c) e (d)).

Em conseqüência, não se poderia esperar que o nosso algoritmo de reconhecimento, ou qualquer outro, pudesse reconhecê-los (podemos facilmente afirmar que, o que é praticamente impossível de ser reconhecido pelo ser humano, também o é pelas máquinas!). Por tal motivo, realizamos uma inspeção visual em todos os caracteres existentes, procurando construir uma nova base de dados, eliminando, desta forma, todos os caracteres que consideramos serem muito difíceis para o reconhecimento por um humano (caracteres espúrios). Foi eliminado 31,87% do total de caracteres existentes na base de dados do Cenparmi.

Submetemos então o conjunto de treinamento a uma análise, sendo extraído do mesmo todos os dígitos espúrios.

É válido notar que estes caracteres tiveram seus formatos extremamente alterados devido aos processos de aquisição da imagem e/ou às distorções que cada um dos algoritmos aplicados anteriormente inseriram nas matrizes.

4.3 A Fase de Extração dos Padrões

Após a fase de preparação dos dados, o nosso sistema efetuou a extração de padrões (regras de associação) sobre o conjunto de caracteres, para cada um dos dígitos existentes. Esta etapa também é denominada de fase de treinamento do sistema.

Nesta seção, detalharemos todo o processo de extração dos padrões das classes existentes no banco de dados do Cenparmi (dígitos de 0 à 9).

4.3.1 Geração dos conjuntos de treinamento e do conjunto de teste

Nesta etapa, geramos o conjunto de treinamento e de teste para que depois, implementado o algoritmo apriori*,* pudéssemos extrair os padrões de cada uma das classes de dígitos existentes (de 0 à 9).

Para construirmos o conjunto de treinamento e de teste, elaboramos um programa que permite separar os ‘n’ caracteres do arquivo de um dígito ‘d’ qualquer, em ‘m’ caracteres para o conjunto de teste e ‘n – m’ caracteres para o conjunto de treinamento. O parâmetro ‘m’, igual para todos os dígitos, é inserido pelo usuário. Na maioria dos trabalhos estudados, o parâmetro ‘m’ representa cerca de 20% a 30% do total de exemplos de caracteres existentes.

Este procedimento de separação dos dez arquivos de dígitos existentes gerou dez arquivos com caracteres de teste e dez arquivos com caracteres de treinamento. Portanto, cada um dos dígitos possui seu próprio arquivo de treinamento e seu próprio arquivo de testes.

Tabela 4.2 – Distribuição de caracteres, para cada um dos dígitos, nos 10 arquivos do conjunto de treinamento.

Dígito	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
QC[1]	1494	2858	1160	1345	355	318	359	848	459	250	9446
PT[1]	15,81	30,25	12,28	14,23	3,75	3,36	3,80	8,97	4,85	2,64	100

Cada arquivo de treinamento ficou com uma quantidade diferente de caracteres porque, separado o número de caracteres para o teste, procuramos aproveitar o restante dos caracteres de cada um dos dígitos para o treinamento, pois quanto maior for o número de caracteres no conjunto de treinamento de um dígito, maior será a diversidade de formas que este dígito é escrito, e portanto, mais representativo será o padrão (regras de associação) extraído para esse dígito. A Tab 4.2 mostra a distribuição de caracteres, por dígito, no conjunto de treinamento.

4.3.2 Linearização das matrizes binárias

Durante a fase de treinamento, cada arquivo, contendo os caracteres devidamente preparados (fase de preparação dos dados), é treinado separadamente. Ao entrar no sistema, o arquivo de treinamento é transformado em uma matriz ‘M’, onde cada linha ‘k’ de ‘M’ representa um caractere existente na base de treinamento. Por exemplo, o arquivo de treinamento do dígito ‘0’ possui ‘1494’ caracteres (Tab 4.2), logo a matriz ‘M’ contém ‘1494’ linhas.

Cada linha ‘k’ de ‘M’ conterá ‘c’ colunas, sendo ‘c’ igual ao número de linhas de pixels do caractere digitalizado multiplicado pelo número de colunas de pixels. Como normalizamos todos os caracteres em um tamanho de 16 x 16 *pixels*, cada linha ‘k’ de ‘M’ possui 256 (16 multiplicado por 16) colunas.

A Fig 4.9 ilustra a transformação de um caractere, P(16x16) , do arquivo de treinamento do dígito ‘0’ em uma linha ‘k’ da matriz ‘M’.

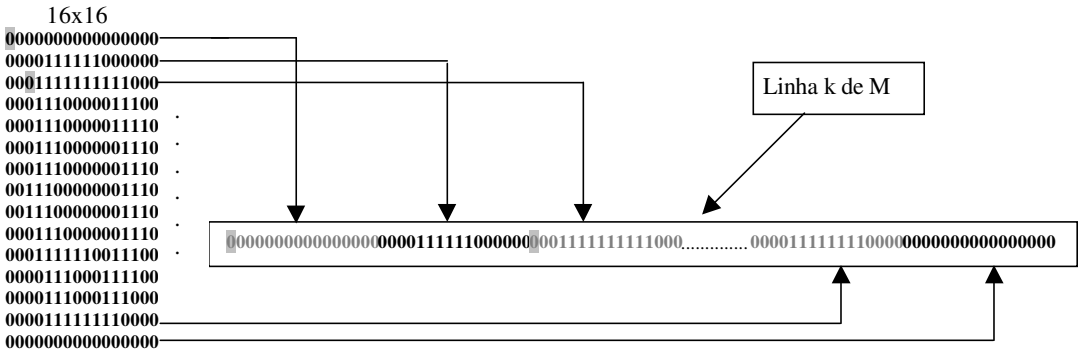

Figura 4.9 – Transformação de uma matriz 16 x 16 do conjunto de treinamento em uma linha ‘k’ da matriz ‘M’, com 256 colunas

Notemos que um pixel (ou posição) ‘P(i,j)’ do caractere da base de treinamento é mapeado na matriz ‘M’ na posição M(k,Pn), sendo Pn dado pela seguinte fórmula:

$$Pn = N(i-1) + j,$$

onde ‘N’ é o tamanho da normalização e ‘i’ e ‘j’ são respectivamente o número da linha e da coluna onde se encontra o pixel no caractere P(i,j).

Por exemplo, o caractere ‘0’, da base de treinamento, ilustrado na Fig 4.9, durante a linearização, foi inserido em uma linha ‘k’ da matriz ‘M’. A posição P(1,1), desse caractere foi mapeada para a seguinte posição (coluna) da linha ‘k’ de ‘M’:

$$Pn = 16(1-1) + 1 = 1$$

Do mesmo modo, a posição P(3,1), é mapeada para a posição 33, pois:

$$Pn = 16(3-1) + 1 = 33$$

[1] QC e PT – quantidade de caracteres e percentagem do total de caracteres, respectivamente.

Notemos que usamos para ‘N’, o valor ‘16’, porque normalizamos os caracteres do conjunto de treinamento em um quadrado 16 x 16.

4.3.3 Identificação dos vetores (linhas ‘k’ de ‘M’)

Com o objetivo de identificar qual o dígito é treinado, foram acrescidas aos vetores (linhas de ‘M’), as colunas ‘257’, ‘258’, ‘259’, ..., ‘266’. Assim, todos os vetores representando ‘0’s’, têm suas posições ‘257’ marcadas com ‘1’ e as demais, ‘258’ até ‘266’, marcadas com ‘0’. Já os vetores dos dígitos ‘1’s’, têm suas posições ‘258’ marcadas com ‘1’ e as posições ‘257’ e de ‘259’ até ‘266’ marcadas com ‘0’, e assim por diante.

A Fig 4.10 ilustra um trecho da matriz ‘M’, após a linearização do arquivo de treinamento do dígito ‘8’. O arquivo de treinamento contém 459 caracteres manuscritos que foram transformados em 459 vetores (linhas) da matriz ‘M’.

	Posições											
	1	**2**	**3**	**4**	**...**	**255**	**256**	**257**	**258**	**...**	**265**	**266**
1ª matriz 16x16	1	1	1	0	...	1	1	0	0	...	1	0
2ª matriz 16x16	1	1	1	0	...	1	1	0	0	...	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
459ª matriz 16x16	0	1	1	0	...	1	1	0	0	...	1	0

Figura 4.10 – Parte dos dígitos ‘8’ linearizados com a posição 265 marcada com ‘1’

As primeiras 256 posições de cada vetor representam um caractere (16 x 16) propriamente dito do conjunto de treinamento, e as outras 10 posições criadas são para identificar qual dígito o vetor está representando.

4.3.4 Extração de regras de associação

Analogamente ao exemplo do capítulo 3, em que se procurava extrair regras de associação a partir das vendas das mercadorias de uma farmácia (‘1’s’ e ‘0’s’), a idéia

aqui também é extrair regras de associação entre as posições de ‘1’s’ e ‘0’s’ nos vetores de caracteres.

No exemplo da farmácia, ao se buscar associações entre os itens ‘i’ (produtos) através do algoritmo apriori, geramos o conjunto ‘G’ dos grandes conjuntos, que é composto por todas as combinações dos ‘i’s’ itens existentes que aparecem no banco de dados com uma freqüência maior ou igual a um mínimo pré-estabelecido (suporte mínimo).

Para extrairmos o conjunto ‘G’ dos grandes conjuntos do dígito ‘0’, por exemplo, o sistema aplica o algoritmo apriori sobre os 1494 caracteres ‘0’s’ existentes (na verdade, 1494 vetores de ‘0’s’ e ‘1’s’). Em cada um dos vetores existentes temos 266 posições.

A Fig 4.11 ilustra parte desse conjunto G, após o treinamento do dígito ‘0’, utilizando um suporte mínimo de 0,80 (o que garante uma freqüência bem alta). Esse suporte de 0,80, significa que devem ser extraídas todas as combinações entre as 266 posições existentes, com uma freqüência maior que 80%.

G = { { 93_1 153_0 }, { 93_1 154_0 }, { 93_1 257 }, { 153_0 257 }
{ 154_0 257 }, { 169_0 257 },..., { 93_1 153_0 257 },
{ 93_1 154_0 257 }, ..., { 157_1 169_0 257 }, { 93_1 153_1 169_0 257 }

Figura 4.11 – Conjunto G de grandes conjuntos para a classe ‘0’.

Considere o grande conjunto { 93_1 153_0 } do conjunto G. Sua interpretação é de que as posições 93 e 153 são marcadas com ‘1’ e ‘0’, respectivamente, em no mínimo 80 % dos 1494 vetores para o dígito ‘0’, isto porque o suporte mínimo para a geração de elementos do conjunto G é 0,80.

É válido salientar que, no início de nosso trabalho, geramos regras de associação que consideravam as posições dos caracteres marcadas somente com ‘1’s’, ou seja, nos importávamos somente com o contorno dos caracteres. A partir de alguns testes realizados verificamos que é muito mais expressivo considerarmos também as posições dos caracteres marcadas com ‘0’s’, pois estas também contém semântica sobre os caracteres, tais como cavidades que porventura os mesmos tenham.

De posse do conjunto G, e utilizando o algoritmo da seção 3.3.2, extraímos as regras de associação que representam o dígito ‘0’. A Fig 4.12 ilustra algumas das regras geradas.

A leitura da regra { 157_1 169_0 } ⇒ 257, mostrada na FIG 4.12, é a seguinte: *se posição(157) = 1 e posição(169) = 0, então o dígito é ‘0’*, com uma freqüência mínima de 80%, considerando o conjunto de treinamento dado.

Notemos que todas as regras geradas possuem a posição ‘257’ no conseqüente da regra. Isto acontece porque estamos extraindo o padrão (regras de associação) do dígito ‘0’, e como vimos anteriormente, todos os 1494 vetores do dígito ‘0’ tiveram a posição ‘257’ marcada com ‘1’. Em consequência, o fator de confiança das regras é cem por cento (ver discussão a respeito, na seção 4.5).

Da mesma maneira, quando treinamos os caracteres do dígito ‘1’, geramos regras com a posição ‘258’ no conseqüente, quando treinamos os caracteres do dígito ‘2’, extraímos regras com a posição ‘259’ no conseqüente e assim sucessivamente.

{ 93_1 } ⇒ 257	{ 93_1 153_0 } ⇒ 257
{ 153_0 }⇒ 257	{ 93_1 154_0 } ⇒ 257
{ 154_0 } ⇒ 257	...
{ 169_0 } ⇒ 257	{ 157_1 169_0 } ⇒ 257
...	{ 93_1 153_0 169_0 } ⇒ 257

Figura 4.12 – Algumas regras geradas para o dígito ‘0’

Observando a Fig 4.11, notamos a presença de alguns grandes conjuntos sem o elemento ‘257’ como elemento, como por exemplo: { 93_1 153_0 }. Vale salientar que para este grande conjunto não foi gerada nenhuma regra porque ele não possui a posição ‘257’.

4.3.5 Etapa de refinamento

Em seguida à extração das regras para os dígitos, agregou-se ao sistema de geração de regras uma etapa de refinamento, a qual consiste na eliminação de regras de acordo com as seguintes heurísticas:

- ✓ Se o antecedente de uma regra 'Ri' está presente no antecedente de uma regra 'Rj', 'Ri' deve ser descartada (prevalece a regra mais específica);
- ✓ Quanto maior for o antecedente de uma regra 'Ri', maior será a probabilidade dela ser específica de apenas um dígito, menor a possibilidade de se ter mais de uma regra para dois dígitos diferentes.

Assim, por exemplo, a regra { 93_1 153_0 } ⇒ 257 tem uma expressividade muito maior do que a regra { 93_1 } ⇒ 257, sendo esta última, portanto, descartada.

As duas regras anteriores são do dígito '0'. Imaginemos um caractere '9'. É mais fácil este caractere conter a regra { 93_1 } ⇒ 266, do que ter a regra { 93_1 153_0 } ⇒ 266. Portanto, tendo-se regras mais específicas, a probabilidade de conflito de regras entre os diversos dígitos é bem menor.

Assim, o refinamento do conjunto de regras consistiu em *eliminar aquelas regras que são subconjunto de uma outra regra*. Para as regras da Fig 4.12, por exemplo, foram eliminadas as regras ilustradas na Fig 4.13:

{ 93_1 } ⇒ 257	{ 169_0 } ⇒ 257
{ 153_0 } ⇒ 257	{ 93_1 153_0 } ⇒ 257
{ 154_0 } ⇒ 257	

Figura 4.13 – Regras para o digito '0' eliminadas após o refinamento.

Observando a Fig 4.13, vemos que a regra { 93_1 } ⇒ 257 foi eliminada porque já estava presente na regra { 93_1 153_0 } ⇒ 257, isto é, era subconjunto desta última. O mesmo ocorrendo para as demais.

Após o refinamento das regras para cada um dos arquivos de treinamento existentes, a fase de treinamento termina, tendo como resultado final, o padrão (conjunto de regras de associação) para cada um dos dígitos.

4.4 Resultados do treinamento dos caracteres

Na Tab 4.3 temos o número de regras geradas para cada um dos dígitos, após o treinamento e refinamento das regras.

Observando a Tab 4.3, podemos verificar que, ao treinar o conjunto de caracteres do dígito '2', foi utilizado um supmin de 0,520, gerando um total de 266 regras para o padrão do dígito '2'.

Tabela 4.3 – suporte mínimo e número de regras geradas.

Dígito	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
supmin	0,775	0,988	0,520	0,675	0,540	0,640	0,690	0,810	0,578	0,760
Regras	255	226	266	272	264	268	262	269	267	262

Notemos que foram utilizados diferentes suportes mínimos para cada um dos dígitos. Isto aconteceu porque o dígito '1', por exemplo, tem 2858 caracteres muito semelhantes em sua base de treinamento. Tal semelhança provoca a geração de um conjunto G de grandes conjuntos muito grande para esse dígito e conseqüentemente, a extração de muitas regras. Assim, mesmo utilizando-se um suporte mínimo muito alto, ainda são extraídas muitas regras para o dígito '1', já que ocorre uma freqüência muito grande das mesmas posições na maioria dos vetores.

Para o dígito '2', acontece o contrário. A base de treinamento do '2' possui caracteres muito diferentes quanto a sua topologia. Portanto, se utilizado um suporte muito alto, como por exemplo 0,90, é gerado um conjunto G vazio, tornando impossível a geração de regras. Para enfrentar este problema foi necessário deixar o suporte do dígito '2' bem mais baixo que o do dígito '1', por exemplo.

Os suportes da Tab 4.3 foram escolhidos após vários treinamentos e de acordo com duas metas:

i. deixar o número de regras o mais próximo possível para todos os dígitos, a fim de que, no processo de teste do sistema, não haja um prejuízo ou favorecimento de um determinado dígito por possuir um número menor ou maior de regras de associação;

ii. forçar a geração de regras de tamanho maior objetivando capturar regras que trazem consigo maior informação sobre a topologia dos caracteres.

A Tab 4.4 mostra o total de regras geradas no treinamento de cada um dos dígitos, assim como ilustra também o tamanho das regras para cada um dos dígitos. O tamanho de uma regra é dado pela quantidade de condições existentes no antecedente da regra. Por exemplo, a regra { 93_1 153_1 169_0 } $\Rightarrow$ 257 possui tamanho igual a 3, pois existem 3 condições no antecedente (93_1, 153_1 e 169_0).

Procurou-se extrair, durante a fase de treinamento, um número maior de regras com um tamanho grande, porque regras com tamanho maior levam consigo maior expressividade. Com certeza, uma regra de tamanho 6 é muito mais representativa do que uma regra de tamanho 1, pois a primeira irá representar mais pontos do dígito do que a segunda.

Tabela 4.4 – Quantidade de regras x tamanho de regras para cada um dos dígitos

	Tamanho da Regra							
Dígito	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total**
0	-	1	13	162	39	23	17	**255**
1	-	-	56	171	-	-	-	**226**
2	-	16	142	103	5	-	-	**266**
3	-	8	117	89	52	6	-	**272**
4	-	7	101	105	42	9	-	**264**
5	-	3	122	129	14	-	-	**268**
6	-	8	37	127	59	31	-	**262**
7	-	-	80	126	63	-	-	**269**
8	-	23	187	56	1	-	-	**267**
9	-	-	65	79	117	6	-	**262**

Todas as regras da Tab 4.4 foram utilizadas para formar os padrões de cada um dos dígitos.

4.5 Discussão sobre o Fator de Confiança das regras

Até então vimos que as regras de associação foram geradas baseadas na freqüência com que as posições dos caracteres continham ‘0’s’ e ‘1’s’, ou seja, utilizamos a medida do suporte. Todavia, quando falamos das medidas estatísticas envolvidas no algoritmo apriori, no capítulo 3, citamos o fator de confiança como uma outra medida. Então por que ela não é utilizada em nosso trabalho para gerar as regras de associações bem como foi realizado com as regras para os produtos da farmácia ?

A resposta para esta questão é que, em nosso contexto, esta medida, que expressa a força de uma regra, tem valor constante igual a 1 (100%).

Exemplificando, imaginemos o elemento { 93_1 153_1 169_0 257 } da Fig 4.11. Este elemento gera a regra 93_1 153_1 169_0 $\Rightarrow$ 257 e só aparece no conjunto G de grandes conjuntos porque as posições ‘93, 153, 169, 257’ estão marcadas respectivamente com ‘1’, ‘1’, ‘0’ e ‘1’ e com uma freqüência maior que o suporte mínimo especificado.

Para medir o fator de confiança, expresso na equação 3.2, dividimos o número de registros onde aparecem todas as posições ‘93, 153, 169, 257’ (antecedente e conseqüente) marcadas com ‘1’, ‘1’, ‘0’ e ‘1’, respectivamente, pelo número de registros onde aparecem as posições ‘93, 153, 169’ com ‘1’, ‘1’, ‘0’ (antecedente), respectivamente. Como a posição ‘257’ está marcada com ‘1’ em todos os registros (para indicar que estamos treinando caracteres ‘0’) existentes, o valor desta divisão vai ser igual a 1. Por conseguinte, como só são geradas regras daqueles elementos do conjunto G dos grandes conjuntos que contém a posição ‘257’, generaliza-se o seguinte: em relação a fatores de confiança, no nosso contexto de reconhecimento de caracteres manuscritos, ele é sempre 1 (ou 100%). Na prática, isto significa que, em nosso estudo não é preciso se preocupar com fatores de confiança [Carvalho1999b].

Note que em nosso trabalho, o conseqüente da regra possui apenas um elemento, porque desejamos obter para cada dígito as regras que os representem univocamente. Não é de nosso interesse extrairmos regras com mais de um elemento no conseqüente, pois isto iria gerar conflitos no momento de decidir a qual dígito a regra pertence [Carvalho1999b].

4.6 O problema da explosão de elementos do conjunto G

Imaginemos um número qualquer de caracteres ‘0’s’ idênticos, selecionados para realizar o treinamento do dígito ‘0’. Todas as posições possíveis para se avaliar durante o treinamento são as seguintes: { {1_0}, {1_1}, {2_0}, {2_1}, {3_0}, {3_1}, ..., {255_0}, {255_1}, {256_0}, {256_1}, {257} }. Como estamos treinando o dígito ‘0’, as posições de 258 a 266 são deixadas de fora porque aqui não fazem sentido.

Portanto, inicialmente o conjunto G dos grandes conjuntos de tamanho ‘1’ tem o equivalente a 513 elementos (as primeiras 256 posições, ora marcadas com ‘0’, ora marcadas com ‘1’ mais a posição ‘257’ marcada com ‘1’, ou seja, 256*2+1 = 513).

Considerando supmin, o suporte mínimo utilizado para extrair as regras de associação, o conjunto G dos grandes conjuntos de tamanho ‘1’ ficará com 257 elementos. Isto acontece porque caso tenhamos o grande conjunto de tamanho ‘1’ {1_0}, com suporte maior que supmin (caso supmin seja maior que 0,5), certamente {1_1} terá um suporte menor que supmin não aparecendo, portanto no conjunto ‘G’ dos grandes conjuntos de tamanho ‘1’. Ou seja, se a posição ‘1’ dos caracteres ‘0’ de treinamento aparece marcada com um ‘0’, com uma freqüência maior que supmin, então a posição ‘1’ desses caracteres marcadas com um ‘1’ terá uma freqüência menor que supmin.

Logo, teremos 256 grandes conjuntos de tamanho ‘1’ ({1_0} ou {1_1}, {2_0} ou {2_1}, {3_0} ou {3_1}, ..., {255_0} ou {255_1}, {256_0} ou {256_1}) com um suporte maior que o supmin. Além desses 256 grandes conjuntos será adicionado ao conjunto ‘G’, o seguinte grande conjunto: {257}, pois trata-se do treinamento do dígito ‘0’.

Quando for gerado o possível conjunto G de tamanho 2, ele poderá ter a seguinte forma { {1_0,2_1}, {1_0,3_0}, {1_0,4_0}, ..., {1_0,256_1}, {1_0,257}, ..., {2_1,3_0}, {2_1,4_0}, ..., {2_1,256_1}, {2_1,257}, ..., {256_1,257} }.

Generalizando, a quantidade de elementos em um possível conjunto G de tamanho ‘n’ é uma combinação do número de elementos do conjunto G de tamanho 1 (*Ne*), tomados ‘n a n’.

Número de elementos do conjunto G de tamanho ‘n’ $\rightarrow C_{Ne,n} = Ne!/(Ne-n)!*n!$

Portanto o possível conjunto G de tamanho 2 teria, em nosso caso, 32.896 elementos, para o pior caso (supmin tão baixo a ponto de capturar todas combinações).

$$C_{257,2} = 257!/(257-2)!*2! = 32.896$$

Já para o possível conjunto G de tamanho igual a 8 teríamos 419.456.452.932.000 elementos ($C_{257,8} = 257!/(257-8)!*8!$), e isto levaria a um consumo de memória muito grande e a um acréscimo considerável no tempo utilizado para a geração de regras. Este número tende a crescer sempre em casos onde os exemplos de treinamento são muito semelhantes.

0000000000000000
0000000000000000
0000111111000000
0000111111110000
0000**11100111**1000
0001**11000011**1000
0011**10000001**1100
0011**10000001**1100
0011**10000001**1100
0111**00000001**1100
0111**00000001**1100
0011**10000011**1000
0001110111111000
0000111110000000
0000000000000000
0000000000000000

Figura 4.14 – Região contendo os ‘0’s’ que foram considerados na fase de treinamento

A fim de evitar esta explosão de elementos no conjunto G foram realizadas algumas adaptações em nosso sistema. Analisando-se os caracteres existentes na base de treinamento de todos os dígitos, verificamos que na maioria deles as linhas e as colunas 1, 2, 3, 14, 15 e 16 (linhas e colunas das bordas das matrizes) estão marcadas com 0, ou seja tais posições não possuem relevância para o sistema pois não acarreta em regras distintas para os dígitos. Logo, concluímos que as posições centrais marcadas com ‘0’ seriam de muito mais expressividade que as posições periféricas marcadas com ‘0’, já que estas últimas tendem a aparecer em todos os caracteres. Portanto, a fim de buscar regras mais expressivas e também diminuir o número de elementos possíveis em um

conjunto G de tamanho ‘k’, foram consideradas apenas as posições marcadas com ‘0’ pertencentes às linhas e colunas 5,6,7,8,9,10,11,12, formando o quadrado sombreado na Fig 4.14. Notamos, desde já, a dependência do algoritmo apriori em relação a bases de dados utilizada para o reconhecimento de caracteres manuscritos, assim como ocorre também nas redes neurais e nos algoritmos sintáticos.

Vale salientar que para as posições marcadas com ‘1’, foram consideradas todas as 16 linhas e 16 colunas dos caracteres. Pois, como já vimos, as posições marcadas com ‘1’ representam os contornos dos caracteres e, portanto, exercem uma grande relevância em qualquer linha que estejam situadas.

4.7 Fator de relevância de uma regra

Após a geração do arquivo de regras, notamos que, em alguns casos, um mesmo antecedente encontrava-se em regras de vários dígitos. Digamos que o antecedente {108_0 109_1} apareça nas regras tanto do numeral 7 como do 9, isto é, {108_0 109_1} $\Rightarrow$ 264 e {108_0 109_1} $\Rightarrow$ 266. Então, levamos em consideração que tal regra deveria ter sua força diminuída, pois consideramos que uma regra unívoca para um determinado dígito tem mais força que regras que apareçam em mais de um dígito. Para inserirmos esta informação na regra criamos uma medida chamada *fator de relevância (F) de uma regra Ri de um dígito Dj*, que é dada pela seguinte expressão:

$$F(Rij) = 1 - \frac{\sum_{k=0, k \neq j}^{k=|Q|-1} pik}{|Q|-1}$$

onde F(Rij) é igual ao fator de relevância de uma regra i do dígito j; |Q| é igual ao número de classes existentes (em nosso caso, são os dez dígitos – de 0 a 9), p_{ik} é igual a 1 se o antecedente da regra *Rij* está presente em alguma regra que classifique o dígito D*k*; p_{ik} = 0 em caso contrário. Notemos que, para o nosso estudo de caso, o denominador do cálculo do fator de relevância de uma regra ‘i’ de um caractere ‘j’ será sempre igual a 9 (10 - 1).

Para exemplificar, tomemos a regra 93_1 $\Rightarrow$ 257 da Fig 4.15, ela também aparece nas regras do 2 (93_1 $\Rightarrow$ 259) e do 9 (93_1 $\Rightarrow$ 266), portanto p_2 e p_9 são iguais a

1. Já p_1 e p_3 a p_8 possuem valor igual a 0. O valor de p_0 não é calculado pois estamos avaliando o fator de relevância de uma regra 'i' do dígito 'j', sendo j neste caso o '0'. Ao fazermos o somatório destes valores, o numerador resultante para a expressão é igual a 2. O denominador é constante: 10 – 1 = 9. Ao dividirmos, teremos o seguinte resultado: 2/9 que é igual a 0,22. Finalmente, de posse deste último valor podemos calcular o fator de relevância desta regra para o dígito '0': $F(Ri_0) = 1 - 0{,}22 = \textit{0,78}$.

Salientamos que de acordo com a fórmula do fator de relevância, uma regra só terá valor igual a 1 caso o seu antecedente não esteja repetido no antecedente de nenhuma outra regra para outro dígito. Em contrapartida, uma regra terá valor igual a 0 quando o seu antecedente estiver presente em algum antecedente das regras de todos os outros dígitos.

93_1 ⇒ 257 \$0,78\$	93_1 ⇒ 259 \$0,78\$
153_0 200_1 ⇒ 257 \$1,00\$	93_1 153_0 ⇒ 259 \$1,00\$
154_0 200_1 ⇒ 257 \$1,00\$	93_1 ⇒ 266 \$0,78\$

Figura 4.15 – Porção de regras geradas com F(Rij) entre o sinal \$.

Ao final da fase de treinamento, obtivemos então dez conjuntos de regras, os quais representam os dez padrões existentes, um para cada dígito. Estes conjuntos foram inseridos, seqüencialmente, em um arquivo geral de regras, ou seja, primeiramente as regras do dígito '0', depois as do dígito '1' até as regras do dígito '9'. Este arquivo geral será utilizado pelo sistema de testes para decidir sobre o reconhecimento de uma determinada matriz binária de entrada.

No próximo capítulo, será explicado como o sistema de testes foi projetado, como ele utilizou o fator de relevância das regras, além de fazermos uma extensa análise sobre os resultados obtidos em nossos experimentos. No Anexo III são apresentadas todas as regras geradas (padrões) para todos os dígitos, utilizando como suporte mínimo os valores da Tab.4.3.

Capítulo Cinco

Reconhecimento de Caracteres Manuscritos: Avaliação Experimental

Neste capítulo, descrevemos a etapa de testes de reconhecimento de caracteres numéricos manuscritos, com regras de associação. Os resultados são comparados com outros trabalhos existentes. Finalmente, discorremos sobre os pontos positivos e negativos de se implementar o reconhecimento de manuscritos utilizando a técnica de regras de associação.

5.1 Fase de Testes

Nesta etapa foram selecionados 266 exemplos de cada dígito, formando 10 arquivos diferentes que formam uma base de testes, a fim de verificarmos a eficiência do nosso sistema de reconhecimento de caracteres manuscritos. Escolhemos 266 exemplos, porque este número também é utilizado no trabalho de [Veloso1998], facilitando assim a comparação entre os resultados. É interessante reforçar que estes exemplos são distintos dos caracteres da base de treinamento, com o propósito de tornar os resultados mais semelhantes a uma situação real de aplicação do sistema.

Os dez arquivos coletados foram inseridos em um único arquivo contendo 2660 caracteres (daqui por diante denominado *arquivo de teste*), assim distribuídos: nas primeiras 266 posições, ou seja, da 1ª a 266ª, matrizes com '0's', nas posições de 267ª a 532ª caracteres, matrizes com '1's' e assim sucessivamente, até a posição 2660ª, a qual conterá uma matriz do dígito '9'.

5.2 O funcionamento do sistema de teste

O resultado da fase de treinamento, como vimos no capítulo anterior, é um arquivo contendo os padrões de cada um dos caracteres treinados, ou seja, um arquivo

contendo todas as regras extraídas, cada qual relacionada com o seu próprio fator de relevância. Chamaremos esse arquivo de regras de 'AR', daqui por diante.

Cada caractere contido no arquivo de teste, ao entrar no sistema é, primeiramente, transformado em um vetor linearizado com 256 posições, onde cada vetor desse representa uma linha 'k' de uma matriz 'M'. Como o arquivo de teste montado possui 2660 caracteres, o nosso sistema cria uma matriz 'M' com 2660 linhas 'k'. Lembremos aqui que utilizamos o mesmo procedimento realizado na fase de treinamento dos caracteres. A partir desse momento, pode-se dar início ao processo que decidirá a que algarismo pertence cada caractere de entrada.

O processo é iniciado lendo-se uma linha 'k' (caractere de teste linearizado) da matriz 'M' (conjunto de todos os caracteres de teste) e colocando-a na entrada do sistema. Faz-se então uma varredura de 'AR' e para cada regra deste arquivo é verificado se a mesma está presente na linha 'k' que está na entrada. Caso esteja, o valor do fator de relevância da regra em questão é somado a 'Chance' do conseqüente desta regra (sobre o conceito de Chance, ver adiante).

Imaginemos um vetor (linha 'k' de 'M') referente a um caractere que foi linearizado que esteja presente na entrada do sistema. Vamos supor que, ao varrer todas as regras de AR, apenas as da Fig 5.1 combinam com as regras existentes neste vetor de entrada.

93_1 ⇒ 257 $0,78$	93_1 ⇒ 259 $0,78$
153_0 200_1 ⇒ 257 $1,00$	93_1 153_0 ⇒ 259 $1,00$
154_0 200_1 ⇒ 257 $1,00$	93_1 ⇒ 266 $0,78$

Figura 5.1 – Regras com fatores de relevância computados no vetor chance

Observando a Fig 5.1, constatamos que a regra {154_0 200_1 ⇒ 257} foi encontrada no caractere de teste. A interpretação para este fato é que o caractere de entrada possui as posições '154' e '200' marcadas com '0' e '1', respectivamente. E tal regra é uma das regras que formam o padrão do dígito '0', pois tem a posição '257' no conseqüente.

Do mesmo modo, este caractere de entrada possui as posições '93' e '153' marcadas com '1' e '0', respectivamente (regra {93_1 153_0 ⇒ 259}). Entretanto, essa

regra faz parte do padrão do dígito ‘2’, pois possui como conseqüente a posição ‘259’. Portanto, já que esse caractere de entrada possui em sua topologia regras referentes tanto ao padrão do dígito ‘0’ quanto ao padrão do dígito ‘2’ e também ao padrão do dígito ‘9’ (regra {93_1 ⇒ 266}), como decidir a que classe (dígito) pertence este caractere de entrada.

Com o objetivo de decidir a que classe pertence um caractere de entrada, implementamos o conceito de um vetor denominado ‘*chance*’. Esse vetor computará, para cada uma das classes de dígito existentes, a soma dos *fatores de relevância* de cada uma das regras de AR que combinem com o caractere de entrada. Assim, quando uma regra ‘r’ de AR, pertencente ao padrão de um dígito ‘d’, estiver presente no caractere de entrada, o fator de relevância desta regra será acrescentado à probabilidade (chance) do caractere de entrada representar o dígito ‘d’.

Inicialmente, criamos o vetor denominado ‘c*hance*’ com dez posições, marcado inicialmente com o valor ‘0’ em todas as suas posições. Desta forma, este vetor será responsável pelo armazenamento da Chance do caractere de entrada representar um ‘0’, ou um ‘1’, ou um ‘2’,..., ou um ‘9’ (classes existentes de acordo com a base de dados do Cenparmi).

Vamos supor que a regra { 93_1 ⇒ 257 $0,78$ } tenha sido encontrada no vetor de entrada, então a posição ‘0’ do vetor chance é incrementada com o valor do fator de relevância desta regra:

$$chance[0] = chance[0] + 0,78.$$

Por que incrementar a posição ‘0’ do vetor chance? Porque a regra {93_1 ⇒ 257 $0,78$} que ‘casou’ (combinou) com o caractere de entrada tem como conseqüente a posição ‘257’, regra esta, portanto, gerada durante a fase de treinamento do dígito ‘0’, regra que pertence ao padrão do dígito ‘0’.

Supondo ainda que exista uma outra regra do ‘0’ a {153_0 200_1 ⇒ 257 $1,00$} que esteja presente no caractere de entrada, o vetor chance então é atualizado novamente na posição ‘0’, resultando no seguinte:

$$chance[0] = chance[0] + 1,00 = chance[0] = 1,78$$

Estes cálculos são realizados analisando-se todas as regras de AR que combinam com o vetor de entrada. Após varrer AR, temos , para o primeiro caractere de teste, o vetor chance com os seguintes valores:

chance[0] = 1,78	chance[2] = 0,78	chance[4] = 0	chance[6] = 0	chance[8] = 0
chance[1] = 0	chance[3] = 0	chance[5] = 0	chance[7] = 0	chance[9] = 0,78

Observando-se o vetor chance acima gerado, concluímos que o caractere de teste atual é um '0', pois o valor da posição '0' no vetor chance é maior que todas as outras posições existentes. Portanto, podemos afirmar que, em nosso sistema, a soma dos fatores de relevância das regras com posições coincidentes as do caractere de entrada, determinará a classe do mesmo.

5.3 Conjunto de posições efetivas na fase de teste

Recordemos que as matrizes binárias de entrada têm tamanho igual a 16 linhas por 16 colunas. Assim como foi realizado nos caracteres utilizados na fase de treinamento (Fig 4.14), durante a fase de testes, foram consideradas somente as posições marcadas com '0', pertencentes às linhas e às colunas de 5 a 12. Portanto, as posições marcadas com '0', que se encontravam nas linhas e colunas de 1 a 4 e de 13 a 16 foram eliminadas dos caracteres de teste.

5.4 Análise dos resultados dos testes

Na Tab 5.1, estão ilustrados os resultados obtidos durante a fase de testes. Ela contém a informação de quanto porcento dos caracteres de um determinado numeral foram reconhecidos corretamente, bem como os conflitos ocorridos no reconhecimento destes dígitos.

Por exemplo, dos 266 caracteres de teste que representam o numeral '6', 80,83% foram reconhecidos como '6', isto é, adequadamente. Os 19,17% dos caracteres restantes não foram reconhecidos corretamente, ou seja, foram confundidos com outros dígitos. Verificando estes conflitos do numeral '6', notamos que 8,27% dos caracteres foram confundidos com o numeral '5', 6,39% foram reconhecidos como sendo o dígito '8', 2,63% confundidos com o dígito '1', 1,13% com o dígito '3' e 0,38% com o dígito '0' e com o dígito '2'.

Tabela 5.1 – Resultado dos testes realizados com 266 caracteres de cada dígito.

	NUMERAIS DE TESTE									
REC[1]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**94,74**	0,00	2,63	1,50	3,01	1,13	0,38	0,38	0,75	0,00
1	0,00	**99,25**	11,65	0,00	8,65	3,01	2,63	0,38	3,76	2,26
2	0,75	0,75	**64,29**	12,78	5,64	1,13	0,38	8,65	2,63	3,76
3	3,38	0,00	1,50	**72,93**	1,88	25,94	1,13	2,63	5,64	1,88
4	0,38	0,00	4,14	0,75	**46,99**	1,50	0,00	1,50	2,63	9,77
5	0,00	0,00	0,75	1,13	0,38	**59,77**	8,27	1,13	3,01	1,88
6	0,75	0,00	0,38	0,00	1,50	4,14	**80,83**	0,00	3,01	2,63
7	0,00	0,00	12,41	9,02	4,51	0,75	0,00	**80,45**	7,14	18,80
8	0,00	0,00	1,50	1,13	0,38	2,63	6,39	0,00	**68,80**	2,63
9	0,00	0,00	0,75	0,75	27,07	0,00	0,00	4,89	2,63	**56,39**

Observando-se ainda a Tab 5.1, constatamos excelentes resultados do sistema de regras de associação no reconhecimento dos dígitos ‘0’ e ‘1’ e bons resultados para os dígitos ‘6’ e ‘7’. Isto acontece porque os caracteres de tais dígitos não apresentam grandes variações em sua topologia nos conjuntos de treinamento e teste. Devido a este fato, foi possível gerar, durante a fase de treinamento do ‘1’, um bom número de regras que aparecem com grande freqüência (suporte alto) também nos dígitos de teste.

Analisando-se a coluna do numeral ‘5’, notamos que, embora a maioria dos caracteres ‘5’ de entrada (59,77% dos 266 caracteres de teste) foi reconhecida corretamente, há uma grande proporção (25,94%) destes caracteres que foram reconhecidos como se fossem o numeral ‘3’. Isto acontece devido a existência de uma grande quantidade de caracteres de teste do dígito ‘5’ que carregam consigo mais regras do numeral ‘3’ que do próprio ‘5’. A Fig 5.2 ilustra um exemplo de um numeral ‘5’ que foi reconhecido incorretamente como ‘3’.

Este reconhecimento errôneo deve-se ao fato do numeral ‘5’ da Fig 5.2 conter um traçado (que está sombreado na figura) muito parecido com os contornos verificados na maioria dos caracteres do numeral ‘3’. Ou seja, o suporte que utilizamos para o numeral ‘5’, 65% (vide Tab 4.3), na fase de treinamento, não foi um valor suficiente para capturar muitas regras que aparecem no traçado sombreado da Fig 5.2 desta matriz

[1] - Dígito reconhecido.

de teste. Ao contrário, o suporte para o dígito ‘3’, 67,5% foi suficiente para extrair as regras do contorno sombreado da Fig 5.2. Desta forma, quando esta matriz ‘5’ de teste entrou no sistema para ser reconhecida acabou combinando um número de regras maiores para o dígito ‘3’, ao invés do ‘5’, fazendo com que a posição ‘3’ do vetor chance ficasse com um valor maior que a posição ‘5’, ocasionando o reconhecimento errado desta matriz.

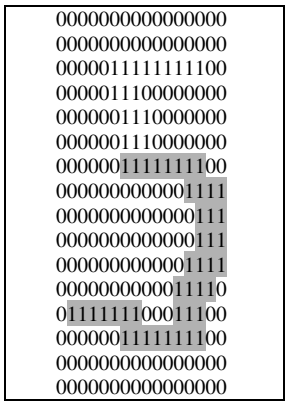

Figura 5.2 – Matriz binária do dígito ‘5’ reconhecida como numeral ‘3’.

Assim, outras matrizes de teste do dígito ‘5’ que possuem um contorno idêntico ou muito parecido ao da Fig 5.2. foram classificadas de forma equivocada.

Pelo mesmo motivo, verificamos um conflito no reconhecimento do numeral ‘4’, no qual uma proporção muito grande (27,07%) dos caracteres foi reconhecida como se fosse um dígito ‘9’.

Reparemos agora como este problema persiste também entre os dígitos ‘9’ e ‘7’, isto é, uma enorme quantidade de caracteres ‘9’ (18,80%) é reconhecida como ‘7’. A seguir a Fig 5.3 ilustra este conflito, por meio de uma análise sobre os contornos do numeral ‘7’ (a) e do numeral ‘9’ (b) e (c).

A Fig 5.3 (a) ilustra uma matriz de um dígito 7. Como a maioria dos ‘7’s’ da base de treinamento são semelhantes eles acabam gerando muitas regras contendo as posições da região sombreada da figura.

Agora, se notarmos as Fig’s 5.3 (b) (c) que mostram dois caracteres ‘9’, atentamos para o fato de que os mesmos não são semelhantes porque possuem suas posições marcadas com ‘1’s’e ‘0’s’ em locais bem diferentes. Esta falta de semelhança entre os dígitos ‘9’s’ é comum, impossibilitando a geração de regras extremamente

confiáveis, durante a fase de treinamento, ao contrário do que acontece com o dígito ‘7’. Recorde-se que, tanto o suporte mínimo para as regras do dígito '9' (0,76), quanto o suporte mínimo para as regras do dígito '7' (0,81), são razoavelmente altos e não muito díspares, o que permite ao sistema reconhecer, às vezes, o '9' como '7'.

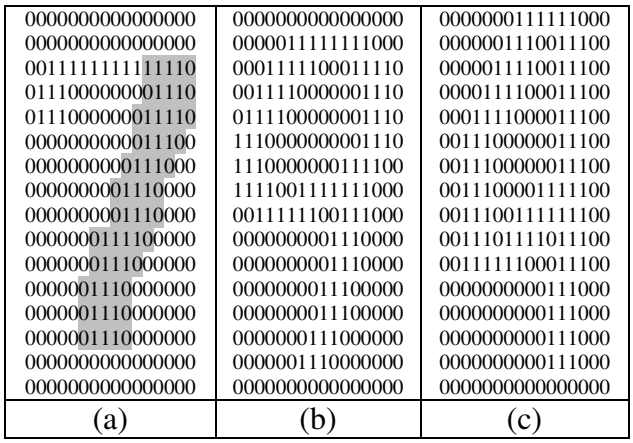

Figura 5.3 – Conflito de regras entre os caracteres

Existem alguns ‘9’s’, tal como o ilustrado na Fig 5.3 (b), que se assemelham à região sombreada da Fig 5.3 (a) e quando submetidos ao teste acabam combinando suas regras com mais regras do ‘7’ do que do próprio ‘9’, gerando um valor para a Chance de ‘7’ maior que para a Chance de 9, provocando um erro de avaliação.

A taxa de reconhecimento do numeral ‘4’ (regras com o mais baixo suporte) foi muito baixa em relação às demais, apenas 46,99% dos caracteres de teste foram reconhecidos corretamente. Após uma investigação nos caracteres de teste e de treinamento do numeral em questão, ressaltamos uma diversidade topológica gigantesca entre a maioria destes dígitos, sendo esta a razão principal pela baixa taxa de reconhecimento. As Fig’s 5.4 (a) (b) (c) (d) e (e), ilustram cinco exemplos de caracteres ‘4’, extremamente diferentes um do outro, que aparecem em nossa base de dados. Esta grande diversidade topológica entre os caracteres do numeral ‘4’ impossibilita a geração de regras relevantes, isto é, regras que aparecem com grande freqüência para o numeral. Assim, durante a fase de treinamento, utilizou-se um suporte mínimo muito baixo, que provocou a geração de regras pouco relevantes, regras estas que acabam não sendo suficientes para identificar univocamente o dígito ‘4’.

0000000000011100	0000000011100000	0000000000000000	0000000000000000	0000000000111000
0000000000011100	0000000011100000	0000000000001110	0000001100000000	0000000000111000
0011100000011100	0000000011100000	0000000000011100	0000001110000000	0000000111111000
0011100000011100	0000000011100000	0111000000011100	0000001110000000	0000011110111000
0011100000011100	0000000011100000	0111000000111000	1110001110000000	0000111000111000
0011100000011100	1110000011100000	0011100000111000	1110001110000000	0001110000111000
0011100000011100	1110000011100000	0011111111110000	1110000111000000	0001110111111100
0011101111111100	1110000001110000	0000000001110000	1110000111100000	0011111110001110
0001111100011100	1111000001110000	0000000011100000	1110000011100000	0000000000001110
0000000000011100	0111111111110000	0000000011100000	1110000011100000	0000000000001110
0000000000111000	0000111110111000	0000000011100000	1110001111100000	0000000000001110
0000000000111000	0000000000111000	0000000111000000	0111011101110000	0000000000001110
0000000000111000	0000000000111000	0000000111000000	0111011100111000	0000000000001110
0000000000111000	0000000000011100	0000001110000000	0011111000111100	0000000000011110
0000000000111000	0000000000001100	0000000000000000	0000000000000000	0000000000011100
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
(a)	(b)	(c)	(b)	(c)

Figura 5.4 – Exemplos de caracteres ‘4’ com diferenças topológicas entre os mesmos

Uma outra razão para o baixo reconhecimento do ‘4’ é que muitos caracteres foram reconhecidos como ‘9’ pelo mesmo motivo que os caracteres ‘5’ foram confundidos com os caracteres ‘3’ e os caracteres ‘9’ confundidos com os caracteres ‘7’, como explicado anteriormente.

O problema dos caracteres que não são reconhecidos corretamente, de forma geral, é determinado pela grande dependência do sistema com relação às posições dos ‘0’s’ e ‘1’s’ na matriz do numeral, ou seja, o padrão de um dígito é determinado pelas regras geradas durante o treinamento e baseiam-se nas posições dos 1’s e 0’s. Portanto, se um dígito de teste estiver um pouco deslocado ou um pouco diferente do normal (o que é comum em nossa base de testes), o sistema encontrará dificuldades em fazer o reconhecimento [Carvalho1999a]. Esta constatação pode ser verificada através da Fig 5.4, observando-se as diversas posições que os contornos dos caracteres ‘4’ ocupam na matriz 16 x 16.

A conseqüência crucial desta diversidade topológica de caracteres de um mesmo algarismo é que fomos obrigados a gerar regras durante a fase de treinamento com um valor de suporte mínimo muito baixo para os dígitos ‘2’, ‘4’ e ‘8’ a fim de capturarmos algumas regras. Desta forma, as regras extraídas acabam não sendo muito relevantes. Exemplificando, uma regra extraída com suporte de 53%, como ocorre no dígito ‘2’, aparece apenas em 53% dos caracteres existentes na base de treinamento o que é pouco expressivo.

Ao contrário, a maioria das regras do dígito ‘1’ possui suporte de 99%, o que quer dizer que, no mínimo, em 99% dos caracteres de treinamento, essas regras

aparecem. Logo, se tais regras aparecem em 99% dos caracteres de treinamento é muito provável que ocorram também nos caracteres de teste, justificando assim a excelente taxa de acerto do dígito '1'.

Embora o suporte utilizado na extração de regras durante a fase de treinamento dos dígitos '9' e '5' sejam mais altos que os dos numerais '2', '4' e '8', eles também não obtiveram resultados muito bons. Isto se deve às semelhanças topológicas já comentados anteriormente com os caracteres dos numerais '7' e '3' respectivamente (Fig 5.2 e Fig 5.3).

5.4.1 Análise da quantidade de regras geradas na fase de treinamento

A Tab 4.4, ilustrada no capítulo anterior, mostra o número de regras geradas durante a fase de treinamento para cada um dos dígitos. O resultado do teste efetuado, mostrado na Tab 5.1, foi realizado sobre esse conjunto de regras que contém em média 261 regras para cada um dos dígitos.

Parece provável que uma maneira de solucionar os problemas que ocasionam um reconhecimento ruim para alguns dígitos, seria gerar padrões (regras de associação) mais representativos para cada um dos dígitos.

Mas, o que seriam padrões mais representativos para um dígito? Em nosso contexto, esses padrões seriam regras com um antecedente muito grande. Imaginemos uma regra 'r1' de tamanho 'x' e uma regra 'r2' de tamanho 'x+y', sendo ambas parte do padrão do dígito 'd1'. A probabilidade de 'r1' aparecer no padrão do dígito 'd2' ('d1'≠ 'd2') é maior que a probabilidade de 'r2' ser encontrada no padrão de 'd2'. Portanto, um caractere de teste 'c', ao entrar no sistema, contendo a regra 'r1', pode ter conflito no seu reconhecimento, pois 'r1' tem maiores chances que 'r2' de fazer parte do padrão tanto de 'd1' quanto de 'd2'.

Assim, para se obter regras de maior expressividade, durante a fase de treinamento, é necessário aumentar o suporte para a extração das regras. Desta forma, o número de regras gerado seria menor, porém as regras mais representativas. Consequentemente, mais difícil de entrar em conflito com regras de outros dígitos. Aumentando-se o suporte para extração de regras, estamos capturando regras que estão presentes em muitos caracteres de treinamento do mesmo dígito, ou seja, aparecem com muita freqüência, expressando, portanto, uma relevância muito maior.

Dígito	**...**	**2**	**...**	**7**	**...**	**9**
Regras	**...**	35_1 ⇒ 259 $1,00$	**...**	51_1 ⇒ 264 $1,00$	**...**	47_0 ⇒ 266 $1,00$
	...	37_1 ⇒ 259 $1,00$	**...**	56_0 ⇒ 264 $1,00$	**...**	137_0 ⇒ 266 $1,00$
	...	38_1 ⇒ 259 $1,00$	**...**	91_1 ⇒ 264 $1,00$	**...**	139_1 ⇒ 266 $1,00$
	...	54_0 ⇒ 259 $1,00$	**...**	93_1 ⇒ 264 $1,00$	**...**	140_1 ⇒ 266 $1,00$

Figura 5.5 – Problema da geração de regras com suporte muito alto

Infelizmente e contrariando nossas previsões iniciais, a solução de aumento de suporte não se mostrou eficiente, porque primeiro, como já foi dito, existe uma diversidade enorme entre os caracteres existentes. Portanto, o dígito '4', tomando-se como exemplo, tanto na base de treinamento quanto na base de testes possui caracteres de diversos formatos. Ao aumentar-se demais o suporte para este dígito, 80% por exemplo, nenhuma regra é gerada, pois não há posições que aparecem com uma freqüência de no mínimo 80% dos caracteres de treinamento existentes.

Assim, tentemos baixar o suporte lentamente de modo a capturar poucas regras para os dígitos. Seguindo esta linha de raciocínio nos confrontamos com o seguinte problema. Pelo motivo de os caracteres existirem das mais variadas formas, as regras capturadas para a maioria dos dígitos (exceto os dígitos '0', '1' '6' e '7'), durante a fase de treinamento, são regras pequenas, de tamanho 1 (com uma posição no antecedente) e no máximo de tamanho 2 (com duas posições no antecedente). Consequentemente, como as regras têm um antecedente pequeno, é muito mais fácil que uma matriz de entrada qualquer contenha em suas posições regras de vários dígitos diferentes, gerando muitos conflitos.

A Fig. 5.5 ilustra este problema. Nessa figura, podemos observar que ao ser avaliada, a matriz de teste do dígito 7 combina com três regras pertencentes ao padrão

do dígito ‘2’, com três regras do padrão do dígito ‘7’ e também com três regras do padrão do dígito ‘9’ (regras sombreadas) gerando o mesmo valor no vetor chance para estes três dígitos, impedindo uma conclusão sobre esta matriz de entrada. Poderia acontecer também de, por exemplo, chance[2] ou chance[9] ficar com o valor maior que chance[7], bastando para isso que uma das regras do padrão do dígito ‘7’ não combinasse com a matriz de entrada, o que é muito fácil de acontecer devido às diversas formas dos caracteres de entrada.

Para que possamos capturar regras com antecedentes maiores é preciso baixar mais o suporte. Contudo, se optarmos por baixar demais o suporte, a fim de gerarmos regras com um antecedente grande e por conseqüência regras bastante significativas, extraímos também inúmeras regras de tamanho pequeno (antecedente igual a 1 e 2), ou seja, com pouca expressividade. Tais regras aumentam a contribuição para que haja conflitos. Além disso, se baixarmos o suporte em demasia, construímos um padrão, com regras que aparecem com pouca freqüência na base de treinamento, não constituindo, portanto, um padrão confiável.

Desta forma, realizamos diversos treinamentos para cada um dos dígitos, utilizando suportes mínimos diversos, extraindo um conjunto de 10 padrões (padrão do dígito ‘0’, padrão do dígito ‘1’,..., padrão do dígito ‘9’) para cada treinamento. Após gerarmos algumas dezenas de conjuntos de 10 padrões, utilizamos a base de testes para verificarmos qual desses conjuntos se comportaria melhor, ou seja, qual deles seria responsável por uma taxa de acerto maior dos caracteres de teste.

O teste que apresentou a melhor taxa de acerto, foi aquele onde utilizamos os suportes mínimos ilustrados na Tab 4.3 para gerar o conjunto de 10 padrões. É importante salientar aqui que após toda a variação de suportes, notamos que os melhores resultados dos testes foram obtidos deixando o número de regras geradas para cada um dos dígitos em torno de 250 regras durante a fase de treinamento.

O maior problema que o nosso sistema enfrentou com a base de dados utilizada é a enorme falta de similaridade entre os caracteres de um mesmo dígito. Depois de uma análise minuciosa nos caracteres da base de dados do Cenparmi, pôde-se constatar que os caracteres dos dígitos ‘0’, ‘1’, ‘3’, ‘6’ e ‘7’, os quais obtiveram resultados mais significativos em nossos testes são aqueles que demonstram uma maior similaridade entre os caracteres do mesmo dígito. Uma constatação da maior

similaridade entre estes dígitos foi o suporte mais alto que utilizamos para a extração dos padrões de cada um desses dígitos, durante a fase de treinamento do sistema.

Observando-se a Tab 5.1 constatamos excelentes resultados do sistema de regras de associação no reconhecimento dos dígitos '0' e '1'. Isto acontece porque tais dígitos não apresentam grandes variações nos conjuntos de treinamento e teste. Por este motivo, durante a fase de treinamento do '0' e do '1' são geradas um número de regras que aparecem com enorme freqüência nestes caracteres e que aparecem também nos caracteres de teste.

5.5 Comparação dos nossos resultados com trabalhos de análise sintática

Utilizamos os resultados obtidos em nosso melhor teste (Tab 5.1) para compará-los com outros trabalhos. Na Tab 5.2 são mostrados os resultados do nosso experimento, e sua comparação com os resultados dos testes dos algoritmos de [Gomes1994] e [Veloso1998], que utilizam a tecnologia de análise sintática. Ambos os algoritmos de análise sintática foram utilizados em [Veloso1998] para reconhecer caracteres manuscritos da base de dados do Cenparmi.

Voltando à Tab 5.2, os resultados ilustram a percentagem de caracteres que foram reconhecidos corretamente durante a fase de testes, por diferentes técnicas. Por exemplo, o valor 94,74%, da quarta coluna, significa que 94,74% dos 266 caracteres '0' testados foram reconhecidos pelo sistema de regras de associação como '0', ou seja, corretamente. Os 5,26% caracteres restantes não foram reconhecidos corretamente, isto é, foram confundidos com os outros numerais existentes (ver Tab 5.1).

Fazendo-se uma análise comparativa entre os resultados do nosso trabalho e do algoritmo de [Gomes1994] percebe-se claramente que o reconhecimento de caracteres numéricos manuscritos utilizando regras de associação mostrou-se muito mais eficaz que o algoritmo sintático, pois para todos os dígitos em nosso trabalho obtivemos um reconhecimento com um percentual bem maior.

O algoritmo de [Gomes1994] foi concebido para reconhecer caracteres de teste de uma base de dados que continha algarismos provindos de cheques bancários, onde geralmente as pessoas procuram escrever os dígitos com um maior cuidado. Em [Veloso1998], este algoritmo é usado para fazer reconhecimento de manuscritos com uma base de dados (base do Cenparmi) com maior diversidade nas formas topológicas dos caracteres, apresentando resultados muito ruins (ilustrados na 1ª coluna da Tab 5.2).

Tabela 5.2 – Comparação dos resultados do nosso trabalho com análise sintática (em percentagem)

Numeral	Algoritmo sintático de [Gomes1994]	Algoritmo sintático de [Veloso1998]	Regras de associação	Melhor Desempenho
0	71,80	71,80	94,74	**Regras de associação**
1	92,48	88,34	99,25	**Regras de associação**
2	18,04	68,04	64,29	Algoritmo sintático
3	18,42	58,64	72,93	**Regras de associação**
4	23,30	68,42	46,99	Algoritmo sintático
5	0	65,03	59,77	Algoritmo sintático
6	0,37	63,15	80,83	**Regras de associação**
7	4,13	67,29	80,45	**Regras de associação**
8	45,48	58,64	68,80	**Regras de associação**
9	30,45	54,13	56,39	**Regras de associação**
Média	**30,45**	**66,35**	**72,44**	

A 3ª coluna da Tab 5.2 mostra as taxas de acerto do algoritmo sintático desenvolvido por [Veloso1998]. Esse algoritmo foi implementado com base no algoritmo de [Gomes1994]. Entretanto, em [Veloso1998], foi realizada uma inspeção visual nos caracteres de treinamento do Cenparmi a fim de identificar características peculiares para cada um dos dígitos e construir um algoritmo com maior eficácia. Ao compararmos os resultados do algoritmo sintático desenvolvido por [Gomes1994] e por [Veloso1998] verificamos um aumento significativo de qualidade deste último.

Confrontando os resultados obtidos por [Veloso1998] com o sistema baseado em regras de associação vemos que este último obteve um melhor desempenho para os dígitos '0,1,3,6,7,8,9'. Contudo, não conseguiu reconhecer melhor os numerais '2,4 e 5'. O algoritmo sintático conseguiu capturar melhor as características desses três numerais.

Todavia, é esperado que o algoritmo sintático implementado por [Veloso1998] não obtenha resultados muito satisfatórios em bases diferentes, assim como o algoritmo

de [Gomes1994] não conseguiu bons resultados com a base do Cenparmi. Isso ocorre porque o algoritmo sintático de [Veloso1998] é estritamente dependente da topologia dos caracteres encontrados na base do Cenparmi, já que as decisões tomadas no algoritmo foram concebidas observando-se as características topológicas dos caracteres desta base de dados.

Observando-se a média de acertos da Tab 5.1, notamos que o sistema baseado em regras de associação mostrou-se claramente mais eficaz que os sistemas de reconhecimento baseados em análise sintática.

Além da dependência da base de dados utilizada, o reconhecimento de caracteres manuscritos a partir da análise sintática depende essencialmente de um especialista humano, o qual deve ser capaz de perceber as diversas características da geometria dos caracteres para poder assim formar um algoritmo sintático.

Ao contrário, os sistemas de reconhecimento de manuscritos baseados em regras de associação independem tanto de um especialista humano quanto da base de dados utilizada. As características dos dígitos são extraídas automaticamente durante a fase de treinamento, através da geração das regras de associação. Os sistemas que possuem uma fase de treinamento automática mostram-se mais eficazes, de maneira geral, que o sistema de análise sintática. Entretanto, tal como os algoritmos de análise sintática, o nosso algoritmo baseado em regras de associação também é extremamente dependente da natureza dos conjuntos de treinamento e de teste. Além disso, bem como os algoritmos de análise sintática e redes neurais, durante a etapa de preparação dos dados, o sistema de regras de associação depende de um especialista humano e/ou da base de dados utilizada.

5.6 Comparação dos nossos resultados com redes neurais

A Tab 5.3 ilustra a comparação entre os resultados do nosso trabalho com os resultados obtidos por [Veloso1998], utilizando redes neurais. Os resultados deste trabalho foram conseqüência de um treinamento realizado com 4000 caracteres e testado com 2660 caracteres, utilizando-se uma rede neural multicamada com propagação direta, com um parâmetro (erro médio) para interromper o treinamento igual a 0,001.

Fazendo-se uma análise comparativa entre o nosso sistema e esse sistema de reconhecimento baseado em redes neurais, chegamos a conclusão que apenas os dígitos

‘0,1 e 7’ obtiveram um reconhecimento mais eficaz através do uso de regras de associação. Para os demais dígitos, a rede neural obteve um melhor desempenho.

Tabela 5.3 – Comparação dos resultados do nosso trabalho com redes neurais (em percentagem)

Numeral	Redes neurais	Regras de associação	Melhor Desempenho
0	90,94	94,74	**Regras de associação**
1	96,60	99,25	**Regras de associação**
2	83,02	64,29	Redes neurais
3	88,67	72,93	Redes neurais
4	96,60	46,99	Redes neurais
5	77,36	59,77	Redes neurais
6	89,06	80,83	Redes neurais
7	68,31	80,45	**Regras de associação**
8	84,31	68,80	Redes neurais
9	71,70	56,39	Redes neurais
Média	**84,72**	**72,44**	

Estes melhores resultados da rede neural refletem que o treinamento da mesma mostrou-se mais eficaz, ou seja, gerou padrões, através do ajuste dos pesos, mais significativos e confiáveis que os padrões (regras) gerados pelo sistema de regras de associação.

O sistema baseado em redes neurais mostrou ser muito menos suscetível à diversidade topológica dos caracteres existentes na base do Cenparmi, do que o sistema de regras de associação. Quando os caracteres de treinamento são muito diferentes entre si, o sistema de regras de associação tende a gerar padrões pouco significativos, com um suporte mínimo baixo. As redes neurais, através de sua estrutura complexa, conseguem reter nas conexões de seus neurônios um conhecimento mais significativo que as regras de associação.

Podemos constatar esta suscetibilidade do nosso sistema ao observar que os dígitos que tiveram um reconhecimento melhor são aqueles que possuem entre todos os seus caracteres, uma maior homogeneidade em suas características topológicas, que são

os casos dos excelentes resultados do ‘0’ e do ‘1’, além das ótimas taxas de acerto dos dígitos ‘6’ e ‘7’.

A prova de que o sistema de regras de associação mostrou-se menos eficaz que o sistema de reconhecimento através de redes neurais está na média de acertos. O sistema de redes neurais de [Veloso1998] obteve uma média de acerto de 84,72%, enquanto o sistema de regras de associação atingiu uma média de 72,44%. Nos trabalhos de [Correia2000a] e [Correia2000b], nos quais uma rede neural multicamada também é utilizada como classificador, a média de acerto de reconhecimento de caracteres numéricos manuscritos utilizando a base do Cenparmi ficou entre 93 e 94,7%, comprovando, desta forma, o excelente desempenho das redes neurais no reconhecimento de caracteres manuscritos.

Com o sistema de redes neurais, analisando a Tab 5.3, observamos que nove das dez classes existentes conseguem taxas de acerto acima de 70%. Com regras de associação, o quadro é bem diferente, apenas cinco das classes atingem reconhecimento correto maior que 70%. Além disso, alguns numerais tiveram um reconhecimento extremamente baixo, principalmente o ‘4’, o ‘2’ e o ‘9’, o que comprometeria excessivamente um sistema real de reconhecimento de caracteres manuscritos.

5.7 Discussão sobre a quantidade de caracteres treinados

Antes de conseguirmos chegar aos resultados da Tab 5.1, realizamos muitos outros testes e treinamentos até encontrarmos resultados mais satisfatórios.

Extraímos os padrões de cada um dos dígitos utilizando vários tamanhos para o conjunto de treinamento. Treinamos cada um dos dígitos com conjuntos de 250 caracteres, 500 caracteres, 700 caracteres e o máximo possível de caracteres para cada um dos dígitos.

Para alguns dígitos, obtivemos um pequeno aumento (de 0% a 3%) na percentagem de caracteres reconhecidos corretamente enquanto aumentávamos o número de caracteres do conjunto de treinamento. Todavia, para outros dígitos houve também um pequeno decréscimo da taxa de acerto à medida que aumentávamos o número de caracteres treinados. Logo, podemos dizer que, utilizando o sistema de regras de associação, não haverá uma taxa de reconhecimento de caracteres muito maior, se a base de treinamento for muito grande.

Mesmo assim, retirados os 2660 caracteres de teste, procuramos utilizar o máximo de caracteres possível em nossa base de dados para gerarmos os padrões de cada um dos dígitos. Desta forma, contamos com uma maior diversidade de escrita dos numerais, representando melhor uma aplicação do mundo real.

5.8 A aplicação de um critério de rejeição

Até então, falamos de dois estados no qual um caractere de entrada, em nosso sistema, pode ser classificado, são eles, *acerto* e *erro*, ou seja, um caractere de teste pode ser reconhecido corretamente ou de forma equivocada. Por exemplo, uma matriz do algarismo '2' se for reconhecida como tal terá sido uma matriz que obteve o reconhecimento correto, senão terá sido uma matriz classificada erroneamente.

Observando outros trabalhos sobre o mesmo tema, decidimos incluir em nosso sistema o conceito de um outro estado de reconhecimento: a *rejeição*, isto é, quando uma matriz de entrada tiver dificuldade de ser reconhecida como um determinado dígito ela deverá ser tratada como uma rejeição.

A idéia da rejeição está apoiada no fato de que se tivermos um sistema real de reconhecimento de caracteres manuscritos, é muito mais valioso que haja uma rejeição do que um erro no instante do reconhecimento, já que se pudermos separar os caracteres rejeitados podemos encaminhá-los a um outro sistema de decisão ou até mesmo para que um ser humano os avalie e faça a decisão necessária.

Em nosso sistema, a rejeição ocorrerá quando o vetor chance, resultante da entrada de um caractere de teste, tiver em pelo menos uma das nove posições restantes, um valor muito próximo ao maior valor de chance.

Após analisar vários vetores chance, para cada um dos caracteres de testes, verificamos que, em nosso caso, alguns caracteres de entrada geram um vetor chance muito 'confuso', ou seja, um vetor no qual nenhuma de suas posições consegue atingir um valor consideravelmente mais alto do que as outras.

Exemplificando, vamos supor que um caractere de entrada qualquer provoque a formação do vetor chance da Fig 5.6.

Seguindo a nossa linha de raciocínio até então (de dois estados: acerto e erro), classificaríamos esta matriz como sendo o algarismo '0', visto que a posição '0' do vetor chance tem um maior valor sobre as demais. Entretanto, ao analisarmos os valores do vetor chance nas posições '4' e '5' verificamos que elas tiveram resultados muito

semelhantes aos da posição ‘0’. Logo, não há uma garantia relevante de que esta matriz de entrada seja realmente um dígito ‘0’, pois apenas uma regra a mais que combinasse com o ‘4’ ou com o ‘5’ poderia mudar a conclusão sobre esta matriz de entrada para um destes dois numerais.

chance[0] = 96,02	chance[2] = 11,56	chance[4] = 95,85	chance[6] = 0	chance[8] = 0
chance[1] = 32,00	chance[3] = 25,03	chance[5] = 95,58	chance[7] = 0	chance[9] = 0

Figura 5.6 – Vetor chance confuso

Portanto, definimos que o nosso sistema apenas classificará um caractere de entrada, quando o maior valor ‘v1’ de chance, referente à posição ‘p1’, for mais alto que o segundo maior valor ‘v2’ de chance referente à posição ‘p2’, acrescido de uma folga ‘F’, como ilustrado na Fig 5.7.

Se chance[p1] > (chance[p2] + F) ***então***
    Matriz é reconhecida como p1
Senão
    Matriz é rejeitada

Figura 5.7 – Critério de rejeição aplicado nos testes

O grande problema do nosso sistema, utilizando o conceito de rejeição é determinar o valor a ser usado para a variável de folga (F). Depois de realizarmos vários testes e experimentarmos muitos valores para a variável F, obtivemos um melhor resultado admitindo o valor ‘10’ para ela. Isto é, para que uma matriz seja reconhecida, com acerto ou erro, é preciso que o valor mais alto do vetor chance seja maior que o segundo maior valor deste vetor acrescido de 10.

A Tab 5.4 (a) ilustra a taxa de erro e acerto para cada um dos dígitos no teste realizado sem a utilização de critério de rejeição (Tab 5.1), além de mostrar também as taxas de acerto, erro e rejeição (Tab 5.4 (b)) no teste realizado com critério de rejeição, utilizando o valor ‘10’ para a variável ‘F’.

Ao analisarmos os resultados do teste realizado utilizando-se critério de rejeição, Tab 5.4 (b), verificamos uma diminuição na taxa de acerto do sistema, pois alguns caracteres que eram reconhecidos corretamente (Tab 5.4 (a)), não são mais devido ao critério de rejeição utilizado. Por exemplo, um caractere de teste que gere o vetor chance da Fig 5.6, não é mais classificado como sendo o dígito '0', é classificado agora como caractere rejeitado (desconhecido). Pelo mesmo motivo, a quantidade de erros ocorrida no sistema também acaba caindo.

Em testes efetuados com um valor para 'F' bem maior que 10, a taxa de erro diminui bastante, porque mais caracteres são rejeitados. Entretanto, a quantidade de matrizes reconhecidas corretamente também diminui muito, podendo chegar a um ponto onde não haja sentido em reconhecer tão poucos caracteres.

Tabela 5.4 – Resultados dos testes realizados com rejeição (a) e sem rejeição (b)

	Teste sem critério de rejeição		**Teste com critério de rejeição**		
Dígito	**Acerto (%)**	**Erro (%)**	**Acerto (%)**	**Erro (%)**	**Rejeição (%)**
0	94,74	5,26	93,23	4,14	2,63
1	99,25	0,75	99,25	0,00	0,75
2	64,29	35,71	62,78	30,08	7,14
3	72,93	27,07	69,55	19,17	11,28
4	46,99	53,01	43,98	44,74	11,28
5	59,77	40.23	53,38	34,59	12,03
6	80,83	19,17	74,81	16,92	8,27
7	80,45	19,55	77,44	16,17	6,39
8	68,80	31,20	63,16	20,30	16,54
9	56,39	43,61	51,50	37,22	11,28
Média	72,44	27,56	68,90	22,33	8,77
(a)			(b)		

Os resultados do teste da Tab 5.4, utilizando critério de rejeição, conseguem ser melhores que os resultados de análise sintática de [Gomes1994] para todos os dígitos e melhor que a análise sintática de [Veloso1998] para os dígitos '0,1,3,6,7,8' e melhor que o sistema de redes neurais, em [Veloso1998], para os dígitos '0,1,7'.

A média de acerto do nosso sistema, levando-se em conta a rejeição dos caracteres conseguiu ser um pouco melhor que a média de acerto de [Veloso1998] (66,35) e foi mais alta que o dobro da média de acerto de [Gomes1994] (30,45%).

O nosso sistema também conseguiu rejeitar um número de caracteres maior (8,77%) que o sistema de análise sintática de [Veloso1998], que obteve taxa de rejeição

igual a 5,75%. Lembremos que, em sistemas de reconhecimento de manuscritos, é muito mais eficaz rejeitar caracteres do que errar.

Quanto à taxa de erro, o sistema de regras de associação apresentou um valor menor (22,33%) que o algoritmo sintático implementado em [Veloso1998] (27,89%). Comparando com o algoritmo sintático de [Gomes1994] que teve uma taxa de erro de 21,35%, o sistema de regras de associação errou um pouco mais que este último (22,33%), porém teve uma taxa de acerto igual ao dobro deste, além de não rejeitar tantos caracteres.

Entretanto, o melhor desempenho do nosso sistema, utilizando critério de rejeição, em relação aos algoritmos sintáticos não pôde ser também constatado em relação aos resultados obtidos por [Veloso1998], utilizando uma rede neural multicamada com propagação direta. O sistema com redes neurais conseguiu uma taxa de acerto de 84,72%, além de ter obtido uma taxa de erro muito pequena, 4,53%, mostrando assim sua maior eficácia em relação às regras de associação no reconhecimento de padrões.

Podemos resumir a discussão desta seção da seguinte forma: com a aplicação de critérios de rejeição, o sistema de reconhecimento de caracteres manuscritos, baseado em regras de associação, erra menos, porém acerta menos!

5.9 Tempo de Treinamento e de Teste

No trabalho de [Lecun1995] sobre reconhecimento de caracteres manuscritos são realizados vários treinamentos e testes utilizando 11 diferentes classificadores neurais. Os tempos para o treinamento destes classificadores sobre uma base de dados de 60.000 caracteres utilizando uma máquina SPARC 10 são muito grandes variando de 12 horas a 35 dias.

Em nosso sistema, como mostrado na Tab 5.5, obtivemos um tempo relativamente menor para treinar os 9446 caracteres: 20 minutos, 53 segundos e 760 milésimos de segundo. Os tempos relacionados abaixo foram obtidos sobre uma máquina de porte pequeno: um computador com microprocessador Cyrix 686 200Mhz, com 40Mb de memória RAM.

Nessa tabela também mostramos os três tempos parciais para chegar à geração de padrões para um determinado dígito, são eles: o tempo para extrair o grande conjunto ‘G’ dos grandes conjuntos; o tempo para extrair as regras dos elementos deste conjunto

'G'; e finalmente o tempo para realizar o refinamento dessas regras (discutido no capítulo 4).

Com relação ao tempo para o reconhecimento dos caracteres de teste, dos 11 classificadores neurais utilizados em [LeCun1995], 6 deles demoraram de 0.5 a 2 segundos para fazer o reconhecimento. Os outros 5 classificadores, tiveram tempos que variaram entre 0.001 a 0.1 segundo para reconhecer uma única matriz de entrada.

Tabela 5.5 – Tempo de treinamento com o sistema de regras de associação.

Dígito	**Conjunto G**	**Extração das Regras**	**Poda de Regras**	**Tempo Total**
0	03:43:440	00:07:910	00:33:780	04:25:130
1	02:38:130	00:01:650	00:06:260	02:46:040
2	02:00:340	00:01:760	00:08:510	02:10:610
3	02:49:890	00:03:080	00:14:560	03:07:530
4	01:17:330	00:03:030	00:25:930	01:36:290
5	00:35:760	00:01:920	00:07:960	00:45:640
6	01:23:000	00:05:980	00:32:130	02:01:110
7	01:18:770	00:03:300	00:14:890	01:36:960
8	01:17:230	00:01:210	00:06:150	01:24:590
9	00:41:960	00:03:290	00:14:610	00:59:860
Total	**17:45:850**	**00:33:130**	**02:44:780**	**20:53:760**

Em nosso sistema de reconhecimento de manuscritos, testamos 2660 caracteres e obtivemos um tempo total de 2 minutos, 56 segundos e 910 milésimos de segundo para reconhecer todos eles. Calculando-se o tempo de apenas um caractere de entrada ser reconhecido, chegamos ao valor aproximado de 252 milésimos de segundo, o que é um tempo bom quando comparado aos primeiros seis classificadores usados por [LeCun1995] e um pouco maior que os outros cinco classificadores utilizados nesse trabalho.

Capítulo Seis

Conclusão

Apesar da crescente difusão das tecnologias de informação, informações 'em papel' como formulários, memorandos e outros documentos continuam abundantes. Seu processo de conversão para a mídia eletrônica tem se revelado caro e improdutivo, devido principalmente à intervenção, no processo, de pessoal despreparado. Este trabalho teve como objetivo pesquisar a viabilidade do uso pioneiro da técnica de data mining, conhecida como regras de associação, para o reconhecimento automático de caracteres manuscritos. Até onde foi nossa pesquisa bibliográfica, podemos afirmar que é um sistema pioneiro.

Além disso, este trabalho propôs modificações no método de extração de regras de associação convencional, a fim de adaptá-lo para o reconhecimento de caracteres manuscritos, contribuindo, desta forma, à literatura de regras de associação. Em particular, propusemos a descoberta de regras de associação, referindo-se não apenas à presença de itens, mas também à *ausência de itens* em algumas posições da matriz de caracteres. Também elaboramos o cálculo de um *fator de relevância* para regras de associação, como medida de seu poder discriminatório para fins de classificação.

É de contribuição deste trabalho também, a elaboração e implementação de um algoritmo para fazer a centralização de cada um dos caracteres da base de dados dentro de uma matriz de pixels de tamanho N x N, onde N é o número de linhas e colunas da matriz.

A demonstração da viabilidade da utilização de regras de associação consistiu em contrapor os resultados obtidos com essa técnica aos resultados de duas outras abordagens de reconhecimento de caracteres numéricos, algoritmos sintáticos e redes neurais.

No geral, regras de associação mostraram-se mais confiáveis que algoritmos sintáticos. Ao observar os resultados de nossos testes, verificamos que obtivemos uma taxa média de acerto maior que os algoritmos sintáticos de [Gomes1994] e [Veloso1998].

Um outro ponto extremamente importante a favor do sistema com regras de associação é o fato de ele dispensar o ser humano da necessidade de extrair características dos caracteres. Bem ao contrário dos algoritmos sintáticos, os padrões (regras de associação) para os dígitos treinados foram extraídos automaticamente.

Regras de associação também não levam em conta os tipos de caracteres (numérico, alfabético, caracteres especiais) sendo treinados e testados. Embora tenhamos realizado um estudo utilizando caracteres numéricos manuscritos do Cenparmi, fazer um algoritmo de geração de regras de associação evoluir para reconhecer caracteres de quaisquer outros tipos não introduz nenhuma dificuldade adicional no algoritmo em si. Esta é outra vantagem enorme em relação aos algoritmos sintáticos, que são baseados na geometria e na topologia dos caracteres ora manipulados.

Como conseqüência da extração automática de padrões, o tempo de treinamento do sistema de regras de associação é infinitamente menor que o dos sistemas baseados em algoritmos sintáticos, pois estes últimos dependem essencialmente da habilidade do ser humano em atentar para as peculiaridades dos caracteres o que pode ser um processo muito demorado.

Todavia, tal como os algoritmos sintáticos, o nosso sistema é extremamente dependente da morfologia dos caracteres, para poder gerar padrões confiáveis. Mais precisamente, sua taxa de reconhecimento de caracteres manuscritos decresce a medida em que um mesmo caractere é escrito de maneira mais e mais variada, morfologicamente (topologia, posição e outros parâmetros, conforme discutidos no corpo da dissertação). Infelizmente, o cenário real é bem variado (pessoas diferentes escrevem diferente!).

No que concerne à utilização de redes neurais para o reconhecimento de caracteres manuscritos, estas últimas se revelaram bem menos dependentes da variedade da mesma classe de caracteres, quando comparadas com análise sintática e regras de associação. O reconhecimento, via regras de associação, conseguiu melhores resultados

em apenas 3 dígitos: ‘0’, ‘1’ e ‘7’. Isto aconteceu porque as bases destes 3 dígitos contém caracteres muito homogêneos, tanto de teste quanto de treinamento.

Além da dependência excessiva em relação à morfologia dos caracteres, existem dois outros fatores críticos em nosso sistema. O primeiro está relacionado com a definição, durante a fase de treinamento, do suporte mínimo adequado para gerar os padrões de cada uma das classes de dígito (em nosso estudo de caso, os dígitos ‘0’,’1’, ‘2’,...,’9’). Não existe um critério previamente determinado para se aplicar um suporte mínimo. Chegamos aos valores da Tab 4.3, já discutidos no capítulo 4, após efetuarmos diversos testes, com diferentes valores de suportes mínimos, para cada uma das classes existentes.

A segunda questão crítica em nosso sistema está relacionada com a ausência de um critério para definir o valor da variável folga ‘F’, ilustrada na Fig 5.7, descrita no capítulo 5, usada para rejeitar caracteres de teste. O valor que encontramos em nossos testes foi resultado de inúmeras tentativas para obter melhores taxas de reconhecimento.

Devido a todos os problemas citados, somos forçados a concluir, de nossa pesquisa, que regras de associação ainda não são uma tecnologia apropriada para se fazer o reconhecimento de caracteres manuscritos. Os resultados de nossos testes ficaram muito abaixo das taxas de acerto obtidas com redes neurais.

Melhorar a acurácia dos algoritmos de regras de associação para o reconhecimento de caracteres manuscritos depende, essencialmente, da etapa de preparação dos dados, em que algoritmos de processamento de imagem tentariam uniformizar os caracteres de uma mesma classe. Se esses algoritmos não forem capazes disto, certamente o sistema de regras de associação não conseguirá resultados significativos.

Os algoritmos que utilizamos, para tornar mais uniformes os caracteres da mesma classe da base de dados do Cenparmi, se mostraram incapazes de atingir esses objetivos, facilitando a posterior geração das mesmas regras de associação para diferentes caracteres, levando às relativamente baixas taxas de reconhecimento obtidas.

Ao término desta dissertação, soubemos dos promissores resultados obtidos por [Anibal2000]. Neste trabalho, foi utilizado o algoritmo de classificação C 5.0 [See5_1999] para fazer o reconhecimento de caracteres manuscritos, usando a mesma base de dados do Cenparmi. Este sistema obteve um desempenho excelente, atingindo

uma taxa média de acerto por volta de 92%, o que ilustra a viabilidade da aplicação da tecnologia de data mining no reconhecimento de padrões.

6.1 Sugestões para trabalhos futuros

Após a experiência obtida com as regras de associação para o reconhecimento de caracteres manuscritos, vislumbramos alguns esforços que podem ser feitos, visando melhorar o desempenho dos algoritmos de regras de associação, no contexto citado. Dentre eles:

✓ Como visto no capítulo anterior, foi utilizado um suporte mínimo diferente para cada dígito a fim de extrair suas regras de associação. Se este suporte mínimo for muito baixo acaba gerando muitas regras com antecedente pequeno, o que provoca futuros conflitos no momento da classificação. Ao contrário, se este suporte for muito alto, nenhuma ou poucas regras de tamanho grande são extraídas. Sabendo-se que regras de tamanho grande possuem muito mais semântica que regras de tamanho pequeno, uma maneira de gerar mais regras grandes é definir diferentes suportes mínimos para gerar regras de um determinado tamanho. Ou seja, definir um suporte mínimo para extrair regras de tamanho 1, outro para regras de tamanho 2 e assim sucessivamente. Desta forma, é possível usar um suporte bastante alto para extrair poucas regras de tamanho 1 e 2, por exemplo, e usar suportes mais baixos para garimpar apenas regras com antecedentes maiores que 3.

✓ Pesquisar a eficiência de utilizar a medida do suporte de cada regra de associação como critério de desempate. Isto é, quando um caractere de entrada ficar confuso com dois ou mais dígitos, seria possível utilizar o valor do suporte de cada regra que *casou* com o caractere de entrada para tentar classifica-lo de maneira mais confiável, já que o suporte de cada regra é uma medida com grande semântica em nosso contexto.

✓ Desenvolver novos algoritmos de normalização, rotação e dilatação, a fim de que eles possam gerar caracteres com menos ‘ruído’ e, portanto, ajudar em taxas mais altas de reconhecimento;

✓ Explorar outros tamanhos para as matrizes dos caracteres. Em nosso trabalho, utilizamos matrizes de caracteres normalizadas em 16 linhas por 16 colunas, para extrair tanto os padrões de cada um dos dígitos quanto para fazer os testes. Por outro lado, as matrizes dos caracteres da base de dados treinada e testada têm diversos tamanhos, como 45 linhas por 23 colunas. Quando uma matriz é transformada deste tamanho para 16x16, acaba perdendo muito das características do dígito original. Entretanto, é provável que se consiga caracteres menos irregulares topologicamente se aumentarmos o tamanho das matrizes para a normalização. Espera-se que, desta forma, os caracteres ficarão mais fiéis aos originais, prejudicando menos o futuro reconhecimento. Maiores matrizes para a normalização dos caracteres, 20 linhas por 20 colunas, e 32 linhas e 32 colunas, são utilizadas em [LeCun1995] e [Yuccer1993], respectivamente. Entretanto, será preciso ter em conta que matrizes grandes podem levar a uma explosão de grandes conjuntos para as regras de associação, como discutido no capítulo 4.

✓ Investigar sistemas híbridos. Em [Lee1999], é apresentado um sistema híbrido para o reconhecimento de caracteres numéricos manuscritos, com análise sintática e redes neurais. Seus resultados alcançaram uma taxa média de reconhecimento de 92,40%. Pode-se construir um sistema semelhante, substituindo-se análise sintática por regras de associação. É de se esperar que, desta forma, os resultados sejam melhores que os obtidos com os sistemas híbridos com análise sintática, a julgar pelas vantagens das regras de associação sobre esta última.

✓ Avaliar o sistema de reconhecimento via regras de associação utilizando outras bases de dados que não a do Cenparmi. Por exemplo, testar o sistema com numerais provindos de cheques bancários, como em [Lee1999], ou ainda com outras bases de caracteres de outros centros de pesquisa como o CEDAR, o NIST (United States National Institute of Standards and Technology), etc.

Anexo I: O Algoritmo Apriori

O algoritmo apriori, ilustrado na Fig I.1, gera um conjunto G dos grandes conjuntos de itens da base dados com suporte maior que um suporte mínimo especificado. De posse deste conjunto G será possível extrair as regras de associação dos itens.

1) G_1 = {Conjunto de grandes conjuntos de tamanho 1}
2) Para (k = 2; G_{k-1} <> ∅; K++) faça
3) início
4) C_k = GeraCandidatos(G_{k-1});
5) Para todas transações t ∈ D faça
6) início
7) C_t = subconjunto (C_k, t);
8) Para todos candidatos c ∈ C_t do
9) c.count++;
10) fim
11) G_k = {c ∈ a C_k | c.count >= minsup}
12) fim
13) ConjuntoResposta = União de todos os $G_{k's}$;

Figura I.1 – Algoritmo Apriori

A primeira ação do algoritmo (iteração k igual a 1) é identificar todos os itens existentes, formando o conjunto G de tamanho igual a 1. Utilizando o exemplo da farmácia, descrito no capítulo 3, teríamos:

$$G_1 = \{ A, E, M \}$$

Após a geração do conjunto G de tamanho igual a 1, deve-se gerar os próximos conjuntos G de tamanho 2, 3, 4, ..., n. Assim, para se constituir o conjunto G de tamanho 2 (k igual a 2), é preciso gerar todas as combinações de itens de tamanho 2 possíveis, com os itens de G_1 a fim de formar o conjunto C_k, candidatos a G_k. O conjunto C_k é construído de acordo com o algoritmo *GeraCandidatos*, descrito posteriormente neste anexo. Generalizando: os itens de C_k são gerados a partir dos itens do conjunto G de tamanho k-1.

Portanto, o nosso conjunto C_2, teria os seguintes elementos:

$$C_2 = \{ \{AE\}, \{AM\}, \{EM\} \}$$

Ainda na mesma iteração, são extraídos deste conjunto C_k, todos os itens que aparecem nas *tuplas* do conjunto de dados D, formando o conjunto C_t. E para todo elemento de C_t é mantido um contador a fim de armazenar em quantas tuplas determinado elemento de C_t foi encontrado. Assim o conjunto C_t de C_2 seria o seguinte:

$$C_t = \{ \{AE \mid 8\}, \{AM \mid 7\}, \{EM \mid 7\} \} ;$$

onde {AE | 8} significa que os itens ‘A’ e ‘E’ aparecem em oito transações da base de dados. E este número 8 é definido como a freqüência de um conjunto, ou o suporte do mesmo.

Para finalizar, varre-se os elementos de C_t, selecionando aqueles com um suporte maior que o suporte mínimo (supmin) especificado, gerando finalmente, o conjunto G dos grandes conjuntos de tamanho k.. Supondo que o suporte mínimo para o nosso exemplo seja 6, o conjunto G_2 gerado será o seguinte:

$$G_2 = \{ \{AE \mid 8\}, \{AM \mid 7\}, \{EM \mid 7\} \}$$

Observamos que, neste caso, o conjunto G_2 é o mesmo conjunto C_t para k igual a 2. Todavia, isto só aconteceu porque todos os elementos de C_t possuíam um suporte maior que o suporte mínimo.

Como o conjunto G_2 não é vazio, o laço é realizado mais uma vez. Para k igual a 3 teremos o seguinte conjunto C_3:

$$C_3 = \{ \{AEM\} \}$$

O conjunto C_t, baseado em C_3 é descrito a seguir:

$$Ct = \{ \{AEM \mid 6\} \}$$

Como o único elemento de Ct possui um suporte maior que supmin, o conjunto G3 será:

$$G3 = \{ \{AEM \mid 6\} \}$$

No passo k igual a 4 não será gerado nenhum conjunto pois não há como extrair um novo conjunto a partir de G_3. Logo, o algoritmo saíra do laço e fará a união de todos os conjuntos G's gerados:

$$G = G_2 \cup G_3$$
$$G = \{ \{AE \mid 8\}, \{AM \mid 7\}, \{EM \mid 7\}, \{AEM \mid 6\} \}$$

É importante salientar que o conjunto G_1 não entrou no conjunto G porque não será útil para a geração de regras, já que não poderá montar regras com antecedente e conseqüente, usando apenas um elemento.

A geração dos Candidatos: o algoritmo GeraCandidatos

De acordo com [Mongiovi1998], a idéia deste algoritmo é unir os elementos de G_{k-1}, 2 a 2 e conservar apenas aqueles em que todos os seus subconjuntos de tamanho k-1 pertençam a G_{k-1}. Ele se dá em dois passos: junção e poda.

Junção:

Insert into C_k
From $G_{k-1}(p)$, G_{k-1} (q) {elementos p e q de G_{k-1}}
Select p.item1 = q.item1, ..., p.itemk-2 = q.itemk-2, p.itemk-1 < q.itemk-1;

Exemplificando, vamos supor que o conjunto G_{k-1} seja constituído dos seguintes elementos:

$$G_{k-1} = \{ ABC, ACD, ADE, ABD, ABE, BDE \}$$

Tomando-se dois elementos deste conjunto, por exemplo: ABC e ABE, teremos segundo as regras de junção a seguinte situação:

A = A ; B = B; C < E; gerando, portanto um elemento para o conjunto C_k. O raciocínio para os outros elementos de C_k é o mesmo:

$$C_k = \{ ABCD, ABCE, ABDE \}$$

Poda:

A idéia por trás da poda é eliminar todo c pertencente a C_k tal que algum (k-1) subconjunto de c não pertence a G_{k-1}.

$C_k = \{c \in C_k \mid \forall s \subset c, s \text{ pertence } G_{k-1}\}$

para todo $c \in C_k$ faça

para todo $s \subset c$ e $|s| = k\text{-}1$ faça

Se $s \notin G_{k-1}$ então elimina c de C_k.

Tomemos o elemento ABCD do conjunto C_k, ele deve ser eliminado porque embora ABC e ABD estejam presentes em G_{k-1}, o outro subconjunto de tamanho k-1, o BCD não encontra-se em G_{k-1}. Pelo mesmo motivo ABCE foi eliminado também do conjunto. Assim, Ck teria a seguinte forma:

$$Ck = \{ ABDE \}$$

Anexo II: Algumas Matrizes de dígitos

Dígito '0'				
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000011111000000	0000000000000000
0000011111000000	0000001111110000	0001111111100000	0001111111110000	0000000000110000
0000111011100000	0000111100111000	0011110001110000	0011110001110000	0000000111111000
0001111001110000	0001111000111000	0011100000111000	0011100000111100	0000001111011100
0001110000111000	0001110000111000	0011100000111000	0111100000011100	0000111100001110
0001110000111000	0011100000111000	0011100000111100	0111000000011100	0001110000011110
0001110000111000	0011100000011100	0011100000011100	0111000000001110	0011110000011100
0011100000111000	0011100000011100	0011100000011100	0111000000001110	0011100000001110
0011100000011100	0011100000011100	0011100000011100	0111000000001110	1111100000000111
0011100000011100	0011100000011100	0011100000111100	0111000000011100	1111100000001111
0011100000111100	0011110000011100	0011100001111000	0011110000111100	0011100000011110
0001110011111000	0000111111111100	0011100011110000	0001111011110000	0011100001111000
0001111111100000	0000000111000000	0001111111100000	0000011111000000	0001111111100000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
Dígito '1'				
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000001110000000	0000000000000000	0000001110000000	0000001110000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000000111000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000000111000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000000111000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000001110000000	0000000000000000	0000001110000000	0000001110000000	0000001110000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000001100000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000001110000000	0000001110000000	0000001110000000	0000001110000000	0000001110000000
0000001110000000	0000001110000000	0000000000000000	0000001110000000	0000001110000000
0000000000000000	0000001110000000	0000000000000000	0000000000000000	0000001110000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
Dígito '2'				
0000000000000000	0111100000000000	0000000000000000	0000000000000000	0000000000000000
0000011100000000	0001111100000000	0000000000000000	0001110000000000	0000000000000000
0000111110000000	0000001110000000	0000001111111000	0111111100000000	0000011111110000
0011111110000000	0000000111110000	0000111100111000	0111011110001110	0000111101111000
0111000111000000	0000000001110000	0011110000111000	0111001111001110	0000000000111000
0111000111000000	0000000001110000	0111000000111000	0000000111001110	0000000000111000
0000000111000000	0000111111100000	0000000001111000	0000000111011110	0000000001111000
0000000111000000	0011110000000000	0000000001110000	0000000111111100	0000000001110000
0000000111000000	1111000000000000	0000000011110000	0000000011111100	0110000011110000
0001111111111110	1110000000000000	0000000111100000	0000000011111000	0011110111100000
0111110111000000	1111100000000000	0000001111001110	0000000011100000	0000111110000000
1110000111000000	0001111100000000	0000111111111100	0000111111000000	0111111111100000
1110001110000000	0000001111110000	0011111000000000	0000111111000000	0000000001111100
1110001110000000	0000000000111110	0111000000000000	0000111111000000	0000000000011110
1111111100000000	0000000000001110	0000000000000000	0000111000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

Dígito '3'

0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000011111111000	0000000111111000	0000000000000000	0000011111000000
0000011111110000	0111111100011110	0000011111111000	0000011110000000	0000000111111100
0000000001110000	0000000000001110	0001111100111000	0000111111111100	0000000011111000
0000000001110000	0000000000011110	0011110000111000	0000000000011100	1111100000111110
0000000011111000	0000000001111000	0111100000111000	0000000000001100	0001111110001111
0000000000011100	0000000111110000	1110000001110000	0000000000011100	0000001111111111
0000000000001110	0000000111110000	0000000001110000	0000000000011100	0000000011111111
0000000000001110	0000000111111100	0000000001111110	0000001111111100	0000000011110000
0000000000001110	0000000000011100	0000000000000110	0000011100011100	0000000001111000
0000000000001110	0000000000011100	0000000000001110	0000000000001110	0000000000011100
0111100000011110	0000000001111100	0000000000011110	0000000000001110	1111100000011100
0111100111111100	0000001111100000	0000000000111000	0000000000011100	0001111100011100
0000111000000000	0000110000000000	0000000111110000	0111111111111100	0000011111111100
0000000000000000	0000110000000000	0000011111000000	0000000000000000	0000000011110000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

Dígito '4'

0000000000001110	0000001110000000	0000000000010000	0000001110000000	0000011100000000
0000000000001110	0000001110000000	0000001110000000	0000001110000000	0000011100000000
0000000000011100	0000001110000000	0000001110000000	0000001111000000	0000001110000000
0000000000011100	0000000111000000	0000000111000000	0000000111001110	0000000111000000
0111000000111000	0000000111000000	0000000111000000	0000000111011100	0000000111001110
0110000000111000	0000000111011100	0000000111000000	0000000011111100	0000000111011100
0111000000111000	1110000011111000	0000000011100111	0000000001110000	0000000011111100
0111000000111000	1110000001110000	1110000011111110	0111000001110000	1100000011000000
0011111001110000	1110000001110000	1110000011111000	0111000001110000	1100000111110000
0000011111110000	0111000111110000	1110000001110000	0111000011000000	1110011111110000
0000000001110000	0111011111110000	1111111111110000	0111001111110000	1111111001110000
0000000011100000	0111111001111000	0000000001110000	0011111101110000	0000000001111000
0000000011100000	0000000000111000	0000000001110000	0011111000111000	0000000000111000
0000000011100000	0000000000111000	0000000001110000	0000000000111000	0000000000011100
0000000011100000	0000000000011100	0000000001110000	0000000000011000	0000000000011100
0000000011100000	0000000000011100	0000000001110000	0000000000000000	0000000000000000

Dígito '5'

0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000111000	0000000000011100	0000111111100000	0000000111100000	0000000001110000
0000000001110000	0000000001111000	0001111011111100	0000001111000000	0000000111100000
0000000111100000	0000000111100000	0001110000001100	0000111110000000	0000001111000000
0000111110000000	0000011111000000	0001111110000000	0011111000000000	0000011100000000
0001110000000000	0011111000000000	0000000011110000	0111100000000000	0000111000000000
0111100000000000	0011100000000000	0000000000111100	0111000000000000	0001110000000000
0111000000000000	0011000000000000	0000000000011100	0111000000000000	0011110000000000
0111111111000000	0011100000000000	0000000000001100	0111000111110000	0011100000000000
0000000011110000	0001111111110000	0000000000011100	0011101110111100	0011100000000000
0000000001110000	0000000000111100	0111100000011100	0011111100011100	0011111111111100
0000000000111000	0000000000011100	0011100000111000	0000000000011100	0000000000011100
0000000001111000	0000000000111100	0001111011110000	0000000000111100	0000000000011100
0000000011110000	0001100011111000	0000011111000000	0000000111110000	0000011000111100
0001111111100000	0001111111000000	0000000000000000	0000011111000000	0000000111110000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

Dígito '6'

0000000000000000	0000000000000000	0000000111000000	0000000000000000	0000000000000000
0000000001110000	0000001110000000	0000011110000000	0000001110000000	0000000000000000
0000000111100000	0000011100000000	0000111100000000	0000011100000000	0000000000000000
0000001111000000	0000111000000000	0000111000000000	0000111000000000	0000000000000000
0000111100000000	0001110000000000	0001110000000000	0000111000000000	0000111000000000
0001111000000000	0011110000000000	0011100000000000	0000111000000000	0000111000000000
0011100000000000	0011100000000000	0011100000000000	0011110001111100	0001110000000000
0111100000000000	0111000001111110	0011100000000000	0111100011101110	0011100000111100
0111000011111000	1110000111101110	0011100011111100	0111000111101110	0011100011110000
1110001111111110	1110001110001110	0011101111011100	1110000111001110	0111101111001100
1110111100001110	1110011100011110	0011101110011100	1110000111001110	0111001111111000
0111111000001110	1110011101111100	0001111111111000	1110000111111100	0111000111110000
0011111100011110	0111111111110000	0011100000000000	0111000111000000	0111111110000000
0000001111111100	0000001111000000	0011100000000000	0111111110000000	0000000000000000
0000000000000000	0000000000000000	0011100000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

Dígito '7'				
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000111111100	0000001111111100	0000000000000000	0000000000001110
0000000111111100	0000111110011100	0000111100001110	0000000001111100	0000000001111110
0111111110011110	0111111000001110	0111110000001110	0000001111111100	1111111110011110
0000000000001110	0000000000001110	0000000000001110	0011110000011100	1110000000011100
0000000000001110	0000000000001110	0000000000001110	0000000000011100	1110000000111100
0000000000011100	0000000000001110	0000000000011100	0000000000011100	1110000001111000
0000000000011100	0000000000001110	0000000000011100	0000000000111000	0000000001110000
0000000000111000	0000000000111100	0000000000011100	0000000001111000	0000000001110000
0000000001110000	0000000000111000	0000000000111000	0000000001110000	0000000011110000
0000000011100000	0000000001111000	0000000001110000	0000000011110000	0000000011100000
0000000111000000	0000000001110000	0000000011100000	0000000011100000	0000000111000000
0000000111000000	0000000001110000	0000000111000000	0000000011100000	0000000111000000
0000000000000000	0000000011110000	0000000111000000	0000000000000000	0000000111000000
0000000000000000	0000000011110000	0000000000000000	0000000000000000	0000001110000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
Dígito '8'				
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000011111110000	0001111000000000	0001111111110000	0001111111100000	0000011111110000
0011110000111100	0011110000000000	0011100000111100	0011110011100000	0001111000111100
0111000000011100	0111100000111000	0011110000011100	0011100001110000	0001110000011100
0111000000011100	0111000000111000	0001111000111100	0001110001110000	0001110000011100
0110000000001100	0111000001111000	0000011111110000	0000111001111110	0000111100011100
0111100000111100	0011110001110000	0000000111110000	0000111111100000	0000001111111000
0011111000111000	0000111111100000	0000000111100000	0000111011100000	0000001111110000
0000011111110000	0000011111100000	0000001111110000	0011111001110000	0000001111000000
0000001111110000	0000011111111000	0000001111110000	0111100001111000	0001111111000000
0000011101111000	0001111100011000	0000011100111000	1110000000111000	0011100011100000
0000111000111000	0011110000111000	0000011100111000	1110000000011000	0011100011100000
0001110000111000	0011100001110000	0000011111110000	1111110000111000	0011100111000000
0001110001111000	0011101111100000	0000000111110000	0000111111111000	0011111111000000
0000111111110000	0001111100000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
Dígito '9'				
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000111111110	0000001111100000	0000000111100000	0000000000000000	0000000111111100
0000011110001110	0001111101111000	0001111000000000	0000011111110000	0001111100011100
0001111000001110	0111110000111000	0011100000001110	0001111100111000	0011100000001110
0011100000001110	0111000000011100	1111000000001110	0011111100111000	0111100000011110
0111000000001110	0111000000011100	1110000000001110	0011101110011100	0111000000111100
0111000000011110	0111101111111100	1110000000001110	0011101111111100	0111000001111100
0110000011111110	0001111100111000	0111000000011100	0001111110111000	0111000011111100
0111111111111110	0000000000111000	0001111111111100	0000000000111000	0111111111011100
0000000000111110	0000000000111000	0000000001110000	0000000000111000	0001111100011100
0000000000111000	0000000001111000	0000000001110000	0000000000111000	0000000000011100
0000000000111000	0000000011110000	0000000001110000	0000000001110000	0000000000011100
0000000000111000	0000000011100000	0000000011100000	0000000011110000	0000000000011100
0000000001110000	0000000111000000	0000000011100000	0000000111100000	0000000000011000
0000000000000000	0000001110000000	0000000111000000	0000000000000000	0000000000011000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

Anexo III: Regras geradas no treinamento referente aos resultados do teste da TAB 5.1, com seus respectivos suportes

Regras	Suporte
{ 188_1 } » 257	#0,78#
{ 93_1 104_0 } » 257	#0,78#
{ 93_1 139_0 } » 257	#0,78#
{ 116_1 121_0 } » 257	#0,78#
{ 116_1 136_0 } » 257	#0,78#
{ 116_1 137_0 } » 257	#0,78#
{ 116_1 138_0 } » 257	#0,78#
{ 116_1 152_0 } » 257	#0,78#
{ 116_1 153_0 } » 257	#0,78#
{ 116_1 168_0 } » 257	#0,78#
{ 116_1 169_0 } » 257	#0,78#
{ 142_1 152_0 } » 257	#0,78#
{ 142_1 168_0 } » 257	#0,78#
{ 157_1 167_0 } » 257	#0,78#
{ 93_1 105_0 126_1 } » 257	#0,78#
{ 93_1 105_0 132_1 } » 257	#0,78#
{ 93_1 105_0 142_1 } » 257	#0,78#
{ 93_1 105_0 148_1 } » 257	#0,78#
{ 93_1 105_0 164_1 } » 257	#0,78#
{ 93_1 106_0 132_1 } » 257	#0,78#
{ 93_1 106_0 142_1 } » 257	#0,78#
{ 93_1 106_0 148_1 } » 257	#0,78#
{ 93_1 106_0 164_1 } » 257	#0,78#
{ 93_1 109_1 135_0 } » 257	#0,78#
{ 93_1 109_1 151_0 } » 257	#0,78#
{ 93_1 120_0 126_1 } » 257	#0,78#
{ 93_1 120_0 148_1 } » 257	#0,78#
{ 93_1 121_0 126_1 } » 257	#0,78#
{ 93_1 121_0 132_1 } » 257	#0,78#
{ 93_1 121_0 142_1 } » 257	#0,78#
{ 93_1 121_0 148_1 } » 257	#0,78#
{ 93_1 121_0 164_1 } » 257	#0,78#
{ 93_1 122_0 132_1 } » 257	#0,78#
{ 93_1 122_0 142_1 } » 257	#0,78#
{ 93_1 122_0 148_1 } » 257	#0,78#
{ 93_1 122_0 164_1 } » 257	#0,78#
{ 93_1 123_0 126_1 } » 257	#0,78#
{ 93_1 126_1 136_0 } » 257	#0,78#
{ 93_1 126_1 152_0 } » 257	#0,78#
{ 93_1 126_1 154_0 } » 257	#0,78#
{ 93_1 126_1 168_0 } » 257	#0,78#
{ 93_1 126_1 169_0 } » 257	#0,78#
{ 93_1 132_1 136_0 } » 257	#0,78#
{ 93_1 132_1 137_0 } » 257	#0,78#
{ 93_1 132_1 138_0 } » 257	#0,78#
{ 93_1 132_1 152_0 } » 257	#0,78#
{ 93_1 132_1 153_0 } » 257	#0,78#
{ 93_1 132_1 154_0 } » 257	#0,78#
{ 93_1 132_1 168_0 } » 257	#0,78#
{ 93_1 132_1 169_0 } » 257	#0,78#
{ 93_1 136_0 142_1 } » 257	#0,78#
{ 93_1 136_0 148_1 } » 257	#0,78#
{ 93_1 136_0 164_1 } » 257	#0,78#
{ 93_1 137_0 142_1 } » 257	#0,78#
{ 93_1 137_0 148_1 } » 257	#0,78#
{ 93_1 137_0 164_1 } » 257	#0,78#
{ 93_1 138_0 142_1 } » 257	#0,78#
{ 93_1 138_0 148_1 } » 257	#0,78#
{ 93_1 138_0 164_1 } » 257	#0,78#
{ 93_1 142_1 153_0 } » 257	#0,78#
{ 93_1 142_1 154_0 } » 257	#0,78#
{ 93_1 142_1 169_0 } » 257	#0,78#
{ 93_1 148_1 151_0 } » 257	#0,78#
{ 93_1 148_1 152_0 } » 257	#0,78#
{ 93_1 148_1 153_0 } » 257	#0,78#
{ 93_1 148_1 154_0 } » 257	#0,78#
{ 93_1 148_1 168_0 } » 257	#0,78#
{ 93_1 148_1 169_0 } » 257	#0,78#
{ 93_1 151_0 157_1 } » 257	#0,78#
{ 93_1 152_0 164_1 } » 257	#0,78#
{ 93_1 153_0 164_1 } » 257	#0,78#
{ 93_1 154_0 164_1 } » 257	#0,78#
{ 93_1 164_1 168_0 } » 257	#0,78#
{ 93_1 164_1 169_0 } » 257	#0,78#
{ 109_1 120_0 148_1 } » 257	#0,78#
{ 109_1 121_0 126_1 } » 257	#0,78#
{ 109_1 121_0 132_1 } » 257	#0,78#
{ 109_1 121_0 148_1 } » 257	#0,78#
{ 109_1 121_0 164_1 } » 257	#0,78#
{ 109_1 122_0 132_1 } » 257	#0,78#
{ 109_1 122_0 142_1 } » 257	#0,78#
{ 109_1 122_0 148_1 } » 257	#0,78#
{ 109_1 122_0 164_1 } » 257	#0,78#
{ 109_1 125_1 135_0 } » 257	#0,78#
{ 109_1 125_1 151_0 } » 257	#0,78#
{ 109_1 126_1 136_0 } » 257	#0,78#
{ 109_1 126_1 154_0 } » 257	#0,78#
{ 109_1 126_1 169_0 } » 257	#0,78#
{ 109_1 132_1 136_0 } » 257	#0,78#
{ 109_1 132_1 137_0 } » 257	#0,78#
{ 109_1 132_1 138_0 } » 257	#0,78#
{ 109_1 132_1 152_0 } » 257	#0,78#
{ 109_1 132_1 153_0 } » 257	#0,78#
{ 109_1 132_1 154_0 } » 257	#0,78#
{ 109_1 132_1 168_0 } » 257	#0,78#
{ 109_1 132_1 169_0 } » 257	#0,78#
{ 109_1 136_0 148_1 } » 257	#0,78#
{ 109_1 136_0 164_1 } » 257	#0,78#
{ 109_1 137_0 142_1 } » 257	#0,78#
{ 109_1 137_0 164_1 } » 257	#0,78#
{ 109_1 138_0 142_1 } » 257	#0,78#
{ 109_1 138_0 164_1 } » 257	#0,78#
{ 109_1 142_1 153_0 } » 257	#0,78#
{ 109_1 148_1 152_0 } » 257	#0,78#
{ 109_1 148_1 153_0 } » 257	#0,78#
{ 109_1 148_1 154_0 } » 257	#0,78#
{ 109_1 148_1 168_0 } » 257	#0,78#
{ 109_1 148_1 169_0 } » 257	#0,78#
{ 109_1 151_0 157_1 } » 257	#0,78#
{ 109_1 153_0 164_1 } » 257	#0,78#
{ 109_1 164_1 169_0 } » 257	#0,78#
{ 125_1 136_0 148_1 } » 257	#0,78#
{ 125_1 148_1 152_0 } » 257	#0,78#
{ 125_1 148_1 153_0 } » 257	#0,78#
{ 125_1 148_1 169_0 } » 257	#0,78#
{ 125_1 151_0 157_1 } » 257	#0,78#
{ 126_1 136_0 157_1 } » 257	#0,78#
{ 126_1 137_0 142_1 } » 257	#0,78#
{ 126_1 137_0 173_1 } » 257	#0,78#
{ 126_1 138_0 142_1 } » 257	#0,78#
{ 126_1 138_0 173_1 } » 257	#0,78#
{ 126_1 142_1 153_0 } » 257	#0,78#
{ 126_1 142_1 169_0 } » 257	#0,78#
{ 126_1 152_0 157_1 } » 257	#0,78#
{ 126_1 153_0 157_1 } » 257	#0,78#
{ 126_1 154_0 157_1 } » 257	#0,78#
{ 126_1 157_1 169_0 } » 257	#0,78#
{ 126_1 169_0 173_1 } » 257	#0,78#
{ 132_1 135_0 148_1 } » 257	#0,78#
{ 132_1 136_0 148_1 } » 257	#0,79#
{ 132_1 136_0 157_1 } » 257	#0,78#
{ 132_1 137_0 157_1 } » 257	#0,78#
{ 132_1 137_0 173_1 } » 257	#0,78#
{ 132_1 138_0 148_1 } » 257	#0,79#
{ 132_1 138_0 157_1 } » 257	#0,78#
{ 132_1 138_0 173_1 } » 257	#0,78#
{ 132_1 148_1 151_0 } » 257	#0,78#
{ 132_1 148_1 152_0 } » 257	#0,78#
{ 132_1 148_1 153_0 } » 257	#0,79#
{ 132_1 148_1 154_0 } » 257	#0,78#
{ 132_1 148_1 168_0 } » 257	#0,78#
{ 132_1 148_1 169_0 } » 257	#0,79#
{ 132_1 152_0 157_1 } » 257	#0,78#
{ 132_1 153_0 157_1 } » 257	#0,78#
{ 132_1 153_0 164_1 } » 257	#0,78#
{ 132_1 153_0 173_1 } » 257	#0,78#

{ 132_1 154_0 157_1 } » 257	#0,78#
{ 132_1 154_0 173_1 } » 257	#0,78#
{ 132_1 157_1 168_0 } » 257	#0,78#
{ 132_1 157_1 169_0 } » 257	#0,78#
{ 132_1 169_0 173_1 } » 257	#0,78#
{ 141_1 151_0 157_1 } » 257	#0,78#
{ 142_1 153_0 157_1 } » 257	#0,78#
{ 142_1 154_0 157_1 } » 257	#0,78#
{ 142_1 157_1 169_0 } » 257	#0,78#
{ 142_1 169_0 173_1 } » 257	#0,78#
{ 148_1 151_0 157_1 } » 257	#0,78#
{ 148_1 151_0 164_1 } » 257	#0,78#
{ 148_1 152_0 157_1 } » 257	#0,78#
{ 148_1 152_0 164_1 } » 257	#0,78#
{ 148_1 152_0 173_1 } » 257	#0,78#
{ 148_1 153_0 157_1 } » 257	#0,78#
{ 148_1 153_0 164_1 } » 257	#0,79#
{ 148_1 153_0 173_1 } » 257	#0,78#
{ 148_1 154_0 157_1 } » 257	#0,78#
{ 148_1 154_0 164_1 } » 257	#0,78#
{ 148_1 154_0 173_1 } » 257	#0,78#
{ 148_1 157_1 168_0 } » 257	#0,78#
{ 148_1 157_1 169_0 } » 257	#0,78#
{ 148_1 164_1 167_0 } » 257	#0,78#
{ 148_1 164_1 168_0 } » 257	#0,78#
{ 148_1 164_1 169_0 } » 257	#0,78#
{ 148_1 168_0 173_1 } » 257	#0,78#
{ 148_1 169_0 173_1 } » 257	#0,78#
{ 157_1 164_1 168_0 } » 257	#0,78#
{ 157_1 164_1 169_0 } » 257	#0,78#
{ 93_1 105_0 109_1 173_1 } » 257	#0,78#
{ 93_1 105_0 157_1 173_1 } » 257	#0,78#
{ 93_1 106_0 109_1 126_1 } » 257	#0,78#
{ 93_1 109_1 120_0 141_1 } » 257	#0,78#
{ 93_1 109_1 121_0 173_1 } » 257	#0,78#
{ 93_1 109_1 122_0 126_1 } » 257	#0,78#
{ 93_1 109_1 126_1 137_0 } » 257	#0,78#
{ 93_1 109_1 126_1 138_0 } » 257	#0,78#
{ 93_1 109_1 126_1 153_0 } » 257	#0,78#
{ 93_1 109_1 136_0 173_1 } » 257	#0,78#
{ 93_1 109_1 152_0 173_1 } » 257	#0,78#
{ 93_1 109_1 168_0 173_1 } » 257	#0,78#
{ 93_1 120_0 125_1 141_1 } » 257	#0,78#
{ 93_1 120_0 141_1 157_1 } » 257	#0,78#
{ 93_1 120_0 157_1 173_1 } » 257	#0,78#
{ 93_1 121_0 157_1 173_1 } » 257	#0,78#
{ 93_1 122_0 125_1 173_1 } » 257	#0,78#
{ 93_1 122_0 126_1 157_1 } » 257	#0,78#
{ 93_1 126_1 137_0 157_1 } » 257	#0,78#
{ 93_1 126_1 138_0 157_1 } » 257	#0,78#
{ 93_1 136_0 157_1 173_1 } » 257	#0,78#
{ 93_1 152_0 157_1 173_1 } » 257	#0,78#
{ 93_1 157_1 168_0 173_1 } » 257	#0,78#
{ 109_1 120_0 125_1 173_1 } » 257	#0,78#
{ 109_1 120_0 157_1 173_1 } » 257	#0,78#
{ 109_1 125_1 137_0 148_1 } » 257	#0,78#
{ 109_1 125_1 138_0 148_1 } » 257	#0,78#
{ 109_1 125_1 152_0 173_1 } » 257	#0,78#
{ 109_1 125_1 168_0 173_1 } » 257	#0,78#
{ 109_1 136_0 141_1 173_1 } » 257	#0,78#
{ 109_1 152_0 157_1 173_1 } » 257	#0,78#
{ 109_1 157_1 168_0 173_1 } » 257	#0,78#
{ 125_1 141_1 152_0 173_1 } » 257	#0,78#
{ 125_1 141_1 168_0 173_1 } » 257	#0,78#
{ 125_1 152_0 157_1 173_1 } » 257	#0,78#
{ 125_1 157_1 168_0 173_1 } » 257	#0,78#
{ 132_1 137_0 148_1 164_1 } » 257	#0,78#
{ 141_1 152_0 157_1 173_1 } » 257	#0,78#
{ 141_1 157_1 168_0 173_1 } » 257	#0,78#
{ 93_1 106_0 109_1 125_1 173_1 } » 257	#0,78#
{ 93_1 106_0 109_1 157_1 173_1 } » 257	#0,78#
{ 93_1 109_1 120_0 125_1 157_1 } » 257	#0,78#
{ 93_1 109_1 122_0 157_1 173_1 } » 257	#0,78#
{ 93_1 109_1 125_1 137_0 173_1 } » 257	#0,78#
{ 93_1 109_1 125_1 138_0 173_1 } » 257	#0,78#
{ 93_1 109_1 125_1 153_0 173_1 } » 257	#0,78#
{ 93_1 109_1 125_1 154_0 173_1 } » 257	#0,78#
{ 93_1 109_1 125_1 169_0 173_1 } » 257	#0,78#
{ 93_1 109_1 137_0 157_1 173_1 } » 257	#0,78#
{ 93_1 109_1 138_0 157_1 173_1 } » 257	#0,78#
{ 93_1 109_1 153_0 157_1 173_1 } » 257	#0,78#
{ 93_1 109_1 154_0 157_1 173_1 } » 257	#0,78#
{ 93_1 109_1 157_1 169_0 173_1 } » 257	#0,78#
{ 109_1 120_0 125_1 141_1 157_1 } » 257	#0,78#
{ 109_1 121_0 125_1 141_1 173_1 } » 257	#0,78#
{ 109_1 121_0 125_1 157_1 173_1 } » 257	#0,78#
{ 109_1 121_0 141_1 157_1 173_1 } » 257	#0,78#
{ 109_1 122_0 125_1 141_1 173_1 } » 257	#0,78#
{ 109_1 122_0 125_1 157_1 173_1 } » 257	#0,78#
{ 109_1 122_0 141_1 157_1 173_1 } » 257	#0,78#
{ 109_1 125_1 136_0 157_1 173_1 } » 257	#0,78#
{ 125_1 136_0 141_1 157_1 173_1 } » 257	#0,78#
{ 93_1 105_0 109_1 125_1 141_1 157_1 } » 257	#0,78#
{ 93_1 106_0 109_1 125_1 141_1 157_1 } » 257	#0,78#
{ 93_1 109_1 121_0 125_1 141_1 157_1 } » 257	#0,78#
{ 93_1 109_1 122_0 125_1 141_1 157_1 } » 257	#0,78#
{ 93_1 109_1 125_1 136_0 141_1 157_1 } » 257	#0,78#
{ 93_1 109_1 125_1 137_0 141_1 157_1 } » 257	#0,78#
{ 93_1 109_1 125_1 138_0 141_1 157_1 } » 257	#0,78#
{ 93_1 109_1 125_1 141_1 152_0 157_1 } » 257	#0,78#
{ 93_1 109_1 125_1 141_1 153_0 157_1 } » 257	#0,78#
{ 93_1 109_1 125_1 141_1 154_0 157_1 } » 257	#0,78#
{ 93_1 109_1 125_1 141_1 157_1 168_0 } » 257	#0,78#
{ 93_1 109_1 125_1 141_1 157_1 169_0 } » 257	#0,78#
{ 109_1 125_1 137_0 141_1 157_1 173_1 } » 257	#0,78#
{ 109_1 125_1 138_0 141_1 157_1 173_1 } » 257	#0,78#
{ 109_1 125_1 141_1 153_0 157_1 173_1 } » 257	#0,78#
{ 109_1 125_1 141_1 154_0 157_1 173_1 } » 257	#0,78#
{ 109_1 125_1 141_1 157_1 169_0 173_1 } » 257	#0,78#
{ 72_1 75_0 } » 258	#0,99#
{ 72_1 76_0 } » 258	#0,99#
{ 72_1 77_0 } » 258	#0,99#
{ 72_1 84_0 } » 258	#0,99#
{ 72_1 85_0 } » 258	#0,99#
{ 72_1 91_0 } » 258	#0,99#
{ 72_1 92_0 } » 258	#0,99#
{ 72_1 93_0 } » 258	#0,99#
{ 72_1 100_0 } » 258	#0,99#
{ 72_1 101_0 } » 258	#0,99#
{ 72_1 108_0 } » 258	#0,99#
{ 72_1 109_0 } » 258	#0,99#
{ 72_1 116_0 } » 258	#0,99#
{ 72_1 117_0 } » 258	#0,99#
{ 72_1 123_0 } » 258	#0,99#
{ 72_1 124_0 } » 258	#0,99#
{ 72_1 125_0 } » 258	#0,99#
{ 72_1 132_0 } » 258	#0,99#
{ 72_1 133_0 } » 258	#0,99#
{ 72_1 139_0 } » 258	#0,99#
{ 72_1 140_0 } » 258	#0,99#
{ 72_1 141_0 } » 258	#0,99#
{ 72_1 148_0 } » 258	#0,99#
{ 72_1 149_0 } » 258	#0,99#
{ 72_1 155_0 } » 258	#0,99#
{ 72_1 156_0 } » 258	#0,99#
{ 72_1 157_0 } » 258	#0,99#
{ 72_1 164_0 } » 258	#0,99#
{ 72_1 165_0 } » 258	#0,99#
{ 72_1 171_0 } » 258	#0,99#
{ 72_1 172_0 } » 258	#0,99#
{ 72_1 173_0 } » 258	#0,99#
{ 72_1 180_0 } » 258	#0,99#
{ 72_1 181_0 } » 258	#0,99#
{ 72_1 187_0 } » 258	#0,99#
{ 72_1 188_0 } » 258	#0,99#
{ 72_1 189_0 } » 258	#0,99#
{ 72_1 196_0 } » 258	#0,99#
{ 72_1 197_0 } » 258	#0,99#
{ 72_1 203_0 } » 258	#0,99#
{ 72_1 204_0 } » 258	#0,99#
{ 72_1 205_0 } » 258	#0,99#
{ 88_1 107_0 } » 258	#0,99#
{ 88_1 139_0 } » 258	#0,99#
{ 88_1 187_0 } » 258	#0,99#
{ 88_1 203_0 } » 258	#0,99#
{ 104_1 107_0 } » 258	#0,99#
{ 104_1 187_0 } » 258	#0,99#
{ 104_1 203_0 } » 258	#0,99#
{ 120_1 187_0 } » 258	#0,99#

{ 136_1 187_0 } » 258	#0,99#	{ 104_1 120_1 196_0 } » 258	#0,99#	{ 136_1 152_1 164_0 } » 258	#0,99#
{ 152_1 187_0 } » 258	#0,99#	{ 104_1 120_1 197_0 } » 258	#0,99#	{ 136_1 152_1 165_0 } » 258	#0,99#
{ 168_1 187_0 } » 258	#0,99#	{ 104_1 120_1 204_0 } » 258	#0,99#	{ 136_1 152_1 171_0 } » 258	#0,99#
{ 168_1 203_0 } » 258	#0,99#	{ 104_1 120_1 205_0 } » 258	#0,99#	{ 136_1 152_1 172_0 } » 258	#0,99#
{ 184_1 187_0 } » 258	#0,99#	{ 120_1 123_0 136_1 } » 258	#0,99#	{ 136_1 152_1 173_0 } » 258	#0,99#
{ 184_1 203_0 } » 258	#0,99#	{ 120_1 123_0 152_1 } » 258	#0,99#	{ 136_1 152_1 180_0 } » 258	#0,99#
{ 88_1 91_0 120_1 } » 258	#0,99#	{ 120_1 124_0 136_1 } » 258	#0,99#	{ 136_1 152_1 181_0 } » 258	#0,99#
{ 88_1 92_0 120_1 } » 258	#0,99#	{ 120_1 124_0 152_1 } » 258	#0,99#	{ 136_1 152_1 188_0 } » 258	#0,99#
{ 88_1 93_0 120_1 } » 258	#0,99#	{ 120_1 125_0 136_1 } » 258	#0,99#	{ 136_1 152_1 189_0 } » 258	#0,99#
{ 88_1 100_0 120_1 } » 258	#0,99#	{ 120_1 125_0 152_1 } » 258	#0,99#	{ 136_1 152_1 196_0 } » 258	#0,99#
{ 88_1 101_0 120_1 } » 258	#0,99#	{ 120_1 132_0 136_1 } » 258	#0,99#	{ 136_1 152_1 197_0 } » 258	#0,99#
{ 88_1 108_0 120_1 } » 258	#0,99#	{ 120_1 132_0 152_1 } » 258	#0,99#	{ 136_1 152_1 204_0 } » 258	#0,99#
{ 88_1 109_0 120_1 } » 258	#0,99#	{ 120_1 133_0 136_1 } » 258	#0,99#	{ 136_1 152_1 205_0 } » 258	#0,99#
{ 88_1 116_0 120_1 } » 258	#0,99#	{ 120_1 133_0 152_1 } » 258	#0,99#	{ 152_1 155_0 168_1 } » 258	#0,99#
{ 88_1 117_0 120_1 } » 258	#0,99#	{ 120_1 136_1 139_0 } » 258	#0,99#	{ 152_1 155_0 184_1 } » 258	#0,99#
{ 88_1 120_1 123_0 } » 258	#0,99#	{ 120_1 136_1 140_0 } » 258	#0,99#	{ 152_1 156_0 168_1 } » 258	#0,99#
{ 88_1 120_1 124_0 } » 258	#0,99#	{ 120_1 136_1 141_0 } » 258	#0,99#	{ 152_1 156_0 184_1 } » 258	#0,99#
{ 88_1 120_1 125_0 } » 258	#0,99#	{ 120_1 136_1 148_0 } » 258	#0,99#	{ 152_1 157_0 168_1 } » 258	#0,99#
{ 88_1 120_1 132_0 } » 258	#0,99#	{ 120_1 136_1 149_0 } » 258	#0,99#	{ 152_1 157_0 184_1 } » 258	#0,99#
{ 88_1 120_1 133_0 } » 258	#0,99#	{ 120_1 136_1 155_0 } » 258	#0,99#	{ 152_1 164_0 168_1 } » 258	#0,99#
{ 88_1 120_1 140_0 } » 258	#0,99#	{ 120_1 136_1 156_0 } » 258	#0,99#	{ 152_1 164_0 184_1 } » 258	#0,99#
{ 88_1 120_1 141_0 } » 258	#0,99#	{ 120_1 136_1 157_0 } » 258	#0,99#	{ 152_1 165_0 168_1 } » 258	#0,99#
{ 88_1 120_1 148_0 } » 258	#0,99#	{ 120_1 136_1 164_0 } » 258	#0,99#	{ 152_1 165_0 184_1 } » 258	#0,99#
{ 88_1 120_1 149_0 } » 258	#0,99#	{ 120_1 136_1 165_0 } » 258	#0,99#	{ 152_1 168_1 171_0 } » 258	#0,99#
{ 88_1 120_1 155_0 } » 258	#0,99#	{ 120_1 136_1 171_0 } » 258	#0,99#	{ 152_1 168_1 172_0 } » 258	#0,99#
{ 88_1 120_1 156_0 } » 258	#0,99#	{ 120_1 136_1 172_0 } » 258	#0,99#	{ 152_1 168_1 173_0 } » 258	#0,99#
{ 88_1 120_1 157_0 } » 258	#0,99#	{ 120_1 136_1 173_0 } » 258	#0,99#	{ 152_1 168_1 180_0 } » 258	#0,99#
{ 88_1 120_1 164_0 } » 258	#0,99#	{ 120_1 136_1 180_0 } » 258	#0,99#	{ 152_1 168_1 181_0 } » 258	#0,99#
{ 88_1 120_1 165_0 } » 258	#0,99#	{ 120_1 136_1 181_0 } » 258	#0,99#	{ 152_1 168_1 188_0 } » 258	#0,99#
{ 88_1 120_1 171_0 } » 258	#0,99#	{ 120_1 136_1 188_0 } » 258	#0,99#	{ 152_1 168_1 189_0 } » 258	#0,99#
{ 88_1 120_1 172_0 } » 258	#0,99#	{ 120_1 136_1 189_0 } » 258	#0,99#	{ 152_1 168_1 196_0 } » 258	#0,99#
{ 88_1 120_1 173_0 } » 258	#0,99#	{ 120_1 136_1 196_0 } » 258	#0,99#	{ 152_1 168_1 197_0 } » 258	#0,99#
{ 88_1 120_1 180_0 } » 258	#0,99#	{ 120_1 136_1 197_0 } » 258	#0,99#	{ 152_1 168_1 204_0 } » 258	#0,99#
{ 88_1 120_1 181_0 } » 258	#0,99#	{ 120_1 136_1 203_0 } » 258	#0,99#	{ 152_1 168_1 205_0 } » 258	#0,99#
{ 88_1 120_1 188_0 } » 258	#0,99#	{ 120_1 136_1 204_0 } » 258	#0,99#	{ 152_1 171_0 184_1 } » 258	#0,99#
{ 88_1 120_1 189_0 } » 258	#0,99#	{ 120_1 136_1 205_0 } » 258	#0,99#	{ 152_1 172_0 184_1 } » 258	#0,99#
{ 88_1 120_1 196_0 } » 258	#0,99#	{ 120_1 139_0 152_1 } » 258	#0,99#	{ 152_1 173_0 184_1 } » 258	#0,99#
{ 88_1 120_1 197_0 } » 258	#0,99#	{ 120_1 140_0 152_1 } » 258	#0,99#	{ 152_1 180_0 184_1 } » 258	#0,99#
{ 88_1 120_1 204_0 } » 258	#0,99#	{ 120_1 141_0 152_1 } » 258	#0,99#	{ 152_1 181_0 184_1 } » 258	#0,99#
{ 88_1 120_1 205_0 } » 258	#0,99#	{ 120_1 148_0 152_1 } » 258	#0,99#	{ 152_1 184_1 188_0 } » 258	#0,99#
{ 104_1 108_0 120_1 } » 258	#0,99#	{ 120_1 149_0 152_1 } » 258	#0,99#	{ 152_1 184_1 189_0 } » 258	#0,99#
{ 104_1 109_0 120_1 } » 258	#0,99#	{ 120_1 152_1 155_0 } » 258	#0,99#	{ 152_1 184_1 196_0 } » 258	#0,99#
{ 104_1 116_0 120_1 } » 258	#0,99#	{ 120_1 152_1 156_0 } » 258	#0,99#	{ 152_1 184_1 197_0 } » 258	#0,99#
{ 104_1 117_0 120_1 } » 258	#0,99#	{ 120_1 152_1 157_0 } » 258	#0,99#	{ 152_1 184_1 204_0 } » 258	#0,99#
{ 104_1 120_1 123_0 } » 258	#0,99#	{ 120_1 152_1 164_0 } » 258	#0,99#	{ 152_1 184_1 205_0 } » 258	#0,99#
{ 104_1 120_1 124_0 } » 258	#0,99#	{ 120_1 152_1 165_0 } » 258	#0,99#	{ 53_1 } » 259	#0,53#
{ 104_1 120_1 125_0 } » 258	#0,99#	{ 120_1 152_1 171_0 } » 258	#0,99#	{ 122_1 } » 259	#0,53#
{ 104_1 120_1 132_0 } » 258	#0,99#	{ 120_1 152_1 172_0 } » 258	#0,99#	{ 138_1 } » 259	#0,60#
{ 104_1 120_1 133_0 } » 258	#0,99#	{ 120_1 152_1 173_0 } » 258	#0,99#	{ 139_1 } » 259	#0,60#
{ 104_1 120_1 139_0 } » 258	#0,99#	{ 120_1 152_1 180_0 } » 258	#0,99#	{ 140_1 } » 259	#0,56#
{ 104_1 120_1 140_0 } » 258	#0,99#	{ 120_1 152_1 181_0 } » 258	#0,99#	{ 152_1 } » 259	#0,57#
{ 104_1 120_1 141_0 } » 258	#0,99#	{ 120_1 152_1 188_0 } » 258	#0,99#	{ 153_1 } » 259	#0,61#
{ 104_1 120_1 148_0 } » 258	#0,99#	{ 120_1 152_1 189_0 } » 258	#0,99#	{ 155_1 } » 259	#0,59#
{ 104_1 120_1 149_0 } » 258	#0,99#	{ 120_1 152_1 196_0 } » 258	#0,99#	{ 166_1 } » 259	#0,55#
{ 104_1 120_1 155_0 } » 258	#0,99#	{ 120_1 152_1 197_0 } » 258	#0,99#	{ 167_1 } » 259	#0,60#
{ 104_1 120_1 156_0 } » 258	#0,99#	{ 120_1 152_1 203_0 } » 258	#0,99#	{ 171_1 } » 259	#0,58#
{ 104_1 120_1 157_0 } » 258	#0,99#	{ 120_1 152_1 204_0 } » 258	#0,99#	{ 181_1 } » 259	#0,56#
{ 104_1 120_1 164_0 } » 258	#0,99#	{ 120_1 152_1 205_0 } » 258	#0,99#	{ 196_1 } » 259	#0,55#
{ 104_1 120_1 165_0 } » 258	#0,99#	{ 136_1 139_0 152_1 } » 258	#0,99#	{ 201_1 } » 259	#0,62#
{ 104_1 120_1 171_0 } » 258	#0,99#	{ 136_1 140_0 152_1 } » 258	#0,99#	{ 202_1 } » 259	#0,55#
{ 104_1 120_1 172_0 } » 258	#0,99#	{ 136_1 141_0 152_1 } » 258	#0,99#	{ 214_1 } » 259	#0,54#
{ 104_1 120_1 173_0 } » 258	#0,99#	{ 136_1 148_0 152_1 } » 258	#0,99#	{ 37_1 85_0 } » 259	#0,52#
{ 104_1 120_1 180_0 } » 258	#0,99#	{ 136_1 149_0 152_1 } » 258	#0,99#	{ 37_1 86_0 } » 259	#0,55#
{ 104_1 120_1 181_0 } » 258	#0,99#	{ 136_1 152_1 155_0 } » 258	#0,99#	{ 37_1 87_0 } » 259	#0,52#
{ 104_1 120_1 188_0 } » 258	#0,99#	{ 136_1 152_1 156_0 } » 258	#0,99#	{ 37_1 100_0 } » 259	#0,55#
{ 104_1 120_1 189_0 } » 258	#0,99#	{ 136_1 152_1 157_0 } » 258	#0,99#	{ 37_1 103_0 } » 259	#0,52#

{ 37_1 118_0 } » 259	#0,53#	{ 58_1 87_0 } » 259	#0,57#	{ 185_1 186_1 } » 259	#0,53#
{ 38_1 40_1 } » 259	#0,53#	{ 58_1 88_0 } » 259	#0,53#	{ 185_1 205_0 } » 259	#0,54#
{ 38_1 70_0 } » 259	#0,54#	{ 58_1 100_0 } » 259	#0,53#	{ 197_1 198_1 } » 259	#0,53#
{ 38_1 71_0 } » 259	#0,56#	{ 58_1 101_0 } » 259	#0,55#	{ 198_1 199_1 } » 259	#0,55#
{ 38_1 84_0 } » 259	#0,53#	{ 58_1 102_0 } » 259	#0,57#	{ 199_1 200_1 } » 259	#0,53#
{ 38_1 85_0 } » 259	#0,58#	{ 58_1 103_0 } » 259	#0,56#	{ 37_1 38_1 101_0 } » 259	#0,53#
{ 38_1 88_0 } » 259	#0,54#	{ 58_1 116_0 } » 259	#0,55#	{ 37_1 38_1 102_0 } » 259	#0,52#
{ 38_1 104_0 } » 259	#0,54#	{ 58_1 117_0 } » 259	#0,56#	{ 37_1 38_1 116_0 } » 259	#0,53#
{ 38_1 119_0 } » 259	#0,56#	{ 58_1 118_0 } » 259	#0,55#	{ 37_1 38_1 117_0 } » 259	#0,52#
{ 38_1 134_0 } » 259	#0,53#	{ 58_1 119_0 } » 259	#0,53#	{ 38_1 39_1 86_0 } » 259	#0,56#
{ 38_1 148_0 } » 259	#0,53#	{ 58_1 132_0 } » 259	#0,53#	{ 38_1 39_1 87_0 } » 259	#0,54#
{ 38_1 205_0 } » 259	#0,54#	{ 59_1 85_0 } » 259	#0,55#	{ 38_1 39_1 100_0 } » 259	#0,56#
{ 39_1 41_1 } » 259	#0,52#	{ 59_1 89_0 } » 259	#0,54#	{ 38_1 39_1 101_0 } » 259	#0,58#
{ 39_1 70_0 } » 259	#0,54#	{ 59_1 92_1 } » 259	#0,52#	{ 38_1 39_1 102_0 } » 259	#0,57#
{ 39_1 71_0 } » 259	#0,56#	{ 59_1 120_0 } » 259	#0,57#	{ 38_1 39_1 103_0 } » 259	#0,54#
{ 39_1 72_0 } » 259	#0,53#	{ 59_1 132_0 } » 259	#0,57#	{ 38_1 39_1 116_0 } » 259	#0,57#
{ 39_1 84_0 } » 259	#0,54#	{ 59_1 133_0 } » 259	#0,55#	{ 38_1 39_1 117_0 } » 259	#0,57#
{ 39_1 85_0 } » 259	#0,59#	{ 59_1 134_0 } » 259	#0,52#	{ 38_1 39_1 118_0 } » 259	#0,55#
{ 39_1 88_0 } » 259	#0,56#	{ 60_1 72_0 } » 259	#0,54#	{ 38_1 39_1 132_0 } » 259	#0,54#
{ 39_1 104_0 } » 259	#0,55#	{ 60_1 73_0 } » 259	#0,54#	{ 38_1 39_1 133_0 } » 259	#0,52#
{ 39_1 119_0 } » 259	#0,58#	{ 60_1 76_1 } » 259	#0,54#	{ 39_1 40_1 86_0 } » 259	#0,54#
{ 39_1 120_0 } » 259	#0,52#	{ 60_1 87_0 } » 259	#0,55#	{ 39_1 40_1 87_0 } » 259	#0,53#
{ 39_1 134_0 } » 259	#0,54#	{ 60_1 88_0 } » 259	#0,55#	{ 39_1 40_1 100_0 } » 259	#0,54#
{ 39_1 148_0 } » 259	#0,53#	{ 60_1 89_0 } » 259	#0,54#	{ 39_1 40_1 101_0 } » 259	#0,56#
{ 39_1 205_0 } » 259	#0,54#	{ 60_1 101_0 } » 259	#0,53#	{ 39_1 40_1 102_0 } » 259	#0,56#
{ 40_1 71_0 } » 259	#0,55#	{ 60_1 102_0 } » 259	#0,55#	{ 39_1 40_1 103_0 } » 259	#0,53#
{ 40_1 72_0 } » 259	#0,53#	{ 60_1 103_0 } » 259	#0,55#	{ 39_1 40_1 116_0 } » 259	#0,56#
{ 40_1 85_0 } » 259	#0,56#	{ 60_1 104_0 } » 259	#0,54#	{ 39_1 40_1 117_0 } » 259	#0,55#
{ 40_1 88_0 } » 259	#0,56#	{ 60_1 117_0 } » 259	#0,53#	{ 39_1 40_1 118_0 } » 259	#0,53#
{ 40_1 104_0 } » 259	#0,54#	{ 60_1 118_0 } » 259	#0,53#	{ 39_1 40_1 132_0 } » 259	#0,52#
{ 40_1 119_0 } » 259	#0,56#	{ 60_1 119_0 } » 259	#0,52#	{ 40_1 41_1 86_0 } » 259	#0,52#
{ 40_1 133_0 } » 259	#0,57#	{ 75_1 85_0 } » 259	#0,53#	{ 40_1 41_1 87_0 } » 259	#0,53#
{ 40_1 134_0 } » 259	#0,53#	{ 75_1 86_0 } » 259	#0,57#	{ 40_1 41_1 100_0 } » 259	#0,52#
{ 41_1 59_1 } » 259	#0,53#	{ 75_1 100_0 } » 259	#0,56#	{ 40_1 41_1 101_0 } » 259	#0,54#
{ 41_1 70_0 } » 259	#0,53#	{ 75_1 104_0 } » 259	#0,57#	{ 40_1 41_1 102_0 } » 259	#0,55#
{ 41_1 84_0 } » 259	#0,52#	{ 75_1 118_0 } » 259	#0,58#	{ 40_1 41_1 103_0 } » 259	#0,53#
{ 41_1 85_0 } » 259	#0,57#	{ 75_1 119_0 } » 259	#0,56#	{ 40_1 41_1 116_0 } » 259	#0,54#
{ 41_1 104_0 } » 259	#0,58#	{ 75_1 120_0 } » 259	#0,53#	{ 40_1 41_1 117_0 } » 259	#0,54#
{ 41_1 120_0 } » 259	#0,54#	{ 75_1 132_0 } » 259	#0,56#	{ 40_1 41_1 118_0 } » 259	#0,52#
{ 41_1 132_0 } » 259	#0,59#	{ 75_1 133_0 } » 259	#0,54#	{ 41_1 42_1 71_0 } » 259	#0,53#
{ 41_1 133_0 } » 259	#0,58#	{ 76_1 85_0 } » 259	#0,53#	{ 41_1 42_1 72_0 } » 259	#0,53#
{ 41_1 134_0 } » 259	#0,54#	{ 76_1 105_0 } » 259	#0,55#	{ 41_1 42_1 86_0 } » 259	#0,53#
{ 41_1 148_0 } » 259	#0,52#	{ 76_1 132_0 } » 259	#0,55#	{ 41_1 42_1 87_0 } » 259	#0,55#
{ 42_1 43_1 } » 259	#0,55#	{ 76_1 133_0 } » 259	#0,53#	{ 41_1 42_1 88_0 } » 259	#0,54#
{ 42_1 70_0 } » 259	#0,53#	{ 91_1 100_0 } » 259	#0,55#	{ 41_1 42_1 100_0 } » 259	#0,53#
{ 42_1 73_0 } » 259	#0,53#	{ 91_1 132_0 } » 259	#0,56#	{ 41_1 42_1 101_0 } » 259	#0,55#
{ 42_1 76_1 } » 259	#0,53#	{ 91_1 133_0 } » 259	#0,54#	{ 41_1 42_1 102_0 } » 259	#0,56#
{ 42_1 85_0 } » 259	#0,56#	{ 92_1 132_0 } » 259	#0,56#	{ 41_1 42_1 103_0 } » 259	#0,56#
{ 42_1 89_0 } » 259	#0,55#	{ 92_1 133_0 } » 259	#0,54#	{ 41_1 42_1 116_0 } » 259	#0,54#
{ 42_1 120_0 } » 259	#0,58#	{ 107_1 123_1 } » 259	#0,53#	{ 41_1 42_1 117_0 } » 259	#0,55#
{ 42_1 132_0 } » 259	#0,59#	{ 107_1 132_0 } » 259	#0,55#	{ 41_1 42_1 118_0 } » 259	#0,55#
{ 42_1 133_0 } » 259	#0,57#	{ 107_1 133_0 } » 259	#0,54#	{ 41_1 42_1 119_0 } » 259	#0,53#
{ 42_1 134_0 } » 259	#0,54#	{ 108_1 132_0 } » 259	#0,55#	{ 42_1 59_1 71_0 } » 259	#0,53#
{ 42_1 148_0 } » 259	#0,53#	{ 108_1 133_0 } » 259	#0,55#	{ 42_1 59_1 72_0 } » 259	#0,55#
{ 43_1 72_0 } » 259	#0,55#	{ 123_1 132_0 } » 259	#0,55#	{ 42_1 59_1 86_0 } » 259	#0,52#
{ 43_1 87_0 } » 259	#0,55#	{ 123_1 133_0 } » 259	#0,54#	{ 42_1 59_1 87_0 } » 259	#0,55#
{ 43_1 88_0 } » 259	#0,55#	{ 124_1 132_0 } » 259	#0,52#	{ 42_1 59_1 88_0 } » 259	#0,55#
{ 43_1 101_0 } » 259	#0,53#	{ 154_1 169_1 } » 259	#0,53#	{ 42_1 59_1 100_0 } » 259	#0,52#
{ 43_1 102_0 } » 259	#0,55#	{ 154_1 170_1 } » 259	#0,53#	{ 42_1 59_1 101_0 } » 259	#0,54#
{ 43_1 103_0 } » 259	#0,55#	{ 168_1 169_1 } » 259	#0,55#	{ 42_1 59_1 102_0 } » 259	#0,56#
{ 43_1 104_0 } » 259	#0,53#	{ 169_1 170_1 } » 259	#0,55#	{ 42_1 59_1 103_0 } » 259	#0,56#
{ 43_1 116_0 } » 259	#0,52#	{ 169_1 185_1 } » 259	#0,52#	{ 42_1 59_1 104_0 } » 259	#0,53#
{ 43_1 117_0 } » 259	#0,53#	{ 169_1 205_0 } » 259	#0,53#	{ 42_1 59_1 116_0 } » 259	#0,54#
{ 43_1 118_0 } » 259	#0,53#	{ 170_1 205_0 } » 259	#0,53#	{ 42_1 59_1 117_0 } » 259	#0,55#
{ 58_1 71_0 } » 259	#0,54#	{ 182_1 183_1 } » 259	#0,53#	{ 42_1 59_1 118_0 } » 259	#0,55#
{ 58_1 86_0 } » 259	#0,54#	{ 183_1 184_1 } » 259	#0,53#	{ 42_1 59_1 119_0 } » 259	#0,53#

{ 59_1 75_1 102_0 } » 259	#0,53#	{ 39_1 154_0 } » 260	#0,68#	{ 108_1 148_0 } » 260	#0,70#
{ 59_1 76_1 87_0 } » 259	#0,52#	{ 39_1 158_1 } » 260	#0,68#	{ 108_1 149_0 } » 260	#0,69#
{ 59_1 76_1 88_0 } » 259	#0,53#	{ 39_1 164_0 } » 260	#0,70#	{ 108_1 150_0 } » 260	#0,69#
{ 59_1 76_1 102_0 } » 259	#0,53#	{ 39_1 165_0 } » 260	#0,72#	{ 108_1 151_0 } » 260	#0,69#
{ 59_1 76_1 103_0 } » 259	#0,53#	{ 39_1 166_0 } » 260	#0,72#	{ 108_1 152_0 } » 260	#0,69#
{ 75_1 87_0 91_1 } » 259	#0,54#	{ 39_1 167_0 } » 260	#0,72#	{ 108_1 153_0 } » 260	#0,69#
{ 75_1 88_0 91_1 } » 259	#0,53#	{ 39_1 168_0 } » 260	#0,72#	{ 108_1 164_0 } » 260	#0,68#
{ 75_1 91_1 101_0 } » 259	#0,52#	{ 39_1 169_0 } » 260	#0,70#	{ 108_1 165_0 } » 260	#0,69#
{ 75_1 91_1 102_0 } » 259	#0,54#	{ 39_1 170_0 } » 260	#0,68#	{ 108_1 166_0 } » 260	#0,70#
{ 75_1 91_1 103_0 } » 259	#0,54#	{ 39_1 183_0 } » 260	#0,68#	{ 108_1 167_0 } » 260	#0,70#
{ 75_1 91_1 116_0 } » 259	#0,52#	{ 40_1 116_0 } » 260	#0,68#	{ 108_1 168_0 } » 260	#0,70#
{ 75_1 91_1 117_0 } » 259	#0,53#	{ 40_1 132_0 } » 260	#0,68#	{ 108_1 169_0 } » 260	#0,69#
{ 76_1 86_0 92_1 } » 259	#0,52#	{ 40_1 148_0 } » 260	#0,70#	{ 125_1 132_0 } » 260	#0,70#
{ 76_1 87_0 92_1 } » 259	#0,56#	{ 40_1 149_0 } » 260	#0,69#	{ 125_1 133_0 } » 260	#0,68#
{ 76_1 87_0 108_1 } » 259	#0,52#	{ 40_1 150_0 } » 260	#0,69#	{ 125_1 142_1 } » 260	#0,68#
{ 76_1 92_1 100_0 } » 259	#0,52#	{ 40_1 151_0 } » 260	#0,69#	{ 125_1 150_0 } » 260	#0,72#
{ 76_1 92_1 101_0 } » 259	#0,54#	{ 40_1 152_0 } » 260	#0,70#	{ 125_1 151_0 } » 260	#0,72#
{ 76_1 92_1 116_0 } » 259	#0,54#	{ 40_1 153_0 } » 260	#0,70#	{ 125_1 164_0 } » 260	#0,70#
{ 76_1 92_1 117_0 } » 259	#0,55#	{ 40_1 164_0 } » 260	#0,70#	{ 125_1 169_0 } » 260	#0,71#
{ 76_1 92_1 118_0 } » 259	#0,55#	{ 40_1 165_0 } » 260	#0,71#	{ 125_1 170_0 } » 260	#0,69#
{ 76_1 92_1 119_0 } » 259	#0,55#	{ 40_1 166_0 } » 260	#0,71#	{ 125_1 174_1 } » 260	#0,68#
{ 76_1 92_1 120_0 } » 259	#0,52#	{ 40_1 167_0 } » 260	#0,71#	{ 125_1 183_0 } » 260	#0,68#
{ 76_1 108_1 117_0 } » 259	#0,52#	{ 40_1 168_0 } » 260	#0,71#	{ 125_1 184_0 } » 260	#0,68#
{ 76_1 108_1 118_0 } » 259	#0,52#	{ 40_1 169_0 } » 260	#0,70#	{ 141_1 173_1 } » 260	#0,68#
{ 76_1 108_1 119_0 } » 259	#0,52#	{ 42_1 116_0 } » 260	#0,68#	{ 141_1 182_0 } » 260	#0,69#
{ 91_1 101_0 107_1 } » 259	#0,53#	{ 42_1 132_0 } » 260	#0,69#	{ 157_1 181_0 } » 260	#0,68#
{ 91_1 102_0 107_1 } » 259	#0,55#	{ 43_1 71_0 } » 260	#0,69#	{ 157_1 182_0 } » 260	#0,71#
{ 91_1 103_0 107_1 } » 259	#0,55#	{ 43_1 86_0 } » 260	#0,68#	{ 157_1 184_0 } » 260	#0,72#
{ 91_1 104_0 107_1 } » 259	#0,53#	{ 43_1 100_0 } » 260	#0,68#	{ 157_1 189_1 } » 260	#0,68#
{ 91_1 107_1 116_0 } » 259	#0,53#	{ 43_1 116_0 } » 260	#0,69#	{ 158_1 159_1 } » 260	#0,68#
{ 91_1 107_1 117_0 } » 259	#0,54#	{ 43_1 132_0 } » 260	#0,70#	{ 158_1 215_1 } » 260	#0,70#
{ 91_1 107_1 118_0 } » 259	#0,54#	{ 43_1 133_0 } » 260	#0,68#	{ 173_1 180_0 } » 260	#0,71#
{ 91_1 107_1 119_0 } » 259	#0,53#	{ 43_1 148_0 } » 260	#0,72#	{ 174_1 186_0 } » 260	#0,69#
{ 92_1 101_0 108_1 } » 259	#0,54#	{ 43_1 149_0 } » 260	#0,71#	{ 174_1 215_1 } » 260	#0,68#
{ 92_1 105_0 108_1 } » 259	#0,53#	{ 43_1 150_0 } » 260	#0,71#	{ 189_1 203_1 } » 260	#0,70#
{ 92_1 108_1 116_0 } » 259	#0,54#	{ 43_1 151_0 } » 260	#0,71#	{ 189_1 204_1 } » 260	#0,71#
{ 92_1 108_1 117_0 } » 259	#0,55#	{ 43_1 152_0 } » 260	#0,72#	{ 189_1 216_1 } » 260	#0,68#
{ 92_1 108_1 118_0 } » 259	#0,55#	{ 43_1 164_0 } » 260	#0,70#	{ 214_1 215_1 } » 260	#0,69#
{ 92_1 108_1 119_0 } » 259	#0,55#	{ 43_1 165_0 } » 260	#0,71#	{ 215_1 216_1 } » 260	#0,71#
{ 92_1 108_1 120_0 } » 259	#0,53#	{ 43_1 166_0 } » 260	#0,72#	{ 216_1 217_1 } » 260	#0,69#
{ 108_1 118_0 124_1 } » 259	#0,53#	{ 43_1 168_0 } » 260	#0,73#	{ 43_1 153_0 158_1 } » 260	#0,68#
{ 108_1 119_0 124_1 } » 259	#0,52#	{ 43_1 169_0 } » 260	#0,72#	{ 43_1 154_0 158_1 } » 260	#0,68#
{ 76_1 88_0 92_1 108_1 } » 259	#0,52#	{ 43_1 170_0 } » 260	#0,69#	{ 43_1 158_1 167_0 } » 260	#0,68#
{ 76_1 89_0 92_1 108_1 } » 259	#0,53#	{ 43_1 174_1 } » 260	#0,69#	{ 125_1 148_0 158_1 } » 260	#0,68#
{ 76_1 92_1 102_0 108_1 } » 259	#0,53#	{ 43_1 183_0 } » 260	#0,69#	{ 125_1 149_0 158_1 } » 260	#0,68#
{ 76_1 92_1 103_0 108_1 } » 259	#0,53#	{ 43_1 184_0 } » 260	#0,68#	{ 125_1 152_0 158_1 } » 260	#0,68#
{ 76_1 92_1 104_0 108_1 } » 259	#0,52#	{ 60_1 148_0 } » 260	#0,69#	{ 125_1 153_0 158_1 } » 260	#0,69#
{ 44_1 } » 260	#0,68#	{ 60_1 149_0 } » 260	#0,68#	{ 125_1 154_0 158_1 } » 260	#0,69#
{ 76_1 } » 260	#0,70#	{ 60_1 150_0 } » 260	#0,68#	{ 125_1 158_1 165_0 } » 260	#0,68#
{ 91_1 } » 260	#0,70#	{ 60_1 151_0 } » 260	#0,68#	{ 125_1 158_1 166_0 } » 260	#0,68#
{ 92_1 } » 260	#0,70#	{ 60_1 152_0 } » 260	#0,69#	{ 125_1 158_1 167_0 } » 260	#0,68#
{ 106_1 } » 260	#0,68#	{ 60_1 153_0 } » 260	#0,69#	{ 125_1 158_1 168_0 } » 260	#0,68#
{ 109_1 } » 260	#0,69#	{ 60_1 165_0 } » 260	#0,68#	{ 141_1 142_1 153_0 } » 260	#0,68#
{ 126_1 } » 260	#0,69#	{ 60_1 166_0 } » 260	#0,69#	{ 141_1 142_1 154_0 } » 260	#0,68#
{ 188_1 } » 260	#0,71#	{ 60_1 167_0 } » 260	#0,70#	{ 141_1 142_1 158_1 } » 260	#0,68#
{ 39_1 71_0 } » 260	#0,68#	{ 60_1 168_0 } » 260	#0,70#	{ 141_1 149_0 158_1 } » 260	#0,71#
{ 39_1 86_0 } » 260	#0,68#	{ 60_1 169_0 } » 260	#0,69#	{ 141_1 149_0 174_1 } » 260	#0,68#
{ 39_1 100_0 } » 260	#0,68#	{ 107_1 148_0 } » 260	#0,68#	{ 141_1 150_0 158_1 } » 260	#0,71#
{ 39_1 116_0 } » 260	#0,68#	{ 107_1 152_0 } » 260	#0,68#	{ 141_1 150_0 174_1 } » 260	#0,68#
{ 39_1 132_0 } » 260	#0,68#	{ 107_1 153_0 } » 260	#0,68#	{ 141_1 151_0 158_1 } » 260	#0,71#
{ 39_1 148_0 } » 260	#0,71#	{ 107_1 165_0 } » 260	#0,68#	{ 141_1 151_0 174_1 } » 260	#0,68#
{ 39_1 149_0 } » 260	#0,70#	{ 107_1 166_0 } » 260	#0,68#	{ 141_1 152_0 157_1 } » 260	#0,68#
{ 39_1 150_0 } » 260	#0,71#	{ 107_1 167_0 } » 260	#0,69#	{ 141_1 153_0 157_1 } » 260	#0,68#
{ 39_1 151_0 } » 260	#0,71#	{ 107_1 168_0 } » 260	#0,69#	{ 141_1 154_0 157_1 } » 260	#0,68#
{ 39_1 152_0 } » 260	#0,71#	{ 107_1 169_0 } » 260	#0,68#	{ 141_1 155_0 158_1 } » 260	#0,70#
{ 39_1 153_0 } » 260	#0,71#	{ 108_1 132_0 } » 260	#0,68#	{ 141_1 158_1 164_0 } » 260	#0,69#

{ 141_1 158_1 165_0 } » 260	#0,71#
{ 141_1 158_1 183_0 } » 260	#0,68#
{ 141_1 158_1 184_0 } » 260	#0,68#
{ 141_1 165_0 174_1 } » 260	#0,68#
{ 142_1 148_0 173_1 } » 260	#0,68#
{ 142_1 148_0 189_1 } » 260	#0,68#
{ 142_1 152_0 189_1 } » 260	#0,68#
{ 142_1 158_1 180_0 } » 260	#0,68#
{ 142_1 165_0 173_1 } » 260	#0,68#
{ 142_1 165_0 189_1 } » 260	#0,68#
{ 142_1 166_0 173_1 } » 260	#0,68#
{ 142_1 168_0 173_1 } » 260	#0,68#
{ 157_1 158_1 164_0 } » 260	#0,69#
{ 157_1 158_1 165_0 } » 260	#0,70#
{ 157_1 158_1 166_0 } » 260	#0,71#
{ 157_1 158_1 167_0 } » 260	#0,71#
{ 157_1 158_1 168_0 } » 260	#0,71#
{ 157_1 158_1 169_0 } » 260	#0,70#
{ 157_1 158_1 170_0 } » 260	#0,68#
{ 157_1 158_1 173_1 } » 260	#0,69#
{ 157_1 158_1 174_1 } » 260	#0,69#
{ 157_1 158_1 183_0 } » 260	#0,68#
{ 157_1 164_0 173_1 } » 260	#0,69#
{ 157_1 165_0 173_1 } » 260	#0,71#
{ 157_1 165_0 174_1 } » 260	#0,69#
{ 157_1 166_0 173_1 } » 260	#0,71#
{ 157_1 166_0 174_1 } » 260	#0,70#
{ 157_1 167_0 173_1 } » 260	#0,72#
{ 157_1 167_0 174_1 } » 260	#0,70#
{ 157_1 168_0 173_1 } » 260	#0,72#
{ 157_1 168_0 174_1 } » 260	#0,70#
{ 157_1 169_0 173_1 } » 260	#0,71#
{ 157_1 169_0 174_1 } » 260	#0,70#
{ 157_1 170_0 173_1 } » 260	#0,69#
{ 157_1 170_0 174_1 } » 260	#0,69#
{ 157_1 173_1 174_1 } » 260	#0,68#
{ 157_1 173_1 183_0 } » 260	#0,68#
{ 158_1 165_0 216_1 } » 260	#0,69#
{ 158_1 166_0 203_1 } » 260	#0,68#
{ 158_1 166_0 216_1 } » 260	#0,69#
{ 158_1 167_0 203_1 } » 260	#0,68#
{ 158_1 167_0 216_1 } » 260	#0,69#
{ 158_1 168_0 203_1 } » 260	#0,68#
{ 158_1 168_0 216_1 } » 260	#0,69#
{ 158_1 169_0 216_1 } » 260	#0,68#
{ 158_1 170_0 216_1 } » 260	#0,68#
{ 158_1 173_1 181_0 } » 260	#0,68#
{ 158_1 173_1 182_0 } » 260	#0,70#
{ 158_1 173_1 184_0 } » 260	#0,70#
{ 158_1 174_1 180_0 } » 260	#0,72#
{ 158_1 174_1 203_1 } » 260	#0,68#
{ 158_1 174_1 204_1 } » 260	#0,68#
{ 158_1 174_1 216_1 } » 260	#0,68#
{ 158_1 181_0 189_1 } » 260	#0,68#
{ 173_1 174_1 182_0 } » 260	#0,69#
{ 173_1 174_1 184_0 } » 260	#0,71#
{ 173_1 182_0 189_1 } » 260	#0,69#
{ 173_1 183_0 189_1 } » 260	#0,70#
{ 173_1 184_0 189_1 } » 260	#0,71#
{ 173_1 185_0 189_1 } » 260	#0,69#
{ 174_1 181_0 189_1 } » 260	#0,70#
{ 174_1 183_0 190_1 } » 260	#0,68#
{ 174_1 184_0 190_1 } » 260	#0,68#
{ 141_1 148_0 158_1 174_1 } » 260	#0,68#
{ 141_1 152_0 158_1 174_1 } » 260	#0,68#
{ 141_1 153_0 158_1 174_1 } » 260	#0,69#

{ 141_1 154_0 158_1 174_1 } » 260	#0,69#
{ 141_1 158_1 166_0 174_1 } » 260	#0,68#
{ 141_1 158_1 167_0 174_1 } » 260	#0,68#
{ 141_1 158_1 168_0 174_1 } » 260	#0,68#
{ 141_1 158_1 169_0 174_1 } » 260	#0,68#
{ 141_1 158_1 170_0 174_1 } » 260	#0,68#
{ 142_1 148_0 158_1 174_1 } » 260	#0,75#
{ 142_1 149_0 158_1 174_1 } » 260	#0,74#
{ 142_1 150_0 158_1 174_1 } » 260	#0,74#
{ 142_1 151_0 158_1 174_1 } » 260	#0,74#
{ 142_1 152_0 158_1 173_1 } » 260	#0,68#
{ 142_1 152_0 158_1 174_1 } » 260	#0,75#
{ 142_1 153_0 158_1 173_1 } » 260	#0,68#
{ 142_1 154_0 158_1 173_1 } » 260	#0,69#
{ 142_1 155_0 158_1 173_1 } » 260	#0,68#
{ 142_1 158_1 164_0 174_1 } » 260	#0,73#
{ 142_1 158_1 165_0 174_1 } » 260	#0,75#
{ 142_1 158_1 166_0 174_1 } » 260	#0,75#
{ 142_1 158_1 166_0 189_1 } » 260	#0,68#
{ 142_1 158_1 167_0 173_1 } » 260	#0,68#
{ 142_1 158_1 169_0 174_1 } » 260	#0,75#
{ 142_1 158_1 169_0 189_1 } » 260	#0,68#
{ 142_1 158_1 174_1 181_0 } » 260	#0,68#
{ 142_1 158_1 174_1 182_0 } » 260	#0,70#
{ 142_1 158_1 174_1 183_0 } » 260	#0,72#
{ 142_1 158_1 174_1 184_0 } » 260	#0,71#
{ 142_1 158_1 174_1 185_0 } » 260	#0,68#
{ 142_1 169_0 174_1 189_1 } » 260	#0,68#
{ 158_1 164_0 173_1 174_1 } » 260	#0,70#
{ 158_1 164_0 174_1 189_1 } » 260	#0,72#
{ 158_1 165_0 173_1 174_1 } » 260	#0,71#
{ 158_1 165_0 174_1 189_1 } » 260	#0,74#
{ 158_1 166_0 173_1 174_1 } » 260	#0,72#
{ 158_1 166_0 174_1 189_1 } » 260	#0,74#
{ 158_1 167_0 173_1 174_1 } » 260	#0,72#
{ 158_1 167_0 173_1 189_1 } » 260	#0,68#
{ 158_1 168_0 173_1 174_1 } » 260	#0,72#
{ 158_1 168_0 173_1 189_1 } » 260	#0,68#
{ 158_1 169_0 173_1 174_1 } » 260	#0,72#
{ 158_1 169_0 174_1 189_1 } » 260	#0,74#
{ 158_1 170_0 173_1 174_1 } » 260	#0,71#
{ 158_1 170_0 174_1 190_1 } » 260	#0,68#
{ 158_1 171_0 174_1 189_1 } » 260	#0,69#
{ 158_1 173_1 174_1 183_0 } » 260	#0,68#
{ 158_1 173_1 174_1 189_1 } » 260	#0,69#
{ 158_1 174_1 182_0 189_1 } » 260	#0,70#
{ 158_1 174_1 183_0 189_1 } » 260	#0,71#
{ 158_1 174_1 184_0 189_1 } » 260	#0,72#
{ 158_1 174_1 185_0 189_1 } » 260	#0,69#
{ 142_1 153_0 158_1 174_1 189_1 } » 260	#0,68#
{ 142_1 154_0 158_1 174_1 189_1 } » 260	#0,68#
{ 142_1 155_0 158_1 174_1 189_1 } » 260	#0,68#
{ 142_1 158_1 167_0 174_1 189_1 } » 260	#0,68#
{ 142_1 158_1 168_0 174_1 189_1 } » 260	#0,68#
{ 142_1 158_1 170_0 174_1 189_1 } » 260	#0,68#
{ 74_1 } » 261	#0,55#
{ 100_1 } » 261	#0,57#
{ 114_1 } » 261	#0,57#
{ 132_1 } » 261	#0,58#
{ 147_1 } » 261	#0,59#

{ 202_1 } » 261	#0,54#
{ 204_1 } » 261	#0,55#
{ 75_1 91_1 } » 261	#0,55#
{ 83_1 102_0 } » 261	#0,55#
{ 83_1 103_0 } » 261	#0,54#
{ 83_1 196_0 } » 261	#0,55#
{ 83_1 197_0 } » 261	#0,54#
{ 90_1 102_0 } » 261	#0,56#
{ 90_1 103_0 } » 261	#0,56#
{ 91_1 102_0 } » 261	#0,57#
{ 91_1 196_0 } » 261	#0,56#
{ 91_1 197_0 } » 261	#0,55#
{ 91_1 198_0 } » 261	#0,55#
{ 92_1 103_0 } » 261	#0,56#
{ 92_1 108_1 } » 261	#0,58#
{ 92_1 123_1 } » 261	#0,56#
{ 92_1 196_0 } » 261	#0,57#
{ 92_1 197_0 } » 261	#0,56#
{ 92_1 198_0 } » 261	#0,55#
{ 99_1 104_0 } » 261	#0,55#
{ 99_1 108_1 } » 261	#0,54#
{ 99_1 124_1 } » 261	#0,55#
{ 99_1 154_1 } » 261	#0,54#
{ 99_1 181_0 } » 261	#0,54#
{ 99_1 182_0 } » 261	#0,54#
{ 99_1 199_0 } » 261	#0,54#
{ 106_1 107_1 } » 261	#0,55#
{ 106_1 139_1 } » 261	#0,57#
{ 106_1 154_1 } » 261	#0,54#
{ 107_1 115_1 } » 261	#0,55#
{ 107_1 170_1 } » 261	#0,54#
{ 107_1 181_0 } » 261	#0,54#
{ 107_1 182_0 } » 261	#0,55#
{ 108_1 115_1 } » 261	#0,57#
{ 108_1 119_0 } » 261	#0,55#
{ 108_1 154_1 } » 261	#0,54#
{ 108_1 182_0 } » 261	#0,54#
{ 108_1 199_0 } » 261	#0,57#
{ 115_1 116_1 } » 261	#0,57#
{ 115_1 119_0 } » 261	#0,56#
{ 115_1 138_1 } » 261	#0,55#
{ 115_1 140_1 } » 261	#0,57#
{ 115_1 154_1 } » 261	#0,57#
{ 115_1 171_1 } » 261	#0,56#
{ 115_1 199_0 } » 261	#0,57#
{ 116_1 123_1 } » 261	#0,55#
{ 116_1 196_0 } » 261	#0,55#
{ 116_1 197_0 } » 261	#0,54#
{ 122_1 170_1 } » 261	#0,55#
{ 122_1 182_0 } » 261	#0,54#
{ 122_1 199_0 } » 261	#0,57#
{ 123_1 131_1 } » 261	#0,57#
{ 123_1 134_0 } » 261	#0,57#
{ 123_1 135_0 } » 261	#0,56#
{ 123_1 150_0 } » 261	#0,56#
{ 123_1 166_0 } » 261	#0,56#
{ 123_1 172_1 } » 261	#0,56#
{ 123_1 180_0 } » 261	#0,58#
{ 124_1 138_1 } » 261	#0,54#
{ 124_1 154_1 } » 261	#0,57#
{ 124_1 172_1 } » 261	#0,55#
{ 124_1 182_0 } » 261	#0,54#
{ 124_1 199_0 } » 261	#0,58#
{ 124_1 200_0 } » 261	#0,54#
{ 131_1 154_1 } » 261	#0,54#
{ 131_1 196_0 } » 261	#0,55#

{ 131_1 197_0 } » 261	#0,55#
{ 131_1 198_0 } » 261	#0,55#
{ 137_1 138_1 } » 261	#0,55#
{ 138_1 153_1 } » 261	#0,55#
{ 138_1 181_0 } » 261	#0,54#
{ 138_1 182_0 } » 261	#0,54#
{ 138_1 199_0 } » 261	#0,58#
{ 139_1 150_0 } » 261	#0,54#
{ 139_1 153_1 } » 261	#0,55#
{ 139_1 166_0 } » 261	#0,55#
{ 139_1 172_1 } » 261	#0,55#
{ 139_1 180_0 } » 261	#0,56#
{ 140_1 199_0 } » 261	#0,56#
{ 153_1 154_1 } » 261	#0,59#
{ 153_1 196_0 } » 261	#0,57#
{ 153_1 197_0 } » 261	#0,55#
{ 154_1 169_1 } » 261	#0,54#
{ 154_1 173_0 } » 261	#0,55#
{ 154_1 181_0 } » 261	#0,55#
{ 154_1 182_0 } » 261	#0,55#
{ 154_1 183_0 } » 261	#0,54#
{ 154_1 189_0 } » 261	#0,55#
{ 155_1 181_0 } » 261	#0,56#
{ 156_1 196_0 } » 261	#0,55#
{ 156_1 197_0 } » 261	#0,55#
{ 156_1 198_0 } » 261	#0,55#
{ 170_1 173_0 } » 261	#0,54#
{ 170_1 189_0 } » 261	#0,55#
{ 171_1 172_1 } » 261	#0,55#
{ 171_1 203_1 } » 261	#0,54#
{ 172_1 188_1 } » 261	#0,54#
{ 186_1 189_0 } » 261	#0,55#
{ 186_1 198_0 } » 261	#0,54#
{ 187_1 196_0 } » 261	#0,55#
{ 187_1 197_0 } » 261	#0,55#
{ 187_1 200_0 } » 261	#0,55#
{ 187_1 203_1 } » 261	#0,55#
{ 83_1 86_0 99_1 } » 261	#0,56#
{ 83_1 87_0 99_1 } » 261	#0,56#
{ 90_1 106_1 122_1 } » 261	#0,54#
{ 91_1 104_0 107_1 } » 261	#0,55#
{ 91_1 107_1 155_1 } » 261	#0,54#
{ 99_1 102_0 115_1 } » 261	#0,61#
{ 99_1 102_0 123_1 } » 261	#0,55#
{ 99_1 102_0 139_1 } » 261	#0,55#
{ 99_1 103_0 115_1 } » 261	#0,60#
{ 99_1 103_0 123_1 } » 261	#0,55#
{ 99_1 103_0 139_1 } » 261	#0,55#
{ 99_1 115_1 123_1 } » 261	#0,55#
{ 99_1 115_1 139_1 } » 261	#0,54#
{ 99_1 115_1 196_0 } » 261	#0,57#
{ 99_1 115_1 197_0 } » 261	#0,56#
{ 99_1 115_1 198_0 } » 261	#0,55#
{ 99_1 123_1 139_1 } » 261	#0,54#
{ 99_1 123_1 196_0 } » 261	#0,55#
{ 99_1 123_1 197_0 } » 261	#0,54#
{ 99_1 139_1 155_1 } » 261	#0,54#
{ 106_1 122_1 123_1 } » 261	#0,54#
{ 106_1 122_1 138_1 } » 261	#0,56#
{ 107_1 108_1 123_1 } » 261	#0,57#
{ 107_1 118_0 123_1 } » 261	#0,55#
{ 107_1 119_0 123_1 } » 261	#0,54#
{ 107_1 120_0 123_1 } » 261	#0,54#
{ 107_1 122_1 123_1 } » 261	#0,58#
{ 107_1 122_1 139_1 } » 261	#0,54#
{ 107_1 123_1 124_1 } » 261	#0,56#
{ 107_1 123_1 140_1 } » 261	#0,55#
{ 107_1 123_1 199_0 } » 261	#0,57#
{ 107_1 155_1 196_0 } » 261	#0,54#
{ 108_1 123_1 196_0 } » 261	#0,57#
{ 108_1 123_1 197_0 } » 261	#0,56#
{ 108_1 123_1 198_0 } » 261	#0,56#
{ 108_1 124_1 196_0 } » 261	#0,55#
{ 108_1 124_1 197_0 } » 261	#0,55#
{ 108_1 124_1 198_0 } » 261	#0,55#
{ 108_1 140_1 156_1 } » 261	#0,54#
{ 108_1 155_1 171_1 } » 261	#0,55#
{ 115_1 118_0 131_1 } » 261	#0,55#
{ 115_1 123_1 124_1 } » 261	#0,54#
{ 115_1 123_1 196_0 } » 261	#0,55#
{ 115_1 123_1 197_0 } » 261	#0,55#
{ 115_1 123_1 198_0 } » 261	#0,55#
{ 122_1 123_1 154_1 } » 261	#0,55#
{ 122_1 123_1 196_0 } » 261	#0,57#
{ 122_1 123_1 197_0 } » 261	#0,57#
{ 122_1 123_1 198_0 } » 261	#0,57#
{ 122_1 138_1 154_1 } » 261	#0,57#
{ 122_1 138_1 196_0 } » 261	#0,55#
{ 122_1 139_1 154_1 } » 261	#0,55#
{ 122_1 139_1 171_1 } » 261	#0,54#
{ 122_1 155_1 171_1 } » 261	#0,55#
{ 123_1 124_1 156_1 } » 261	#0,54#
{ 123_1 124_1 171_1 } » 261	#0,55#
{ 123_1 124_1 196_0 } » 261	#0,57#
{ 123_1 124_1 197_0 } » 261	#0,57#
{ 123_1 124_1 198_0 } » 261	#0,58#
{ 123_1 138_1 197_0 } » 261	#0,58#
{ 123_1 139_1 181_0 } » 261	#0,55#
{ 123_1 139_1 182_0 } » 261	#0,56#
{ 123_1 139_1 183_0 } » 261	#0,55#
{ 123_1 139_1 200_0 } » 261	#0,55#
{ 123_1 140_1 171_1 } » 261	#0,55#
{ 123_1 154_1 199_0 } » 261	#0,55#
{ 123_1 155_1 200_0 } » 261	#0,54#
{ 123_1 170_1 186_1 } » 261	#0,54#
{ 124_1 139_1 196_0 } » 261	#0,55#
{ 124_1 139_1 197_0 } » 261	#0,54#
{ 124_1 139_1 198_0 } » 261	#0,55#
{ 124_1 140_1 171_1 } » 261	#0,55#
{ 124_1 140_1 196_0 } » 261	#0,55#
{ 124_1 140_1 197_0 } » 261	#0,55#
{ 124_1 140_1 198_0 } » 261	#0,55#
{ 131_1 139_1 155_1 } » 261	#0,55#
{ 138_1 139_1 197_0 } » 261	#0,58#
{ 138_1 154_1 170_1 } » 261	#0,57#
{ 138_1 154_1 196_0 } » 261	#0,56#
{ 138_1 154_1 197_0 } » 261	#0,55#
{ 138_1 154_1 198_0 } » 261	#0,55#
{ 139_1 140_1 154_1 } » 261	#0,54#
{ 139_1 140_1 196_0 } » 261	#0,55#
{ 139_1 140_1 197_0 } » 261	#0,55#
{ 139_1 140_1 198_0 } » 261	#0,54#
{ 139_1 154_1 199_0 } » 261	#0,56#
{ 139_1 155_1 182_0 } » 261	#0,56#
{ 139_1 155_1 183_0 } » 261	#0,56#
{ 139_1 155_1 186_1 } » 261	#0,55#
{ 139_1 155_1 200_0 } » 261	#0,57#
{ 139_1 170_1 186_1 } » 261	#0,55#
{ 140_1 154_1 155_1 } » 261	#0,54#
{ 140_1 155_1 196_0 } » 261	#0,54#
{ 140_1 156_1 172_1 } » 261	#0,56#
{ 154_1 155_1 199_0 } » 261	#0,55#
{ 154_1 170_1 186_1 } » 261	#0,58#
{ 154_1 170_1 196_0 } » 261	#0,56#
{ 154_1 170_1 197_0 } » 261	#0,55#
{ 154_1 170_1 198_0 } » 261	#0,55#
{ 155_1 156_1 171_1 } » 261	#0,55#
{ 155_1 156_1 172_1 } » 261	#0,54#
{ 155_1 170_1 186_1 } » 261	#0,54#
{ 155_1 171_1 200_0 } » 261	#0,55#
{ 171_1 187_1 198_0 } » 261	#0,54#
{ 171_1 187_1 199_0 } » 261	#0,54#
{ 91_1 103_0 107_1 123_1 } » 261	#0,55#
{ 91_1 107_1 123_1 139_1 } » 261	#0,55#
{ 107_1 123_1 138_1 139_1 } » 261	#0,54#
{ 107_1 123_1 139_1 196_0 } » 261	#0,55#
{ 107_1 123_1 139_1 197_0 } » 261	#0,56#
{ 107_1 123_1 155_1 197_0 } » 261	#0,54#
{ 108_1 123_1 124_1 139_1 } » 261	#0,56#
{ 108_1 123_1 124_1 140_1 } » 261	#0,55#
{ 108_1 123_1 139_1 155_1 } » 261	#0,57#
{ 108_1 124_1 139_1 140_1 } » 261	#0,55#
{ 108_1 124_1 139_1 155_1 } » 261	#0,55#
{ 108_1 124_1 140_1 155_1 } » 261	#0,54#
{ 115_1 123_1 139_1 155_1 } » 261	#0,55#
{ 122_1 123_1 138_1 139_1 } » 261	#0,54#
{ 122_1 123_1 139_1 155_1 } » 261	#0,57#
{ 123_1 138_1 139_1 196_0 } » 261	#0,54#
{ 123_1 138_1 139_1 198_0 } » 261	#0,54#
{ 123_1 139_1 140_1 156_1 } » 261	#0,54#
{ 123_1 139_1 154_1 170_1 } » 261	#0,55#
{ 123_1 139_1 154_1 196_0 } » 261	#0,56#
{ 123_1 139_1 154_1 197_0 } » 261	#0,55#
{ 123_1 139_1 154_1 198_0 } » 261	#0,56#
{ 123_1 139_1 155_1 156_1 } » 261	#0,56#
{ 123_1 139_1 155_1 170_1 } » 261	#0,55#
{ 123_1 139_1 155_1 196_0 } » 261	#0,61#
{ 123_1 139_1 155_1 199_0 } » 261	#0,59#
{ 123_1 139_1 171_1 196_0 } » 261	#0,54#
{ 123_1 154_1 155_1 198_0 } » 261	#0,54#
{ 123_1 155_1 171_1 196_0 } » 261	#0,54#
{ 123_1 155_1 171_1 199_0 } » 261	#0,54#
{ 124_1 139_1 155_1 171_1 } » 261	#0,55#
{ 124_1 140_1 155_1 156_1 } » 261	#0,54#
{ 138_1 139_1 155_1 171_1 } » 261	#0,55#
{ 139_1 140_1 155_1 156_1 } » 261	#0,56#
{ 139_1 140_1 155_1 171_1 } » 261	#0,55#
{ 139_1 154_1 155_1 170_1 } » 261	#0,55#
{ 139_1 154_1 155_1 196_0 } » 261	#0,57#
{ 139_1 154_1 155_1 197_0 } » 261	#0,56#
{ 139_1 154_1 155_1 198_0 } » 261	#0,56#
{ 139_1 155_1 170_1 171_1 } » 261	#0,54#
{ 139_1 155_1 171_1 196_0 } » 261	#0,57#
{ 139_1 155_1 171_1 199_0 } » 261	#0,56#
{ 107_1 123_1 139_1 154_1 155_1 } » 261	#0,54#
{ 107_1 123_1 139_1 155_1 171_1 } » 261	#0,57#
{ 107_1 123_1 139_1 155_1 198_0 } » 261	#0,54#
{ 123_1 124_1 139_1 140_1 155_1 } » 261	#0,55#
{ 123_1 138_1 139_1 154_1 155_1 } » 261	#0,55#
{ 123_1 139_1 154_1 155_1 171_1 } » 261	#0,55#
{ 123_1 139_1 155_1 171_1 187_1 } » 261	#0,55#

{ 123_1 139_1 155_1 171_1 197_0 } » 261	#0,54#
{ 123_1 139_1 155_1 171_1 198_0 } » 261	#0,55#
{ 84_1 } » 262	#0,66#
{ 116_1 } » 262	#0,66#
{ 141_1 } » 262	#0,65#
{ 41_1 56_1 } » 262	#0,64#
{ 41_1 59_0 } » 262	#0,69#
{ 41_1 90_0 } » 262	#0,67#
{ 41_1 91_0 } » 262	#0,69#
{ 41_1 92_0 } » 262	#0,69#
{ 41_1 165_0 } » 262	#0,68#
{ 41_1 166_0 } » 262	#0,69#
{ 41_1 167_0 } » 262	#0,68#
{ 41_1 168_0 } » 262	#0,69#
{ 41_1 169_0 } » 262	#0,68#
{ 41_1 170_0 } » 262	#0,66#
{ 41_1 180_0 } » 262	#0,66#
{ 41_1 181_0 } » 262	#0,69#
{ 41_1 185_0 } » 262	#0,68#
{ 42_1 89_0 } » 262	#0,65#
{ 42_1 90_0 } » 262	#0,69#
{ 42_1 91_0 } » 262	#0,71#
{ 42_1 92_0 } » 262	#0,72#
{ 42_1 152_0 } » 262	#0,65#
{ 42_1 164_0 } » 262	#0,67#
{ 42_1 165_0 } » 262	#0,71#
{ 42_1 166_0 } » 262	#0,71#
{ 42_1 170_0 } » 262	#0,67#
{ 42_1 180_0 } » 262	#0,69#
{ 42_1 181_0 } » 262	#0,71#
{ 43_1 75_0 } » 262	#0,65#
{ 43_1 76_0 } » 262	#0,65#
{ 43_1 77_0 } » 262	#0,65#
{ 55_1 60_0 } » 262	#0,68#
{ 55_1 89_0 } » 262	#0,64#
{ 55_1 90_0 } » 262	#0,66#
{ 55_1 91_0 } » 262	#0,67#
{ 55_1 92_0 } » 262	#0,68#
{ 55_1 164_0 } » 262	#0,64#
{ 55_1 165_0 } » 262	#0,66#
{ 55_1 166_0 } » 262	#0,66#
{ 55_1 167_0 } » 262	#0,65#
{ 55_1 168_0 } » 262	#0,66#
{ 55_1 169_0 } » 262	#0,66#
{ 55_1 173_1 } » 262	#0,64#
{ 55_1 180_0 } » 262	#0,68#
{ 55_1 181_0 } » 262	#0,69#
{ 55_1 182_0 } » 262	#0,70#
{ 55_1 185_0 } » 262	#0,67#
{ 55_1 204_1 } » 262	#0,64#
{ 56_1 89_0 } » 262	#0,68#
{ 56_1 90_0 } » 262	#0,71#
{ 56_1 91_0 } » 262	#0,72#
{ 56_1 164_0 } » 262	#0,67#
{ 56_1 165_0 } » 262	#0,70#
{ 56_1 166_0 } » 262	#0,70#
{ 56_1 167_0 } » 262	#0,69#
{ 56_1 168_0 } » 262	#0,70#
{ 56_1 169_0 } » 262	#0,69#
{ 56_1 170_0 } » 262	#0,66#
{ 56_1 180_0 } » 262	#0,69#
{ 56_1 181_0 } » 262	#0,72#
{ 56_1 186_0 } » 262	#0,65#
{ 56_1 198_0 } » 262	#0,66#
{ 56_1 199_0 } » 262	#0,65#
{ 56_1 218_1 } » 262	#0,65#
{ 57_1 75_0 } » 262	#0,65#
{ 57_1 76_0 } » 262	#0,65#
{ 57_1 77_0 } » 262	#0,65#
{ 69_1 74_0 } » 262	#0,65#
{ 69_1 75_0 } » 262	#0,66#
{ 69_1 76_0 } » 262	#0,66#
{ 69_1 77_0 } » 262	#0,67#
{ 69_1 93_0 } » 262	#0,64#
{ 69_1 183_0 } » 262	#0,64#
{ 69_1 184_0 } » 262	#0,65#
{ 70_1 89_0 } » 262	#0,66#
{ 70_1 90_0 } » 262	#0,67#
{ 70_1 91_0 } » 262	#0,68#
{ 70_1 92_0 } » 262	#0,69#
{ 70_1 165_0 } » 262	#0,67#
{ 70_1 166_0 } » 262	#0,67#
{ 70_1 167_0 } » 262	#0,67#
{ 70_1 168_0 } » 262	#0,68#
{ 70_1 169_0 } » 262	#0,67#
{ 70_1 170_0 } » 262	#0,65#
{ 70_1 180_0 } » 262	#0,67#
{ 70_1 181_0 } » 262	#0,69#
{ 70_1 182_0 } » 262	#0,70#
{ 70_1 185_0 } » 262	#0,68#
{ 71_1 75_0 } » 262	#0,65#
{ 71_1 76_0 } » 262	#0,65#
{ 71_1 77_0 } » 262	#0,66#
{ 85_1 90_0 } » 262	#0,65#
{ 85_1 91_0 } » 262	#0,65#
{ 85_1 92_0 } » 262	#0,65#
{ 85_1 93_0 } » 262	#0,66#
{ 85_1 181_0 } » 262	#0,66#
{ 85_1 182_0 } » 262	#0,67#
{ 85_1 183_0 } » 262	#0,68#
{ 85_1 184_0 } » 262	#0,68#
{ 85_1 185_0 } » 262	#0,66#
{ 85_1 189_1 } » 262	#0,64#
{ 100_1 181_0 } » 262	#0,64#
{ 100_1 182_0 } » 262	#0,64#
{ 100_1 183_0 } » 262	#0,66#
{ 100_1 184_0 } » 262	#0,66#
{ 157_1 164_0 } » 262	#0,64#
{ 157_1 204_1 } » 262	#0,65#
{ 158_1 169_0 } » 262	#0,64#
{ 158_1 174_1 } » 262	#0,66#
{ 173_1 197_0 } » 262	#0,64#
{ 173_1 198_0 } » 262	#0,65#
{ 173_1 199_0 } » 262	#0,64#
{ 174_1 181_0 } » 262	#0,65#
{ 174_1 204_1 } » 262	#0,65#
{ 188_1 203_1 } » 262	#0,67#
{ 189_1 190_1 } » 262	#0,64#
{ 189_1 197_0 } » 262	#0,67#
{ 189_1 198_0 } » 262	#0,69#
{ 189_1 199_0 } » 262	#0,68#
{ 189_1 200_0 } » 262	#0,64#
{ 189_1 217_1 } » 262	#0,66#
{ 204_1 217_1 } » 262	#0,65#
{ 216_1 217_1 } » 262	#0,65#
{ 217_1 218_1 } » 262	#0,66#
{ 218_1 219_1 } » 262	#0,64#
{ 41_1 42_1 60_0 } » 262	#0,68#
{ 41_1 42_1 61_0 } » 262	#0,68#
{ 41_1 42_1 74_0 } » 262	#0,66#
{ 41_1 42_1 75_0 } » 262	#0,69#
{ 41_1 42_1 76_0 } » 262	#0,69#
{ 41_1 42_1 77_0 } » 262	#0,69#
{ 41_1 42_1 93_0 } » 262	#0,65#
{ 41_1 42_1 182_0 } » 262	#0,65#
{ 41_1 42_1 183_0 } » 262	#0,66#
{ 41_1 42_1 184_0 } » 262	#0,66#
{ 41_1 55_1 77_0 } » 262	#0,64#
{ 41_1 60_0 189_1 } » 262	#0,66#
{ 41_1 61_0 189_1 } » 262	#0,66#
{ 41_1 74_0 189_1 } » 262	#0,66#
{ 41_1 75_0 189_1 } » 262	#0,67#
{ 41_1 75_0 204_1 } » 262	#0,64#
{ 41_1 76_0 189_1 } » 262	#0,67#
{ 41_1 76_0 204_1 } » 262	#0,64#
{ 41_1 77_0 173_1 } » 262	#0,64#
{ 41_1 77_0 189_1 } » 262	#0,68#
{ 41_1 77_0 204_1 } » 262	#0,65#
{ 41_1 183_0 189_1 } » 262	#0,65#
{ 41_1 184_0 189_1 } » 262	#0,65#
{ 42_1 56_1 60_0 } » 262	#0,64#
{ 42_1 56_1 61_0 } » 262	#0,66#
{ 42_1 56_1 75_0 } » 262	#0,66#
{ 42_1 56_1 76_0 } » 262	#0,66#
{ 42_1 56_1 77_0 } » 262	#0,66#
{ 42_1 60_0 189_1 } » 262	#0,66#
{ 42_1 61_0 173_1 } » 262	#0,64#
{ 42_1 61_0 189_1 } » 262	#0,68#
{ 42_1 61_0 204_1 } » 262	#0,66#
{ 42_1 74_0 189_1 } » 262	#0,66#
{ 42_1 75_0 173_1 } » 262	#0,65#
{ 42_1 75_0 189_1 } » 262	#0,69#
{ 42_1 75_0 204_1 } » 262	#0,67#
{ 42_1 76_0 173_1 } » 262	#0,65#
{ 42_1 76_0 189_1 } » 262	#0,69#
{ 42_1 76_0 204_1 } » 262	#0,67#
{ 42_1 77_0 173_1 } » 262	#0,66#
{ 42_1 77_0 189_1 } » 262	#0,70#
{ 42_1 77_0 204_1 } » 262	#0,68#
{ 42_1 93_0 189_1 } » 262	#0,66#
{ 42_1 93_0 204_1 } » 262	#0,65#
{ 42_1 167_0 189_1 } » 262	#0,64#
{ 42_1 168_0 189_1 } » 262	#0,65#
{ 42_1 169_0 189_1 } » 262	#0,65#
{ 42_1 173_1 189_1 } » 262	#0,64#
{ 42_1 182_0 189_1 } » 262	#0,66#
{ 42_1 183_0 189_1 } » 262	#0,67#
{ 42_1 183_0 204_1 } » 262	#0,64#
{ 42_1 184_0 189_1 } » 262	#0,68#
{ 42_1 184_0 204_1 } » 262	#0,65#
{ 42_1 185_0 189_1 } » 262	#0,66#
{ 42_1 185_0 204_1 } » 262	#0,65#
{ 42_1 189_1 204_1 } » 262	#0,64#
{ 55_1 56_1 61_0 } » 262	#0,64#
{ 55_1 56_1 74_0 } » 262	#0,66#
{ 55_1 56_1 75_0 } » 262	#0,66#
{ 55_1 56_1 76_0 } » 262	#0,66#
{ 55_1 56_1 77_0 } » 262	#0,67#
{ 55_1 56_1 93_0 } » 262	#0,64#
{ 55_1 56_1 183_0 } » 262	#0,64#
{ 55_1 56_1 184_0 } » 262	#0,65#
{ 55_1 74_0 189_1 } » 262	#0,64#
{ 55_1 75_0 189_1 } » 262	#0,64#
{ 55_1 76_0 189_1 } » 262	#0,64#
{ 55_1 77_0 189_1 } » 262	#0,65#
{ 56_1 61_0 189_1 } » 262	#0,66#

{ 56_1 61_0 204_1 } » 262	#0,66#	{ 173_1 181_0 189_1 204_1 } » 262	#0,64#	{ 70_1 76_0 189_1 } » 263	#0,69#
{ 56_1 70_1 75_0 } » 262	#0,64#	{ 173_1 182_0 189_1 204_1 } » 262	#0,67#	{ 70_1 77_0 85_1 } » 263	#0,69#
{ 56_1 70_1 76_0 } » 262	#0,64#	{ 173_1 183_0 188_1 189_1 } » 262	#0,64#	{ 70_1 77_0 116_1 } » 263	#0,69#
{ 56_1 70_1 77_0 } » 262	#0,65#	{ 173_1 183_0 189_1 204_1 } » 262	#0,69#	{ 70_1 77_0 147_1 } » 263	#0,69#
{ 56_1 74_0 189_1 } » 262	#0,66#	{ 173_1 184_0 188_1 189_1 } » 262	#0,64#	{ 70_1 77_0 189_1 } » 263	#0,69#
{ 56_1 74_0 204_1 } » 262	#0,66#	{ 173_1 184_0 189_1 204_1 } » 262	#0,70#	{ 70_1 85_1 89_0 } » 263	#0,70#
{ 56_1 75_0 173_1 } » 262	#0,64#	{ 173_1 185_0 189_1 204_1 } » 262	#0,70#	{ 70_1 85_1 90_0 } » 263	#0,69#
{ 56_1 75_0 189_1 } » 262	#0,68#	{ 173_1 186_0 189_1 204_1 } » 262	#0,68#	{ 70_1 85_1 91_0 } » 263	#0,69#
{ 56_1 75_0 204_1 } » 262	#0,68#	{ 157_1 } » 263	#0,72#	{ 70_1 85_1 92_0 } » 263	#0,69#
{ 56_1 76_0 173_1 } » 262	#0,64#	{ 197_1 } » 263	#0,74#	{ 70_1 85_1 93_0 } » 263	#0,69#
{ 56_1 76_0 189_1 } » 262	#0,69#	{ 198_1 } » 263	#0,71#	{ 70_1 85_1 100_1 } » 263	#0,69#
{ 56_1 76_0 204_1 } » 262	#0,69#	{ 199_1 } » 263	#0,73#	{ 70_1 89_0 116_1 } » 263	#0,69#
{ 56_1 77_0 173_1 } » 262	#0,65#	{ 215_1 } » 263	#0,72#	{ 70_1 89_0 147_1 } » 263	#0,69#
{ 56_1 77_0 189_1 } » 262	#0,69#	{ 216_1 } » 263	#0,72#	{ 70_1 89_0 189_1 } » 263	#0,69#
{ 56_1 77_0 204_1 } » 262	#0,69#	{ 217_1 } » 263	#0,72#	{ 70_1 90_0 116_1 } » 263	#0,69#
{ 56_1 92_0 189_1 } » 262	#0,65#	{ 218_1 } » 263	#0,72#	{ 70_1 90_0 147_1 } » 263	#0,69#
{ 56_1 92_0 204_1 } » 262	#0,65#	{ 70_1 104_0 } » 263	#0,71#	{ 70_1 90_0 189_1 } » 263	#0,69#
{ 56_1 93_0 189_1 } » 262	#0,66#	{ 70_1 105_0 } » 263	#0,72#	{ 70_1 91_0 116_1 } » 263	#0,69#
{ 56_1 93_0 204_1 } » 262	#0,66#	{ 70_1 106_0 } » 263	#0,70#	{ 70_1 91_0 147_1 } » 263	#0,69#
{ 56_1 182_0 189_1 } » 262	#0,64#	{ 70_1 107_0 } » 263	#0,70#	{ 70_1 91_0 189_1 } » 263	#0,69#
{ 56_1 182_0 204_1 } » 262	#0,64#	{ 70_1 108_0 } » 263	#0,69#	{ 70_1 92_0 189_1 } » 263	#0,69#
{ 56_1 183_0 189_1 } » 262	#0,66#	{ 70_1 131_1 } » 263	#0,69#	{ 70_1 93_0 189_1 } » 263	#0,69#
{ 56_1 183_0 204_1 } » 262	#0,66#	{ 85_1 106_0 } » 263	#0,70#	{ 85_1 89_0 100_1 } » 263	#0,70#
{ 56_1 184_0 189_1 } » 262	#0,66#	{ 85_1 107_0 } » 263	#0,70#	{ 85_1 89_0 116_1 } » 263	#0,69#
{ 56_1 184_0 204_1 } » 262	#0,66#	{ 85_1 108_0 } » 263	#0,69#	{ 85_1 89_0 147_1 } » 263	#0,70#
{ 56_1 185_0 189_1 } » 262	#0,65#	{ 85_1 119_0 } » 263	#0,69#	{ 85_1 90_0 100_1 } » 263	#0,70#
{ 56_1 185_0 204_1 } » 262	#0,66#	{ 85_1 189_1 } » 263	#0,70#	{ 85_1 90_0 116_1 } » 263	#0,70#
{ 56_1 189_1 204_1 } » 262	#0,65#	{ 86_1 90_0 } » 263	#0,71#	{ 85_1 90_0 147_1 } » 263	#0,70#
{ 70_1 74_0 189_1 } » 262	#0,64#	{ 86_1 91_0 } » 263	#0,71#	{ 85_1 90_0 163_1 } » 263	#0,69#
{ 70_1 75_0 189_1 } » 262	#0,67#	{ 86_1 92_0 } » 263	#0,71#	{ 85_1 91_0 100_1 } » 263	#0,70#
{ 70_1 75_0 204_1 } » 262	#0,66#	{ 86_1 93_0 } » 263	#0,71#	{ 85_1 91_0 116_1 } » 263	#0,69#
{ 70_1 76_0 173_1 } » 262	#0,64#	{ 86_1 105_0 } » 263	#0,70#	{ 85_1 91_0 147_1 } » 263	#0,70#
{ 70_1 76_0 189_1 } » 262	#0,68#	{ 86_1 106_0 } » 263	#0,70#	{ 85_1 92_0 100_1 } » 263	#0,70#
{ 70_1 76_0 204_1 } » 262	#0,66#	{ 100_1 132_1 } » 263	#0,69#	{ 85_1 92_0 116_1 } » 263	#0,69#
{ 70_1 77_0 173_1 } » 262	#0,64#	{ 101_1 120_0 } » 263	#0,70#	{ 85_1 92_0 147_1 } » 263	#0,70#
{ 70_1 77_0 189_1 } » 262	#0,68#	{ 116_1 121_0 } » 263	#0,70#	{ 85_1 93_0 100_1 } » 263	#0,70#
{ 70_1 77_0 204_1 } » 262	#0,66#	{ 116_1 204_1 } » 263	#0,69#	{ 85_1 93_0 116_1 } » 263	#0,69#
{ 70_1 93_0 189_1 } » 262	#0,64#	{ 132_1 164_1 } » 263	#0,70#	{ 85_1 93_0 147_1 } » 263	#0,70#
{ 70_1 183_0 189_1 } » 262	#0,65#	{ 132_1 173_1 } » 263	#0,70#	{ 85_1 100_1 104_0 } » 263	#0,70#
{ 70_1 184_0 189_1 } » 262	#0,64#	{ 132_1 180_1 } » 263	#0,69#	{ 85_1 100_1 105_0 } » 263	#0,69#
{ 157_1 165_0 173_1 } » 262	#0,66#	{ 147_1 164_1 } » 263	#0,71#	{ 85_1 100_1 116_1 } » 263	#0,70#
{ 157_1 166_0 173_1 } » 262	#0,67#	{ 148_1 164_1 } » 263	#0,72#	{ 85_1 100_1 163_1 } » 263	#0,69#
{ 157_1 167_0 173_1 } » 262	#0,67#	{ 148_1 174_1 } » 263	#0,71#	{ 85_1 104_0 116_1 } » 263	#0,69#
{ 157_1 168_0 173_1 } » 262	#0,68#	{ 148_1 190_1 } » 263	#0,70#	{ 85_1 104_0 147_1 } » 263	#0,70#
{ 157_1 168_0 189_1 } » 262	#0,65#	{ 158_1 190_1 } » 263	#0,70#	{ 85_1 116_1 147_1 } » 263	#0,70#
{ 157_1 173_1 181_0 } » 262	#0,65#	{ 163_1 164_1 } » 263	#0,72#	{ 85_1 147_1 163_1 } » 263	#0,70#
{ 173_1 174_1 189_1 } » 262	#0,64#	{ 164_1 174_1 } » 263	#0,69#	{ 100_1 104_0 174_1 } » 263	#0,69#
{ 173_1 180_0 189_1 } » 262	#0,67#	{ 164_1 180_1 } » 263	#0,71#	{ 100_1 105_0 189_1 } » 263	#0,71#
{ 173_1 182_0 188_1 } » 262	#0,66#	{ 164_1 189_1 } » 263	#0,71#	{ 100_1 106_0 116_1 } » 263	#0,70#
{ 173_1 184_0 203_1 } » 262	#0,64#	{ 174_1 175_1 } » 263	#0,69#	{ 100_1 106_0 189_1 } » 263	#0,69#
{ 173_1 185_0 188_1 } » 262	#0,64#	{ 174_1 180_1 } » 263	#0,72#	{ 100_1 107_0 116_1 } » 263	#0,70#
{ 173_1 189_1 203_1 } » 262	#0,64#	{ 180_1 190_1 } » 263	#0,70#	{ 100_1 107_0 189_1 } » 263	#0,69#
{ 174_1 182_0 189_1 } » 262	#0,64#	{ 190_1 196_1 } » 263	#0,69#	{ 100_1 108_0 116_1 } » 263	#0,70#
{ 174_1 183_0 189_1 } » 262	#0,66#	{ 70_1 74_0 85_1 } » 263	#0,70#	{ 100_1 108_0 189_1 } » 263	#0,69#
{ 174_1 184_0 189_1 } » 262	#0,67#	{ 70_1 74_0 116_1 } » 263	#0,70#	{ 100_1 109_0 147_1 } » 263	#0,70#
{ 174_1 185_0 189_1 } » 262	#0,66#	{ 70_1 74_0 147_1 } » 263	#0,70#	{ 100_1 116_1 174_1 } » 263	#0,69#
{ 188_1 189_1 204_1 } » 262	#0,66#	{ 70_1 74_0 189_1 } » 263	#0,70#	{ 100_1 119_0 189_1 } » 263	#0,70#
{ 189_1 203_1 204_1 } » 262	#0,66#	{ 70_1 75_0 85_1 } » 263	#0,70#	{ 100_1 120_0 147_1 } » 263	#0,70#
{ 189_1 204_1 205_1 } » 262	#0,68#	{ 70_1 75_0 100_1 } » 263	#0,69#	{ 100_1 147_1 190_1 } » 263	#0,70#
{ 189_1 204_1 218_1 } » 262	#0,65#	{ 70_1 75_0 116_1 } » 263	#0,70#	{ 101_1 104_0 116_1 } » 263	#0,70#
{ 157_1 169_0 173_1 189_1 } » 262	#0,65#	{ 70_1 75_0 147_1 } » 263	#0,70#	{ 101_1 105_0 189_1 } » 263	#0,70#
{ 157_1 170_0 173_1 189_1 } » 262	#0,66#	{ 70_1 75_0 163_1 } » 263	#0,69#	{ 101_1 106_0 189_1 } » 263	#0,69#
{ 157_1 173_1 182_0 189_1 } » 262	#0,64#	{ 70_1 75_0 189_1 } » 263	#0,70#	{ 101_1 107_0 163_1 } » 263	#0,69#
{ 157_1 173_1 183_0 189_1 } » 262	#0,66#	{ 70_1 76_0 85_1 } » 263	#0,70#	{ 101_1 108_0 116_1 } » 263	#0,70#
{ 157_1 173_1 184_0 189_1 } » 262	#0,66#	{ 70_1 76_0 116_1 } » 263	#0,69#	{ 101_1 109_0 116_1 } » 263	#0,69#
{ 157_1 173_1 185_0 189_1 } » 262	#0,65#	{ 70_1 76_0 147_1 } » 263	#0,69#	{ 101_1 116_1 132_1 } » 263	#0,70#

{ 101_1 116_1 174_1 } » 263	#0,69#
{ 101_1 116_1 190_1 } » 263	#0,70#
{ 101_1 163_1 189_1 } » 263	#0,69#
{ 116_1 119_0 132_1 } » 263	#0,69#
{ 116_1 119_0 190_1 } » 263	#0,69#
{ 116_1 120_0 132_1 } » 263	#0,70#
{ 116_1 120_0 174_1 } » 263	#0,70#
{ 116_1 120_0 190_1 } » 263	#0,69#
{ 116_1 132_1 148_1 } » 263	#0,69#
{ 116_1 147_1 173_1 } » 263	#0,69#
{ 116_1 173_1 189_1 } » 263	#0,69#
{ 116_1 189_1 205_1 } » 263	#0,69#
{ 131_1 134_0 147_1 } » 263	#0,69#
{ 131_1 146_1 147_1 } » 263	#0,69#
{ 131_1 147_1 173_1 } » 263	#0,69#
{ 132_1 147_1 148_1 } » 263	#0,70#
{ 132_1 148_1 163_1 } » 263	#0,70#
{ 132_1 148_1 189_1 } » 263	#0,70#
{ 132_1 174_1 189_1 } » 263	#0,71#
{ 132_1 174_1 190_1 } » 263	#0,71#
{ 132_1 189_1 205_1 } » 263	#0,69#
{ 132_1 190_1 205_1 } » 263	#0,69#
{ 146_1 147_1 163_1 } » 263	#0,69#
{ 147_1 148_1 163_1 } » 263	#0,70#
{ 147_1 162_1 163_1 } » 263	#0,69#
{ 147_1 163_1 180_1 } » 263	#0,72#
{ 147_1 163_1 204_1 } » 263	#0,71#
{ 147_1 189_1 204_1 } » 263	#0,70#
{ 158_1 163_1 179_1 } » 263	#0,70#
{ 158_1 174_1 179_1 } » 263	#0,70#
{ 158_1 174_1 189_1 } » 263	#0,71#
{ 163_1 173_1 179_1 } » 263	#0,71#
{ 163_1 174_1 196_1 } » 263	#0,70#
{ 163_1 174_1 204_1 } » 263	#0,69#
{ 163_1 179_1 180_1 } » 263	#0,70#
{ 163_1 179_1 195_1 } » 263	#0,69#
{ 163_1 179_1 204_1 } » 263	#0,69#
{ 163_1 180_1 189_1 } » 263	#0,70#
{ 163_1 189_1 196_1 } » 263	#0,69#
{ 163_1 189_1 204_1 } » 263	#0,71#
{ 163_1 190_1 204_1 } » 263	#0,69#
{ 173_1 174_1 179_1 } » 263	#0,69#
{ 173_1 179_1 189_1 } » 263	#0,69#
{ 174_1 189_1 204_1 } » 263	#0,71#
{ 174_1 190_1 204_1 } » 263	#0,70#
{ 179_1 190_1 205_1 } » 263	#0,70#
{ 189_1 190_1 204_1 } » 263	#0,71#
{ 189_1 204_1 205_1 } » 263	#0,69#
{ 85_1 100_1 131_1 147_1 } » 263	#0,69#
{ 100_1 103_0 131_1 147_1 } » 263	#0,69#
{ 100_1 104_0 116_1 163_1 } » 263	#0,69#
{ 100_1 104_0 147_1 189_1 } » 263	#0,69#
{ 100_1 105_0 116_1 131_1 } » 263	#0,69#
{ 100_1 105_0 116_1 147_1 } » 263	#0,70#
{ 100_1 106_0 131_1 147_1 } » 263	#0,70#
{ 100_1 106_0 147_1 163_1 } » 263	#0,70#
{ 100_1 107_0 131_1 147_1 } » 263	#0,70#
{ 100_1 107_0 147_1 163_1 } » 263	#0,69#
{ 100_1 108_0 131_1 147_1 } » 263	#0,70#
{ 100_1 108_0 147_1 163_1 } » 263	#0,69#
{ 100_1 115_1 131_1 147_1 } » 263	#0,69#
{ 100_1 116_1 147_1 189_1 } » 263	#0,70#
{ 100_1 131_1 147_1 174_1 } » 263	#0,70#
{ 100_1 147_1 163_1 174_1 } » 263	#0,70#
{ 100_1 147_1 163_1 179_1 } » 263	#0,70#
{ 101_1 105_0 116_1 147_1 } » 263	#0,70#

{ 101_1 105_0 116_1 163_1 } » 263	#0,69#
{ 101_1 106_0 116_1 147_1 } » 263	#0,70#
{ 101_1 106_0 116_1 163_1 } » 263	#0,69#
{ 101_1 107_0 116_1 147_1 } » 263	#0,69#
{ 101_1 116_1 131_1 147_1 } » 263	#0,69#
{ 101_1 116_1 147_1 163_1 } » 263	#0,72#
{ 101_1 116_1 147_1 189_1 } » 263	#0,69#
{ 115_1 131_1 147_1 163_1 } » 263	#0,69#
{ 116_1 119_0 147_1 174_1 } » 263	#0,69#
{ 116_1 119_0 147_1 189_1 } » 263	#0,70#
{ 116_1 120_0 147_1 189_1 } » 263	#0,70#
{ 116_1 131_1 189_1 190_1 } » 263	#0,69#
{ 116_1 132_1 147_1 174_1 } » 263	#0,69#
{ 116_1 132_1 147_1 189_1 } » 263	#0,71#
{ 116_1 132_1 147_1 190_1 } » 263	#0,69#
{ 116_1 132_1 163_1 179_1 } » 263	#0,69#
{ 116_1 132_1 163_1 189_1 } » 263	#0,70#
{ 116_1 147_1 163_1 205_1 } » 263	#0,69#
{ 116_1 174_1 189_1 190_1 } » 263	#0,70#
{ 131_1 147_1 163_1 205_1 } » 263	#0,69#
{ 132_1 147_1 163_1 174_1 } » 263	#0,71#
{ 132_1 147_1 163_1 179_1 } » 263	#0,71#
{ 132_1 147_1 163_1 189_1 } » 263	#0,72#
{ 132_1 147_1 163_1 190_1 } » 263	#0,71#
{ 132_1 147_1 189_1 190_1 } » 263	#0,69#
{ 132_1 163_1 189_1 190_1 } » 263	#0,70#
{ 147_1 158_1 163_1 174_1 } » 263	#0,70#
{ 147_1 163_1 173_1 174_1 } » 263	#0,70#
{ 147_1 163_1 173_1 189_1 } » 263	#0,70#
{ 147_1 163_1 179_1 196_1 } » 263	#0,70#
{ 147_1 163_1 179_1 205_1 } » 263	#0,69#
{ 147_1 163_1 189_1 205_1 } » 263	#0,71#
{ 147_1 163_1 190_1 205_1 } » 263	#0,72#
{ 147_1 173_1 174_1 189_1 } » 263	#0,70#
{ 147_1 174_1 190_1 205_1 } » 263	#0,69#
{ 147_1 189_1 190_1 205_1 } » 263	#0,70#
{ 163_1 173_1 174_1 189_1 } » 263	#0,71#
{ 163_1 174_1 190_1 205_1 } » 263	#0,70#
{ 163_1 189_1 190_1 205_1 } » 263	#0,71#
{ 173_1 174_1 189_1 190_1 } » 263	#0,70#
{ 174_1 189_1 190_1 205_1 } » 263	#0,70#
{ 100_1 104_0 116_1 131_1 147_1 } » 263	#0,70#
{ 100_1 104_0 131_1 147_1 163_1 } » 263	#0,70#
{ 100_1 105_0 131_1 147_1 163_1 } » 263	#0,70#
{ 100_1 116_1 119_0 131_1 147_1 } » 263	#0,69#
{ 100_1 116_1 131_1 147_1 163_1 } » 263	#0,71#
{ 100_1 119_0 131_1 147_1 163_1 } » 263	#0,70#
{ 100_1 131_1 147_1 163_1 189_1 } » 263	#0,69#
{ 116_1 119_0 131_1 147_1 163_1 } » 263	#0,72#
{ 116_1 120_0 131_1 147_1 163_1 } » 263	#0,69#
{ 116_1 131_1 132_1 147_1 163_1 } » 263	#0,70#
{ 116_1 131_1 147_1 163_1 174_1 } » 263	#0,71#
{ 116_1 131_1 147_1 163_1 179_1 } » 263	#0,72#
{ 116_1 131_1 147_1 163_1 189_1 } »	#0,71#

263	
{ 116_1 131_1 147_1 163_1 190_1 } » 263	#0,70#
{ 116_1 147_1 163_1 174_1 179_1 } » 263	#0,70#
{ 116_1 147_1 163_1 174_1 189_1 } » 263	#0,70#
{ 116_1 147_1 163_1 174_1 190_1 } » 263	#0,70#
{ 116_1 147_1 163_1 179_1 189_1 } » 263	#0,70#
{ 116_1 147_1 163_1 179_1 190_1 } » 263	#0,69#
{ 116_1 147_1 163_1 189_1 190_1 } » 263	#0,71#
{ 131_1 147_1 163_1 174_1 179_1 } » 263	#0,70#
{ 131_1 147_1 163_1 174_1 189_1 } » 263	#0,71#
{ 131_1 147_1 163_1 174_1 190_1 } » 263	#0,70#
{ 131_1 147_1 163_1 179_1 189_1 } » 263	#0,70#
{ 131_1 147_1 163_1 189_1 190_1 } » 263	#0,71#
{ 131_1 147_1 174_1 189_1 190_1 } » 263	#0,69#
{ 147_1 163_1 174_1 179_1 189_1 } » 263	#0,72#
{ 147_1 163_1 174_1 179_1 190_1 } » 263	#0,70#
{ 147_1 163_1 174_1 189_1 190_1 } » 263	#0,73#
{ 147_1 163_1 179_1 189_1 190_1 } » 263	#0,71#
{ 163_1 174_1 179_1 189_1 190_1 } » 263	#0,70#
{ 45_1 103_0 } » 264	#0,81#
{ 45_1 104_0 } » 264	#0,81#
{ 45_1 118_0 } » 264	#0,82#
{ 45_1 119_0 } » 264	#0,82#
{ 45_1 133_0 } » 264	#0,81#
{ 45_1 134_0 } » 264	#0,82#
{ 45_1 135_0 } » 264	#0,81#
{ 45_1 148_0 } » 264	#0,82#
{ 45_1 149_0 } » 264	#0,82#
{ 45_1 150_0 } » 264	#0,82#
{ 45_1 164_0 } » 264	#0,82#
{ 45_1 165_0 } » 264	#0,82#
{ 45_1 166_0 } » 264	#0,81#
{ 45_1 180_0 } » 264	#0,82#
{ 45_1 181_0 } » 264	#0,82#
{ 45_1 196_0 } » 264	#0,82#
{ 45_1 197_0 } » 264	#0,81#
{ 61_1 87_0 } » 264	#0,84#
{ 61_1 101_0 } » 264	#0,84#
{ 61_1 152_0 } » 264	#0,83#
{ 61_1 183_0 } » 264	#0,82#
{ 62_1 89_0 } » 264	#0,81#
{ 62_1 90_0 } » 264	#0,81#
{ 62_1 103_0 } » 264	#0,81#
{ 62_1 104_0 } » 264	#0,82#
{ 62_1 105_0 } » 264	#0,82#
{ 62_1 117_0 } » 264	#0,81#
{ 62_1 118_0 } » 264	#0,82#
{ 62_1 119_0 } » 264	#0,82#
{ 62_1 120_0 } » 264	#0,82#

{ 62_1 133_0 } » 264	#0,82#	{ 61_1 109_1 151_0 } » 264	#0,81#	{ 93_1 124_1 180_0 } » 264	#0,82#
{ 62_1 134_0 } » 264	#0,82#	{ 61_1 109_1 164_0 } » 264	#0,81#	{ 93_1 124_1 181_0 } » 264	#0,82#
{ 62_1 135_0 } » 264	#0,82#	{ 61_1 109_1 165_0 } » 264	#0,81#	{ 93_1 124_1 196_0 } » 264	#0,82#
{ 62_1 136_0 } » 264	#0,81#	{ 61_1 109_1 166_0 } » 264	#0,81#	{ 93_1 124_1 197_0 } » 264	#0,81#
{ 62_1 148_0 } » 264	#0,82#	{ 61_1 109_1 180_0 } » 264	#0,81#	{ 93_1 133_0 140_1 } » 264	#0,81#
{ 62_1 149_0 } » 264	#0,82#	{ 61_1 109_1 181_0 } » 264	#0,81#	{ 93_1 134_0 140_1 } » 264	#0,82#
{ 62_1 150_0 } » 264	#0,82#	{ 61_1 109_1 196_0 } » 264	#0,81#	{ 93_1 135_0 140_1 } » 264	#0,82#
{ 62_1 151_0 } » 264	#0,82#	{ 77_1 87_0 93_1 } » 264	#0,83#	{ 93_1 136_0 140_1 } » 264	#0,82#
{ 62_1 164_0 } » 264	#0,82#	{ 77_1 93_1 101_0 } » 264	#0,83#	{ 93_1 140_1 148_0 } » 264	#0,82#
{ 62_1 165_0 } » 264	#0,82#	{ 77_1 93_1 116_0 } » 264	#0,85#	{ 93_1 140_1 149_0 } » 264	#0,82#
{ 62_1 166_0 } » 264	#0,82#	{ 77_1 93_1 183_0 } » 264	#0,82#	{ 93_1 140_1 150_0 } » 264	#0,82#
{ 62_1 180_0 } » 264	#0,82#	{ 77_1 119_0 140_1 } » 264	#0,81#	{ 93_1 140_1 151_0 } » 264	#0,82#
{ 62_1 181_0 } » 264	#0,82#	{ 77_1 136_0 140_1 } » 264	#0,81#	{ 93_1 140_1 164_0 } » 264	#0,82#
{ 62_1 182_0 } » 264	#0,81#	{ 77_1 140_1 150_0 } » 264	#0,81#	{ 93_1 140_1 165_0 } » 264	#0,82#
{ 62_1 196_0 } » 264	#0,82#	{ 77_1 140_1 151_0 } » 264	#0,81#	{ 93_1 140_1 166_0 } » 264	#0,82#
{ 62_1 197_0 } » 264	#0,81#	{ 77_1 140_1 164_0 } » 264	#0,81#	{ 93_1 140_1 167_0 } » 264	#0,81#
{ 77_1 205_0 } » 264	#0,82#	{ 77_1 140_1 165_0 } » 264	#0,81#	{ 93_1 140_1 180_0 } » 264	#0,82#
{ 78_1 88_0 } » 264	#0,82#	{ 77_1 140_1 166_0 } » 264	#0,81#	{ 93_1 140_1 181_0 } » 264	#0,82#
{ 78_1 89_0 } » 264	#0,83#	{ 77_1 140_1 180_0 } » 264	#0,81#	{ 93_1 140_1 182_0 } » 264	#0,81#
{ 78_1 102_0 } » 264	#0,83#	{ 77_1 140_1 181_0 } » 264	#0,81#	{ 93_1 140_1 196_0 } » 264	#0,82#
{ 78_1 103_0 } » 264	#0,83#	{ 77_1 140_1 196_0 } » 264	#0,81#	{ 93_1 140_1 197_0 } » 264	#0,82#
{ 78_1 116_0 } » 264	#0,81#	{ 78_1 90_0 109_1 } » 264	#0,81#	{ 109_1 117_0 140_1 } » 264	#0,82#
{ 78_1 117_0 } » 264	#0,83#	{ 78_1 104_0 109_1 } » 264	#0,81#	{ 109_1 118_0 140_1 } » 264	#0,83#
{ 78_1 121_0 } » 264	#0,83#	{ 78_1 105_0 109_1 } » 264	#0,81#	{ 109_1 119_0 140_1 } » 264	#0,83#
{ 78_1 132_0 } » 264	#0,83#	{ 78_1 109_1 118_0 } » 264	#0,81#	{ 109_1 120_0 140_1 } » 264	#0,83#
{ 78_1 133_0 } » 264	#0,84#	{ 78_1 109_1 119_0 } » 264	#0,81#	{ 109_1 121_0 140_1 } » 264	#0,83#
{ 78_1 136_0 } » 264	#0,83#	{ 78_1 109_1 120_0 } » 264	#0,81#	{ 109_1 132_0 140_1 } » 264	#0,81#
{ 78_1 167_0 } » 264	#0,82#	{ 78_1 109_1 134_0 } » 264	#0,81#	{ 109_1 133_0 140_1 } » 264	#0,82#
{ 78_1 182_0 } » 264	#0,83#	{ 78_1 109_1 135_0 } » 264	#0,81#	{ 109_1 134_0 140_1 } » 264	#0,83#
{ 78_1 197_0 } » 264	#0,84#	{ 78_1 109_1 148_0 } » 264	#0,81#	{ 109_1 135_0 140_1 } » 264	#0,83#
{ 78_1 198_0 } » 264	#0,81#	{ 78_1 109_1 149_0 } » 264	#0,81#	{ 109_1 136_0 140_1 } » 264	#0,83#
{ 93_1 205_0 } » 264	#0,83#	{ 78_1 109_1 150_0 } » 264	#0,81#	{ 109_1 140_1 148_0 } » 264	#0,83#
{ 109_1 137_0 } » 264	#0,82#	{ 78_1 109_1 151_0 } » 264	#0,81#	{ 109_1 140_1 149_0 } » 264	#0,83#
{ 109_1 183_0 } » 264	#0,82#	{ 78_1 109_1 164_0 } » 264	#0,81#	{ 109_1 140_1 150_0 } » 264	#0,83#
{ 124_1 132_0 } » 264	#0,83#	{ 78_1 109_1 165_0 } » 264	#0,81#	{ 109_1 140_1 151_0 } » 264	#0,83#
{ 124_1 167_0 } » 264	#0,82#	{ 78_1 109_1 166_0 } » 264	#0,81#	{ 109_1 140_1 164_0 } » 264	#0,83#
{ 124_1 182_0 } » 264	#0,83#	{ 78_1 109_1 180_0 } » 264	#0,81#	{ 109_1 140_1 165_0 } » 264	#0,83#
{ 140_1 152_0 } » 264	#0,84#	{ 78_1 109_1 181_0 } » 264	#0,81#	{ 109_1 140_1 166_0 } » 264	#0,83#
{ 140_1 198_0 } » 264	#0,82#	{ 78_1 109_1 196_0 } » 264	#0,81#	{ 109_1 140_1 167_0 } » 264	#0,81#
{ 155_1 164_0 } » 264	#0,82#	{ 93_1 101_0 109_1 } » 264	#0,82#	{ 109_1 140_1 180_0 } » 264	#0,83#
{ 155_1 165_0 } » 264	#0,82#	{ 93_1 103_0 124_1 } » 264	#0,81#	{ 109_1 140_1 181_0 } » 264	#0,83#
{ 155_1 166_0 } » 264	#0,82#	{ 93_1 103_0 140_1 } » 264	#0,81#	{ 109_1 140_1 182_0 } » 264	#0,82#
{ 155_1 180_0 } » 264	#0,82#	{ 93_1 104_0 124_1 } » 264	#0,82#	{ 109_1 140_1 196_0 } » 264	#0,83#
{ 155_1 181_0 } » 264	#0,82#	{ 93_1 104_0 140_1 } » 264	#0,82#	{ 109_1 140_1 197_0 } » 264	#0,82#
{ 155_1 196_0 } » 264	#0,82#	{ 93_1 105_0 124_1 } » 264	#0,82#	{ 61_1 77_1 88_0 93_1 } » 264	#0,82#
{ 171_1 180_0 } » 264	#0,82#	{ 93_1 105_0 140_1 } » 264	#0,82#	{ 61_1 77_1 89_0 93_1 } » 264	#0,83#
{ 171_1 181_0 } » 264	#0,82#	{ 93_1 109_1 116_0 } » 264	#0,83#	{ 61_1 77_1 90_0 93_1 } » 264	#0,82#
{ 171_1 182_0 } » 264	#0,82#	{ 93_1 117_0 140_1 } » 264	#0,81#	{ 61_1 77_1 93_1 102_0 } » 264	#0,82#
{ 171_1 196_0 } » 264	#0,82#	{ 93_1 118_0 124_1 } » 264	#0,82#	{ 61_1 77_1 93_1 103_0 } » 264	#0,83#
{ 171_1 197_0 } » 264	#0,82#	{ 93_1 118_0 140_1 } » 264	#0,82#	{ 61_1 77_1 93_1 104_0 } » 264	#0,84#
{ 61_1 74_0 77_1 } » 264	#0,81#	{ 93_1 119_0 124_1 } » 264	#0,82#	{ 61_1 77_1 93_1 105_0 } » 264	#0,83#
{ 61_1 77_1 116_0 } » 264	#0,81#	{ 93_1 119_0 140_1 } » 264	#0,82#	{ 61_1 77_1 93_1 117_0 } » 264	#0,82#
{ 61_1 77_1 121_0 } » 264	#0,82#	{ 93_1 120_0 124_1 } » 264	#0,82#	{ 61_1 77_1 93_1 118_0 } » 264	#0,83#
{ 61_1 90_0 109_1 } » 264	#0,81#	{ 93_1 120_0 140_1 } » 264	#0,82#	{ 61_1 77_1 93_1 119_0 } » 264	#0,84#
{ 61_1 93_1 121_0 } » 264	#0,81#	{ 93_1 121_0 140_1 } » 264	#0,82#	{ 61_1 77_1 93_1 120_0 } » 264	#0,83#
{ 61_1 104_0 109_1 } » 264	#0,81#	{ 93_1 124_1 133_0 } » 264	#0,81#	{ 61_1 77_1 93_1 132_0 } » 264	#0,82#
{ 61_1 105_0 109_1 } » 264	#0,81#	{ 93_1 124_1 134_0 } » 264	#0,82#	{ 61_1 77_1 93_1 133_0 } » 264	#0,83#
{ 61_1 109_1 118_0 } » 264	#0,81#	{ 93_1 124_1 135_0 } » 264	#0,82#	{ 61_1 77_1 93_1 134_0 } » 264	#0,83#
{ 61_1 109_1 119_0 } » 264	#0,81#	{ 93_1 124_1 136_0 } » 264	#0,81#	{ 61_1 77_1 93_1 135_0 } » 264	#0,84#
{ 61_1 109_1 120_0 } » 264	#0,81#	{ 93_1 124_1 148_0 } » 264	#0,82#	{ 61_1 77_1 93_1 136_0 } » 264	#0,83#
{ 61_1 109_1 134_0 } » 264	#0,81#	{ 93_1 124_1 149_0 } » 264	#0,82#	{ 61_1 77_1 93_1 148_0 } » 264	#0,84#
{ 61_1 109_1 135_0 } » 264	#0,81#	{ 93_1 124_1 150_0 } » 264	#0,82#	{ 61_1 77_1 93_1 149_0 } » 264	#0,83#
{ 61_1 109_1 136_0 } » 264	#0,81#	{ 93_1 124_1 151_0 } » 264	#0,82#	{ 61_1 77_1 93_1 150_0 } » 264	#0,84#
{ 61_1 109_1 148_0 } » 264	#0,81#	{ 93_1 124_1 164_0 } » 264	#0,82#	{ 61_1 77_1 93_1 151_0 } » 264	#0,83#
{ 61_1 109_1 149_0 } » 264	#0,81#	{ 93_1 124_1 165_0 } » 264	#0,82#	{ 61_1 77_1 93_1 164_0 } » 264	#0,84#
{ 61_1 109_1 150_0 } » 264	#0,81#	{ 93_1 124_1 166_0 } » 264	#0,82#	{ 61_1 77_1 93_1 165_0 } » 264	#0,84#

{ 61_1 77_1 93_1 166_0 } » 264	#0,83#	{ 37_1 71_0 } » 265	#0,59#	{ 83_1 88_0 } » 265	#0,61#
{ 61_1 77_1 93_1 167_0 } » 264	#0,82#	{ 37_1 72_0 } » 265	#0,59#	{ 83_1 99_1 } » 265	#0,61#
{ 61_1 77_1 93_1 180_0 } » 264	#0,84#	{ 38_1 52_1 } » 265	#0,59#	{ 83_1 125_0 } » 265	#0,59#
{ 61_1 77_1 93_1 181_0 } » 264	#0,84#	{ 38_1 71_0 } » 265	#0,60#	{ 83_1 135_1 } » 265	#0,61#
{ 61_1 77_1 93_1 182_0 } » 264	#0,83#	{ 38_1 72_0 } » 265	#0,59#	{ 83_1 138_1 } » 265	#0,60#
{ 61_1 77_1 93_1 196_0 } » 264	#0,84#	{ 38_1 204_1 } » 265	#0,60#	{ 83_1 151_1 } » 265	#0,58#
{ 61_1 77_1 93_1 197_0 } » 264	#0,83#	{ 38_1 218_1 } » 265	#0,58#	{ 83_1 185_0 } » 265	#0,60#
{ 61_1 77_1 93_1 198_0 } » 264	#0,81#	{ 44_1 204_1 } » 265	#0,59#	{ 83_1 205_1 } » 265	#0,58#
{ 77_1 88_0 93_1 109_1 } » 264	#0,82#	{ 45_1 57_0 } » 265	#0,59#	{ 83_1 214_1 } » 265	#0,59#
{ 77_1 89_0 93_1 109_1 } » 264	#0,84#	{ 45_1 61_1 } » 265	#0,62#	{ 83_1 218_1 } » 265	#0,60#
{ 77_1 90_0 93_1 109_1 } » 264	#0,84#	{ 45_1 71_0 } » 265	#0,59#	{ 83_1 219_1 } » 265	#0,61#
{ 77_1 93_1 102_0 109_1 } » 264	#0,83#	{ 45_1 72_0 } » 265	#0,62#	{ 84_1 87_0 } » 265	#0,61#
{ 77_1 93_1 103_0 109_1 } » 264	#0,84#	{ 45_1 185_0 } » 265	#0,58#	{ 84_1 100_1 } » 265	#0,59#
{ 77_1 93_1 104_0 109_1 } » 264	#0,84#	{ 45_1 204_1 } » 265	#0,61#	{ 84_1 136_1 } » 265	#0,59#
{ 77_1 93_1 105_0 109_1 } » 264	#0,85#	{ 51_1 67_1 } » 265	#0,58#	{ 84_1 204_1 } » 265	#0,63#
{ 77_1 93_1 106_0 109_1 } » 264	#0,81#	{ 51_1 71_0 } » 265	#0,60#	{ 84_1 219_1 } » 265	#0,58#
{ 77_1 93_1 109_1 117_0 } » 264	#0,83#	{ 51_1 72_0 } » 265	#0,59#	{ 92_1 107_1 } » 265	#0,59#
{ 77_1 93_1 109_1 118_0 } » 264	#0,84#	{ 52_1 53_1 } » 265	#0,59#	{ 100_1 204_1 } » 265	#0,59#
{ 77_1 93_1 109_1 119_0 } » 264	#0,85#	{ 52_1 56_0 } » 265	#0,61#	{ 107_1 122_1 } » 265	#0,63#
{ 77_1 93_1 109_1 120_0 } » 264	#0,85#	{ 52_1 57_0 } » 265	#0,62#	{ 107_1 123_1 } » 265	#0,59#
{ 77_1 93_1 109_1 121_0 } » 264	#0,84#	{ 52_1 67_1 } » 265	#0,61#	{ 107_1 135_1 } » 265	#0,59#
{ 77_1 93_1 109_1 132_0 } » 264	#0,83#	{ 52_1 73_0 } » 265	#0,60#	{ 107_1 185_0 } » 265	#0,58#
{ 77_1 93_1 109_1 133_0 } » 264	#0,84#	{ 52_1 87_0 } » 265	#0,58#	{ 107_1 204_1 } » 265	#0,61#
{ 77_1 93_1 109_1 134_0 } » 264	#0,84#	{ 52_1 135_1 } » 265	#0,61#	{ 118_1 204_1 } » 265	#0,59#
{ 77_1 93_1 109_1 135_0 } » 264	#0,85#	{ 52_1 136_1 } » 265	#0,61#	{ 119_1 204_1 } » 265	#0,58#
{ 77_1 93_1 109_1 136_0 } » 264	#0,84#	{ 52_1 137_1 } » 265	#0,60#	{ 120_1 121_1 } » 265	#0,62#
{ 77_1 93_1 109_1 148_0 } » 264	#0,85#	{ 52_1 138_1 } » 265	#0,60#	{ 120_1 136_1 } » 265	#0,60#
{ 77_1 93_1 109_1 149_0 } » 264	#0,84#	{ 52_1 189_1 } » 265	#0,59#	{ 120_1 185_0 } » 265	#0,58#
{ 77_1 93_1 109_1 150_0 } » 264	#0,85#	{ 52_1 205_1 } » 265	#0,61#	{ 120_1 204_1 } » 265	#0,61#
{ 77_1 93_1 109_1 151_0 } » 264	#0,84#	{ 53_1 204_1 } » 265	#0,59#	{ 120_1 218_1 } » 265	#0,61#
{ 77_1 93_1 109_1 152_0 } » 264	#0,81#	{ 61_1 71_0 } » 265	#0,63#	{ 121_1 122_1 } » 265	#0,61#
{ 77_1 93_1 109_1 164_0 } » 264	#0,85#	{ 61_1 72_0 } » 265	#0,66#	{ 121_1 125_0 } » 265	#0,59#
{ 77_1 93_1 109_1 165_0 } » 264	#0,85#	{ 61_1 73_0 } » 265	#0,62#	{ 121_1 135_1 } » 265	#0,58#
{ 77_1 93_1 109_1 166_0 } » 264	#0,85#	{ 61_1 77_1 } » 265	#0,62#	{ 121_1 136_1 } » 265	#0,61#
{ 77_1 93_1 109_1 167_0 } » 264	#0,83#	{ 61_1 136_1 } » 265	#0,59#	{ 121_1 137_1 } » 265	#0,61#
{ 77_1 93_1 109_1 180_0 } » 264	#0,85#	{ 61_1 137_1 } » 265	#0,58#	{ 121_1 185_0 } » 265	#0,61#
{ 77_1 93_1 109_1 181_0 } » 264	#0,85#	{ 61_1 185_0 } » 265	#0,59#	{ 121_1 204_1 } » 265	#0,63#
{ 77_1 93_1 109_1 182_0 } » 264	#0,84#	{ 61_1 204_1 } » 265	#0,63#	{ 121_1 217_1 } » 265	#0,58#
{ 77_1 93_1 109_1 196_0 } » 264	#0,85#	{ 61_1 214_1 } » 265	#0,58#	{ 121_1 218_1 } » 265	#0,62#
{ 77_1 93_1 109_1 197_0 } » 264	#0,84#	{ 61_1 218_1 } » 265	#0,58#	{ 121_1 219_1 } » 265	#0,59#
{ 77_1 93_1 109_1 198_0 } » 264	#0,81#	{ 61_1 219_1 } » 265	#0,60#	{ 122_1 123_1 } » 265	#0,60#
{ 24_1 } » 265	#0,60#	{ 67_1 68_1 } » 265	#0,60#	{ 122_1 125_0 } » 265	#0,58#
{ 25_1 } » 265	#0,63#	{ 67_1 73_0 } » 265	#0,58#	{ 122_1 135_1 } » 265	#0,58#
{ 26_1 } » 265	#0,64#	{ 67_1 135_1 } » 265	#0,59#	{ 122_1 136_1 } » 265	#0,58#
{ 27_1 } » 265	#0,64#	{ 67_1 136_1 } » 265	#0,62#	{ 122_1 137_1 } » 265	#0,61#
{ 28_1 } » 265	#0,59#	{ 67_1 137_1 } » 265	#0,60#	{ 122_1 138_1 } » 265	#0,63#
{ 43_1 } » 265	#0,59#	{ 67_1 138_1 } » 265	#0,58#	{ 122_1 185_0 } » 265	#0,62#
{ 60_1 } » 265	#0,62#	{ 67_1 185_0 } » 265	#0,61#	{ 122_1 189_1 } » 265	#0,58#
{ 62_1 } » 265	#0,58#	{ 67_1 218_1 } » 265	#0,61#	{ 122_1 204_1 } » 265	#0,64#
{ 76_1 } » 265	#0,65#	{ 67_1 219_1 } » 265	#0,60#	{ 122_1 205_1 } » 265	#0,59#
{ 91_1 } » 265	#0,62#	{ 68_1 73_0 } » 265	#0,61#	{ 122_1 218_1 } » 265	#0,61#
{ 101_1 } » 265	#0,59#	{ 68_1 83_1 } » 265	#0,59#	{ 122_1 219_1 } » 265	#0,60#
{ 106_1 } » 265	#0,64#	{ 68_1 84_1 } » 265	#0,61#	{ 123_1 204_1 } » 265	#0,60#
{ 108_1 } » 265	#0,61#	{ 68_1 87_0 } » 265	#0,58#	{ 134_1 135_1 } » 265	#0,64#
{ 117_1 } » 265	#0,63#	{ 68_1 122_1 } » 265	#0,58#	{ 134_1 150_1 } » 265	#0,58#
{ 133_1 } » 265	#0,59#	{ 68_1 135_1 } » 265	#0,60#	{ 134_1 185_0 } » 265	#0,59#
{ 153_1 } » 265	#0,64#	{ 68_1 136_1 } » 265	#0,59#	{ 134_1 204_1 } » 265	#0,60#
{ 154_1 } » 265	#0,65#	{ 68_1 137_1 } » 265	#0,59#	{ 135_1 138_1 } » 265	#0,58#
{ 156_1 } » 265	#0,64#	{ 68_1 138_1 } » 265	#0,59#	{ 135_1 150_1 } » 265	#0,61#
{ 165_1 } » 265	#0,59#	{ 68_1 185_0 } » 265	#0,61#	{ 135_1 151_1 } » 265	#0,61#
{ 166_1 } » 265	#0,64#	{ 68_1 189_1 } » 265	#0,60#	{ 135_1 184_0 } » 265	#0,58#
{ 171_1 } » 265	#0,59#	{ 68_1 205_1 } » 265	#0,61#	{ 135_1 186_0 } » 265	#0,58#
{ 196_1 } » 265	#0,64#	{ 68_1 218_1 } » 265	#0,61#	{ 135_1 189_1 } » 265	#0,60#
{ 198_1 } » 265	#0,60#	{ 68_1 219_1 } » 265	#0,63#	{ 135_1 201_0 } » 265	#0,58#
{ 37_1 52_1 } » 265	#0,59#	{ 77_1 204_1 } » 265	#0,59#	{ 135_1 205_1 } » 265	#0,60#
{ 37_1 56_0 } » 265	#0,60#	{ 83_1 84_1 } » 265	#0,62#	{ 135_1 214_1 } » 265	#0,63#

{ 135_1 217_1 } » 265	#0,58#	{ 68_1 72_0 204_1 } » 265	#0,58#	{ 83_1 87_0 } » 266	#0,78#
{ 135_1 218_1 } » 265	#0,64#	{ 83_1 87_0 100_1 } » 265	#0,58#	{ 83_1 164_0 } » 266	#0,80#
{ 135_1 220_1 } » 265	#0,59#	{ 83_1 87_0 136_1 } » 265	#0,58#	{ 83_1 165_0 } » 266	#0,81#
{ 136_1 151_1 } » 265	#0,61#	{ 83_1 87_0 204_1 } » 265	#0,58#	{ 83_1 166_0 } » 266	#0,80#
{ 136_1 152_1 } » 265	#0,59#	{ 83_1 136_1 137_1 } » 265	#0,58#	{ 83_1 167_0 } » 266	#0,78#
{ 136_1 188_1 } » 265	#0,59#	{ 135_1 136_1 137_1 } » 265	#0,60#	{ 83_1 199_0 } » 266	#0,78#
{ 136_1 189_1 } » 265	#0,59#	{ 135_1 136_1 204_1 } » 265	#0,60#	{ 84_1 93_1 } » 266	#0,77#
{ 136_1 201_0 } » 265	#0,59#	{ 135_1 136_1 219_1 } » 265	#0,58#	{ 84_1 180_0 } » 266	#0,80#
{ 136_1 205_1 } » 265	#0,60#	{ 135_1 185_0 204_1 } » 265	#0,60#	{ 84_1 181_0 } » 266	#0,79#
{ 136_1 214_1 } » 265	#0,62#	{ 135_1 204_1 219_1 } » 265	#0,59#	{ 84_1 182_0 } » 266	#0,77#
{ 136_1 217_1 } » 265	#0,61#	{ 136_1 137_1 138_1 } » 265	#0,59#	{ 84_1 197_0 } » 266	#0,80#
{ 136_1 220_1 } » 265	#0,59#	{ 136_1 137_1 185_0 } » 265	#0,58#	{ 84_1 198_0 } » 266	#0,79#
{ 137_1 205_1 } » 265	#0,59#	{ 136_1 137_1 204_1 } » 265	#0,61#	{ 92_1 180_0 } » 266	#0,79#
{ 137_1 214_1 } » 265	#0,59#	{ 136_1 137_1 218_1 } » 265	#0,58#	{ 92_1 181_0 } » 266	#0,78#
{ 137_1 220_1 } » 265	#0,58#	{ 136_1 137_1 219_1 } » 265	#0,59#	{ 92_1 198_0 } » 266	#0,78#
{ 138_1 139_1 } » 265	#0,58#	{ 136_1 185_0 204_1 } » 265	#0,59#	{ 93_1 156_1 } » 266	#0,76#
{ 138_1 155_1 } » 265	#0,59#	{ 136_1 204_1 218_1 } » 265	#0,59#	{ 93_1 164_0 } » 266	#0,79#
{ 138_1 185_0 } » 265	#0,63#	{ 136_1 204_1 219_1 } » 265	#0,61#	{ 93_1 165_0 } » 266	#0,80#
{ 138_1 204_1 } » 265	#0,66#	{ 136_1 218_1 219_1 } » 265	#0,59#	{ 93_1 166_0 } » 266	#0,80#
{ 138_1 205_1 } » 265	#0,59#	{ 137_1 204_1 219_1 } » 265	#0,60#	{ 93_1 167_0 } » 266	#0,78#
{ 138_1 214_1 } » 265	#0,58#	{ 172_1 185_0 204_1 } » 265	#0,59#	{ 93_1 199_0 } » 266	#0,78#
{ 138_1 218_1 } » 265	#0,63#	{ 172_1 188_1 204_1 } » 265	#0,60#	{ 99_1 183_0 } » 266	#0,77#
{ 138_1 219_1 } » 265	#0,63#	{ 173_1 185_0 189_1 } » 265	#0,58#	{ 108_1 164_0 } » 266	#0,78#
{ 139_1 204_1 } » 265	#0,58#	{ 173_1 186_0 204_1 } » 265	#0,58#	{ 108_1 165_0 } » 266	#0,79#
{ 150_1 185_0 } » 265	#0,59#	{ 173_1 189_1 204_1 } » 265	#0,60#	{ 108_1 166_0 } » 266	#0,79#
{ 150_1 204_1 } » 265	#0,59#	{ 173_1 204_1 205_1 } » 265	#0,58#	{ 108_1 167_0 } » 266	#0,78#
{ 151_1 152_1 } » 265	#0,58#	{ 188_1 189_1 204_1 } » 265	#0,59#	{ 108_1 199_0 } » 266	#0,78#
{ 151_1 204_1 } » 265	#0,61#	{ 188_1 204_1 218_1 } » 265	#0,60#	{ 124_1 155_1 } » 266	#0,76#
{ 155_1 204_1 } » 265	#0,60#	{ 188_1 204_1 219_1 } » 265	#0,60#	{ 124_1 164_0 } » 266	#0,77#
{ 172_1 189_1 } » 265	#0,59#	{ 189_1 204_1 205_1 } » 265	#0,66#	{ 124_1 165_0 } » 266	#0,78#
{ 172_1 218_1 } » 265	#0,58#	{ 189_1 204_1 219_1 } » 265	#0,59#	{ 124_1 166_0 } » 266	#0,78#
{ 172_1 219_1 } » 265	#0,58#	{ 189_1 205_1 219_1 } » 265	#0,61#	{ 124_1 167_0 } » 266	#0,77#
{ 181_1 184_0 } » 265	#0,58#	{ 189_1 205_1 220_1 } » 265	#0,59#	{ 124_1 199_0 } » 266	#0,77#
{ 181_1 197_1 } » 265	#0,60#	{ 204_1 205_1 219_1 } » 265	#0,62#	{ 140_1 156_1 } » 266	#0,77#
{ 188_1 205_1 } » 265	#0,59#	{ 204_1 205_1 220_1 } » 265	#0,59#	{ 155_1 180_0 } » 266	#0,77#
{ 189_1 201_0 } » 265	#0,58#	{ 204_1 217_1 218_1 } » 265	#0,61#	{ 155_1 181_0 } » 266	#0,76#
{ 189_1 218_1 } » 265	#0,61#	{ 204_1 218_1 219_1 } » 265	#0,66#	{ 155_1 196_0 } » 266	#0,78#
{ 197_1 200_0 } » 265	#0,60#	{ 204_1 219_1 220_1 } » 265	#0,64#	{ 155_1 197_0 } » 266	#0,77#
{ 197_1 204_1 } » 265	#0,59#	{ 205_1 219_1 220_1 } » 265	#0,60#	{ 156_1 196_0 } » 266	#0,77#
{ 197_1 213_1 } » 265	#0,58#	{ 215_1 216_1 217_1 } » 265	#0,58#	{ 156_1 197_0 } » 266	#0,77#
{ 197_1 214_1 } » 265	#0,62#	{ 216_1 217_1 218_1 } » 265	#0,60#	{ 156_1 198_0 } » 266	#0,76#
{ 203_1 204_1 } » 265	#0,63#	{ 217_1 218_1 219_1 } » 265	#0,59#	{ 171_1 196_0 } » 266	#0,77#
{ 203_1 218_1 } » 265	#0,60#	{ 218_1 219_1 220_1 } » 265	#0,59#	{ 171_1 197_0 } » 266	#0,76#
{ 204_1 214_1 } » 265	#0,67#	{ 173_1 186_0 189_1 205_1 } » 265	#0,59#	{ 53_1 68_1 196_0 } » 266	#0,77#
{ 204_1 215_1 } » 265	#0,62#	{ 38_1 196_0 } » 266	#0,77#	{ 53_1 68_1 197_0 } » 266	#0,77#
{ 204_1 216_1 } » 265	#0,62#	{ 45_1 196_0 } » 266	#0,78#	{ 61_1 68_1 196_0 } » 266	#0,76#
{ 205_1 214_1 } » 265	#0,59#	{ 45_1 197_0 } » 266	#0,76#	{ 61_1 72_0 77_1 } » 266	#0,77#
{ 205_1 218_1 } » 265	#0,61#	{ 52_1 196_0 } » 266	#0,77#	{ 61_1 77_1 124_1 } » 266	#0,77#
{ 213_1 214_1 } » 265	#0,61#	{ 53_1 180_0 } » 266	#0,77#	{ 61_1 77_1 182_0 } » 266	#0,78#
{ 214_1 215_1 } » 265	#0,62#	{ 53_1 181_0 } » 266	#0,76#	{ 61_1 83_1 180_0 } » 266	#0,77#
{ 214_1 218_1 } » 265	#0,63#	{ 53_1 198_0 } » 266	#0,78#	{ 61_1 83_1 197_0 } » 266	#0,77#
{ 214_1 219_1 } » 265	#0,62#	{ 61_1 73_0 } » 266	#0,77#	{ 61_1 108_1 180_0 } » 266	#0,76#
{ 215_1 218_1 } » 265	#0,59#	{ 62_1 196_0 } » 266	#0,77#	{ 61_1 124_1 196_0 } » 266	#0,77#
{ 52_1 68_1 71_0 } » 265	#0,58#	{ 67_1 182_0 } » 266	#0,76#	{ 61_1 124_1 197_0 } » 266	#0,76#
{ 52_1 71_0 204_1 } » 265	#0,61#	{ 67_1 198_0 } » 266	#0,76#	{ 67_1 71_0 83_1 } » 266	#0,77#
{ 52_1 71_0 219_1 } » 265	#0,58#	{ 68_1 87_0 } » 266	#0,78#	{ 67_1 83_1 180_0 } » 266	#0,78#
{ 52_1 72_0 204_1 } » 265	#0,59#	{ 68_1 164_0 } » 266	#0,78#	{ 67_1 83_1 181_0 } » 266	#0,77#
{ 52_1 185_0 204_1 } » 265	#0,59#	{ 68_1 165_0 } » 266	#0,78#	{ 67_1 83_1 196_0 } » 266	#0,79#
{ 52_1 204_1 218_1 } » 265	#0,58#	{ 68_1 166_0 } » 266	#0,78#	{ 67_1 83_1 197_0 } » 266	#0,77#
{ 52_1 204_1 219_1 } » 265	#0,58#	{ 68_1 199_0 } » 266	#0,77#	{ 68_1 71_0 83_1 } » 266	#0,78#
{ 67_1 70_0 83_1 } » 265	#0,60#	{ 77_1 84_1 } » 266	#0,77#	{ 68_1 72_0 83_1 } » 266	#0,78#
{ 67_1 71_0 83_1 } » 265	#0,63#	{ 77_1 87_0 } » 266	#0,76#	{ 68_1 73_0 83_1 } » 266	#0,76#
{ 67_1 71_0 204_1 } » 265	#0,59#	{ 77_1 164_0 } » 266	#0,78#	{ 68_1 77_1 181_0 } » 266	#0,78#
{ 67_1 72_0 83_1 } » 265	#0,59#	{ 77_1 165_0 } » 266	#0,79#	{ 68_1 77_1 182_0 } » 266	#0,76#
{ 67_1 83_1 86_0 } » 265	#0,58#	{ 77_1 166_0 } » 266	#0,78#	{ 68_1 83_1 182_0 } » 266	#0,80#
{ 67_1 83_1 87_0 } » 265	#0,60#	{ 83_1 86_0 } » 266	#0,78#	{ 68_1 83_1 183_0 } » 266	#0,78#

{ 68_1 84_1 196_0 } » 266	#0,76#
{ 68_1 93_1 182_0 } » 266	#0,79#
{ 68_1 93_1 183_0 } » 266	#0,77#
{ 68_1 99_1 198_0 } » 266	#0,76#
{ 68_1 108_1 180_0 } » 266	#0,79#
{ 68_1 108_1 181_0 } » 266	#0,79#
{ 68_1 108_1 182_0 } » 266	#0,78#
{ 68_1 109_1 181_0 } » 266	#0,76#
{ 68_1 124_1 180_0 } » 266	#0,78#
{ 68_1 124_1 181_0 } » 266	#0,78#
{ 68_1 124_1 182_0 } » 266	#0,77#
{ 68_1 124_1 198_0 } » 266	#0,78#
{ 68_1 140_1 196_0 } » 266	#0,76#
{ 77_1 93_1 183_0 } » 266	#0,79#
{ 77_1 99_1 196_0 } » 266	#0,77#
{ 77_1 99_1 197_0 } » 266	#0,77#
{ 77_1 124_1 183_0 } » 266	#0,77#
{ 77_1 140_1 180_0 } » 266	#0,77#
{ 77_1 140_1 181_0 } » 266	#0,77#
{ 77_1 140_1 198_0 } » 266	#0,77#
{ 83_1 84_1 196_0 } » 266	#0,77#
{ 83_1 93_1 183_0 } » 266	#0,78#
{ 83_1 99_1 180_0 } » 266	#0,79#
{ 83_1 99_1 181_0 } » 266	#0,79#
{ 83_1 99_1 182_0 } » 266	#0,77#
{ 83_1 99_1 198_0 } » 266	#0,79#
{ 83_1 108_1 182_0 } » 266	#0,79#
{ 83_1 108_1 183_0 } » 266	#0,76#
{ 83_1 124_1 182_0 } » 266	#0,78#
{ 83_1 124_1 183_0 } » 266	#0,77#
{ 83_1 140_1 180_0 } » 266	#0,77#
{ 83_1 140_1 181_0 } » 266	#0,77#
{ 83_1 140_1 198_0 } » 266	#0,77#
{ 92_1 108_1 196_0 } » 266	#0,77#
{ 92_1 108_1 197_0 } » 266	#0,77#
{ 93_1 99_1 180_0 } » 266	#0,76#
{ 93_1 99_1 197_0 } » 266	#0,78#
{ 93_1 99_1 198_0 } » 266	#0,77#
{ 93_1 108_1 183_0 } » 266	#0,78#
{ 93_1 109_1 183_0 } » 266	#0,80#
{ 93_1 124_1 183_0 } » 266	#0,79#
{ 93_1 140_1 183_0 } » 266	#0,78#
{ 99_1 108_1 196_0 } » 266	#0,78#
{ 99_1 108_1 197_0 } » 266	#0,78#
{ 108_1 109_1 182_0 } » 266	#0,77#
{ 108_1 124_1 183_0 } » 266	#0,80#
{ 108_1 140_1 180_0 } » 266	#0,77#
{ 108_1 140_1 181_0 } » 266	#0,77#
{ 109_1 124_1 183_0 } » 266	#0,76#
{ 109_1 125_1 197_0 } » 266	#0,77#
{ 124_1 140_1 183_0 } » 266	#0,78#
{ 61_1 77_1 83_1 196_0 } » 266	#0,77#
{ 61_1 77_1 93_1 180_0 } » 266	#0,77#
{ 61_1 77_1 93_1 181_0 } » 266	#0,76#
{ 61_1 77_1 93_1 196_0 } » 266	#0,79#
{ 61_1 77_1 93_1 197_0 } » 266	#0,78#
{ 61_1 77_1 93_1 198_0 } » 266	#0,76#
{ 61_1 77_1 108_1 196_0 } » 266	#0,77#
{ 61_1 77_1 108_1 197_0 } » 266	#0,77#
{ 68_1 77_1 83_1 196_0 } » 266	#0,78#

{ 68_1 77_1 83_1 197_0 } » 266	#0,77#
{ 68_1 77_1 93_1 180_0 } » 266	#0,76#
{ 68_1 77_1 93_1 196_0 } » 266	#0,78#
{ 68_1 77_1 93_1 197_0 } » 266	#0,78#
{ 68_1 77_1 93_1 198_0 } » 266	#0,77#
{ 68_1 83_1 93_1 180_0 } » 266	#0,78#
{ 68_1 83_1 93_1 181_0 } » 266	#0,77#
{ 68_1 83_1 93_1 196_0 } » 266	#0,80#
{ 68_1 83_1 93_1 197_0 } » 266	#0,79#
{ 68_1 83_1 93_1 198_0 } » 266	#0,78#
{ 68_1 83_1 99_1 196_0 } » 266	#0,77#
{ 68_1 83_1 99_1 197_0 } » 266	#0,77#
{ 68_1 83_1 108_1 196_0 } » 266	#0,78#
{ 68_1 83_1 108_1 197_0 } » 266	#0,78#
{ 68_1 83_1 108_1 198_0 } » 266	#0,76#
{ 68_1 93_1 108_1 196_0 } » 266	#0,77#
{ 68_1 93_1 108_1 197_0 } » 266	#0,77#
{ 68_1 93_1 109_1 180_0 } » 266	#0,76#
{ 68_1 93_1 109_1 196_0 } » 266	#0,78#
{ 68_1 93_1 109_1 197_0 } » 266	#0,78#
{ 68_1 93_1 109_1 198_0 } » 266	#0,77#
{ 68_1 93_1 124_1 196_0 } » 266	#0,77#
{ 68_1 93_1 124_1 197_0 } » 266	#0,76#
{ 77_1 83_1 93_1 180_0 } » 266	#0,79#
{ 77_1 83_1 93_1 181_0 } » 266	#0,78#
{ 77_1 83_1 93_1 182_0 } » 266	#0,76#
{ 77_1 83_1 93_1 196_0 } » 266	#0,81#
{ 77_1 83_1 93_1 197_0 } » 266	#0,80#
{ 77_1 83_1 93_1 198_0 } » 266	#0,78#
{ 77_1 83_1 108_1 180_0 } » 266	#0,76#
{ 77_1 83_1 108_1 196_0 } » 266	#0,78#
{ 77_1 83_1 108_1 197_0 } » 266	#0,77#
{ 77_1 83_1 124_1 196_0 } » 266	#0,77#
{ 77_1 93_1 108_1 109_1 } » 266	#0,76#
{ 77_1 93_1 108_1 180_0 } » 266	#0,79#
{ 77_1 93_1 108_1 181_0 } » 266	#0,78#
{ 77_1 93_1 108_1 182_0 } » 266	#0,76#
{ 77_1 93_1 108_1 197_0 } » 266	#0,80#
{ 77_1 93_1 108_1 198_0 } » 266	#0,78#
{ 77_1 93_1 109_1 180_0 } » 266	#0,78#
{ 77_1 93_1 109_1 181_0 } » 266	#0,78#
{ 77_1 93_1 109_1 182_0 } » 266	#0,76#
{ 77_1 93_1 109_1 197_0 } » 266	#0,79#
{ 77_1 93_1 109_1 198_0 } » 266	#0,78#
{ 77_1 93_1 124_1 140_1 } » 266	#0,76#
{ 77_1 93_1 124_1 180_0 } » 266	#0,79#
{ 77_1 93_1 124_1 181_0 } » 266	#0,78#
{ 77_1 93_1 124_1 182_0 } » 266	#0,77#
{ 77_1 93_1 124_1 197_0 } » 266	#0,79#
{ 77_1 93_1 124_1 198_0 } » 266	#0,78#
{ 77_1 93_1 140_1 196_0 } » 266	#0,78#
{ 77_1 93_1 140_1 197_0 } » 266	#0,77#
{ 77_1 108_1 124_1 180_0 } » 266	#0,77#
{ 77_1 108_1 124_1 181_0 } » 266	#0,77#
{ 77_1 108_1 124_1 197_0 } » 266	#0,78#
{ 77_1 108_1 124_1 198_0 } » 266	#0,76#
{ 77_1 124_1 140_1 196_0 } » 266	#0,77#
{ 83_1 93_1 99_1 196_0 } » 266	#0,76#
{ 83_1 93_1 108_1 180_0 } » 266	#0,78#
{ 83_1 93_1 108_1 181_0 } » 266	#0,77#

{ 83_1 93_1 108_1 196_0 } » 266	#0,80#
{ 83_1 93_1 108_1 197_0 } » 266	#0,79#
{ 83_1 93_1 108_1 198_0 } » 266	#0,77#
{ 83_1 93_1 109_1 180_0 } » 266	#0,78#
{ 83_1 93_1 109_1 181_0 } » 266	#0,78#
{ 83_1 93_1 109_1 182_0 } » 266	#0,76#
{ 83_1 93_1 109_1 196_0 } » 266	#0,80#
{ 83_1 93_1 109_1 197_0 } » 266	#0,79#
{ 83_1 93_1 109_1 198_0 } » 266	#0,78#
{ 83_1 93_1 124_1 180_0 } » 266	#0,78#
{ 83_1 93_1 124_1 181_0 } » 266	#0,77#
{ 83_1 93_1 124_1 196_0 } » 266	#0,79#
{ 83_1 93_1 124_1 197_0 } » 266	#0,78#
{ 83_1 93_1 124_1 198_0 } » 266	#0,77#
{ 83_1 93_1 140_1 196_0 } » 266	#0,77#
{ 83_1 93_1 140_1 197_0 } » 266	#0,76#
{ 83_1 108_1 124_1 180_0 } » 266	#0,77#
{ 83_1 108_1 124_1 181_0 } » 266	#0,76#
{ 83_1 108_1 124_1 196_0 } » 266	#0,78#
{ 83_1 108_1 124_1 197_0 } » 266	#0,78#
{ 83_1 108_1 124_1 198_0 } » 266	#0,76#
{ 83_1 109_1 124_1 196_0 } » 266	#0,76#
{ 83_1 124_1 140_1 196_0 } » 266	#0,76#
{ 93_1 108_1 109_1 180_0 } » 266	#0,78#
{ 93_1 108_1 109_1 181_0 } » 266	#0,77#
{ 93_1 108_1 109_1 198_0 } » 266	#0,78#
{ 93_1 108_1 124_1 140_1 } » 266	#0,76#
{ 93_1 108_1 124_1 180_0 } » 266	#0,79#
{ 93_1 108_1 124_1 181_0 } » 266	#0,78#
{ 93_1 108_1 124_1 182_0 } » 266	#0,77#
{ 93_1 108_1 124_1 198_0 } » 266	#0,78#
{ 93_1 108_1 140_1 196_0 } » 266	#0,76#
{ 93_1 109_1 124_1 180_0 } » 266	#0,79#
{ 93_1 109_1 124_1 181_0 } » 266	#0,79#
{ 93_1 109_1 124_1 182_0 } » 266	#0,77#
{ 93_1 109_1 124_1 198_0 } » 266	#0,78#
{ 93_1 109_1 125_1 196_0 } » 266	#0,76#
{ 93_1 109_1 140_1 180_0 } » 266	#0,78#
{ 93_1 109_1 140_1 181_0 } » 266	#0,78#
{ 93_1 109_1 140_1 182_0 } » 266	#0,76#
{ 93_1 109_1 140_1 198_0 } » 266	#0,78#
{ 93_1 124_1 140_1 180_0 } » 266	#0,78#
{ 93_1 124_1 140_1 181_0 } » 266	#0,78#
{ 93_1 124_1 140_1 182_0 } » 266	#0,76#
{ 93_1 124_1 140_1 198_0 } » 266	#0,78#
{ 108_1 124_1 140_1 196_0 } » 266	#0,78#
{ 108_1 124_1 140_1 197_0 } » 266	#0,78#
{ 108_1 124_1 140_1 198_0 } » 266	#0,76#
{ 77_1 93_1 108_1 124_1 196_0 } » 266	#0,77#
{ 77_1 93_1 109_1 124_1 196_0 } » 266	#0,77#
{ 93_1 108_1 109_1 124_1 196_0 } » 266	#0,77#
{ 93_1 108_1 109_1 124_1 197_0 } » 266	#0,76#
{ 93_1 109_1 124_1 140_1 196_0 } » 266	#0,77#
{ 93_1 109_1 124_1 140_1 197_0 } » 266	#0,76#

Referências Bibliográficas

[Agrawal1993] Agrawal, Rakesh, Imielinski, T., Swami, A., *Mining Association Rules between Sets of Items in Large Databases*, SIGMOD 5/93, 207-216, Washington, USA, 1993.

[Agrawal1994] Agrawal, Rakesh; Srikant, Ramakrishnan, *Fast Algorithms for Mining Association Rules*, In: 20th VLDB Conference, 487-498, Santiago, Chile, 1994.

[Anibal2000] Anibal, Angel; Mongiovi, Giuseppe; Sampaio, Marcus C., *Utilização de Algoritmos de Classificação para o Reconhecimento de Caracteres Manuscritos*, Relatório Técnico do Departamento de Sistemas e Computação, 2000.

[Baltazar2000] Baltazar Henry, *NBA coaches' latest weapon: Data mining*, PC Week Labas, March, 2000, http://www.zdnet.com/pcweek/stories/news/0,4153,2454689,00.html

[Cabena1997] Cabena, Peter; et al, *Discovering Data Mining from Concept to Implementation*, Prentice Hall PTR, New Jersey, USA, 1997.

[Carvalho1999a] Carvalho, Juliano Varella de; Sampaio, Marcus Costa; Mongiovi, Giuseppe, *Utilizando Técnicas de Data Mining para o Recohecimento de Caracteres Manuscritos*, Proposta de Dissertação de Mestrado, Universidade Federal da Paraíba, Brasil, Fevereiro, 1999.

[Carvalho1999b] Carvalho, Juliano Varella de; Sampaio, Marcus Costa; Mongiovi, Giuseppe, *Utilização de Técnicas de Data Mining para o Reconhecimento de Caracteres Manuscritos*, In: XIV Simpósio Brasileiro de Banco de Dados, 235-249, Florianópolis, Santa Catarina, Brasil, 1999.

[Cavalcanti1998] Cavalcanti, José Homero F., *Sistemas Inteligentes, Redes Neurais Artificiais e Lógica Fuzzy*, Apostila de Aula, Universidade Federal da Paraíba, Campina Grande, 1998.

[Correia2000a] Correia, Suzete; Carvalho, João Marques de, *Optimizing the Recognition Rates of na Unconstrained Handwritten Numerals Using Biorthogonal Spline Wavelets*, In: International Conference of Pattern Recognition, Barcelona, Spain, Setembro, 2000.

[Correia2000b] Correia, Suzete; Carvalho, João Marques de, *Handwritten Numerical Characters Recognition Using Biorthogonal Spline Wavelets*, In: International Conference of Pattern Recognition, Barcelona, Spain, Setembro, 2000.

[Cheung1996] Cheung, David W.; et al, Maintenance of Discovered Association Rules in Large Databases: Na Incremental Updating Technique, ICDE 1996: 106-114.

[Damsels1996] DAMSELS – *Data Mining of Singapore Lifestyle Survey Data (DaMSeLS)*, Fevereiro, 1996, http://www.jsaic.krdl.org.sg/projects/datamining.html

[Dbms1998] DBMS Tools & Strategies for IS Professionals, August, 1998, Vol. 11, number 9, pag 52-61, http://www.dbmsmag.com/9807m00.html

[Fayyad1996] Fayyad, Usama M.; *et al*, *Advances in Knowledge Discovery and Data Mining*, MIT Press, Massachussetts, USA, 1996.

[Fayyad1998] Fayyad Usama B., *Data Mining and Knowledge Discovery*, Na International Journal, Volume 1, Issue 1, 1998. http://www.research.microsoft.com/research/datamine/vol1-1/editorial3.htm

[Feldens1997a] Feldens, Miguel A.; Castilho, José Mauro Volkmer de, *Engenharia da Descoberta de Conhecimento em Bases de Dados: Estudo e Aplicação na Área de Saúde*: Dissertação de mestrado, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil, 1997.

[Feldens1997b] Feldens, Miguel A.; Moraes, Rodrigo L. de; Pavan, Altino; *Data Mining na Gestão Hospitalar*, 1999. http://vidaconsultores.com.br/infovida/DATAM.htn

[Freitas1998a] Freitas, Alex A., *A Mult-Ciuteria Aproach for the of Rule Interestingness*, Data Mining, (Proc. Int. Conf, Rio de Janeiro, Brazil, sep. 1998), 7-20. WIT Press, 1998.

[Freitas1998b] Freitas, Alex A., *Data Mining (Tutorial)*, In: XIII Simpósio Brasileiro de Banco de Dados, Maringá, Brasil, 1998.

[Freitas1998c] Freitas, Alex A., *On Rule Interestingness Measures,* In: ES98, the 18th Annual International Conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, December 1998.

[Gader1996] Gader, Paul D.; Khabou, Mohamed Ali, *Automatic Feature Generation for Handwritten Digit Recognition*, In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 18, no 12, 1256-1261, December, 1996.

[Gomes1996] Gomes, N. R., *Algoritmo Seqüencial para Reconhecimento de Numerais Manuscritos Desconectados Utilizando Redes Neurais*, Dissertação de Mestrado, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica, Departamento de Computação, 1996.

[Gomes1994] Gomes, H. M., *Investigação de Técnicas Automáticas para o Reconhecimento Off-line de Assinaturas*. Dissertação de Mestrado, Universidade Federal de Pernambuco, 1994.

[Gomes1988] Gomes, F. A. C.; Mongiovi, G.; Silva, H. M., *APREND – Um Sistema de Aquisição de Conhecimento a Partir de Exemplos*, 14ª Conferência Latinoamericana de Informática, Buenos Aires, Argentina, 1989.

[Gonzales1992] Gonzales, R. C.; Woods, R. E., *Digital Image Processing*, Addison-Wesley, 1992.

[Grandidier1999] Grandidier F.; et al, *Influence of Word Length on Handwritting Recognition*, In: 5ª International Conference on Document Analysis and Recognition, September, Bangalore, India**,** 1999.

[Guyon1996] Guyon, Isabelle; et al, *Data Sets For OCR and Document Image Undestanding Research*, Handbook on Optical Character Recognition and Document Image Analysis, World Scientific Publishing Company, 1996.

[Han1995] Han, Jiawei; Fu, Yongjian, *Discovering of Multiple-Level Association Rules from Large Databases*, Proc. Of 21st VLDB Conference, Zurique, Suíça, 1995.

[Han1999] Han, J., *Characteristic Rules*, to appear in W. Kloesgen and J. Zytkow (eds.), *Handbook of Data Mining and Knowledge Discovery*, Oxford University Press, 1999. http://db.cs.sfu.ca/sections/publication/kdd/kdd.html

[Hanmandlu1999] Hanmandlu, M.; Mohan, M. K. R.; Kumar H., *Neural Based Handwritten Character Recognition*, In: 5ª International Conference on Document Analysis and Recognition, September**,** Bangalore, India, 1999.

[Hertz1990] Hertz, J.; Krogh, A.; Palmer, R. G., Introduction to the Theory of Neural Computation, Adison-Wesley, Publishing Company, Califórnia, 1990.

[Indurkhia1998] Indurkhia, Nitin; Weiss, Sholom M., *Predictive Data Mining a practical guide*, Morgan Kaufmann Publishers, San Francisco, Califórnia, USA, 1998.

[Kaufman1990] Kaufman, Leonard; Rousseeuw, Peter J., *Finding Groups In Data : Na Introduction to cluster analysis*, Wiley Interscience Publication, 1990.

[Kimball1998] Kimball, Ralph, *Data Warehouse Toolkit – Técnicas para a Construção de Data Warehouses Dimensionais*, Makron Books, 1998.

[Korth1999] Korth, Henry F.; Silberschatz, Abraham; Sudarshan S., *Sistema de Banco de Dados,* Makron Books, São Paulo, 1999.

[Koundour1998] Koundourakis G.; Saraee M.; Theodoulidis C., *Data Mining in Temporal Databases*, Panhellenic Conference on New Information Technology, Athens, Greece, 8-10, October, 1998.

[Kovacs1986] Kovács, Z. L., *Redes Neurais Artificiais – Fundamentos e Aplicações*, Edição acadêmica, São Paulo, 1996.

[Krose1995] Kröse, B. J. A.; Smagt, P. P. Van der, An Introduction to Neural Networks, Seventh Edition, December, 1995.

[LeCun1995] LeCun, Y.; et al, *Learning algorithms for classification: A comparison on handwritten digit recognition*. In: J. H. Oh, C. Kwon, and S. Cho, editors, Neural Networks: The Statistical Mechanics Perspective, pages 261-276. World Scientific, 1995.

[Lee1999] Lee, L. L.; Gomes, N. R., *Automatic Classification of Deformed HandWritten Numeral Characters*, In: 5ª International Conference on Document Analysis and Recognition, September**,** Bangalore, India, 1999.

[Liu1998] Liu, Bing; Hsu, Wynne; Ma, Yiming, *Integrating Classification and Association Rule Mining*, Proc. of the 4th International Conference on Knowledge Discovery & Data Mining, 1998.

[Lixin1999] Lixin, W.; Jing, H.; Ruwei, D., *An Integrated Pattern Recognition System and Its Application*, In: 5ª International Conference on Document Analysis and Recognition, September**,** Bangalore, India, 1999.

[Mongiovi1992] Mongiovi, Giuseppe, *Aquisição Automática de Conhecimento a partir de Exemplos: Uma Abordagem Pragmática*: Tese de Concurso Público para Prof. Titular, Universidade Federal da Paraíba, Campina Grande, 1992.

[Mongiovi1995] Mongiovi, Giuseppe, *Uso de Relevância Semântica na Melhoria da Qualidade dos Resultados Gerados pelos Métodos Indutivos de Aquisição de Conhecimento a partir de Exemplos*: Tese de Doutorado, Universidade Federal da Paraíba, Campina Grande, 1995.

[Mongiovi1998] Mongiovi, Giuseppe, *Data mining*: Notas de aula, Universidade Federal da Paraíba, Campina Grande, 1998.

[Montreal1999] *Bank of Montreal Mines Knowledge From Data*, September, 1999. http://www.techweb.com/wire/story/TWB19990920S0008

[Moxon1996] Moxon, Bruce, *Defining Data Mining*, DBMS Data Warehouse Supplement, August, 1996. http://www.dbmsmag.com/9608d53.html

[Ng1998] Ng Raymond T.; et al, *Exploratory Mining and Pruning Optimizations of Constrained Association Rules,* SIGMOD Conference 1998: 13-24.

[Quinlan1993] Quinlan, J. R., *C4.5: Programs for machine learning*, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[Rummelhart1986] Rummelhart, D.; Hintom, Williams, *Learning Internal Representations by Error Propagation*, Parallel Distributed Processing, MIT Press, Cambridge, 1986.

[Savasere1995] Savasere, Ashok; et al, *An Efficient Algorithm for Mining Association Rules in Large Databases*, In: 21st VLDB Conference, 432-444, Zurich, Swizerland, 1995.

[Scott1998] Scott R.I., et al, *Experiences of using Data Mining in a Banking Application*, IMACS'98, Athens, Greece, September, 1998.

[See5_1999] Rulequest Research 1999**,** *Data Mining Tools See5 and C5.0*, http://www.rulequest.com/see5-info.html, Last updated December 1999.

[SipinaPro1998] Sipina-pro, EPCAD - Entrez dans I``ere de la decision, 1998. http://www.epcad.fr/sipina/indexE.html

[Srikant1996] Srikant, Ramakrishnan; Agrawal Rakesh, *Mining Quantitative Association Rules in Large Relational Tables*, SIGMOD 6/96, 1-12, Montreal, Canadá, 1996.

[Subrahmonia1996] Subrahmonia, Jayashree; et al, *Writer Dependent Recognition of On-Line Uncronstained Handwriting*, ICASSP'96, 1996. http://www.research.ibm.com/handwriting/papers.html/icassp96_writerdep.ps.gz

[Suen1992] Suen, C. Y.; et al, *Computer Recognition of Unconstrained Handwritten Numerals* – Proceeding of the IEEE, 1162-1180, july, 1992.

[Suen1980] Suen, C. Y.; Berthod, M; Mori S., *Automatic Recognition of Handprinted Characters – The State of the Art*, Proceeding of IEEE, april, 1980.

[Tafner1995] Tafner, Malcon A.; Xerez, Marcos de; Rodrigues Filho, Ilson W., *Redes Neurais Artificiais – Introdução e Princípios de Neurocomputação*, Editora EKO, 1995.

[Veloso1998] Veloso, Luciana R., *Reconhecimento de Caracteres Numéricos Manuscritos*: Dissertação de Mestrado, Universidade Federal da Paraíba, Campina Grande, 1998.

[Veloso1999] Veloso, Luciana R.; Carvalho, João Marques de, *Neural versus Syntatic Recognition of Handwritten Numerals*, In: 5ª International Conference on Document Analysis and Recognition, September**,** Bangalore, India, 1999.

[Wüthrich1994] Wüthrich, B., *Knowledge Discovery in Databases*, Kowloon, Hong Kong: The Hong Kong University of Science and Technology, 1994.

[Yen1996] Yen, Show-Jane; Chen, Arbee L. P., *An Efficient Approach to Discovering Knowledge from Large Databases*, PDIS 1996: 8-18.

[Yuceer1993] Yuceer, C.; Oflaser, K., *A Rotation, Scaling and Translaction Invariant Pattern Classification System*, Pattern Recognition, 26(5): 687-710, 1993.