LUIZ VINICIUS MARRA RIBAS

USO DE REDES NEURAIS NA PREVISÃO DE DESVIOS EM ARQUITETURAS SUPERESCALARES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática pelo Curso de Pós-Graduação em Informática, do Setor de Ciências Exatas da Universidade Federal do Paraná, em convênio com o Departamento de Informática da Universidade Estadual de Maringá.

Orientador: Prof. Dr. Maurício F. Figueiredo

Orientador: Prof. Dr. Ronaldo A. L. Gonçalves

CURITIBA

2003

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno *Luiz Vinícius Marra Ribas* , avaliamos o trabalho intitulado, *"Uso de Redes Neurais na Previsão de Desvios em Arquiteturas Superescalares"*, cuja defesa foi realizada no dia 29 de agosto de 2003, às quatorze horas, no Anfiteatro A do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato. (Convênio número 279-00/UFPR de Pós-Graduação entre a UFPR e a UEM - ref. UEM número 1331/2000-UEM).

Curitiba, 29 de agosto de 2003.

Prof. Dr. Maurício Fernandes Figueiredo
DIN/UEM – Orientador

Prof. Dr. Ronaldo Augusto de Lara Gonçalves
DIN/UEM – Orientador

Prof. Dr. Philippe Oliver Alexandre Navaux
INF/UFRGS – Membro Externo

Prof. Dr. Roberto André Hexsel
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

A **Deus**

Por todas as conquistas obtidas e projetos realizados.

Ao Prof. Dr. **Maurício F. Figueiredo**

Por sua orientação a este trabalho.

Ao Prof. Dr. **Ronaldo A. L. Gonçalves**

Por sua orientação, palavras de incentivo e motivação.

A minha esposa **Adriana**

Por seu incentivo, compreensão e apoio incondicional.

Aos meus **pais e irmãos**

Por seus incentivos e votos de sucesso.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE EQUAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APACHE	-	Servidor de Páginas WEB
ASS	-	Designação de Número de Associatividade da Tabela do Previsor
BTB	-	Branch Target Buffer
D-Cache	-	Área de Armazenamento de Dados – Arquitetura Superescalar
DAG	-	Grafos de Dependências
DNE	-	Previsor Perceptron de Dois Níveis por Endereço
DNT	-	Previsor Perceptron de Dois Níveis por tipo
DSV	-	Designação de Desvio Avaliado no Previsor
END	-	Previsor Perceptron por Endereço de Desvio
EST-RES	-	Estações de Reserva – Arquitetura Superescalar
EXEC	-	Unidades Funcionais de Execução – Arquitetura Superescalar
FILA-REORD	-	Fila de Reordenação de Instruções – Arquitetura Superescalar
GAg	-	Modelo de Previsor de Desvios Adaptativo de Dois Níveis
Ghz	-	Gigahertz
I-Cache	-	Área de Armazenamento de Instruções – Arquitetura Superescalar
I-FILA	-	Fila de Instruções – Arquitetura Superescalar
IBM	-	International Business Machines Corporation – Marca Registrada
ILP	-	Instruction Level Parallelism
jpGraph	-	Biblioteca Gráfica para Linguagem PHP
LG_AND	-	Previsor com Organização LOCAL e GLOBAL combinada por E-Lógico
LG_OR	-	Previsor com Organização LOCAL e GLOBAL combinada por OU-Lógico
LIN	-	Designação de Número de Linhas da Tabela do Previsor
LRU	-	Least Recently Used
LTB	-	Loop Termination Buffer
MLP	-	Multi-Layer Perceptron
MOD	-	Designação de Modelo de Previsor
MySql	-	Banco de Dados SQL
ORG	-	Designação de Organização de Previsor
PAg	-	Modelo de Previsor de Desvios Adaptativo de Dois Níveis

PAp	-	Modelo de Previsor de Desvios Adaptativo de Dois Níveis
PC	-	Contador de Programa (Program Counter)
PHP	-	Linguagem de Programação para WEB
PRG	-	Designação de Programa Avaliado no Previsor
RAW	-	Read After Write
SPEC	-	Standard Performance Evaluation Corporation – Pacote de Programas de Avaliação (Benchmarks)
TIP	-	Previsor Perceptron por Tipo de Desvio
ULA	-	Unidade Lógica e Aritmética
UNI	-	Previsor com Perceptron Único
VLR	-	Designação de Valor de Porcentagem de Acerto do Previsor
WAR	-	Write After Read
WAW	-	Write After Write
WEB	-	Página de Informações da Internet

RESUMO

Os processadores comerciais atuais usam técnicas agressivas para a extração do paralelismo em nível de instrução com o objetivo de atingir maior desempenho. Uma destas técnicas, a previsão de desvios, é usada para antecipar a busca de instruções, manter contínuo o fluxo de instruções no pipeline e aumentar as chances de paralelização de instruções. A maioria dos previsores de desvios utiliza algoritmos triviais aplicados a informações comportamentais sobre os desvios contidas em tabelas atualizadas dinamicamente. Uma nova abordagem tem sido investigada recentemente visando substituir estes algoritmos triviais por redes neurais, com o objetivo de prover maior inteligência aos previsores. Os trabalhos realizados com previsores deste tipo ainda são introdutórios e por isso estudos mais profundos devem ser realizados.

O presente trabalho analisa o desempenho da previsão de desvios baseada em rede neural do tipo Perceptron para cinco diferentes modelos de previsores propostos. O modelo UNI realiza a previsão através de um único Perceptron para todas as instruções dos programas. Os modelos TIP e END utilizam vários Perceptrons em tabelas acessadas pelo tipo ou endereço das instruções de desvios, respectivamente. Os modelos DNT e DNE possuem o mecanismo de previsão implementado em dois níveis e são extensões dos respectivos modelos em um nível (TIP e END).

Estes modelos foram avaliados sob diferentes tamanhos de históricos de desvios (2 a 64), diferentes números de linhas (64 a 1024) e graus de associatividade (1 a 16) da tabela de Perceptrons, incluindo diferentes tipos de organização do previsor, LOCAL e GLOBAL, definindo a localização do histórico de desvios nos Perceptrons; e LG_AND e LG_OR, que combinam as saídas de LOCAL e GLOBAL segundo sua função lógica.

As avaliações mostram que os previsores de dois níveis apresentam melhores resultados que os correspondentes de um nível, que o aumento das linhas da tabela para a mesma associatividade apresenta um ganho de desempenho e que há aumento de desempenho com o aumento da associatividade para o mesmo número de linhas da tabela.

Os melhores resultados obtidos foram para programas de ponto flutuante e desvios para frente. As organizações LG_AND e LG_OR não apresentam contribuições representativas na previsão de desvios, ficando os melhores resultados para LOCAL e GLOBAL.

De uma forma geral, o presente trabalho mostrou que o uso do Perceptron na previsão de desvio é atrativo e os resultados são equivalentes àqueles obtidos em trabalhos correlatos.

ABSTRACT

Nowadays, all commercial processors use aggressive techniques to extract high instruction level parallelism and to improve performance. One of these techniques, the branch prediction, is used to anticipate the instruction fetch, to maintain a continuous instruction stream in the pipeline and to increase the chances of obtaining instructions to be executed in parallel. Most of predictors use trivial algorithms, which work with behavior information of branches stored in tables that are updated dynamically. A new approach has been investigated to replace these trivial algorithms to neural networks, with the aim to add more intelligence to branch predictors. Work on this kind of predictor is current and new studies must be realized.

This work analyses the performance of branch prediction Perceptron under five different proposed models. The UNI model makes prediction with a single Perceptron that receives all branch instructions of the programs. The models TIP and END use several Perceptrons arranged in tables indexed by branch type or branch address, respectively. The models DNT and DNE have a two-level prediction mechanism and are extensions of the models TIP and AND, respectively.

All prediction models were evaluated under different configurations for branch history sizes (2 up to 64), number of lines (64 up to 1024) and associativity (1 up to 16) of the Perceptron tables. In addition, different types of predictor organization where LOCAL and GLOBAL, that define the branch history distribution in analyzed Perceptron tables, and LG_AND and LG_OR, that combine LOCAL and GLOBAL outputs.

The results show that two-level predictors have better performance than their one-level counterparts. Also the increase of table size using fixed associativity improves predictor performance. Moreover, the performance of the predictor increases when enlarging associativity to the same table line number.

Floating-point programs have better performance than integer programs. Forward branches have better performance than backward branches. The performance attained using LG_AND and LG_OR organizations is worst than with LOCAL and GLOBAL organizations.

In a general way, this work showed that the use of Perceptron in branch prediction is viable and the results are close to ones from other techniques.

1. Introdução

No processo de evolução dos processadores, grande parte do aumento do desempenho alcançado foi devido a alterações arquiteturais. Os processadores mais antigos executavam menos de uma instrução por ciclo, sub-utilizando os recursos de hardware. Além disso, as instruções eram executadas seqüencialmente e na ordem natural do programa. Posteriormente, com o advento das arquiteturas *pipelined*, os processadores passaram a compartilhar o hardware em diferentes fases do ciclo de instrução, permitindo assim a execução de várias instruções de forma parcialmente paralela.

Contudo, devido ao aumento contínuo da complexidade das aplicações, a necessidade de desenvolver processadores mais eficientes sempre foi uma constante. Ao mesmo tempo, a grande quantidade de códigos seqüenciais já existentes conduziu as pesquisas no sentido de desenvolver novas arquiteturas que fossem capazes de obter alto desempenho sem interferência do programador. Com isso, em uma das tendências arquiteturais, o *pipeline* foi replicado com o objetivo de explorar paralelismo em nível de instruções, conhecido como ILP (*Instruction Level Parallelism*), que é extraído dinamicamente pelo hardware, caracterizando as arquiteturas superescalares.

O ILP refere-se ao número de instruções de um programa que pode ser executado em paralelo a cada ciclo [12, 48, 50, 52] e tem sido usado como medida de desempenho no processo de avaliação arquitetural. Este paralelismo pode ser alcançado quando três condições são satisfeitas: 1) existem instruções independentes dentro do programa; 2) existe mecanismo em hardware capaz de detectar esta independência e 3) existem recursos de hardware disponíveis que possam ser usados para a execução das instruções independentes. Os processadores superescalares utilizam mecanismos sofisticados que permitem o tratamento destas três condições, definindo um modelo de execução capaz de explorar paralelismo nos diferentes estágios do *pipeline* [18, 48].

Uma das condições necessárias para a detecção eficiente de instruções independentes é a existência de um fluxo contínuo de instruções dentro do *pipeline*. Entretanto, este fluxo de instruções pode ser quebrado pelas instruções de desvios presentes no código, que mudam a direção do caminho de execução e causam

atrasos na busca das próximas instruções, com uma freqüência média de uma instrução de desvios a cada cinco instruções do programa [43]. Esta situação fica pior quando os endereços alvos das instruções de desvios não são conhecidos, e pior ainda, quando estas instruções de desvios são condicionais e os resultados das condições também não estão resolvidos. A previsão de desvios também impacta no desempenho do processador, já que uma previsão de desvio incorreta conduz ao esvaziamento do pipeline, ocasionando um atraso no processamento e causando um impacto negativo no desempenho.

Para aliviar esta problemática, um dos mecanismos utilizados nos processadores superescalares para manter seu alto desempenho é a previsão de desvios, que pré-determina com a maior probabilidade possível o caminho a seguir pela instrução de desvio antes que ela tenha sido executada, e que pode ser não-tomado, aquele na seqüência natural do código – sem salto, ou tomado, aquele que é desviado para outra posição de memória – com salto. A previsão de desvios pode ocorrer durante a fase de busca, durante a fase de decodificação ou de forma combinada nestes dois estágios do *pipeline* superescalar [47, 48].

Existem diversas técnicas de previsão de desvios, tanto estáticas quanto dinâmicas. Estas técnicas variam desde a simples opção de se escolher sempre um caminho do desvio, o caminho tomado ou não-tomado [47], passando por estruturas de tabelas simples ou hierarquizadas contendo contadores ou históricos de desvios [6, 30, 56, 57] chegando a novas alternativas baseadas em fórmulas Booleanas [22] e Redes Neurais [23, 24, 29, 33].

Atualmente, diversos grupos de pesquisa se dedicam ao estudo e análise das técnicas existentes de previsão de desvios, procurando melhorar o funcionamento das mesmas ou desenvolver mecanismos alternativos que possam ser mais eficientes e preferencialmente mais simples quanto à implementação em hardware. É conhecida a perda de desempenho causado pelos erros de previsão [56, 57], e assim, qualquer indício de aumento no desempenho dos processadores superescalares que possa surgir pela melhoria na previsão de desvios justifica os esforços nesta área de pesquisa, mesmo que isto possa aumentar o custo da implementação em hardware. Muitos trabalhos que hoje são teóricos e até impossíveis de serem implementados, no futuro, com a evolução tecnológica na confecção de circuitos cada vez menores, estes poderão ser concretizados.

O Perceptron é uma rede neural simples, normalmente constituído por apenas um neurônio, que permite classificar áreas linearmente separáveis e fazer previsões de valores futuros em séries baseadas no tempo [9, 13, 26, 53]. Estudos deste tipo de rede neural como previsor de desvios foram realizados obtendo resultados interessantes [23, 24, 29, 33]. Pelo fato da capacidade de aprendizado, capacidade de fazer previsão, simplicidade de implementação em hardware e resultados satisfatórios preliminarmente obtidos justifica-se o emprego do Perceptron na previsão de desvios. Ainda assim, este é um campo recente de pesquisa e estudos mais profundos se fazem necessários.

O objetivo deste trabalho é de propor e avaliar modelos de previsão de desvios baseados em redes neurais simples, notadamente o Perceptron, de forma a entender o seu comportamento; comparar diferentes métodos de cálculo de previsão em diferentes situações; e avaliar a efetividade de seu uso na previsão de desvios, determinando se há ou não benefícios na previsão com o emprego destes modelos.

Este objetivo é alcançado nas diversas fases do trabalho; inicialmente através do estudo dos conceitos sobre arquitetura superescalar, técnicas de previsão de desvios e conceitos de redes neurais, enfocando a rede neural do tipo Perceptron; em seguida, foram iniciados os experimentos através da implementação de um sistema, e uma variante deste, contendo um previsor baseado no Perceptron, e foram aplicados rastros de execução de *benchmarks* sobre estes sistemas e a efetividade de previsão foi avaliada sobre os seus resultados; através dos resultados obtidos destes sistemas preliminares foram concebidos modelos mais arrojados do previsor, envolvendo conceitos de algumas técnicas de previsão de desvios estudadas e que foram implementadas sobre um simulador de arquitetura superescalar amplamente utilizado; e finalmente, através da execução de *benchmarks* sobre os previsores implementados (modelos UNI, TIP, END, DNT e DNT, descritos posteriormente), foi feita a análise e avaliação de desempenho sobre os resultados obtidos.

O texto está organizado da forma a seguir descrita. O capítulo 2 introduz conceitos básicos sobre paralelismo e processamento superescalar, descreve diversas estruturas utilizadas nas arquiteturas superescalares, suas características e funcionalidades. O capítulo 3 descreve diversas técnicas de previsão de desvios, sua classificação e funcionamento e trabalhos relacionados. No capítulo 4 é definido

o conceito de rede neural, enfocando uma rede neural simples e seu funcionamento, o Perceptron, relatando alguns trabalhos relacionados. O capítulo 5 descreve os trabalhos preliminares sobre um previsor baseado no Perceptron e seus resultados, e a descrição dos modelos de previsor de desvios propostos. O capítulo 6 descreve o simulador sim-bpred utilizado para desenvolvimento e simulação de previsores, as alterações efetuadas sobre o mesmo, descreve os aspectos de implementação dos novos modelos de previsor baseado no Perceptron, como foram feitas as simulações e como foram obtidos os resultados. No capítulo 7 estão os resultados obtidos de cada modelo proposto e sua análise. Finalmente, o capítulo 8 apresenta as conclusões sobre o trabalho e possíveis trabalhos futuros.

2. Arquiteturas Superescalares

A capacidade de execução paralela de instruções e o desempenho dos processadores aumentaram no decorrer do tempo, de acordo com a evolução arquitetural. Esta seqüência evolutiva partiu da simples execução seqüencial de uma única instrução por vez, passando pelas abordagens de *pipeline* e *superpipeline*, chegando à tecnologia de superescalar. Embora possa ser vista desta forma, esta seqüência evolutiva não seguiu nesta ordem de forma estrita.

O conceito de *pipeline* consiste na execução de uma instrução dividida em passos subseqüentes, onde cada parte é executada por um estágio dedicado do hardware, podendo todos estes executar em paralelo. Este modelo assemelha-se a uma linha de produção, onde a matéria prima sofre, ao longo do tempo, alterações por diferentes operações e sai na forma de produto final. É interessante observar que a cada instante tem-se somente uma instrução por vez entrando no *pipeline*, contudo tem-se mais de uma instrução sendo trabalhada ao mesmo tempo em diferentes estágios. Estes conceitos podem ser observados na Figura 1, que apresenta um *pipeline* de cinco estágios e a representação do fluxo de instruções sobre este *pipeline*.

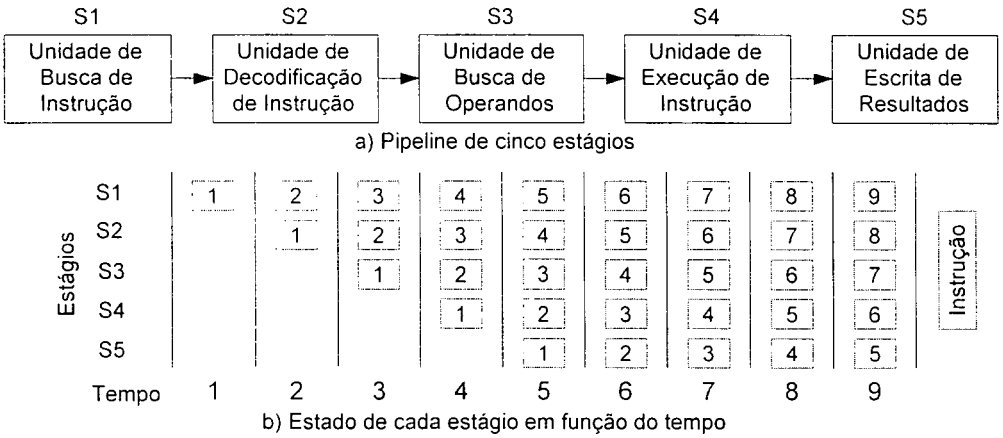


Figura 1 – (a) estágios de um *pipeline*; (b) cada estágio em função do tempo

De forma simples, podemos dizer que o *superpipeline* é o *pipeline* com um número maior de estágios, sendo cada qual executado em um tempo de ciclo menor. Muitos estágios do *superpipeline* realizam tarefas que requerem menos que a metade de um ciclo de processador e mais de um estágio do *pipeline* são executados em cada ciclo, divididos em partes não sobrepostas e cada uma pode

ser executada num pedaço do ciclo. Usando outra abordagem, o conceito de superescalar consiste da execução em paralelo de várias instruções diferentes e independentes por ciclo.

A Figura 2 mostra a execução de seis instruções (i1 a i6) em um *pipeline* de quatro estágios básicos (B – busca, D – decodificação, E – execução e S – escrita dos resultados) em função do tempo. No topo temos a representação do *pipeline*, a parte mais escura na figura representa a fase de execução.

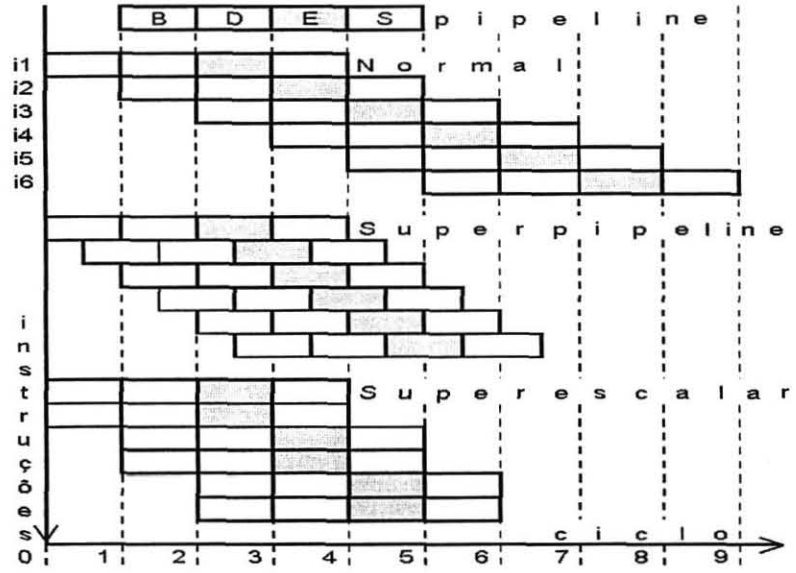


Figura 2 – Modelos *pipeline*, *superpipeline* e superescalar

No primeiro bloco temos o *pipeline* normal usado para comparação. Seu tempo de execução é de nove ciclos para seis instruções. O segundo bloco representa um *superpipeline* de grau dois, ou seja, dois estágios do *pipeline* por ciclo. Seu tempo de execução é de 6,5 ciclos para seis instruções. O último bloco exemplifica a execução superescalar: duas instâncias de cada estágio executam em paralelo. O tempo de execução é de seis ciclos para seis instruções. Tanto o *superpipeline* quanto o superescalar têm o mesmo número de instruções por ciclo, mas o primeiro perde em desempenho no início de cada execução e na ocorrência de desvios. Para diferentes profundidades e largura de *pipeline*, este modelo de execução apresenta-se com a mesma característica observada na Figura 2, diferentes estágios executados em seqüência e múltiplas instâncias do mesmo estágio executando ao mesmo tempo.

2.1. Modelo de Execução e Organização Superescalar

O modelo de execução superescalar, identificado na maioria dos processadores superescalares, que pode ser representado na forma apresentada na Figura 3. O *pipeline* superescalar consiste de múltiplos estágios em que cada um pode manipular múltiplas instruções simultaneamente. A representação do *pipeline* de um processador superescalar com cinco unidades funcionais pode ser visto na Figura 4.

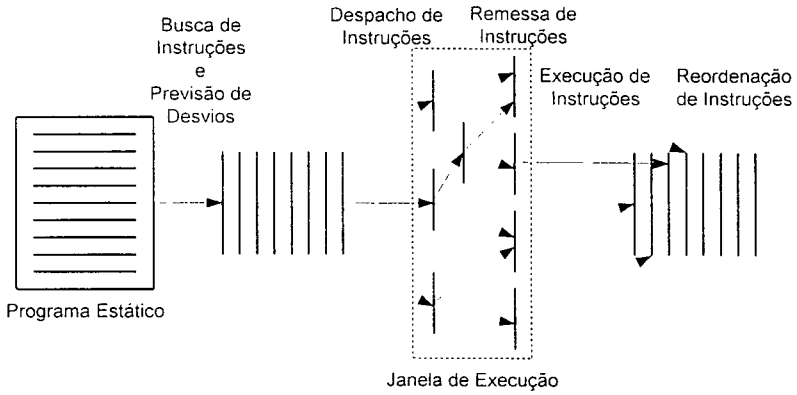


Figura 3 – Modelo de execução usado na maioria dos processadores superescalares

As instruções de desvios condicionam a seqüência de execução e podem parar a alimentação do *pipeline* se a previsão não for feita ou se a previsão for feita incorretamente. A busca de instruções a partir de um programa estático com instruções seqüenciais é feita, incluindo previsão de desvios, e forma um fluxo de instruções dinâmicas. As instruções são então despachadas para a janela de execução (mecanismo onde são identificadas e possivelmente eliminadas as dependências entre as instruções), e são parcialmente ordenadas. Estas instruções são lançadas na ordem definida pelas dependências entre as instruções e pelos recursos disponíveis no momento da execução. Após a execução, as instruções são colocadas de volta na ordem original do programa.

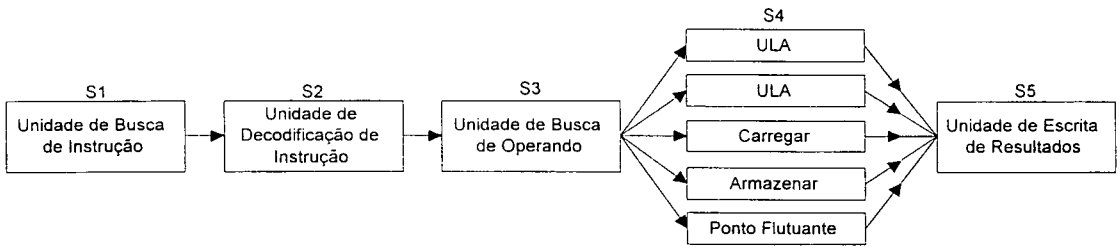


Figura 4 – Processador superescalar com cinco unidades funcionais

Embora as instruções sejam executadas numa ordem diferente daquela definida pelo programa seqüencial, o modelo de execução garante a aparência de execução seqüencial das instruções no final da execução deste programa.

A organização do hardware de um processador superescalar típico, apresenta alguns componentes principais e algumas fases básicas dentro de seu *pipeline* de execução. Segundo Gonçalves [18], em um nível mais amplo, destacam-se as seguintes estruturas manipuladas pela arquitetura superescalar: as *caches* de dados e de instruções, para armazenar dados e código, respectivamente; a fila de instruções (ou de busca), onde as instruções buscadas da *cache* são colocadas para posterior decodificação e despacho para as estações de reserva junto às unidades funcionais; a fila de reordenação, utilizada normalmente para controlar a finalização correta das instruções; e o conjunto de registradores, muitas vezes divididos em agrupamentos de registradores, chamados bancos.

As maiores fases desta micro-arquitetura são: busca, decodificação, despacho, remessa, execução e conclusão da instrução. Pode ocorrer a previsão de desvios durante a busca, durante a decodificação ou de forma combinada nestes dois estágios. A Figura 5 representa as fases de uma arquitetura simplificada do *pipeline* superescalar, os componentes envolvidos e o fluxo da informação dentro deste *pipeline* e entre os componentes. Cada fase da arquitetura superescalar possui sua funcionalidade característica e envolve componentes com características específicas para implementar sua execução.

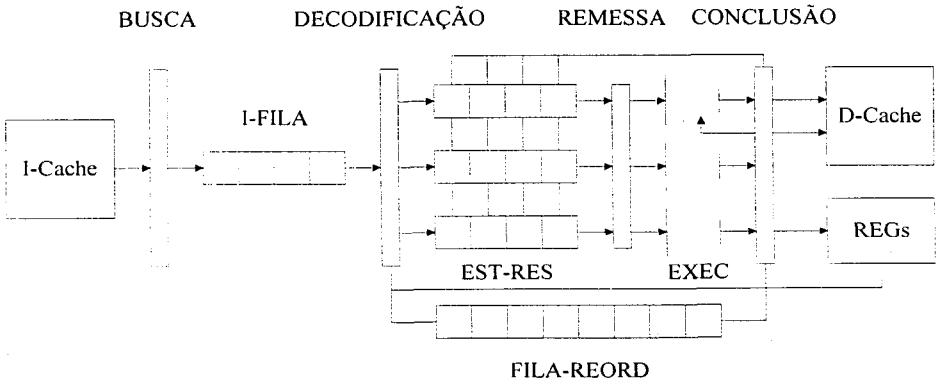


Figura 5 – Arquitetura típica simplificada de um processador superescalar

- **Busca de Instruções**

Esta fase fornece instruções para todo o resto do *pipeline*. As instruções são buscadas na *cache* de instruções e inseridas em uma fila de busca. Pode haver uma pré-busca que antecipa a busca de instruções da *cache* secundária ou memória. Comumente nesta fase é realizada a previsão de desvios, que ocorre em média uma a cada cinco instruções do programa [43], para evitar a parada da busca de instruções no *pipeline*, enquanto se resolve a instrução de desvio, e para permitir a seqüência correta das instruções.

Nesta fase destacam-se: as *caches* de instruções, com instruções buscadas da memória, o mecanismo de busca que permite mais de uma instrução ser buscada ao mesmo tempo, a fila de busca que armazena instruções e serve como suprimento para o *pipeline* e a tabela de previsão de desvios que indica qual caminho a ser tomado e quais as instruções seguintes que serão buscadas.

- **Decodificação e Despacho de Instruções**

A fase de decodificação e despacho de instruções decodifica instruções da fila de instruções e as despacha para as estações de reserva (estruturadas em filas de remessa) junto a diferentes unidades funcionais, para posterior remessa e execução. Normalmente, é nesta fase que os operandos fontes são lidos do banco de registradores da arquitetura, sendo possível controlar as dependências verdadeiras e eliminar as falsas dependências entre instruções, usando alguma variação do método de renomeação de registradores ou do algoritmo de Tomasulo [54]. Durante a decodificação, para cada instrução despachada, uma área é alocada na fila de reordenação, para controlar a finalização precisa das instruções.

- **Remessa ou Lançamento de Instruções**

Das filas de remessa as instruções são remetidas ou lançadas para as unidades funcionais, se já estiverem com todos seus operandos resolvidos. Já nas unidades funcionais, as instruções são executadas em vários ciclos, dependendo do tipo de instrução e podem acessar a memória de dados. No fim da execução os

resultados são enviados para todas as instruções que estão esperando nas filas de remessa e para as respectivas áreas da fila de reordenação.

O desempenho de uma arquitetura superescalar durante a fase de remessa e execução das instruções depende diretamente dos seguintes fatores: topologia das filas de remessa (forma em que as filas encaminham as instruções, em ordem ou fora de ordem), algoritmo de remessa, configuração das unidades funcionais e situações que envolvam acesso à memória.

- **Conclusão das Instruções**

A conclusão de instruções é a fase final do *pipeline* superescalar. Tenta apresentar aparência do modelo de execução seqüencial embora a execução possa ser fora de ordem e especulativa, ocasionadas por instruções independentes e por caminhos seguintes a desvios previstos e ainda não resolvidos, respectivamente. O resultado desta fase modifica o estado lógico do processador (usado para recuperação do estado preciso da máquina em caso de interrupção). A arquitetura superescalar apresenta dois tipos de estados:

- Físico: atualizado conforme as instruções são atualizadas. Não representa o estado correto da arquitetura, devido à possibilidade de execução especulativa e fora de ordem.
- Lógico ou arquitetural: é o estado correto da máquina, estado em que a máquina deveria estar se o programa tivesse sido executado seqüencialmente, no momento da interrupção.

Para um gerenciamento preciso de interrupção, só é permitida uma instrução completar quando todas as anteriores não puderem mais produzir qualquer interrupção. Existem pelo menos duas técnicas bem conhecidas para recuperar o estado da arquitetura, de forma precisa:

- Primeira técnica: os resultados das instruções atualizam imediatamente o estado físico da máquina, no próprio arquivo de registradores da arquitetura. Utiliza uma estrutura auxiliar chamada "tabela de histórico" para armazenar o estado lógico, registrado na estrutura de tempos em tempos, quando da certeza que os resultados não são mais especulativos.

- Segunda técnica: usa a fila de reordenação para recuperar o estado lógico. O estado físico é armazenado diretamente nos campos da fila de reordenação, logo após as instruções serem executadas. O estado lógico, mantido no arquivo de registradores, é atualizado na ordem seqüencial do programa, quando as instruções são removidas da fila de reordenação.

2.2. Fatores Importantes do Desempenho Superescalar

Existem fatores que limitam ou aceleram o desempenho da arquitetura superescalar. A seguir estão descritos os principais limitadores de desempenho dos processadores superescalares e as principais técnicas de aceleração de desempenho neste tipo de arquitetura.

- **Dependências entre Instruções**

Dependências são limitações que ocorrem e restringem o paralelismo. Diferentes instruções podem conflitar pelo uso de recursos (registradores, ULAs, etc) e pode ocorrer a dependência de resultados de algumas instruções em relação a outras, impedindo que instruções sejam encaminhadas para execução em paralelo.

Estas dependências podem ser classificadas em tipos: as chamadas dependências verdadeiras, que são as dependências de dados entre as instruções, e as chamadas falsas dependências ou dependências artificiais que são as dependências de desvios, conflitos de recursos, dependências de saída e antidependências.

- **Dependência de dados:** ocorre quando uma instrução posterior depende dos dados produzidos por uma instrução anterior. Também conhecido como dependência de fluxo, dependência de escrita-leitura ou dependência RAW (*Read After Write hazard*).
- **Dependência de saída:** ocorre quando uma instrução está tentando escrever num registrador que uma instrução prévia ainda não terminou de escrever no mesmo registrador. Também conhecida com dependência WAW (*Write After Write hazard*).

- **Antidependência:** ocorre quando uma instrução está tentando escrever num registrador que uma instrução prévia ainda não terminou de ler. Também conhecida como dependência WAR (*Write After Read hazard*).
- **Dependência de desvio:** ocorre quando instruções posteriores a uma instrução de desvio (tomado ou não-tomado) dependem do resultado produzido pela mesma e não podem ser executadas até o direcionamento do desvio.
- **Conflito de recursos:** o conflito de recursos é a disputa de duas ou mais instruções pelo mesmo recurso (memória, *cache*, via de dados, banco de registradores, unidades funcionais, etc) ao mesmo tempo.

- **Política de Remessa de Instruções**

O termo remeter ou lançar uma instrução refere-se ao processo de iniciar a execução da instrução nas unidades funcionais do processador. Já o termo política de remessa ou lançamento de instrução refere-se ao protocolo usado para lançar a instrução.

A política de remessa gerencia o envio das instruções da fila de remessa para as unidades funcionais e o envio pode ser feito fora de ordem. Esta política pode ser definida como uma verificação, em tempo de execução, pela disponibilidade de dados e recursos [48]. Uma instrução está pronta para executar tão logo seus operandos estejam disponíveis e a unidade funcional também.

As políticas de lançamento de instruções podem ser agrupadas numas das seguintes categorias: lançamento em ordem com finalização em ordem, lançamento em ordem com finalização fora de ordem e lançamento fora de ordem com finalização fora de ordem.

- **Lançamento em ordem com finalização em ordem:** É a política de lançamento mais simples, lançar a instrução na mesma ordem que seria ativada pela execução seqüencial e escrever os resultados na mesma ordem.
- **Lançamento em ordem com finalização fora de ordem:** Com a finalização fora de ordem, qualquer número de instruções pode estar no

estágio de execução ao mesmo tempo, até o máximo do grau de paralelismo da máquina através do uso de todas as unidades funcionais.

- **Lançamento fora de ordem com finalização fora de ordem:** Os estágios de decodificação e execução são separados por uma "janela de instruções". Após a decodificação, a instrução é colocada nesta janela enquanto outras são buscadas e decodificadas. Quando uma unidade funcional torna-se disponível, uma instrução da janela é lançada para a execução, caso não haja conflitos ou dependências.

- **Execução Especulativa**

Execução especulativa é a técnica de executar um trecho de código mesmo antes de se saber se ele vai ser necessário. Esta prática é muito utilizada nos processadores superescalares para manter o *pipeline* cheio e o desempenho de execução.

A execução especulativa pode influenciar no estado lógico da máquina. Uma maneira comum de prevenir que um código especulativo sobrescreva registradores, antes de se saber se ele é desejável, é renomear todos os registradores de destino usados no código especulativo.

- **Fila de Reordenação**

A fila de reordenação garante a reordenação de instruções que foram executadas fora-de-ordem. Esta fila é geralmente implementada segundo o modelo de fila circular com ponteiros de início e fim de fila. Sempre que uma instrução é despachada, uma área para esta instrução é inserida no final desta fila, quando a mesma instrução conclui sua execução, seu resultado é também colocado nesta área. Quando há uma seqüência ininterrupta de instruções concluídas com término normal no início da fila de reordenação, estas instruções podem ser retiradas da fila e os resultados são gravados nos registradores.

Uma instrução incompleta (sem dados) bloqueia o início da fila de reordenação até que seu valor chegue. Claro que, um processador superescalar

precisa ser capaz de colocar novas áreas na fila e tirá-las mais que uma por vez; entretanto, adicionando e removendo áreas novas ainda seguindo a disciplina de fila.

A Figura 6 mostra uma fila de reordenação; áreas são reservadas e liberadas na ordem da fila. Entretanto, os resultados das instruções podem ser colocados na área a qualquer tempo em que elas completem, e os resultados podem ser lidos a qualquer tempo para uso de instruções dependentes.

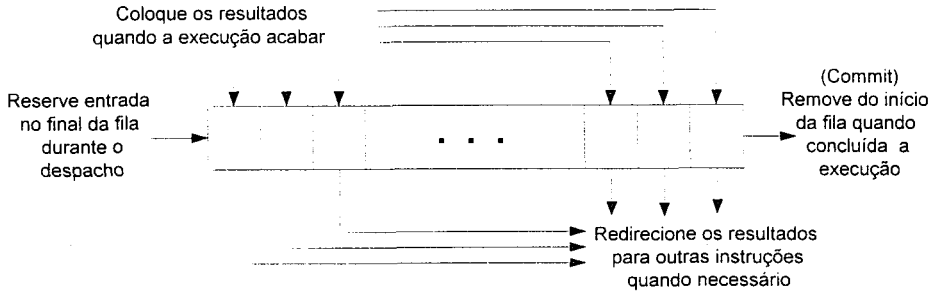


Figura 6 – Representação da fila de reordenação

- **Renomeação de Registradores**

Devido à quantidade limitada de recursos, os processadores armazenam um maior número de valores em um menor número de registradores. Esta característica causa um grande número de conflitos: antidependências e dependências de saída.

O processador superescalar remove estes conflitos através da alocação dinâmica de registradores adicionais, na técnica chamada renomeação de registradores. Em um programa estático, os elementos de armazenamento (registradores e localizações de memória) possuem referências lógicas, e devem ser mapeados (renomeados) para os registradores físicos existentes. Durante este processo, as falsas dependências são eliminadas. Geralmente, o processador aloca um novo registrador para cada novo valor produzido, ou seja, para cada instrução que escreve em um registrador destino.

A Figura 7 mostra um exemplo de renomeação através de DAGs (grafos de dependências) para os trechos de um código antes e após a renomeação. Nestes DAGs, pode-se notar que sem renomeação, as instruções somente podem ser executadas seqüencialmente em função das diversas dependências. Com renomeação, as instruções 1 e 3 podem ser executadas em paralelo. Assim, com o uso desta técnica, um maior paralelismo pode ser extraído dos programas.

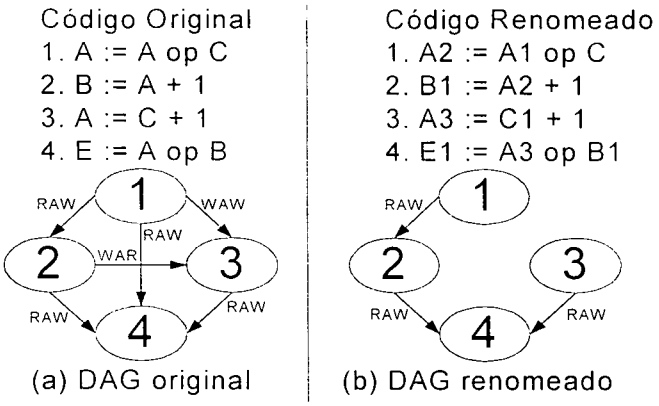


Figura 7 – Relação de dependência entre instruções antes e após a renomeação

Dois métodos de renomeação geralmente são utilizados:

- Uso de um número de Registradores Físicos maior que Lógicos: uma tabela de mapeamento é usada para associar um registrador físico ao registrador lógico corrente (designado pelo compilador). Uma lista de registradores físicos livres é gerenciada para fornecer a renomeação aos operandos de destino das instruções. Conforme os registradores vão sendo liberados, vão retornando à lista de livres. A Figura 8 mostra este método.

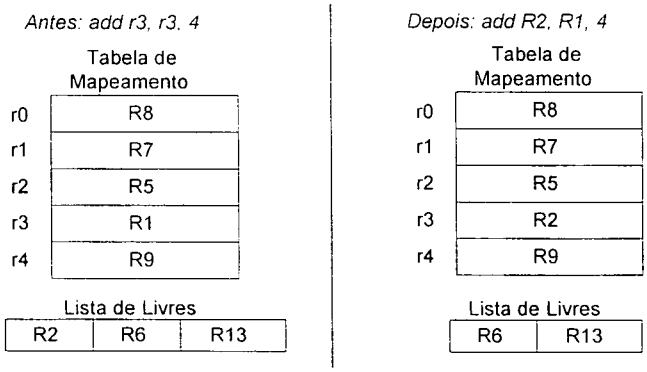


Figura 8 – Renomeação por registradores físicos

- Uso da Fila de Reordenação: é usado um conjunto de registradores físicos de mesmo tamanho que o número de registradores lógicos, com relacionamento um a um. Aproveita-se a fila de reordenação, com uma área para cada instrução ativa e campos para acomodar os operandos destino. Os registradores lógicos são mapeados provisoriamente na fila de reordenação enquanto a instrução ainda é especulativa. Quando a instrução é retirada da fila de reordenação, o registrador lógico é mapeado

diretamente no arquivo físico de registradores. Uma tabela de mapeamento é utilizada para informar o local correto em que se encontra o registrador lógico. Este método esta representado na Figura 9.

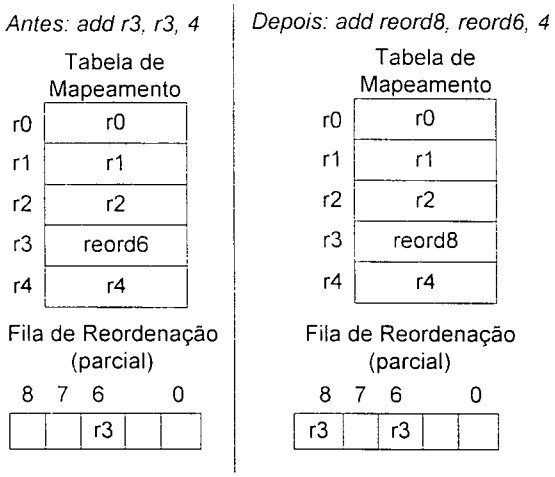


Figura 9 – Renomeação por fila de reordenação

3. O Estado da Arte em Previsão de Desvios

A previsão de desvios é um dos fatores essenciais no desempenho dos processadores superescalares, pois condiciona a sua taxa de desempenho através da previsão do correto caminho que o fluxo de execução deve seguir, evitando o esvaziamento do *pipeline* superescalar quando uma previsão é feita incorretamente, cujo esvaziamento causa um impacto negativo no desempenho final do processamento. Nestes processadores, a execução paralela de diversas instruções determina a capacidade de desempenho do mesmo e algumas instruções podem ser executadas antes mesmo que instruções anteriores tenham sido completadas.

Cada instrução de desvio determina o caminho do fluxo de execução de um programa, direcionando a localização das futuras instruções a serem executadas. Estas futuras instruções somente estarão disponíveis para execução quando a instrução de desvio for completamente executada e seu direcionamento (o caminho a seguir) resolvido e o endereço da próxima instrução conhecido.

As instruções de desvios podem condicionar uma taxa alta ou baixa de lançamento de instruções no *pipeline* superescalar; devido a isto, a direção das futuras instruções a serem buscadas após cada desvio deve ser prevista, e quanto maior for a habilidade do previsor dinamicamente fazer esta previsão, maior será a sua influência na velocidade de execução de um programa e maior o desempenho do processador como um todo. A previsão de direção indica se o resultado da execução do desvio encaminhará o fluxo de execução para instruções anteriores ou posteriores ao desvio.

Além de prever o direcionamento de um desvio a previsão pode determinar o endereço alvo da próxima instrução, antecipando o cálculo deste endereço. A previsão de endereço alvo indica o endereço da instrução a ser executada após o desvio ser completado, acelerando o processamento do *pipeline*, já que este endereço não precisará ser calculado. A Figura 10 apresenta a parte de um código com uma instrução de desvio com os dois possíveis direcionamentos do desvio (caminho tomado ou não-tomado) em ambos os casos em que o salto do desvio é para frente ou para trás. O endereço de destino do desvio a ser computado corresponde ao endereço #d na condição de desvio tomado ou #n+1 quando não-tomado.

A previsão de desvios além de prover maior paralelismo aos processadores superescalares, evita a parada do *pipeline*, possibilitando um fluxo contínuo de instruções, pois a previsão permite o encaminhamento de instruções, possivelmente especulativas, para execução. A previsão também evita as dependências de desvios, já que as instruções posteriores a uma instrução de desvio dependem do resultado produzido por esta e que não podem ser executadas até o conhecimento do caminho a ser seguido pelo desvio.

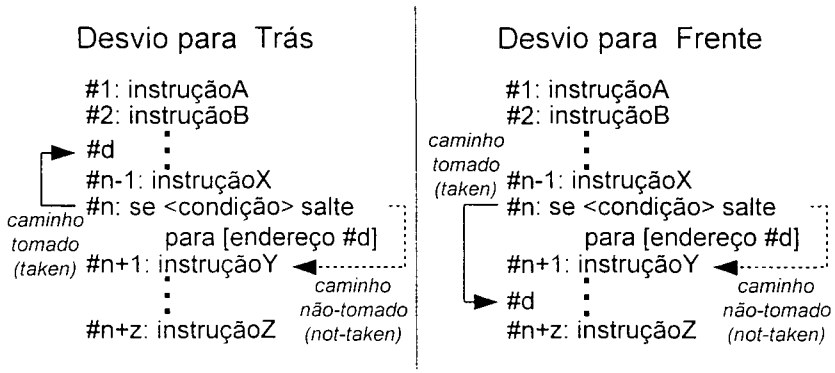


Figura 10 – Direcionamento de desvio para frente ou pra trás

3.1. Técnicas de Previsão

Diversas são as técnicas de previsão de desvios, elaboradas e analisadas por diversos autores. Smith [30] classifica de forma geral estas técnicas em estáticas (a previsão de um desvio é sempre a mesma, toda as vezes que o desvio é encontrado) e dinâmicas (os desvios são dinamicamente previstos durante a execução com base em seu histórico passado). Algumas técnicas estáticas e dinâmicas estudadas por Smith [47] serão descritas a seguir.

3.1.1. Técnicas Estáticas

Destacam-se diferentes estratégias [30] dentre as técnicas de previsão estáticas. A técnica de prever que todos os desvios serão sempre tomados (ou serão sempre não-tomados) normalmente é utilizada como referencial mínimo de comparação. Uma estratégia de previsão de desvios para ser considerada válida deve ter no mínimo desempenho igual a ela. Considerando a técnica em que os

desvios são previstos sempre tomados, pode-se observar que, na instrução de desvio que determina um laço, esta técnica sempre leva a previsões corretas em todas as execuções do desvio, menos ao final da execução do laço.

Um refinamento da técnica anterior é prever que apenas certos tipos de instruções de desvios (definidos pela sua codificação) serão tomados e as demais instruções de desvios serão não-tomados, ou vice-versa [47]. Geralmente esta técnica alcança uma exatidão maior que a anterior, já que o previsor é ajustado pela análise prévia das instruções de desvios dos programas.

Uma outra técnica estática é prever como tomados apenas os desvios que tem direcionamento para endereço de memória anterior ao desvio, e prever não-tomado o desvio no caso inverso [47]. Laços sempre terminam com desvios para endereço anterior, logo a exatidão desta técnica será alta para estes casos. A desvantagem desta técnica, ou outras que fazem uso o endereçamento destino do desvio, é que pode ser necessário, antes que a previsão seja feita, computar ou comparar com o contador de programa o endereço destino.

3.1.2. Técnicas Dinâmicas

Dentre as técnicas de previsão dinâmicas [47] destaca-se a técnica de prever que um desvio seguirá sempre o mesmo caminho da sua última execução; se não tiver sido executado previamente, o desvio é previsto como tomado. Ocasionalmente esta técnica pode ter pior desempenho que a técnica de prever sempre um desvio tomado, já que na instrução de desvio que determina um laço, esta técnica sempre leva a duas previsões incorretas, uma ao final da execução do laço que conduz a um novo direcionamento e outra à próxima decisão a ser feita sobre o mesmo desvio.

Uma outra técnica de previsão dinâmica é a utilização do histórico do desvio [30], armazenado em registradores ou *caches.* Uma possibilidade é manter uma tabela dos últimos *m* mais recentes desvios não-tomados. Se o desvio a ser previsto encontra-se na tabela, ele é previsto não-tomado, de outra forma é considerado como tomado. As áreas da tabela são liberadas conforme os desvios são tomados e novas áreas são inseridas pela política de substituição do menos recentemente usado (*Least Recently Used* - LRU) para os desvios não-tomados.

Outra técnica é manter um bit para cada instrução numa *cache* rápida. Se a instrução for de desvio, o bit é usado para armazenar se ele foi tomado nas últimas execuções. Ou seja, os desvios são previstos como na sua última execução; se ele ainda não foi executado é previsto como tomado, marcando o bit como tomado quando a instrução é colocada pela primeira vez na *cache* rápida.

O refinamento das técnicas dinâmicas apresentadas conduz a novas técnicas que usam memória RAM ao invés de memória associativa para gerenciar de forma mais efetiva as decisões anômalas do previsor. O uso de uma função *hash* sobre o endereço da instrução de desvio resulta em um índice de m bits que endereça uma memória RAM de tamanho 2^m contendo um histórico de bits que representam as saídas mais recentes desta instrução de desvio.

Outra técnica usa contadores em complemento de dois, ao invés de um histórico de bits no previsor [47]. A previsão será de desvio tomado se o bit de sinal do contador acessado for 0 e será não-tomado se o bit de sinal for 1. O contador é incrementado quando o desvio for tomado e decrementado se o desvio for não-tomado. O contador pode ter de um a mais bits, sendo a melhor escolha o contador de 2 bits já que acima de dois bits não há ganho extra nos resultados de desempenho.

3.1.3. Outras Abordagens

Outras técnicas foram descritas nos estudos realizados por Lee e Smith [30] e por Perleberg e Smith [39] e ultrapassam a classificação inicial de previsão estática e dinâmica.

A técnica de desvio adiado (*delayed branch*) [30, 39] tem como idéia executar as n instruções imediatamente seguintes ao desvio, permitindo que instruções válidas sejam executadas até que o caminho do desvio seja conhecido e o endereço de destino calculado. Em seguida a ação é seguir o caminho do desvio ou fracassar o *pipeline*. Entretanto é difícil encontrar instruções seguintes que possam ser executadas antecipadamente, conduzindo normalmente ao preenchimento de instruções nulas no *pipeline*, que não traz ganhos no desempenho.

Um pequeno e rápido *buffer* de laços (loop buffer) [30, 39] mantido no estágio de busca do *pipeline* e utilizado para pré-busca e execução de instruções de laço

pode ser usado para previsão de desvio. Este *buffer* contém instruções seqüencialmente à frente do endereço de busca da instrução sendo executada e reconhece quando o destino de um desvio entra em sua área e conduz esta instrução à execução sem a necessidade de acessar a memória.

O uso de múltiplas filas de instruções [30, 39] serve para evitar a penalidade de uma previsão de desvio errado. Os estágios iniciais do *pipeline* são replicados de forma que ambos os caminhos do desvio podem ser buscados, decodificados e executados. Após a resolução do desvio, os efeitos do caminho incorreto são descartados. Contudo, o custo de duplicação do hardware necessário, o cálculo antecipado do endereço destino do desvio e a forma de tratar a ocorrência de outro desvio antes que um primeiro seja resolvido levam a um custo de efetividade questionável.

Outra técnica é a pré-busca do destino do desvio [30, 39]. Ao invés de replicar estágios do *pipeline*, utiliza-se um estágio adicional para fazer a pré-busca do destino do desvio, para reduzir a penalidade de seguir o caminho incorreto do desvio, levando a um esvaziamento do pipeline e redução do desempenho do processador. Mesmo que a execução tome o caminho errado, as instruções do destino do desvio são buscadas enquanto o desvio e as instruções no caminho não-tomado são executados, sem perda de tempo para a busca de instrução.

A técnica "preparar para desviar" (*prepare to branch*) [30, 39] faz uso de instruções especiais que direcionam o processador à pré-busca do endereço destino do desvio que está por vir, ao invés do endereço seqüencial que vem após o desvio. A efetividade desta técnica depende da habilidade do programador ou do compilador ordenar as instruções corretamente.

A previsão de desvios também pode ser feita em sistemas multiprocessados com o compartilhamento do *pipeline* [39]. Esta arquitetura executa *n* fluxos de instruções simultaneamente, buscando uma instrução de cada um dos *n* fluxos via algoritmo *round robin*. Com um *pipeline* cujo número de estágios seja menor ou igual a *n*, cada instrução, que esteja no mesmo fluxo de instrução, completa sua execução antes que a próxima instrução comece a executar. Da mesma forma, uma instrução de desvio completa antes que a próxima instrução seja buscada, evitando atrasos nas instruções de desvio.

A técnica chamada "empacotamento de desvio" (*branch folding*) [39] usa uma *cache* rápida com instruções que podem estar decodificadas ou não. Com esta área contendo instruções decodificadas, cada instrução que não é de desvio ocupa uma linha desta área e cada entrada possui um campo de próximo endereço apontando para a próxima linha de código a ser executada. Os desvios incondicionais são encapsulados no campo de próximo endereço da linha anterior e não necessita de tempo para execução do desvio porque está na área decodificada. Já os desvios condicionais usam um segundo campo de próximo endereço e fazem uso de previsão de desvio estático. Um bit na instrução de desvio diz se o desvio será tomado ou não e o campo de próximo endereço previsto é selecionado para buscar a próxima linha de micro-código. Se a *cache* não for decodificada, há a necessidade de um processador de desvio no início do *pipeline* e que encapsula o desvio quando seu resultado é conhecido.

3.1.4. Técnicas Mais Utilizadas

As técnicas de previsão de desvios mais amplamente utilizadas e com os melhores resultados utilizam uma tabela para armazenar os destinos de desvios (*Branch Target Buffer* – BTB). A BTB é uma pequena *cache* associada com o estágio de busca do *pipeline*. Ela é organizada em forma de tabela (Figura 11), contendo um rótulo para cada desvio (normalmente seu endereço), informação que permite prever se o desvio será tomado ou não (normalmente um contador ou pequeno histórico), o endereço mais recente de destino do desvio (o último endereço seguido) e possivelmente alguns bytes das instruções no endereço de destino. O estágio de busca de instruções do *pipeline* compara o endereço da instrução de desvio com os campos de endereço da BTB, na forma de um índice, definindo se esta instrução é de desvio ou não.

O desvio é previsto usando a informação do campo de previsão da BTB. Se ele é previsto tomado, o *pipeline* é alimentado com as instruções armazenadas e o processador inicia a busca de instruções do endereço destino deslocado pelo número de bytes das instruções armazenadas na BTB. Se for previsto não-tomado, o processador continua a busca de instruções no endereço após o desvio. Quando o desvio é resolvido no estágio de execução, uma verificação é feita quanto a

corretude da previsão. No caso de acerto, o processador continua seqüencialmente; em caso de erro na previsão ou no caso do endereço destino do desvio ter mudado, o processador deve esvaziar o *pipeline* e reiniciar a busca no caminho correto do desvio e a BTB deve ter seu campo de previsão e/ou de endereço destino atualizados.

Inst. #1	Estat. #1	Dest. #1	AAAA
Inst. #2	Estat. #2	Dest. #2	BBBB
Inst. #3	Estat. #3	Dest. #3	CCCC
⋮	⋮	⋮	⋮
Inst. #n-1	Estat. #n-1	Dest. #n-1	XXXX
Inst. #n	Estat. #n	Dest. #n	YYYY
Endereço da Instrução de Desvio	Estatística da Previsão de Desvio	Endereço de Destino do Desvio	Bytes de Instruções no Endereço Destino

Figura 11 – Área de Destino de Desvio (Branch Target Buffer – BTB)

Variações da BTB que usam um histórico mais complexo de dois níveis foram estudados por Yeh e Patt [56, 57]. A característica principal destas novas técnicas é que a previsão não se baseia somente no comportamento do desvio que se deseja prever, mas também no comportamento em conjunto com outros desvios recentes. Esta técnica foi chamada inicialmente de previsão de desvio por treinamento adaptável de dois níveis (*Two-Level Adaptive Training Branch Prediction*), porque a previsão é baseada, em um primeiro nível, no histórico dos últimos *n* desvios encontrados, em conjunto com as últimas *s* ocorrências de cada padrão particular deste histórico em um segundo nível. A implementação deste método requer duas estruturas principais: o registro de histórico de desvio e a tabela de histórico de padrões, mostradas na Figura 12.

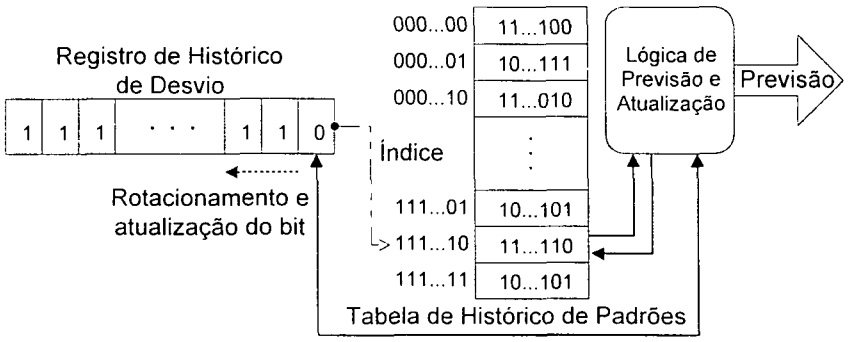


Figura 12 – Estrutura da previsão por treinamento adaptável de dois níveis

Posteriormente Yeh e Patt [56] propuseram alterações a que chamaram de forma abreviada de GAg, PAg e PAp e que estão demonstradas na Figura 13. No previsor GAg há um registrador de histórico global e uma tabela global de padrão de histórico, onde todas as previsões são baseadas, que são atualizados depois que cada desvio diferente é resolvido.

No previsor PAg, para coletar informações individuais do histórico do desvio, há um registrador de histórico associado a cada desvio distinto. Estes históricos estão contidos numa tabela de histórico acessado pelo endereço do desvio e que indexam uma tabela global de padrão de histórico e onde todas as previsões são baseadas. Depois que cada desvio é resolvido o previsor atualiza a linha correspondente ao endereço do desvio na tabela de histórico e na tabela global de histórico de padrões, podendo ocorrer interferência de vários desvios na mesma linha da tabela de padrões.

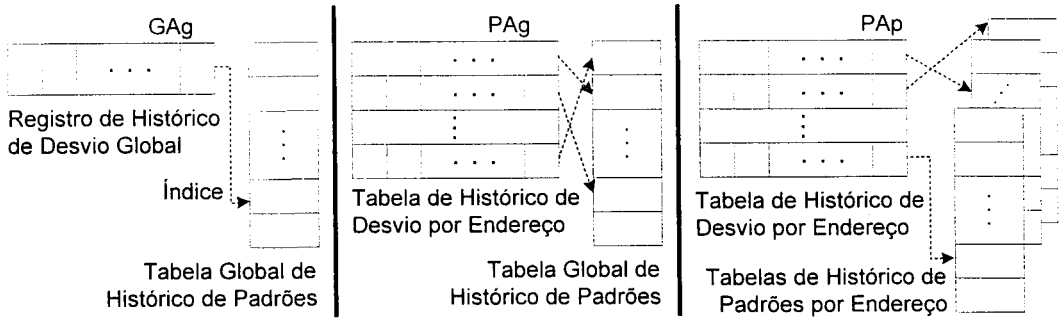


Figura 13 – Variações da previsão por treinamento adaptável de dois níveis

No previsor PAp, além da tabela de histórico de desvios por endereço, cada desvio tem sua própria tabela de padrões, para evitar a interferência entre desvios. Estas tabelas são agrupadas em tabelas de histórico de padrões por endereço e são indexadas pelo padrão do primeiro nível e pelo endereço do desvio, respectivamente.

Novas estratégias de previsão de desvios baseadas no uso redes neurais [23, 24, 29, 33] têm sido objeto de estudo recentemente. Estas estratégias têm utilizado arquiteturas simples de redes neurais, normalmente com apenas uma camada de neurônios. A característica principal destas técnicas é aproveitar a capacidade das redes neurais em aprender. Estruturas simples, como o Perceptron, têm se apresentado de forma atrativa, pois tem a capacidade de atuar sobre longas cadeias de histórico de desvio sem requerer muitos recursos computacionais e de hardware.

Detalhes sobre as redes neurais, incluindo organização, aprendizado e o Perceptron serão discutidos no próximo capítulo.

Trabalhos ainda mais recentes [22] estudam a aplicação de funções booleanas sobre o histórico de desvios, de forma a tentar prever o desvio. Com isso, a previsão de desvio baseada em histórico é vista como um problema de aprender a função booleana de um histórico de desvio que apresenta a melhor previsão. O objetivo da previsão dinâmica é aprender esta função da forma mais rápida possível para prover uma previsão correta.

A área de estudos de previsão de desvios é um campo vasto. Novas metodologias e técnicas estão sempre surgindo como alternativas de implementação, ora simplificando o hardware, ora provendo mais desempenho.

3.2. Trabalhos Relacionados à Previsão de Desvios

Sherwood e Calder [46] trabalharam com desvios que são difíceis de prever através dos métodos de dois níveis tradicionais: os desvios finalizadores de laços, os quais podem causar uma grande quantidade de erros de previsão em programas com laços altamente aninhados. Eles propuseram duas técnicas para tratar deste tipo de desvios: um mecanismo de hardware para auxiliar qualquer previsor de desvios, chamado de LTB (Loop Termination Buffer), capaz de capturar os padrões para estes tipos de desvios e uma técnica de software, chamada Branch Splitting, capaz de quebrar laços grandes, os quais não podem ser capturados através de históricos convencionais, em pequenos laços capturáveis. O uso de LTB reduziu os erros de previsão de 5,4% para 0,2% para o *benchmark* m88ksim e o uso de Branch Splitting reduziu os erros de previsão de 2,6% para 1,3% para o *benchmark* apsi.

Zilles e Sohi [58] descobriram que existe um subconjunto pequeno das instruções estáticas (*backward slices*) que contribui bastante para a degradação do desempenho de muitas aplicações, cujo comportamento não pode ser antecipado pelos previsores de desvios atuais. Eles propuseram uma técnica que remove estas instruções do programa para executá-las em paralelo com o próprio programa, escondendo assim as latências causadas pelas mesmas. Nestes experimentos os erros de previsão ficaram abaixo de 5%.

Fern et al. [14] propuseram um modelo de seleção dinâmica de características de desvios para que os previsores baseados em tabelas possam conter históricos mais relevantes dependendo de cada tipo de desvio, provendo economia do espaço de armazenamento se comparado com as abordagens baseadas em tabelas comumente utilizadas.

Ramirez, Larriba e Valero [41] analisaram o efeito da reordenação de código na previsão de desvios. Eles concluíram que a reordenação de código permite aumentar a ocorrência de desvios não-tomados e com isso melhorar a previsão de desvios. Usando reordenação de código os previsores estáticos podem atingir mais que 80% de acerto e os pequenos previsores dinâmicos podem ser mais eficientes. Em seus experimentos, um previsor *gshare* de 0,5KB pode ser melhorado de 91,4% para 93,6% e um *gskew* de 0,4KB de 93,5% para 94,4%. Infelizmente, para previsores com grandes históricos de desvios, o aumento do número de desvios não-tomados pode degradar o desempenho, tal como no previsor *agree* de 16KB, cujo desempenho diminui de 96,2% para 95,8%. Mesmo assim, devido a outros benefícios tais como o aumento da banda útil da busca e maior estabilidade na *cache*, um processador com previsor *agree* tem um aumento de 8% no desempenho.

Gonçalves et al. [19] apresentaram uma comparação entre arquiteturas superescalar e SMT baseado no impacto da melhora na exatidão da previsão de desvios. Eles analisaram ambas arquiteturas sob diferentes cenários que compreendiam diferentes configurações (i.e. recursos disponíveis) e diferentes acertos de previsão. Embora as arquiteturas SMT e superescalar tenham comportamentos diferentes, eles observaram que o efeito da melhora no acerto na previsão é similar em ambas as arquiteturas.

Aragon et al. [2] propuseram um avaliador de confidência para ser usado em conjunto com qualquer previsor de desvios. O avaliador é capaz de informar se a previsão realizada possui baixa ou alta confidência. Para os desvios de baixa confidência eles propuseram uma Unidade de Inversão de Previsão de Desvios. O mecanismo proposto possui maior efeito para desvios com baixa correlação com históricos passados tais como os desvios finalizadores de laços. Os resultados de simulações mostraram em alguns casos uma redução da ordem de 15% de erro de previsão para *benchmarks* de inteiros do SPEC2000 e aumento de desempenho da

ordem de 9% em processadores superescalares. A principal conclusão é que utilizar o avaliador proposto é melhor do que simplesmente aumentar o tamanho do previsor.

Os Milenkovic e Kulick [35] usaram a ferramenta *VTune Performance Analyzer* da Intel para investigar a organização e tamanho dos previsores de desvios nos processadores Pentium III e Pentium IV através de uma série de experimentos com o microbenchmark *spy* e somente conseguiram descobrir algumas medidas utilizadas em suas estruturas internas. O objetivo do experimento foi o de obter métricas reais que pudessem ser usadas em simulações, embora não tenham tido muito êxito.

4. Fundamentos em Redes Neurais Artificiais

A utilização de implementações de redes neurais em diversas áreas do conhecimento humano vem sendo ampliada de forma significativa. Devido às suas características peculiares, tais como: capacidade de aprendizado, generalização, robustez a ruídos, entre outras. Os sistemas baseados em redes neurais são adotados em aplicações diversas tais como classificação, previsão, otimização, aproximação, entre outras [11].

Segundo Jain e Mao [21] a aplicabilidade das redes neurais é extensa e pode solucionar uma variedade de desafios em problemas computacionais. Entre estes, encontram-se os problemas de prever seqüências em função do tempo, onde se enquadra no problema de se prever o desvio de uma instrução baseado em seu histórico de desvio passado.

As redes neurais evoluíram e se diversificaram em inúmeras implementações diferentes, adequando-se a cada problema específico a ser resolvido. Uma visão clara da estrutura do neurônio artificial e evolução das redes neurais pode ser vista em Kováks [28] e Haykin [20].

4.1. Modelo Biológico de um Neurônio

O neurônio biológico mais usual possui uma estrutura como a ilustrada na Figura 14, constituído por várias ramificações de entrada chamados dendritos, um corpo celular chamado soma e uma ramificação de saída chamada axônio.

Neurônios estabelecem interações celulares por meio de sinapses, que é espaço iônico entre células nervosas. Uma sinapse ocorre usualmente entre a extremidade de um axônio, do neurônio pré-sináptico, e a extremidade de um dendrito, do neurônio pós-sináptico.

Um neurônio pós-sináptico recebe sinais (impulsos) de neurônios pré-sinápticos através dos dendritos (entradas) e transmite sinais gerados pelo seu corpo celular ou soma através do axônio (saída) e suas ramificações. A iteração entre neurônios ocorre por meio de neurotransmissores que fluem pelas sinapses. As sinapses podem ser excitatórias (que propagam o sinal gerado) ou inibitórias (que retêm o sinal gerado). A efetividade das sinapses pode ser ajustada pelos sinais que passam através delas. Desta forma as sinapses aprendem a partir das atividades nas quais estão envolvidas.

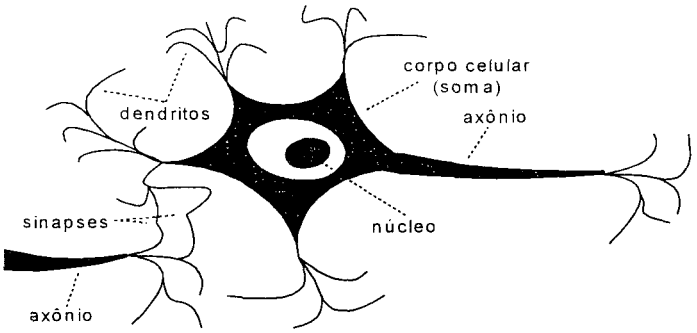


Figura 14 – Representação biológica de um neurônio

O impulso nervoso que passa através do axônio é controlado por um potencial de ação que é determinado por um limiar de disparo. Este controle do potencial de ação, a partir do limiar, determina quando um pulso é propagado pelo axônio [11].

4.2. Modelo Computacional de um Neurônio

A partir do conhecimento do funcionamento biológico do neurônio modelou-se computacionalmente o seu funcionamento. Sua representação computacional pode ser visto na Figura 15.

O neurônio é alimentado por n entradas X_i ($1 \leq i \leq n$) ponderadas por pesos sinápticos W_j ($1 \leq j \leq n$). Os pesos sinápticos representam a eficiência sináptica e definem as correlações de aprendizado para as entradas a eles relacionados. A saída y é definida pela função f cujo argumento é a soma das entradas ponderadas

pelos respectivos pesos e tem como parâmetro o limiar Θ. Assim, o neurônio é representado matematicamente pela Equação 1:

$$y = f(\sum_{j=1}^{n} W_j X_j - \Theta) \qquad \text{(Equação 1)}$$

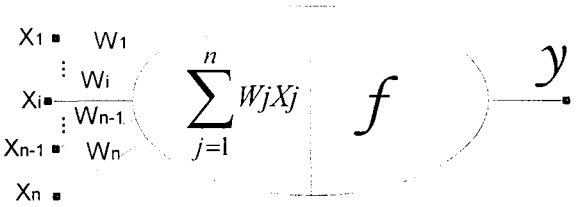


Figura 15 – Representação computacional de um neurônio

Para esta equação, pesos positivos correspondem a sinapses excitatórias e pesos negativos correspondem a sinapses inibitórias. Por questão de simplificação da notação, costuma-se utilizar o limiar Θ como um outro peso $W_0 = -\Theta$ ligado ao neurônio com uma constante de entrada $X_0 = 1$. Assim, a Equação 1 pode ser reduzida para a forma:

$$y = f(W_0 + \sum_{j=1}^{n} W_j X_j) \qquad \text{(Equação 2)}$$

A estrutura computacional do neurônio artificial procura representar na íntegra o modelo simplificado do neurônio biológico: as linhas e conexões da Figura 15 representam os axônios e dendritos, os pesos das conexões representam as sinapses e a função limiar, a atividade do corpo celular.

As redes neurais são formadas por grupos de neurônios organizados em uma estrutura particular. A topologia da estrutura é definida principalmente pelo número de neurônios, pelo número de entradas e de saídas; e a forma como os neurônios estão conectados.

Uma estrutura bastante usual é a estrutura com arranjo de neurônios em camadas. Cada camada é formada com um grupo de neurônios tal que cada neurônio estabelece conexões de entrada apenas com uma camada (camada anterior) e conexões de saída com uma outra distinta camada (camada posterior). A camada de entrada recebe as entradas da rede neural e estabelece conexões apenas com uma camada posterior e a camada de saída estabelece conexões apenas com uma camada anterior e fornece as saídas da rede neural.

4.3. Aprendizado de um Neurônio

Uma das principais características da rede neural é sua capacidade de aprendizado e o processo de aprendizado das redes neurais pode ser visto como um problema de atualizar os pesos sinápticos de forma que a rede possa realizar satisfatoriamente uma tarefa específica [21].

O aprendizado de redes neurais é classificado em três principais paradigmas: supervisionado, não supervisionado e híbrido.

No paradigma de aprendizado supervisionado, a rede opera como se tivesse um professor conduzindo seu aprendizado. A rede é alimentada com valores de entrada e o valor de saída esperado para aquelas entradas. Os novos pesos sinápticos são ajustados para produzir uma saída que mais se aproxime do resultado esperado.

O aprendizado baseado em reforço pode ser usado como uma variação deste método, em que, ao invés de ser fornecida a saída esperada, a rede recebe uma crítica do quão correta sua saída está em relação à saída esperada [21].

No paradigma não supervisionado, somente entradas são fornecidas à rede neural. A rede explora a estrutura oculta nos dados, correlações de padrões entre os dados e organiza padrões em categorias a partir destas correlações.

O paradigma híbrido combina os dois paradigmas anteriores. Parte dos pesos sinápticos são ajustados por aprendizado supervisionado e parte por não supervisionado.

A aprendizagem supervisionada, por exemplo, deve considerar três aspectos práticos e fundamentais: capacidade da rede neural, complexidade dos exemplos e complexidade computacional. A capacidade da rede neural define quantos padrões podem ser armazenados e quais funções e limites de decisão a rede pode formar. A complexidade de exemplos define o número de padrões de treinamento necessários para treinar a rede e garantir uma generalização válida. E a complexidade computacional refere-se ao tempo requerido para a aprendizagem de forma que a rede neural ofereça uma solução.

As técnicas de aprendizado são classificadas em quatro tipos básicos: correção de erro, aprendizado de Boltzmann, aprendizado de Hebb e aprendizado competitivo [20].

O princípio básico do aprendizado por correção de erro é usar o erro gerado (diferencial entre a saída esperada e a saída real do neurônio) para modificar os pesos sinápticos de forma a reduzir este erro.

O objetivo do aprendizado de Boltzmann é ajustar os pesos sinápticos de forma que os estados visíveis dos neurônios que compõem a rede satisfaçam uma particular distribuição de probabilidade desejada. Este é um caso particular da aprendizagem por erro, já que é a diferença entre as saídas de dois neurônios em diferentes condições de operação.

O aprendizado de Hebb, concebido por um biólogo com o mesmo nome, é baseado na observação de experimentos neurobiológicos, e é a mais antiga regra de aprendizagem elaborada [20]. Este aprendizado baseia-se no fato de que o processo de aprendizado de sistemas nervosos complexos pode ser reduzido a um processo puramente local. A intensidade das conexões sinápticas é alterada apenas em função dos erros detectáveis localmente e a mudança nos pesos sinápticos de um neurônio depende da atividade dos neurônios ligados a ele [21, 28].

Na aprendizagem competitiva, os neurônios disputam entre si pela ativação e somente o neurônio vencedor tem sua saída propagada, resultando em apenas uma saída ativa a cada tempo.

Uma visão mais detalhada destas técnicas de aprendizado pode ser obtida em Jain e Mao [21] e Haykin [20].

4.4. Perceptron

O Perceptron foi um dos primeiros modelos de neurônio implementados. A rede neural constituída por tais neurônios é usualmente denominada Perceptron [44], sendo formada por duas camadas, as camadas de entrada e de saída. Este é o conceito adotado como definição para o Perceptron neste trabalho. Outros autores definem o Perceptron com camadas ocultas como Perceptron Multi-Camadas (MLP - *Multi-Layer Perceptron*) [10, 21, 26].

O Perceptron originalmente concebido por Rosemblatt [20, 28] gera um resultado de saída, calculado pela Equação 2, no intervalo [–1;+1], em que a função

de ativação é uma função sinal $f(X)$ na forma [20]: $f(X) = \begin{cases} +1 \; se \; X \geq 0 \\ -1 \; se \; X < 0 \end{cases}$. Este será o modelo de Perceptron utilizado no trabalho.

O Perceptron composto por apenas um neurônio com várias entradas possui a capacidade restrita de classificar elementos por separação linear [24, 44]. Assim, se as possíveis entradas de um Perceptron estiverem contidas em um espaço *n*-dimensional, tal que possam ser agrupadas em dois conjuntos desconexos linearmente separáveis então haverá um hiperplano (uma reta, se *n*=2) que divide este espaço em semi-espaço para o qual o Perceptron responde –1 para todos os elementos de um conjunto e, outro em que a resposta é +1 para todos os elementos do outro conjunto. O hiperplano é definido pela Equação 3:

$$W_0 + \sum_{j=1}^{n} W_j X_j = 0 \qquad \text{(Equação 3)}$$

Rosemblatt provou que quando o padrão de treinamento fornecido ao Perceptron possui classes linearmente separáveis o processo de aprendizagem do Perceptron converge em um número finito de iterações, ajustando os pesos sinápticos e o limiar de ativação do neurônio [21, 24]. Este é o chamado Teorema de Convergência do Perceptron.

No processo de aprendizagem do Perceptron é utilizado o aprendizado por correção de erro, em que os pesos sinápticos inicialmente são atribuídos com valores aleatoriamente gerados, geralmente na faixa $[-½;+½]$ [44]. Normalmente a atualização dos pesos sinápticos ocorre somente na presença de erro entre a saída apresentada e a saída desejada.

O algoritmo de aprendizado do Perceptron pode ser definido pelos seguintes passos:

- Ajustar os pesos e o limiar de ativação com números aleatórios no intervalo estipulado.
- Apresentar cada entrada representada pelo vetor $X(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_i(t) \\ \vdots \\ x_n(t) \end{bmatrix}$ e avaliar a saída $y(t)$ com a saída desejada $d(t)$ para a iteração t.
- Atualizar os pesos sinápticos de acordo com Equação 4:

$W_i(t+1)=W_i(t) + \eta(d(t)- y(t))X_i(t), 0 < n < 1$ (Equação 4)

Onde η é a taxa de aprendizado do Perceptron (o passo de ganho).

O Perceptron pode ser usado para aprender correlações entre a seqüência de saída, baseado numa seqüência de entradas, portanto, sendo interessante em problemas de previsão. É devido a esta característica que o Perceptron é utilizado como um previsor de desvios neste trabalho.

4.5. Trabalhos Relacionados a Redes Neurais e ao Perceptron

Vários trabalhos recentes investigaram o uso de redes neurais do tipo Perceptron em diferentes áreas científicas, inclusive na previsão de desvios em arquiteturas de computadores modernos.

Os Mitra e Pal [36] propuseram um algoritmo para um sistema de suporte à decisão na detecção de diferentes estágios de câncer cervical. A metodologia apresentada por eles é baseada na integração de uma rede modular baseada em conhecimento com algoritmos genéticos. Eles propuseram um algoritmo envolvendo vários módulos de redes neurais Perceptron multi-camadas, sintonizados através de algoritmos genéticos. Resultados demonstraram que a proposta é mais eficiente em termos de redução do tempo de treinamento, do tempo de classificação e da espessura da rede, quando comparado com uma única rede neural Perceptron multi-camadas convencional.

Lee e Zhang [31] propuseram um método para o controle de robôs baseado em comportamento através de programação genética. Devido à dificuldade de utilização em algoritmos genéticos de entradas provenientes de sensores, eles utilizaram um Perceptron simples (de uma camada) para representar cada elemento funcional da programação genética. Com esta técnica, eliminaram a necessidade de mapear a representação simbólica com os valores de entrada dos sensores. Eles verificaram que é possível aos robôs aprenderem as regras corretas de comportamento baseado em informações sensoriais locais e limitadas, sem necessidade de mapas internos.

Koumpis e Renals [27] também realizaram pesquisas com Perceptron na área de reconhecimento de fala em sistema de mensagens de voz (*voicemail*), com o objetivo de transcrevê-las e resumi-las. Eles descreveram um sistema híbrido que combinava capacidades de modelagem temporal de Markov com a capacidade de classificação de padrões do Perceptron multi-camadas. Os testes foram realizados sobre um conjunto de mais de 1800 mensagens de voluntários, coletadas pela IBM. O Perceptron tinha como entradas características acústicas e como saídas 54 classes de fonemas. O sistema proporcionou desempenho competitivo na tarefa de transcrição, quando comparado com método gaussiano, e quanto a tarefa de

elaboração de resumo, o sistema permitiu a extração de informações sutis mesmo na ocorrência de erros na transcrição.

Oliveira et al. [37] apresentaram uma nova abordagem de segmentação aplicada a dígitos escritos a mão. Para avaliar o algoritmo proposto, eles utilizaram três Perceptrons multi-camadas treinados em 3 tipos de características: concavidade, bordas e contornos. A base de treinamento continha 9500 imagens de dígitos isolados escritos manualmente extraídos de 2000 folhas de cheque brasileiros. A avaliação foi feita sobre dígitos isolados ou conectados aos pares e triplos. Os resultados mostraram que 98,5% dos dígitos conectados podem ser segmentados corretamente.

Liu et al. [32] investigaram a interpretação automatizada da atividade dinâmica de objetos em seqüência de imagens em ambientes incertos, tais como tráfego de veículos. Neste trabalho, eles usaram uma rede neural Perceptron convencional de três camadas para identificar e categorizar as trajetórias úteis dos objetos na seqüência de imagens, cujos resultados foram usados posteriormente no processo de interpretação através de lógica nebulosa.

Em outro trabalho interessante, Khardon, Roth e Servedio [25] investigaram algoritmos de aprendizado dentro de domínios lógicos. Eles ponderaram a eficiência computacional com a convergência do Perceptron no espaço de conjunções. Através de demonstrações matemáticas eles concluíram que, quando trabalhando com conjunções de características básicas, o Perceptron pode aumentar a sua expressividade e ser eficiente sobre um número exponencial de conjunções, mas também pode errar exponencialmente mesmo quando aprendendo funções simples, devido ao aumento do número de características e do tempo de computação.

Kuvayev [29] analisou algoritmos de rede neural, inclusive o Perceptron, para a tarefa de previsão de desvios. Ele concluiu que o previsor Perceptron apresentou melhores resultados embora um maior treinamento de dados era necessário para a abordagem das outras redes neurais.

Steven et al. [51] analisaram dois tipos de redes neurais em previsão de desvios, a rede LVQ (Learning Vector Quantification) e a rede Backpropagation. Eles demonstraram que um previsor baseado em rede neural pode alcançar taxas de acerto comparável ao previsor adaptativo de dois níveis, explorando em algumas circunstâncias correlação de informações mais eficientemente com um treinamento

mínimo, e sugeriram que maiores investigações em previsores baseados em redes neurais eram necessárias.

Sbera, Vintan e Florea [45] investigaram um técnica estática de previsão de desvios baseada em redes neurais. Seu previsor não previa o comportamento de um programa baseado na execução prévia do mesmo programa ou baseado no perfil de alguns programas, mas usava o conhecimento obtido de outros programas.

Jiménez e Lin [23] apresentaram um método para a previsão de desvios em arquiteturas de computadores modernos baseado em rede neural Perceptron. O método por eles apresentado mostrou melhores desempenhos do que métodos baseados em contadores de dois bits comumente usados, atingindo ganhos acima de 14% na redução de erros de previsão sobre o previsor Gshare, quando usando rastro de execução de *benchmarks* do SPEC2000. Além disso, o método proposto permitiu a análise de históricos de desvios maiores do que os convencionais.

Este método foi analisado e estendido para funções mais gerais por Majumdar e Weitz [33] e indicaram que não há aumento da complexidade para a implementação de um previsor baseado no Perceptron que representa funções mais gerais embora não haja melhora de desempenho em relação ao previsor Perceptron original. Eles concluíram que os algoritmos de previsão e atualização são mais complexos e não trabalham em poucos ciclos como os do previsor original.

5. Proposta do Trabalho

A objetivo geral deste trabalho é o de propor e avaliar o uso de técnicas de previsão de desvios baseadas em redes neurais do tipo Perceptron, sob diferentes aspectos e situações ainda não analisadas, de forma a entender o seu comportamento com profundidade. A justificativa para se usar o Perceptron é o fato deste ser uma rede neural simples que permite simplicidade de implementação em hardware, possuir a capacidade de fazer previsões, possuir capacidade de aprendizado e já ter sido pesquisada em outros trabalhos relacionados com resultados satisfatórios preliminarmente obtidos.

Para o desenvolvimento teórico deste trabalho, primeiramente foi realizado um estudo da estrutura e organização das arquiteturas superescalares, enfocando especialmente as técnicas de previsão de desvios. Em seguida, os estudos concentraram-se sobre os conceitos de redes neurais e seu funcionamento específico na previsão de séries, enfocando o Perceptron.

Com a finalidade de relacionar os estudos realizados e avaliar o grau de satisfatoriedade quanto ao uso do Perceptron na previsão de desvios, alguns modelos preliminares foram implementados e simulados com resultados satisfatórios mostrando que o presente trabalho poderia prosseguir nesta linha de investigação. O primeiro modelo preliminar foi desenvolvido para extrair parâmetros iniciais de comparação. Através da análise destes resultados pôde-se efetuar melhorias neste modelo gerando assim uma segunda versão, cujos resultados também foram analisados. A análise destes modelos preliminares gerou subsídios para a concepção e implementação de modelos de previsores mais agressivos.

Assim, cinco modelos mais complexos de previsores baseados no Perceptron foram propostos. Estes modelos foram parametrizados em diferentes aspectos estruturais, definindo uma gama distinta de previsores. Estes modelos foram implementados e simulados, provendo métricas de desempenho, as quais foram agrupadas e analisadas com a ajuda de um sistema implementado especificamente para esta função.

As próximas seções descrevem e apresentam os resultados obtidos em todas as fases de desenvolvimento do trabalho, cujo sequenciamento acompanha a ordem cronológica das tarefas realizadas.

5.1. Modelos Preliminares de Estudo

Um primeiro previsor básico foi implementado com facilidade de especulação controlada e foi desenvolvido a partir da alteração de um núcleo de previsão baseado no Perceptron cedido por Jiménez e Lin [23, 24]. Este previsor não se preocupa em restaurar os pesos sinápticos reais após um erro de previsão, repondo o histórico de desvios e atualizando o aprendizado segundo os pesos especulativos correntes. Este previsor é aqui chamado de *Previsor com Especulação Controlada sem Atualização de Peso*. Posteriormente, uma segunda versão de previsor foi implementada com a característica de restaurar, além do histórico especulativo de desvios, os pesos reais utilizados na última execução com acerto da previsão. Este previsor é aqui chamado de *Previsor com Especulação Controlada com Atualização de Peso*. O uso destes dois modelos preliminares de previsores com especulação controlada é relatado em [42].

5.1.1. Implementação dos Modelos Preliminares

O objetivo da implementação dos modelos preliminares é o de avaliar o comportamento do previsor considerando a variação do grau de especulação e do número de desvios obtidos (histórico do desvio). O histórico dos desvios foi usado como entrada para o Perceptron, podendo o mesmo conter valores especulativos (correspondentes às previsões ainda não resolvidas) e/ou reais (correspondente aos resultados dos desvios efetivamente ocorridos), dependendo do número de entradas do Perceptron.

Nos previsores aqui desenvolvidos, os desvios são tratados por um Perceptron, o qual é executado em duas fases principais. A primeira fase corresponde à previsão do desvio, que é calculada em função dos pesos sinápticos do Perceptron e do histórico de desvios. A segunda fase corresponde ao aprendizado do Perceptron, em que o ajuste de peso (peso especulativo) é feito continuamente baseado no histórico especulativo. Ambos previsores permitem fixar o grau de especulação, ou seja, fixar o número exato de desvios que são mantidos em espera nas estações de reserva antes do desvio corrente ser resolvido.

No primeiro modelo implementado, representado na Figura 16, se o desvio que está sendo avaliado não foi ainda resolvido, o previsor apenas atualiza especulativamente seus pesos sinápticos. Caso o desvio avaliado já tenha sido resolvido, então o histórico real é atualizado com o resultado do desvio. Em seguida é analisado se a previsão é correta ou incorreta. Se a previsão for correta, os pesos sinápticos são ajustados segundo os pesos especulativos. Se a previsão for incorreta o histórico especulativo é atualizado para refletir o histórico real e os pesos sinápticos são ainda ajustados segundo os pesos especulativos.

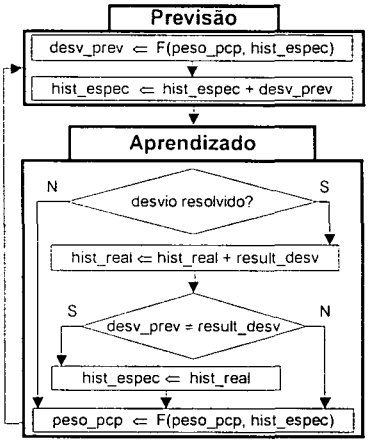


Figura 16 – Previsor com Especulação Controlada sem Atualização de Pesos

O segundo modelo implementado, representado na Figura 17, possui funcionalidade semelhante ao modelo anterior, com a diferença que no caso do previsor errar, além do histórico especulativo ser atualizado para o histórico real, os pesos sinápticos são também ajustados para refletir os pesos reais da última previsão correta realizada. Existe nesta estrutura o armazenamento dos últimos pesos sinápticos reais utilizados na última previsão correta realizada.

Os previsores com especulação controlada foram avaliados usando rastros de desvios obtidos na execução de quatro *benchmarks* de inteiros (*cc1, ijpeg, li e perl*) e de quatro de ponto flutuante (*fpppp, mgrid, swim e wave5*) do pacote de avaliação SPEC [49]. Os *benchmarks* foram executados no simulador sim-outorder do SimpleScalar [7], em que as primeiras 10 milhões de instruções foram desconsideradas e somente os resultados das instruções de desvios (tomado ou não-tomado) das 100 milhões de instruções seguintes foram coletados. A seguir são apresentados os resultados preliminares destas simulações.

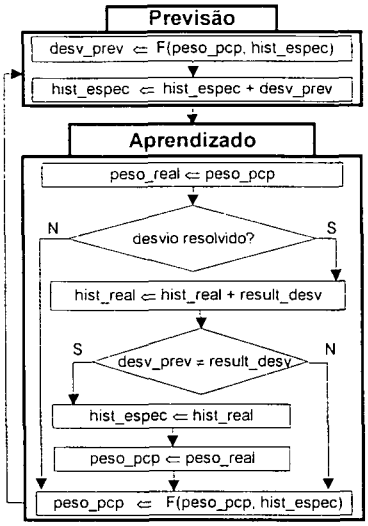


Figura 17 – Previsor com Especulação Controlada com Atualização de Pesos

5.1.2. Resultados Preliminares

Os resultados preliminares representam a porcentagem de acerto do previsor de desvios em relação ao número de desvios realmente tomados durante a execução do *benchmark*. As figuras seguintes mostram os resultados da avaliação dos *benchmarks* de inteiros, ponto flutuante e o conjunto de todos os *benchmarks*, obtidos com a média aritmética simples entre os resultados individuais de cada um.

O eixo x apresenta a variação do grau de especulação, crescendo de 0 (nenhum desvio pendente além do corrente) até 64 (número de desvios continuamente pendentes nas estações de reserva a espera de resolução, além do corrente). O número de entradas utilizadas pelo Perceptron (tamanho do histórico em bits) também foi variado de 2 a 64, representado pelas colunas da esquerda para a direita no gráfico. O eixo y apresenta a taxa de acerto.

Estes resultados foram obtidos considerando a especulação implícita dos programas. Fixar o número de desvios pendentes força a ocorrência de uma situação que pode não ser real, pois a quantidade de desvios pendentes varia dinamicamente com a execução do programa, mas é útil para podermos analisar a variação e o comportamento da taxa de acerto. Os resultados mostram que o acerto do previsor baseado no Perceptron sofre degradação acentuada devido ao aumento do grau de especulação.

Para o primeiro modelo do previsor, a Figura 18 mostra os resultados para os *benchmarks* de inteiro, a Figura 19 mostra os resultados equivalentes para os *benchmarks* de ponto flutuante e a Figura 20 apresenta os resultados médios entre todos os 8 *benchmarks* avaliados.

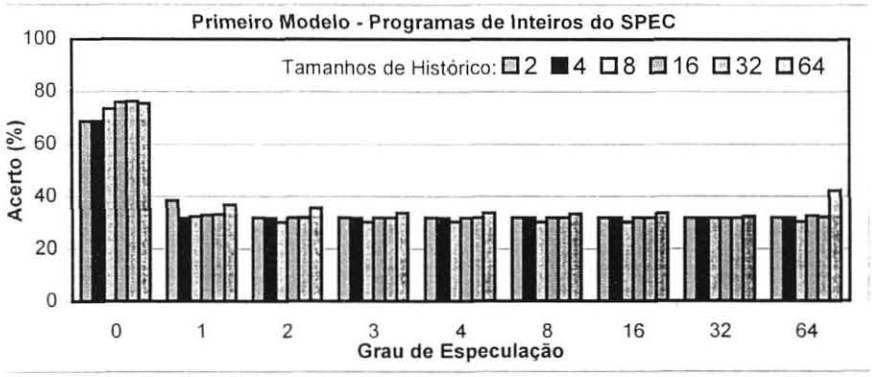


Figura 18 – Resultados do primeiro modelo para *benchmarks* de inteiro

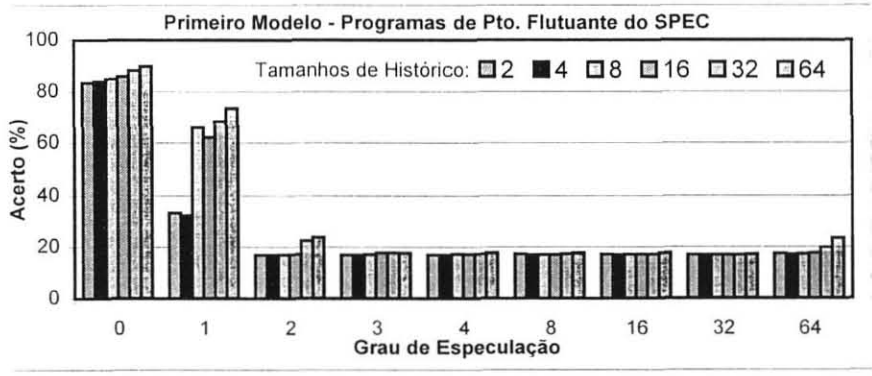


Figura 19 – Resultados do primeiro modelo para *benchmarks* de pto. flutuante

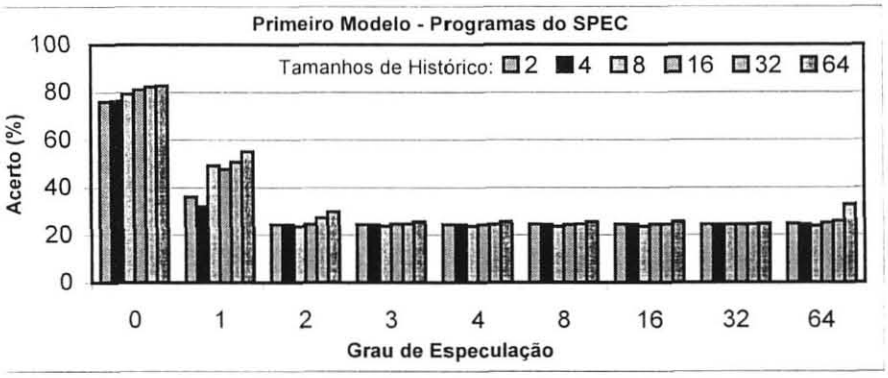


Figura 20 – Resultados do primeiro modelo para os 8 *benchmarks* do SPEC

De forma imediata, é possível observar uma queda acentuada na taxa de acerto do previsor com relação aos resultados do previsor sem especulação. Os percentuais de acerto poucas vezes ultrapassam 60%.

A Figura 21 mostra os resultados para os *benchmarks* de inteiro. A Figura 22 mostra os resultados equivalentes para os *benchmarks* de ponto flutuante. A Figura 23 apresenta os resultados médios entre todos os 8 *benchmarks* para o segundo modelo do previsor.

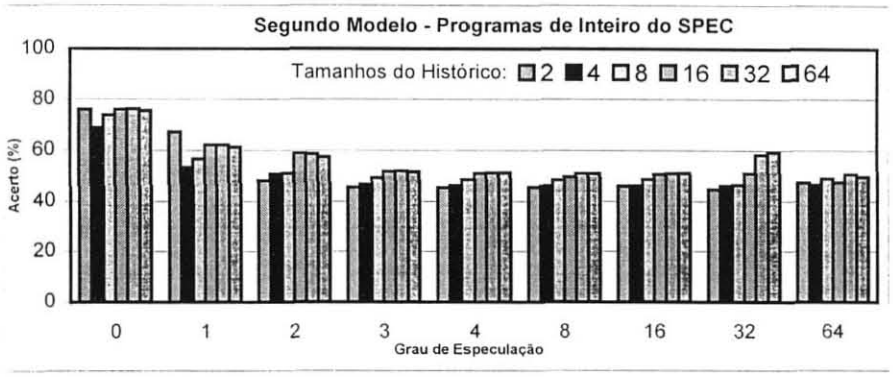


Figura 21 – Resultados do segundo modelo para *benchmarks* de inteiro

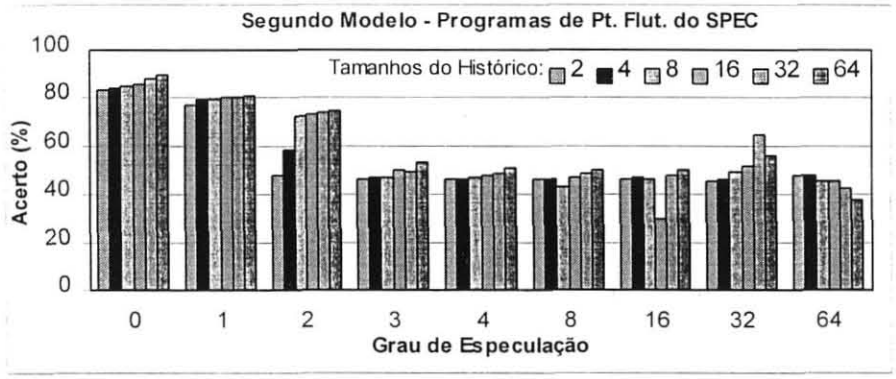


Figura 22 – Resultados do segundo modelo para *benchmarks* de pto. flutuante

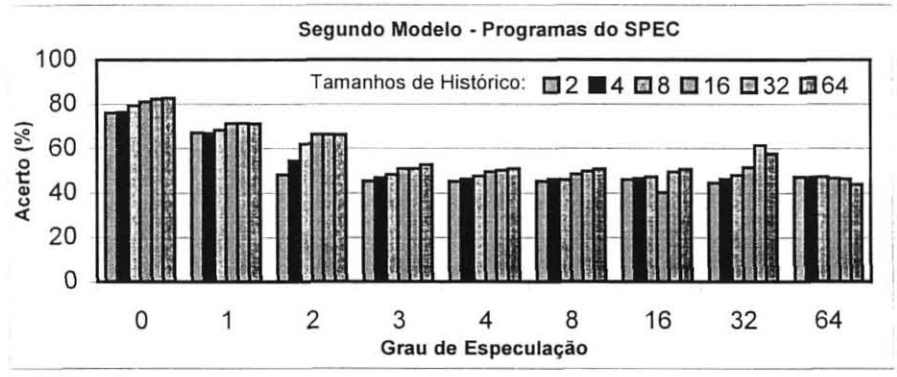


Figura 23 – Resultados do segundo modelo para os 8 *benchmarks* do SPEC

De uma forma geral, podemos observar na Figura 23 que a taxa de acerto do previsor aumenta levemente na medida em que aumenta o tamanho do histórico na entrada do Perceptron (exceto quando o grau de especulação é igual a 64 para

os *benchmarks* de ponto flutuante). Isto é bastante razoável, pois quanto maior é o histórico aceito pelo Perceptron, mais estável é o padrão dos resultados dos desvios. A maior diferença entre os históricos de 0 e 64 foi de 54%, obtida com os *benchmarks* de ponto flutuante quando o grau de especulação é de 2 desvios continuamente pendentes. Além disso, podemos também observar que quando existem poucos desvios pendentes (grau entre 0 e 2), os resultados para ponto flutuante são melhores do que os resultados para inteiros. A maior vantagem é de 32%, obtida para o grau de especulação 1 e histórico 64 (80,95 contra 61,20).

Podemos observar também que a taxa de acerto do previsor sofre redução acentuada quando aumenta o grau de especulação. Para os *benchmarks* de inteiros esta queda ocorre mais rapidamente, atingindo índices de redução sobre o melhor acerto da ordem de 23%, quando o grau aumenta de 0 para 1. Para os *benchmarks* de ponto flutuante este índice de redução atinge o máximo de 39% quando o grau aumenta de 2 para 3.

Vemos claramente que os *benchmarks* de inteiros são mais suscetíveis ao aumento do número de desvios pendentes. De uma forma geral, quanto maior a freqüência de desvios dentro de um programa, maior será o número de desvios pendentes e com isso o desempenho do previsor será prejudicado. Em todas as simulações realizadas, a melhor taxa de acerto foi de 97,8%, obtida pelo *benchmark* de ponto flutuante *mgrid* quando o grau de especulação é 0.

5.1.3. Conclusões e Perspectivas de Continuidade

Os ganhos no desempenho do segundo modelo de previsão sobre o primeiro modelo podem ser vistos nas figuras a seguir (Figura 24, Figura 25 e Figura 26), as quais mostram as diferenças percentuais em porcentagem de acerto na previsão. Podemos observar que o segundo modelo atinge ganhos da ordem de 30% para *benchmarks* de inteiros e da ordem de mais de 50% para *benchmarks* de ponto flutuante. Na média entre todos estes *benchmarks*, os ganhos variam na maioria dos casos na faixa de 20% a 30% na taxa de acerto na previsão de desvios.

Apesar do previsor baseado no Perceptron sofrer uma queda de desempenho quando forçado a trabalhar com desvios especulativos, seu desempenho melhora com a mudança do seu mecanismo de funcionamento.

Embora esta melhora atinja ganhos significativos, os valores máximos alcançados ainda exigem maiores estudos e melhorias. Uma possibilidade é modificar tanto a função quanto o limiar de ativação do Perceptron de forma a reduzir as interferências causadas pelos desvios pendentes. Desta forma, o previsor poderá a cada nova previsão considerar o número de desvios pendentes como um fator de correção dos pesos sinápticos durante o aprendizado.

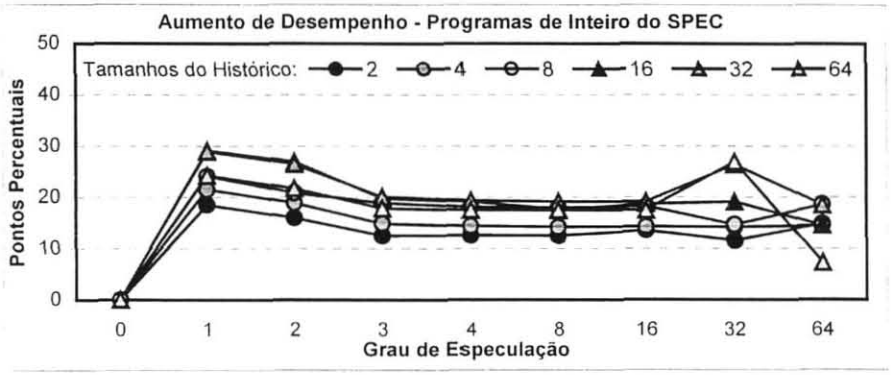


Figura 24 – Ganho de desempenho para *benchmarks* de inteiro

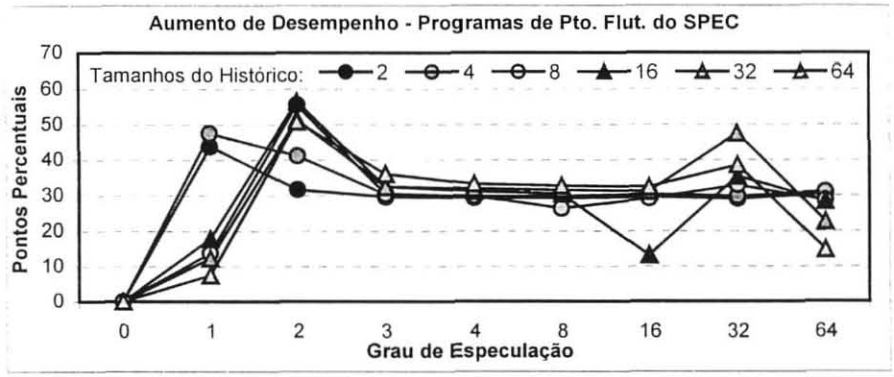


Figura 25 – Ganho de desempenho para *benchmarks* de pto. flutuante

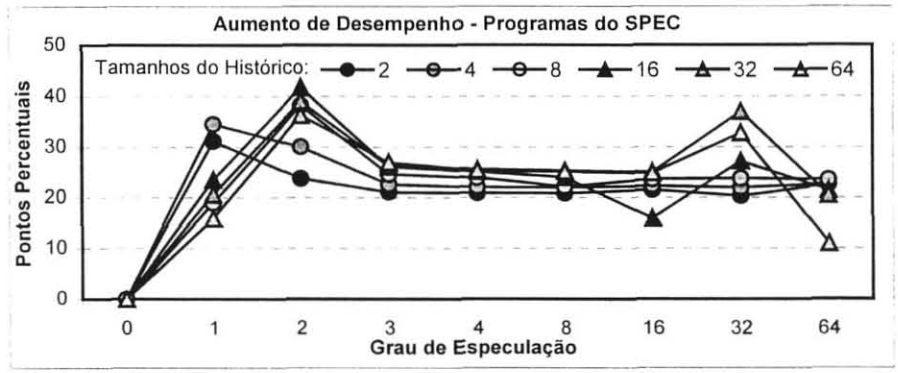


Figura 26 – Ganho de desempenho para os 8 *benchmarks* do SPEC

Este estudo preliminar, embora não tenha sido o foco principal do trabalho, foi realizado para pré-avaliar o uso do Perceptron como um mecanismo de previsão, as simulações mostraram que o Perceptron pode ser usado na previsão de desvios, entretanto, previsores mais agressivos devem ser modelados. Além disso, as simulações sobre rastros não refletem a real situação enfrentada por um previsor na arquitetura superescalar. A implementação do previsor diretamente em um simulador de arquitetura superescalar e a execução sobre programas reais pode refletir mais corretamente a dinâmica do que realmente ocorre em processadores superescalares e os resultados são mais confiáveis. Desta forma, modelos de previsores mais agressivos foram implementados no simulador sim-bpred (veja Capítulo 6.1), os quais são descritos a seguir.

5.2. Modelos de Previsão de Desvios Mais Agressivos

O modelo conceitual básico da previsão de desvios baseada no Perceptron trabalha conforme mostra a Figura 27, onde o Perceptron prevê o resultado da próxima instrução de desvio dentro de um fluxo contínuo de instruções de desvios. O contador de instruções (PC) aponta para o desvio buscado, que deve ser previsto para posterior execução. O Perceptron usa como entrada o histórico dos desvios anteriores ao desvio a ser previsto e faz a previsão baseada em seus pesos sinápticos. Após a execução real do desvio, o histórico é atualizado, deslocando-se no fluxo de desvios de forma a incorporar o novo resultado do desvio. Os pesos sinápticos são ajustados na medida em que o histórico se modifica no tempo, se adaptando aos padrões de 0s e 1s que se formam. Os modelos aqui propostos são baseados neste modelo conceitual e possuem o limiar de ativação (Θ) e a taxa de aprendizado (η) definidos por Jiménez [23, 24]. O limiar de ativação é uma relação do tamanho do histórico (h) definido como $\Theta = 1{,}93h + 14$, obtido de forma empírica por Jiménez [23, 24]. A taxa de aprendizado é $\eta = +1$ ou $\eta = -1$, dependendo da previsão. O número de entradas e de pesos do Perceptron varia de 2 a 64, dependendo do histórico.

Diferentes modelos de previsor foram concebidos para implementação e avaliação. Os modelos UNI e TIP não possuem equivalência com outros previsores

da literatura estudada; o modelo END apresenta modificações sobre um previsor baseado no Perceptron existente [23, 24] e os modelos DNT e DNE aplicam conceitos de previsores de dois níveis [56, 57] existentes sobre os modelos originais TIP e END. O detalhamento de cada modelo proposto é descrito a seguir.

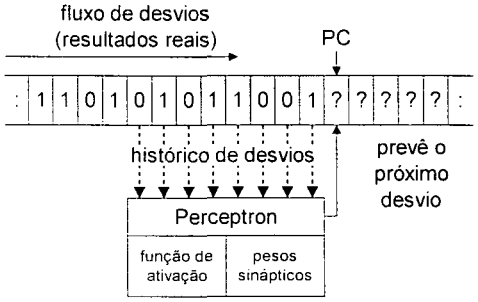


Figura 27 – Modelo Conceitual da Previsão de Desvios

5.2.1. O Modelo UNI

O previsor UNI é um previsor com um único Perceptron, ao qual todas as instruções de desvios são encaminhadas. Este modelo possui uma organização única que armazena o histórico corrente dos desvios (entradas do Perceptron) e os pesos sinápticos do Perceptron. A Figura 28 esquematiza a representação deste modelo de previsor.

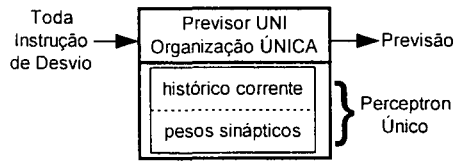


Figura 28 – Representação do previsor com um Perceptron e organização ÚNICA

5.2.2. O Modelo TIP

O previsor TIP corresponde ao previsor que possui um Perceptron associado a cada tipo de desvio condicional do conjunto de instruções da arquitetura superescalar do SimpleScalar. Todo desvio a ser previsto tem seu tipo determinado em tempo de execução e indexa a tabela de Perceptrons. Os tipos de desvio condicionais possíveis na arquitetura do SimpleScalar são: beq, bne, blez, bgtz, bltz, bgez, bc1f e bc1t [7]. Este modelo define um previsor com uma tabela com oito Perceptrons, um para cada tipo de desvio.

Este modelo e os seguintes possuem quatro diferentes tipos de organização, dependentes de como é armazenado o histórico dos desvios no previsor e como a previsão de desvio final é calculada.

A Figura 29 mostra a representação do modelo TIP com organização LOCAL. Nele, cada entrada da tabela de Perceptrons possui associada sua própria área de histórico e de pesos. Cada Perceptron calcula a saída do previsor com base no histórico e pesos locais para cada tipo de desvio.

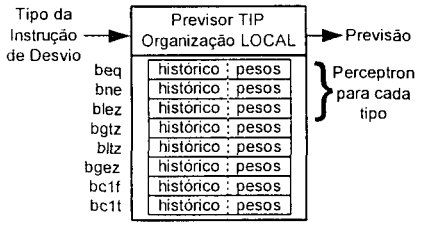


Figura 29 – Representação do previsor por tipo de desvio com organização LOCAL

Na segunda organização do modelo TIP, a GLOBAL, há uma única área de histórico associada a todas as entradas da tabela de Perceptrons e diferentes áreas de pesos associadas a cada Perceptron. Cada Perceptron calcula a saída do previsor com base no histórico global e pesos locais para o tipo de desvio correspondente, atualizando após cada previsão seus pesos locais e o histórico global. Este modelo está representado na Figura 30.

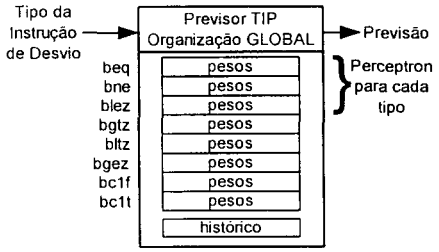


Figura 30 – Representação do previsor por tipo de desvio com organização GLOBAL

O terceiro tipo de organização para o modelo TIP, chamado LG_AND, calcula a saída do previsor fazendo um E-lógico do resultado da previsão feita com a organização LOCAL com o resultado da previsão feita com a organização GLOBAL. A Figura 31 representa este modelo e organização.

O quarto tipo de organização para o modelo TIP, chamado LG_OR, é análogo à organização anterior, contudo calcula a saída do previsor fazendo um OU-lógico dos resultados das previsões feitas com as organizações LOCAL e GLOBAL. A Figura 32 representa este modelo e organização.

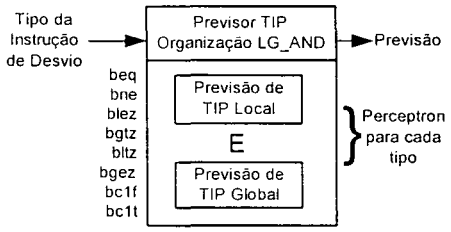


Figura 31 – Representação do previsor por tipo de desvio com organização LG_AND

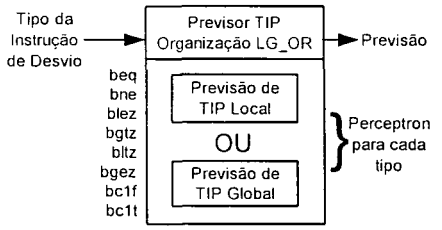


Figura 32 – Representação do previsor por tipo de desvio com organização LG_OR

5.2.3. O Modelo END

O previsor END corresponde ao previsor que define seu resultado com relação a cada endereço diferente de desvio condicional do programa avaliado. Além dos quatro tipos diferentes de organização já apresentados, este previsor bem como os demais (DNT e DNE) foram implementados e avaliados variando o número de linhas e a associatividade da tabela de Perceptrons. Foram quinze os agrupamentos "linhas/associatividade" de tabela utilizados nas simulações, conforme descrito na Tabela 1, mantendo o número máximo de Perceptrons igual a 1024.

Tabela 1 – Relação número de linhas/associatividade definindo o total de células (tamanho) da tabela de Perceptrons para os modelos END e DNE

Células / Linhas	64	128	256	512	1024
64	64/1	64/2	64/4	64/8	64/16
128	-	128/1	128/2	128/4	128/8
256	-	-	256/1	256/2	256/4
512	-	-	-	512/1	512/2
1024	-	-	-	-	1024/1

Um determinado endereço de desvio indexa a tabela de Perceptrons através de uma função *hash* que é a função módulo do endereço de desvio pelo número de linhas da tabela.

Para a associatividade igual a 1 (mapeamento direto), pode ocorrer conflito de indexação para diferentes endereços e a previsão fica comprometida pela interferência de diferentes desvios no mesmo Perceptron. Caso a associatividade empregada seja maior que 1 e ocorra conflito de indexação na mesma linha da

tabela de Perceptrons, o previsor procura seqüencialmente dentro da linha, uma célula de Perceptron já alocada anteriormente para aquele endereço específico, ou uma célula que não esteja sendo utilizada por nenhum desvio e a aloca para este endereço. Não havendo mais células livres, o previsor aplica o algoritmo LRU para substituição de uma célula já ocupada.

Como pode ser visto na Tabela 1, foi aplicado um número mínimo de 64 e um máximo de 1024 células na tabela de Perceptrons deste previsor. A associatividade variou de 1 a 16.

A Figura 33 mostra a representação do modelo do previsor END com organização LOCAL. Nele, cada célula diferente da tabela de Perceptrons referenciada pelo endereço de desvio e pela ocorrência de conflito possui sua própria área de histórico e de pesos. Cada Perceptron calcula a saída do previsor com base no histórico e pesos locais.

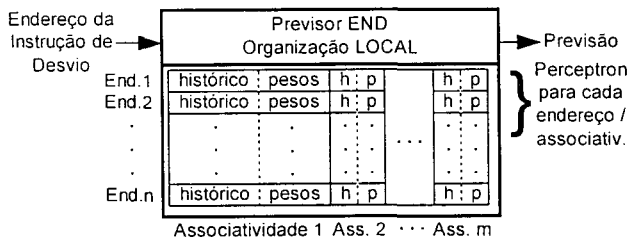


Figura 33 – Representação do previsor por endereço com linhas e associatividade da tabela de Perceptrons variáveis e organização LOCAL para cada endereço

O modelo de previsor END com organização GLOBAL difere do anterior pelo fato de não haver históricos locais e sim um único histórico global associado aos Perceptrons. Este tipo de modelo e organização é o mesmo desenvolvido por Jiménez e Lin [23, 24]. Entretanto, o previsor de Jiménez e Lin foi implementado, de forma limitada, com um número reduzido de linhas e com nível de associatividade fixo em 1 para a tabela de Perceptrons, ocasionando interferência na previsão. A Figura 34 representa este previsor.

A organização LG_AND do modelo END calcula a saída do previsor empregando a função E-lógico do resultado da previsão feita com a organização LOCAL com o resultado da previsão feita com a organização GLOBAL. Esta representação é vista na Figura 35.

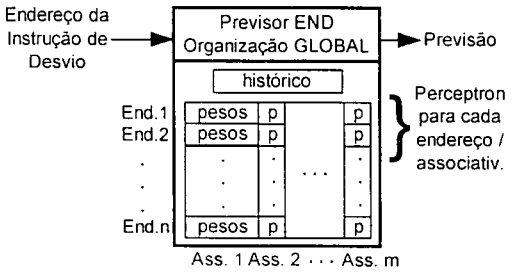


Figura 34 – Representação do previsor por endereço com linhas e associatividade da tabela de Perceptrons variáveis e organização GLOBAL para cada endereço

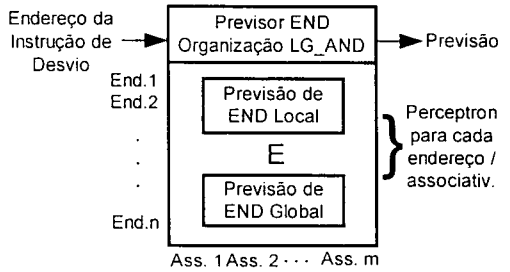


Figura 35 – Representação do previsor por endereço com linhas e associatividade da tabela de Perceptrons variáveis e organização LG_AND para cada endereço

Da mesma forma, a organização LG_OR do modelo END calcula a saída do previsor através de um OU-lógico dos resultados das previsões realizadas pelas organizações LOCAL e GLOBAL. Veja a Figura 36.

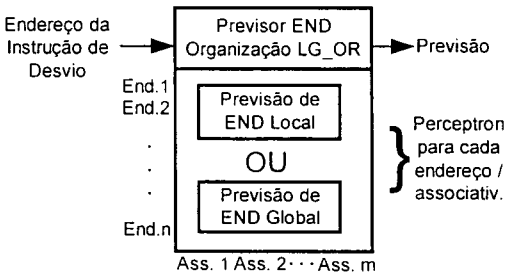


Figura 36 – Representação do previsor por endereço com linhas e associatividade da tabela de Perceptrons variáveis e organização LG_OR para cada endereço

5.2.4. O Modelo DNT

Aproveitando os conceitos de previsões de desvios de dois níveis apresentados por Yeh e Patt [56, 57], foi desenvolvido um modelo de previsor de dois níveis aplicado aos tipos de desvios, nomeado DNT. A característica deste modelo é que a previsão de desvio é feita sobre o comportamento de um padrão de histórico por tipo, da mesma forma que o proposto por Yeh e Patt, entretanto o mecanismo utilizado para fazer a previsão é o Perceptron.

Para cada tipo de desvio há uma tabela que armazena em primeiro nível os respectivos históricos. Cada um destes históricos indexa, no segundo nível, uma tabela de Perceptrons a qual é usada para armazenar o padrão de ocorrência de cada histórico do primeiro nível.

Este tipo de previsor foi implementado e avaliado variando o tamanho da tabela de Perceptrons no segundo nível, de 64 a 1024 linhas, mantendo a associatividade 1.

Para a organização LOCAL, o histórico do padrão (específico por Perceptron) e os pesos associados ao Perceptron são armazenados localmente, conforme visto na Figura 37.

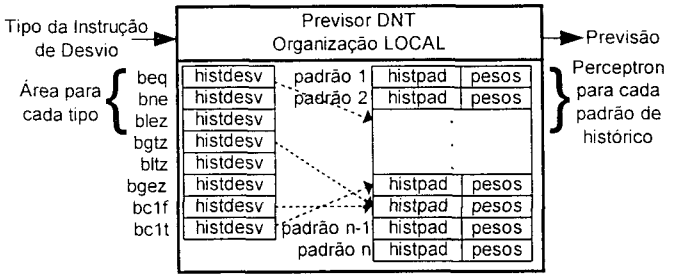


Figura 37 – Representação do previsor de dois níveis por tipo com linhas da tabela de segundo nível de Perceptrons variáveis e organização LOCAL para cada tipo

Para a organização GLOBAL, o histórico do padrão ocorrido anteriormente é armazenado em um único registrador global compartilhado entre todos o Perceptrons e os pesos associados ao Perceptron são armazenados localmente, conforme visto na Figura 38.

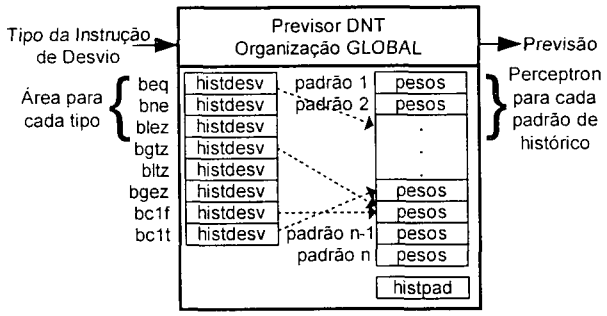


Figura 38 – Representação do previsor de dois níveis por tipo com linhas da tabela de segundo nível de Perceptrons variáveis e organização GLOBAL para cada tipo

A organização LG_AND do modelo DNT calcula a saída do previsor empregando a função E-lógico do resultado da previsão feita com a organização LOCAL com o resultado da previsão feita com a organização GLOBAL, de maneira análoga aos modelos anteriores. A Figura 39 representa esta organização.

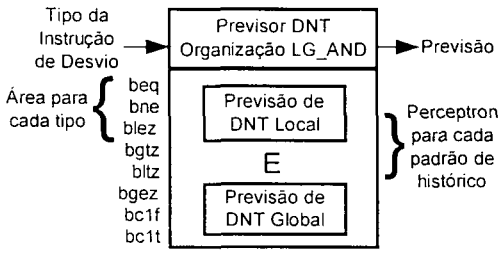


Figura 39 – Representação do previsor de dois níveis por tipo com linhas da tabela de segundo nível de Perceptrons variáveis e organização LG_AND para cada tipo

A organização LG_OR do modelo DNT calcula a saída do previsor através de um OU-lógico dos resultados das previsões LOCAL e GLOBAL, de maneira análoga à organização anterior. A Figura 40 representa este modelo e organização.

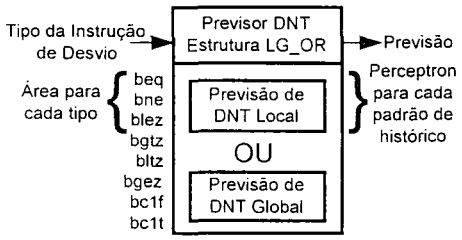


Figura 40 – Representação do previsor de dois níveis por tipo com linhas da tabela de segundo nível de Perceptrons variáveis e organização LG_OR para cada tipo

5.2.5. O Modelo DNE

O modelo do previsor DNE é análogo ao DNT, entretanto o primeiro nível é indexado pela função módulo dos endereços dos desvios pelo número de linhas da tabela, da mesma forma que o modelo de previsor END. A forma de cálculo da previsão e o mecanismo de utilização dos Perceptrons são os mesmos do DNT.

Este tipo de previsor foi implementado e avaliado com quinze diferentes variações de número de linhas e associatividade para a tabela do segundo nível da mesma forma que o modelo END conforme a Tabela 1.

Para a organização LOCAL, o histórico do padrão ocorrido anteriormente e os pesos associados ao Perceptron também são armazenados localmente como no modelo END organização LOCAL. Esta representação está na Figura 41.

Para a organização GLOBAL deste modelo, o histórico do padrão ocorrido anteriormente é armazenado em um registrador global e os pesos associados ao Perceptron são armazenados localmente, conforme visto na Figura 42.

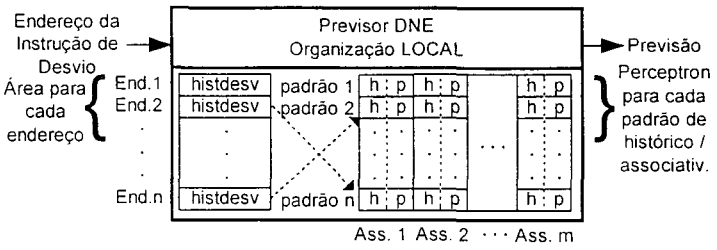


Figura 41 – Representação do previsor de dois níveis por endereço com linhas e associatividade de tabela de segundo nível de Perceptrons variáveis e organização LOCAL

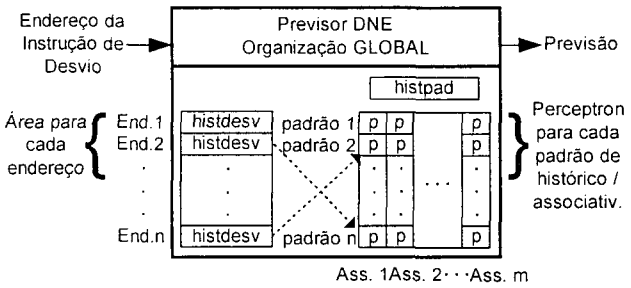


Figura 42 – Representação do previsor de dois niveis por endereço com linhas e associatividade de tabela de segundo nível de Perceptrons variáveis e organização GLOBAL

A organização LG_AND do modelo DNE, representada na Figura 43, calcula a saída do previsor usando a função E-lógico dos resultados das previsões obtidos das organizações LOCAL e GLOBAL, da mesma forma que os modelos anteriores.

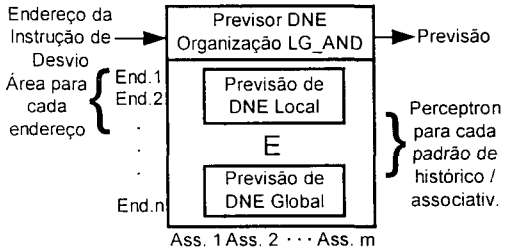


Figura 43 – Representação do previsor de dois níveis por endereço com linhas e associatividade de tabela de segundo nível de Perceptrons variáveis e organização LG_AND

A organização LG_OR do modelo DNE (Figura 44) calcula a saída do previsor usando uma função OU-lógico dos resultados de LOCAL e GLOBAL.

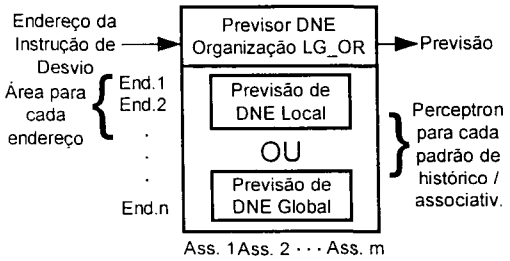


Figura 44 – Representação do previsor de dois níveis por endereço com linhas e associatividade de tabela de segundo nível de Perceptrons variáveis e organização LG_OR

6. Implementação e Simulação

Os diferentes modelos de previsor baseado no Perceptron foram implementados no simulador sim-bpred. A implementação diferenciada sobre o mesmo tipo de previsor baseado no Perceptron permite uma análise diversificada e abrangente do comportamento deste tipo de previsor numa arquitetura superescalar. Para todos os modelos, foram avaliadas de 2 a 64 entradas do Perceptron (histórico dos desvios) no previsor.

6.1. O Simulador sim-bpred

O uso de ferramentas próprias para avaliação do previsor de desvios contribui para o entendimento do mecanismo de previsão e análise do seu funcionamento. Contudo, para uma avaliação segura e criteriosa procurou-se utilizar ferramentas já empregadas na comunidade científica.

Um ambiente de simulação amplamente utilizado nos estudos da arquitetura superescalar é o conjunto de ferramentas SimpleScalar [7]. Ele é composto de bibliotecas de funções e simuladores prontos para avaliação de componentes das arquiteturas superescalares, tais como *caches* de instrução e de dados, unidades funcionais, previsão de desvios e outros.

O sim-bpred é o simulador específico para configurar e avaliar previsores de desvios. A execução desta ferramenta é realizada através de linha de comando passando como parâmetros o tipo de previsor, as configurações das estruturas utilizadas e o *benchmark* específico a ser simulado.

A estrutura de funcionamento do simulador sim-bpred pode ser visto na Figura 45. O *benchmark* é carregado na memória do simulador e suas instruções são executadas uma a uma; as instruções de desvio são encaminhadas ao previsor definido, e este determina se elas serão tomadas ou não-tomadas; após a execução do desvio as estruturas de previsão são atualizadas. A definição da estrutura e as atividades do previsor estão representadas pelos quadros em cinza.

O sim-bpred possui alguns previsores de desvio já implementados em seu código [7]. São eles: não-tomado (*nottaken*), tomado (*taken*), perfeito (*perfect*), bimodal (*bimod*), dois níveis (*2lev*) e combinado (*comb*). Os previsores não-tomado e

tomado correspondem às políticas de prever sempre não-tomado e sempre tomado, respectivamente. O previsor perfeito é a implementação de um previsor ideal, que acerta sempre sua previsão. O previsor bimodal é um previsor BTB com contador de dois bits [6, 30, 39]. O previsor dois níveis corresponde a um previsor adaptativo de dois níveis [56, 57]. E finalmente, o previsor combinado utiliza uma estrutura combinada do previsor bimodal com o de dois níveis [34].

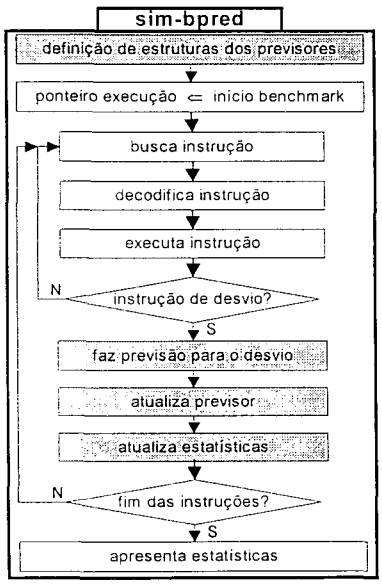


Figura 45 – Estrutura de funcionamento do sim-bpred

6.2. Mudanças no Simulador sim-bpred

Primeiramente, o sim-bpred foi alterado de forma a permitir o descarte de instruções iniciais dos *benchmarks* em execução. Este procedimento possibilita saltar código tendencioso de iniciação para que as previsões sejam feitas na fase de execução estável.

Posteriormente foram implementados os modelos de previsor propostos. Foram alterados para incluir estas implementações: o arquivo sim-bpred.c, que possui o código da estrutura, passagem de parâmetros, funcionamento e estatísticas do simulador e chamada aos códigos dos previsores de desvio; o arquivo bpred.h, que contém o código de interface e definição das estruturas de previsão; e o arquivo bpred.c que contém o código de iniciação e funcionamento dos previsores. Dois novos arquivos foram agregados ao simulador, o arquivo pcp.h que contém o código o código de interface e definição da estrutura dos previsores baseados no

perceptron e o arquivo pcp.c que contém o código de iniciação e funcionamento dos previsores baseados no Perceptron.

O sim-bpred também passou a incluir código para a contabilização da previsão pelo tipo de instrução de desvio existente no conjunto de instruções da arquitetura SimpleScalar. Desta forma, é possível visualizar o comportamento do previsor para cada tipo específico de desvio da arquitetura. A estrutura de funcionamento alterada do sim-bpred pode ser visto na Figura 46. Os quadros em cinza correspondem às áreas alteradas para incluir as diferentes variações do previsor baseado no Perceptron.

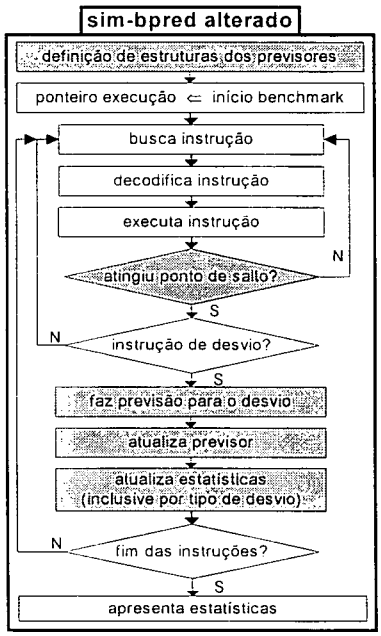


Figura 46 – Estrutura alterada de funcionamento do sim-bpred

6.3. Pormenores das Simulações

Para as simulações e avaliação dos previsores implementados foram utilizados os *benchmarks* do SPEC [49]. Foram usados quatro *benchmarks* de inteiros (*cc1, ijpeg, li e perl*) e quatro *benchmarks* de ponto flutuante (*fpppp, mgrid, swim e wave5*). Os *benchmarks* foram executados no simulador sim-bpred alterado, de forma que as primeiras 100 milhões de instruções dos *benchmarks* foram descartadas e não contabilizadas. As seguintes um bilhão de instruções foram utilizadas para previsão e contabilização de resultados.

Com o intuito de automatizar as tarefas de simulação e agilizar a obtenção de resultados dos diversos *benchmarks* avaliados foram criados scripts para auxiliar na chamada de execução do simulador, parametrizando os diversos modelos, organizações, número de linhas e associatividade das tabelas dos previsores. Os resultados obtidos foram filtrados e organizados em tabelas ordenadas em arquivos texto. Todas as variações possíveis para os diversos parâmetros foram simuladas.

Os cinco modelos de previsores (UNI, TIP, END, DNT e DNE), os cinco tipos de organizações (Única – para UNI; LOCAL, GLOBAL. LG_AND e LG_OR – para TIP, END, DNT e DNE), os diferentes números de linhas de tabela (1 – para UNI; 8 – para TIP; 64, 128, 256, 512 e 1024 – para END, DNT e DNE), as diferentes associatividades (1 – para UNI, TIP, END, DNT e DNE; 2, 4, 8 e 16 – para END e DNE) e os diferentes tamanhos de históricos (2, 4, 8, 16, 32 e 64 – para todos) para os oito diferentes *benchmarks* (*cc1, ijpeg, li, perl, fpppp, mgrid, swim e wave5*) foram combinados e simulados, resultando em 83.520 medidas de desempenho (porcentagem de acerto). Estas medidas foram contabilizadas para o previsor como um todo (prev), para os desvios para frente e para trás (forw e back, respectivamente), para os oito tipos de desvio condicional (beq, bne, blez, bgtz, bltz, bgez, bc1f e bc1t) e para os desvios incondicionais (jump).

Nas tabelas, figuras e textos a serem apresentados a seguir designaremos o modelo de previsão como MOD, o tipo de organização do previsor como ORG, o número de linhas da tabela do previsor como LIN, o número de associatividade da tabela como ASS, o *benchmark* avaliado como PRG, o desvio avaliado como DSV e valor da porcentagem de acerto resultante como VLR. A Tabela 2 sintetiza os valores possíveis para estes parâmetros.

Tabela 2 – Combinação dos valores possíveis dos parâmetros das simulações

<table>
<tr><th>MOD</th><th>ORG</th><th>LIN</th><th>ASS</th><th>PRG</th><th>DSV</th><th>VLR</th></tr>
<tr><td>UNI</td><td>UNICA</td><td>1</td><td>1</td><td rowspan="6">cc1,
ijpeg,
li,
perl,
fpppp,
mgrid,
swim,
wave5.</td><td rowspan="6">prev, beq, bne, blez, bgtz, bltz, bgez, bc1f, bc1t, jump, forw, back.</td><td rowspan="6">Valor representando a porcentagem de acerto.</td></tr>
<tr><td>TIP</td><td rowspan="5">LOCAL,
GLOBAL,
LG_AND,
LG_OR.</td><td>8</td><td>1</td></tr>
<tr><td>DNT</td><td rowspan="4">64,
128,
256,
512,
1024</td><td>1</td></tr>
<tr><td>END</td><td rowspan="3">1, 2, 4, 8, 16
dependendo de LIN.</td></tr>
<tr><td>DNE</td></tr>
</table>

As simulações foram realizadas em servidores com processamento superior a 1 Ghz e levaram semanas de execução contínua para o término das diversas combinações, gerando um total de 1.320 arquivos texto de resultados.

6.4. Metodologia para a Visualização de Resultados

Como visto, o volume de informação obtido pelas simulações para análise foi muito grande. Assim, para facilitar a geração de gráficos e tabelas, os dados inicialmente armazenados em arquivo texto foram transportados para um servidor de banco de dados MySQL [3], onde foi criado uma tabela no formato MOD/ORG/LIN/ASS/PRG/DSV/VLR. Outra tabela auxiliar mantém o valor da média dos resultados do previsor (prev) para todos os *benchmarks*. Através de consultas a este banco de dados, os gráficos foram gerados por um sistema em linguagem PHP [4] sobre o servidor de WEB APACHE [1], utilizando a biblioteca gráfica jpGraph [40], podendo então serem visualizados por um navegador de páginas WEB. Todo o processo de simulação e geração de resultados foi realizado no sistema operacional Linux.

A geração de resultados pôde ser feita de forma dinâmica segundo as escolhas realizadas nas telas de seleção de dados deste sistema de geração de resultados. A interface do sistema de geração de gráficos é vista na Figura 47.

Os gráficos de linha gerados representam a evolução do resultado da previsão (valores de porcentagem de acerto) segundo o aumento do tamanho do histórico de desvios, onde cada linha representa um destes: desvio, *benchmark*, organização, número de linhas e associatividade da tabela ou modelo do previsor segundo o agrupamento utilizado. O gráfico de barra gerado é o gráfico de ações (*stock-charts*) que indica o valor máximo e mínimo, o primeiro e terceiro quartil e a mediana (segundo quartil) dos valores de acerto da média dos *benchmarks* do resultado (prev) de um determinado previsor, individualizados em gráficos distintos segundo seus modelos, organizações, número de linhas e associatividades da tabela (MOD/ORG/LIN/ASS), possibilitando avaliar a dispersão dos resultados obtidos com maior qualidade.

Os gráficos gerados de maior relevância estão nos apêndices de 1 a 7 conforme sua classificação.

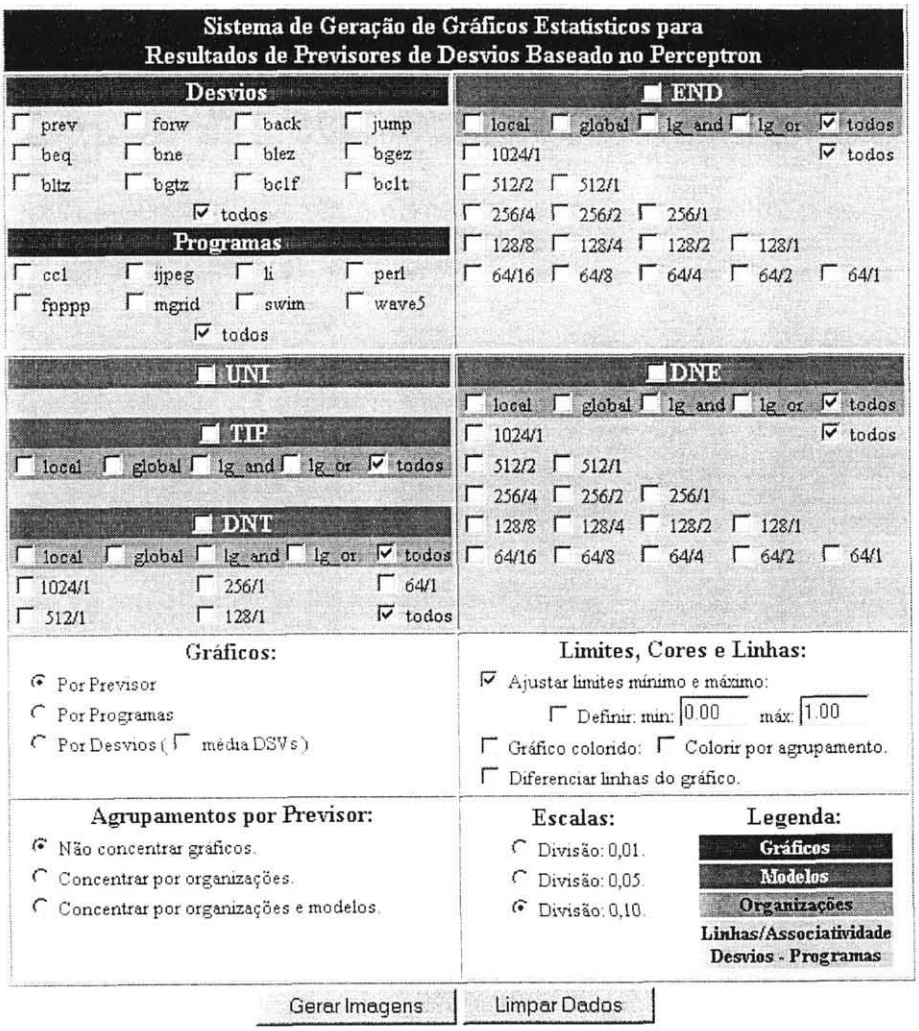


Figura 47 – Interface para geração de gráficos de resultados

7. Avaliação de Desempenho

Para facilitar a análise de desempenho dos previsores fez-se uma contagem da quantidade de instruções de desvios (desv), porcentagens de ocorrência por desvio e o número de instruções por desvios (ipd) para cada *benchmark*, descritas na Tabela 3. Antes de apresentar as especificidades de cada modelo de previsão, algumas características gerais podem ser relatadas conforme segue.

Tabela 3 – Porcentagem de Cada Desvio por Programa

	Porcentagem do Desvio para Cada Programa								Todos PRG
	mgrid	wave5	swim	fpppp	li	perl	ijpeg	cc1	Média Total
desv	12603966	139515952	38179080	15136289	16859505	80928667	75137695	198534259	576895413
prev	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
beq	96,97%	16,27%	36,75%	41,49%	32,83%	46,93%	11,39%	33,98%	30,29%
bne	2,46%	17,45%	14,16%	8,20%	24,13%	29,11%	59,88%	37,26%	30,84%
blez	0,21%	0,01%	0,06%	1,21%	0,00%	0,81%	0,54%	1,38%	0,70%
bgtz	0,02%		0,00%		0,89%	0,00%	0,72%	0,19%	0,19%
bltz	0,02%		0,00%	1,89%	0,00%	0,00%	0,01%	1,42%	0,54%
bgez	0,03%	5,17%	0,00%	0,35%	0,00%	0,82%	9,31%	3,11%	3,66%
bc1f		11,52%	12,14%	10,24%			0,00%		3,86%
bc1t	0,00%	2,91%	1,13%	11,55%			0,00%		1,08%
jump	0,29%	46,67%	35,76%	25,07%	42,15%	22,33%	18,15%	22,67%	28,85%
back	2,83%	48,04%	32,80%	66,52%	47,61%	60,48%	44,83%	61,50%	52,48%
forw	96,88%	5,28%	31,44%	8,41%	10,24%	17,19%	37,02%	15,83%	18,67%
ipd	79,34	7,17	26,20	66,07	4,24	5,24	13,31	5,04	

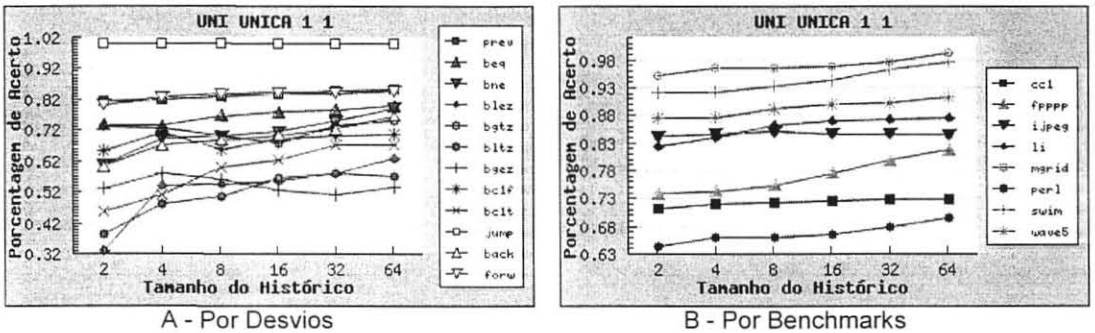
Pela observação dos dados gerados, os previsores END, DNE e DNT apresentaram aumento crescente de desempenho com o aumento do tamanho da tabela de Perceptrons, seja pelo aumento do número de linhas da tabela (LIN) ou pelo aumento da associatividade (ASS).

Para os gráficos de desempenho por *benchmark* foi observado que os melhores resultados sempre foram para *benchmarks* de ponto flutuante (mgrid para UNI e TIP, e swim para END, DNE e DNT) e os piores resultados foram para *benchmarks* de inteiro (perl para UNI e cc1 para os outros modelos). Isto acontece pela maior ocorrência de desvios em um espaçamento de instruções menor (ipd) nos programas de intero que nos programas de ponto flutuante, gerando uma maior interferência entre desvios nos previsores.

Para todos os gráficos por desvio apresentados, os desvios incondicionais (jump) foram previstos com taxa de acerto de 100%, já que seus destinos sempre são conhecidos (desvios sempre tomados).

7.1. Resultados de UNI

Analisando os resultados do previsor com Perceptron único foi possível verificar que, para cada tipo de desvio, a taxa média de acerto de previsão dos resultados por histórico em relação a todos os *benchmarks* ficou em torno de 83,41%. Os desvios para frente (forw) tiveram um desempenho de 83,25% na taxa de acerto contra 69,26% para os desvios para trás (back). Isto mostra que o este previsor teve seu desempenho regido principalmente pelo acerto na previsão dos desvios forw, cujo resultado está muito próximo do valor obtido para o previsor como um todo. Todos os tipos de desvios tiveram as taxas de acerto com resultados abaixo da média, e individualmente foram somente três dos oito tipos possíveis de desvio que atingiram médias acima de 70% (beq: 76,73%, bne: 70,94% e bgtz: 72,14%). Devemos considerar obviamente que os desvios incondicionais foram computados na média do previsor, elevando seu índice. Esta mesma consideração deve ser levada em conta na análise por desvios dos modelos seguintes. O desvio beq teve melhor desempenho e o pior desempenho foi para bltz com 51,40% de taxa de acerto. Todos os desvios ou tiveram taxa de acerto crescente ou taxa estável com variações, para o aumento do histórico de entrada. Estes dados podem ser vistos na Figura 48-A.



A - Por Desvios B - Por Benchmarks

Figura 48 – Resultados por desvios e benchmarks para o modelo UNI

Os resultados da previsão agrupados por *benchmark* podem ser vistos na Figura 48-B. Todos os *benchmarks* tiveram taxa de acerto suavemente crescente para o aumento de histórico. O maior valor da média dos valores por histórico foi para o *benchmark* mgrid com 97,02% e o menor valor foi de 66,76% para perl. Apenas três *benchmarks* (cc1, fpppp e perl) tiveram desempenho abaixo do desempenho médio do previsor. A ordem crescente da média de desempenho para

os *benchmarks* foi: mgrid (97,02%), swim (94,32%), wave5 (89,38%), li (85,71%), ijpeg (84,60%), fpppp (77,10%), cc1 (72,37%), e finalmente perl (66,76%).

7.2. Resultados de TIP

Este modelo de previsor foi avaliado em quatro diferentes tipos de organização, sendo fixadas o número de linhas e a associatividade da tabela. Esta seção mostra os resultados por desvio e por *benchmark* para cada tipo.

O gráfico por desvio da organização LOCAL é mostrado na Figura 49-A. A média do previsor para esta organização foi de 69,50%, com 76,01% de taxa de acerto para os desvios para frente e 39,50% para os desvios para trás. Foi possível observar que o desvio bgtz teve o melhor resultado, com média de 84,56% e foi o único desvio a ter desempenho maior que a média do previsor. O pior resultado foi para o desvio blez, com 1,89% de taxa de acerto. Todos os desvios tiveram crescimento do desempenho suave ou estável com pequena variação para o aumento do histórico.

A organização GLOBAL apresentou um desempenho médio de 69,79% de taxa de acerto. Para o agrupamento por desvios, representado no gráfico da Figura 49-B, é possível observar que os desvios para frente tiveram um acerto de 76,42% e os desvios para trás de 39,88%. O melhor desempenho foi do desvio bgtz com média de 84,81%, sendo o único desvio a ultrapassar a média do previsor. O pior desempenho ficou para o desvio blez com 2,50% de taxa. Todos os desvios tiveram crescimento do desempenho suave ou estável com pequenas variações para o aumento do tamanho do histórico.

A organização LG_AND teve resultado médio de taxa de acerto em 68,16%. O desvio bgtz teve o melhor desempenho e foi o único a ultrapassar a média desta organização com 83,54% de taxa de acerto; e o pior desempenho foi para blez com 1,66%. Alguns dos desvios (beq, blez e bltz) tiveram crescimento do desempenho suave para o aumento do histórico, o desvio bgez teve redução de desempenho (7% do menor ao maior histórico) e os outros desvios mantiveram-se estáveis com pequena oscilação. Os desvios para frente tiveram um desempenho de 74,42% e os desvios para trás 37,22% de taxa média de acerto. A Figura 49-C apresenta o gráfico por desvios para a organização LG_AND do modelo TIP.

Todos os desvios tiveram crescimento do desempenho suave para o aumento do histórico na organização LG_OR. O maior resultado ficou para o desvio bgtz com 85,11% de taxa de acerto e o menor resultado para o desvio blez com 2,61% da média de acerto do desvio para todos os *benchmarks*. Os desvios para frente tiveram desempenho de 77,14% de taxa de acerto enquanto que os desvios para trás atingiram 41,43%. O previsor com a organização LG_OR atingiu a taxa média de acerto de 70,69% na previsão e bgtz foi o único desvio a ultrapassar este valor em seu desempenho. A Figura 49-D apresenta o gráfico por desvio para a organização LG_OR.

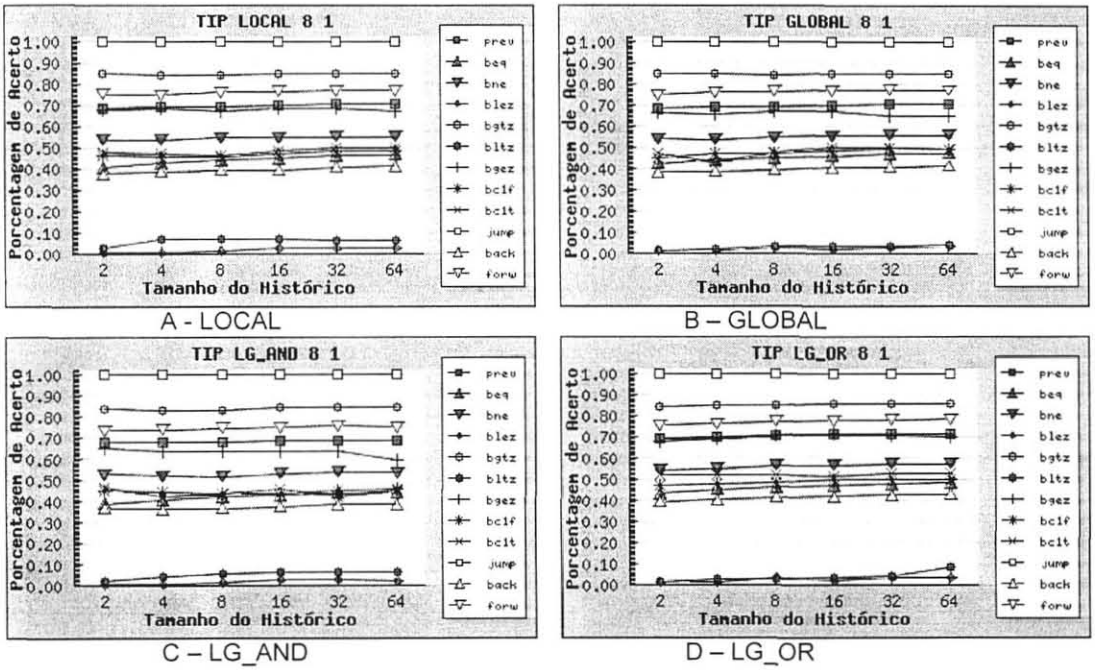


Figura 49 – Resultados por desvios para o modelo TIP

A média de resultados para os *benchmarks* na organização LOCAL ficou estável com pouca variação. O *benchmark* mgrid teve o melhor resultado com 94,98% de média de acerto e o pior resultado foi para cc1 com 48,97%. Outros três *benchmarks* além de mgrid tiveram desempenho acima da média, ijpeg com 71,83%, swim com 91,73% e wave5 com 85,98% de acerto para este modelo e organização. A única variação expressiva foi para o *benchmark* perl com o aumento do histórico de 16 para 32, resultando num salto de 52% para 57% de acerto nesta variação. A Figura 50-A apresenta o desempenho por *benchmarks* da organização LOCAL.

Na Figura 50-B, esta representado o gráfico por *benchmark* para a organização GLOBAL. A média de resultados por *benchmark* no aumento de

histórico foi estável com alguma variação (próximo a 4% do menor para o maior valor obtido por histórico) para o *benchmark* perl e crescimento leve para fpppp e li. O melhor resultado foi de mgrid com média de 95,01% e pior resultado para cc1 com 49,75% de média de acerto. Além de mgrid, ijpeg (72,16%), swim (91,27%) e wave5 (86,16%) ultrapassaram a média.

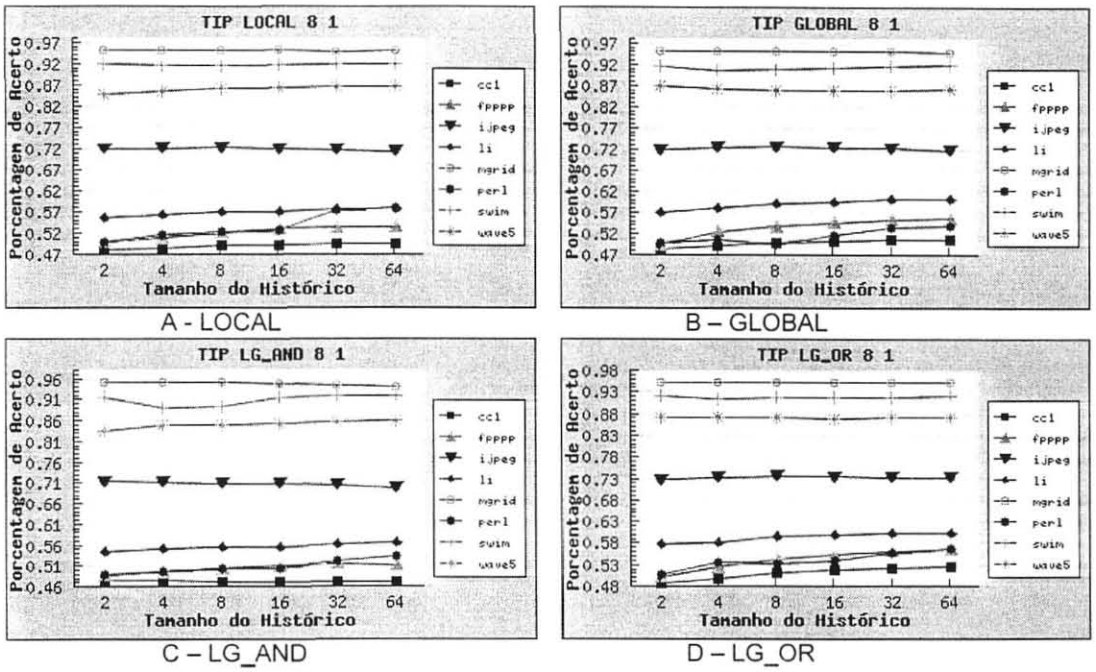
Para a organização LG_AND, o *benchmark* mgrid teve o melhor desempenho com 94,74% de taxa média de acerto, entretanto houve redução de desempenho na ordem de 1% com o aumento de histórico. O *benchmark* ijpeg também teve redução de desempenho em torno de 2%. O *benchmark* swim teve variação de desempenho em torno de 2% mas manteve desempenho estável. Os outros *benchmarks* tiveram crescimento leve. O pior resultado foi para o *benchmark* cc1 com 47,26% de acerto, e o acompanharam com desempenho abaixo da média os *benchmarks* fpppp (50,37%), li (55,50%) e perl (50,78%). A Figura 50-C apresenta estes dados.

Na organização LG_OR, alguns *benchmarks* (cc1, perl e fpppp) tiveram crescimento pouco mais acentuado que os outros. *Benchmarks* como wave5, ijpeg e swim apresentaram variações, mas mantiveram desempenho estável. O *benchmark* mgrid foi o que apresentou maior resultado com 95,04% de acerto e o menor resultado ficou com cc1 com média de 51% de acerto. Além de mgrid, ultrapassaram a média de desempenho os *benchmarks* ijpeg com 73,26%, swim com 91,77% e wave5 com 87,05% de taxa média de desempenho. A Figura 50-D apresenta o gráfico por *benchmarks* para a organização LG_OR.

A melhor organização para este modelo de previsor foi LG_OR com 70,69% de acerto, seguida pela organização GLOBAL com 69,79% de acerto. Em seguida a organização LOCAL atingiu 69,50% de acerto e o pior desempenho ficou para LG_AND com 68,16% de acerto. A diferença entre a organização de maior resultado e a segunda colocada foi de 0,90 pontos percentuais, a diferença da segunda para a terceira colocada foi de 0,29 pontos percentuais e a diferença entre o terceiro e quarto colocado foi de 1,34 pontos percentuais. Da organização com melhor desempenho para a de pior desempenho houve uma diferença de 2,53 pontos.

Fazendo uma média dos resultados obtidos para cada organização é atingido o valor de 69,53% de desempenho de acerto para o modelo de previsor TIP, definindo sua porcentagem de acerto média. Observamos que em todas as

organizações a ordem crescente da média de desempenho para os *benchmarks* foi: mgrid, swim, wave5, ijpeg, li, fpppp, perl e cc1; este comportamento é semelhante ao resultado do modelo UNI.



A - LOCAL B – GLOBAL

C – LG_AND D – LG_OR

Figura 50 – Resultados por benchmarks para o modelo TIP

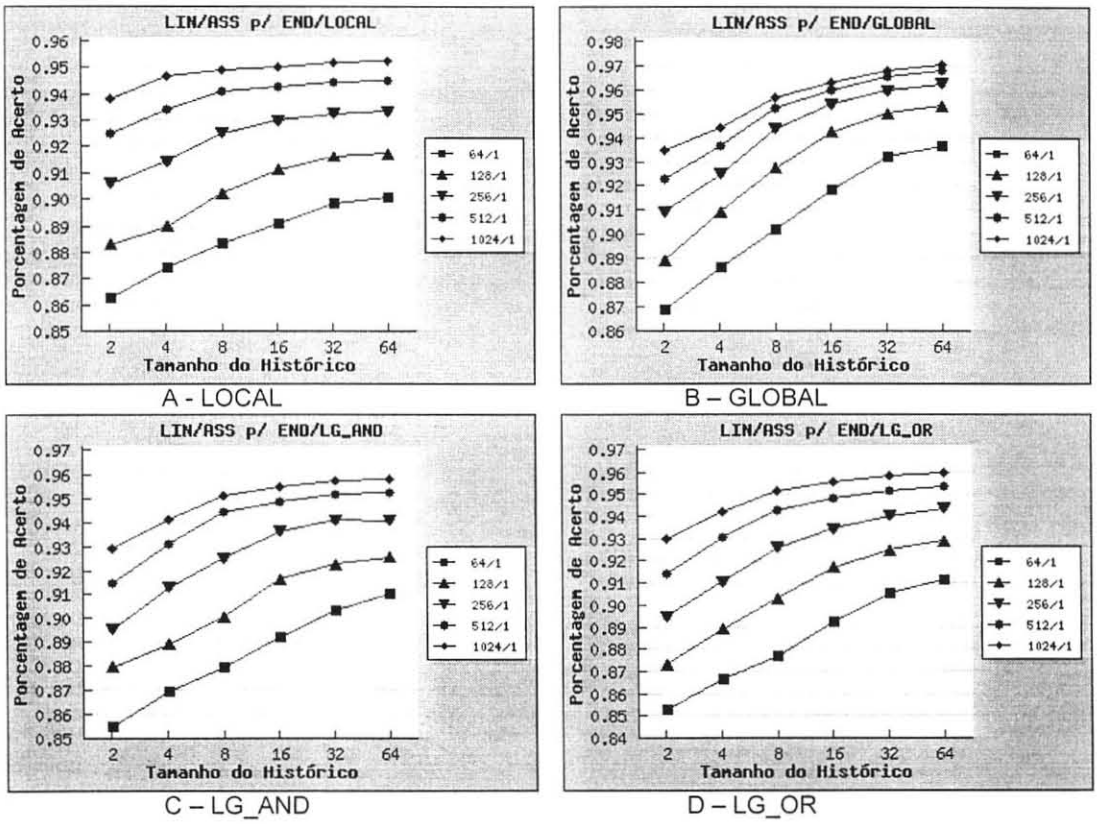
7.3. Resultados de END

Conforme já visto, este modelo de previsor foi avaliado sob diversas combinações de organização, linhas e associatividade de tabela, e dessa forma, a análise individual de cada gráfico resultante desta combinação seria muito extensa. Como o parâmetro de avaliação utilizado para comparar os previsores é o melhor caso para cada modelo, estaremos analisando os melhores casos para os agrupamentos por desvio e por *benchmark* para os quatro tipos de organização.

A Figura 51 apresenta os resultados para aumento de linhas de tabela e associatividade 1 para todas as organizações deste modelo de previsor. Observa-se que há aumento de desempenho com o aumento das linhas da tabela. O melhor resultado encontrado em todas as organizações com associatividade fixa em 1 foi para a relação LIN/ASS em 1024/1. A melhor organização para o modelo END foi a organização GLOBAL com 93,34% de acerto, seguido pelas organizações LOCAL e LG_OR com desempenhos iguais a 91,96% de taxa de acerto e o pior desempenho

ficou para LG_AND com 91,93% de taxa. A diferença da organização GLOBAL em relação à LOCAL foi de 1,38 pontos percentuais, a diferença de LOCAL e LG_OR em relação a LG_AND foi de 0,03 pontos percentuais. Da organização com melhor para a de pior desempenho houve uma diferença de 1,41 pontos percentuais.

O ganho do previsor com maior número de linhas em sua tabela para a que possui menos linhas foi de 7,12% para a organização LOCAL, 5,38% para GLOBAL, 7,19% para LG_AND e 7,40% para LG_OR.



A - LOCAL B – GLOBAL

C – LG_AND D – LG_OR

Figura 51 – Resultados para aumento de linhas da tabela para o modelo END

A porcentagem média de acerto para este modelo de previsor considerando todas as combinações foi de 92,30% de desempenho de acerto. O pior caso atingiu 88,23% de acerto no previsor LOCAL/64/2.

Para todas as organizações, os melhores resultados foram para a relação LIN/ASS com valor 1024/1. Os dados seguintes referem-se a esta relação. Para a organização LOCAL, o melhor resultado obtido foi de 94,80% de taxa média de acerto. Para a organização GLOBAL o melhor resultado obtido foi de 95,63% de taxa. A organização LG_AND obteve taxa média de acerto de 94,87% na previsão. Para a organização LG_OR o melhor resultado obtido foi de 95,96% de acerto.

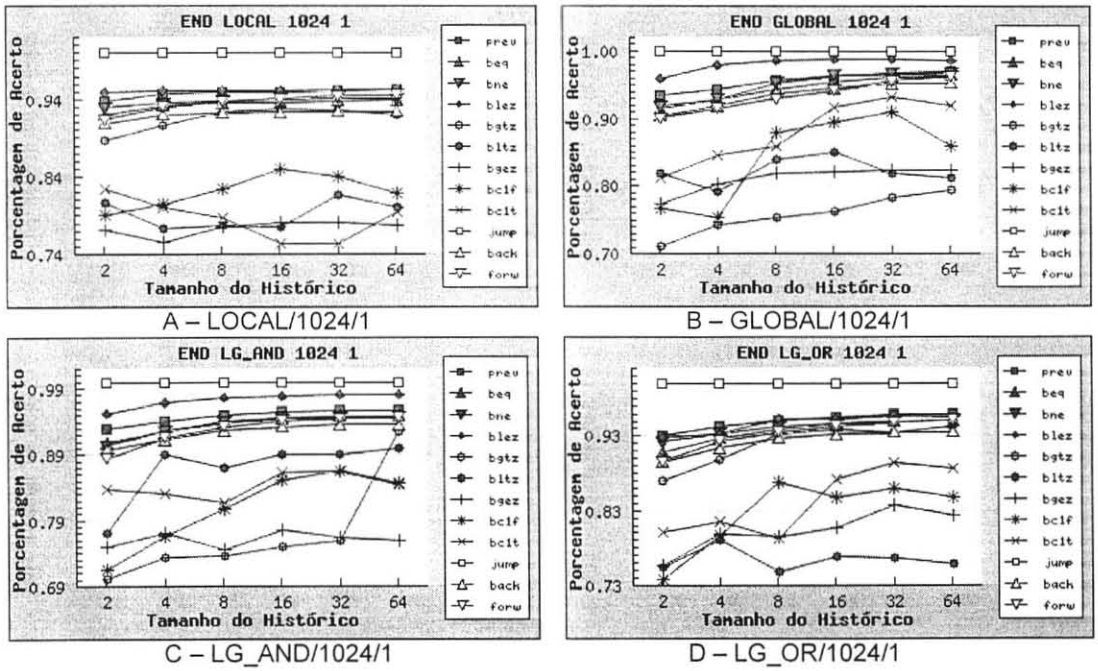
O agrupamento por desvios da estrutura LOCAL, representado na Figura 52-A, apresentou desvios (bltz, bgez, bc1f e bc1t) com grandes variações de desempenho (em torno de 7% do menor para o maior valor obtido por histórico), com redução de desempenho para bc1t, para o aumento do histórico. Os demais desvios tiveram crescimento suave ou estável. A melhor média de resultados foi de blez com 95,15% (único a ultrapassar a média) e o pior resultado foi de bgez com 77,32% de acerto. Os desvios para frente atingiram uma média de acerto de 93,57% de acerto e os desvios para trás o valor de 92,09%.

Na organização GLOBAL, o agrupamento por desvios, representado na Figura 52-B, apresentou desvios (bgtz, bltz, bgez, bc1f e bc1t) com grandes variações (topo de 16% entre menor e maior valor por histórico) de desempenho e os demais com crescimento suave, conforme o aumento do histórico. A melhor média de resultados foi de blez (único a ultrapassar a média do previsor) com 98,16% e o pior resultado foi de bgtz com 75,76% de taxa. Os desvios para frente atingiram uma taxa média de acerto de 93,47% de acerto e os desvios para trás o valor de 93,54%, desempenho equivalente entre ambos.

A Figura 52-C representa o agrupamento por desvios da organização LG_AND, ela mostra que apenas alguns desvios (beq, bne e blez) tiveram crescimento suave, bc1f e bgtz tiveram crescimento acentuado (11% e 22%, respectivamente) e os outros desvios tiveram grandes variações de desempenho, conforme o histórico aumentava. Os desvios para frente atingiram uma média de acerto de 92,83% e os desvios para trás o valor de 92,41% definindo desempenho equivalente para ambos. A melhor média de resultados foi de blez (único acima da média) com 97,28% e o pior resultado foi de bgez com 76,05% de taxa de acerto.

A Figura 52-D apresenta o agrupamento por desvios da organização LG_OR, o gráfico apresentou os desvios bltz, bgez, bc1f e bc1t com grandes variações de desempenho (topo de 17%), sendo que bltz manteve-se estável no crescimento e os outros três desvios apresentaram ganho no desempenho; os demais desvios tiveram crescimento suave, para o aumento do histórico. O pior resultado foi para o desvio bltz com 76,45% de acerto e a melhor média de resultados foi de blez com 94,65%. Nenhum desvio individualmente ultrapassou a média do previsor. Os desvios para trás apresentaram a taxa de desempenho no valor de 92,33% e os desvios para frente atingiram uma média de acerto de 93,31%.

Para a organização LOCAL, o *benchmark* com o melhor desempenho foi swim e o pior desempenho ficou para cc1. Para o agrupamento por *benchmark*, representado pela Figura 53-A, observamos que estes tiveram desempenho crescente suave na maioria dos casos, com um crescimento acentuado em alguns pontos para perl, fpppp e mgrid. O *benchmark* swim apresentou melhor resultado com 99,86% de acerto e a pior média foi para cc1 com 85,02%. Também ficaram abaixo da média, além de cc1, os *benchmarks* ijpeg com 91,43% e li com 93,94% de taxa média de desempenho.



A – LOCAL/1024/1 B – GLOBAL/1024/1

C – LG_AND/1024/1 D – LG_OR/1024/1

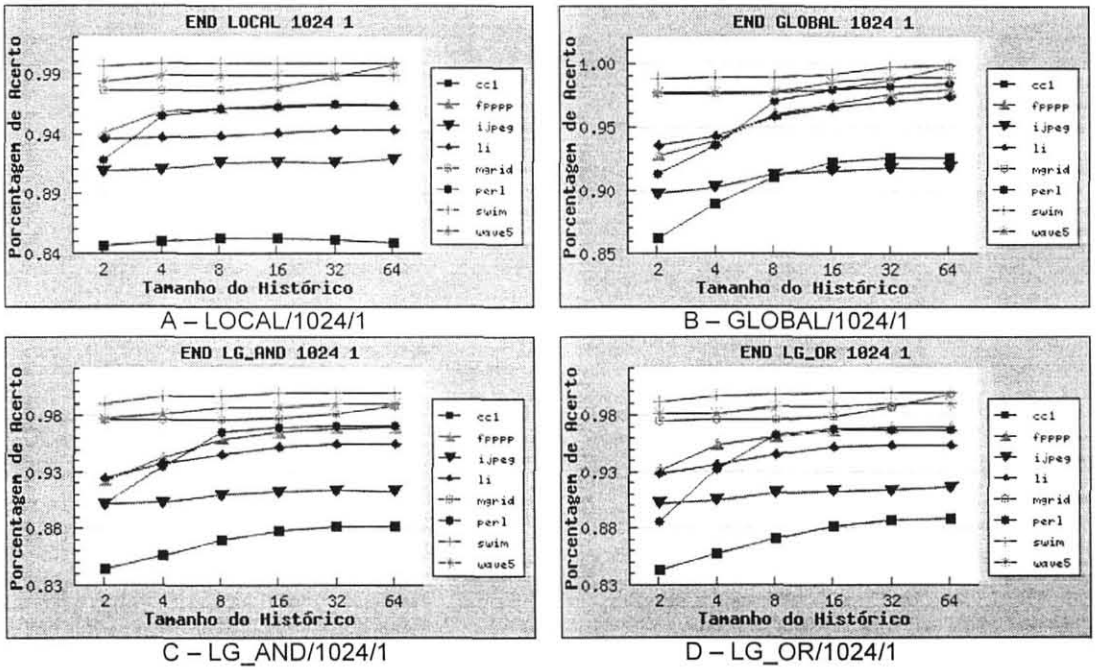
Figura 52 – Melhores resultados por desvios para o modelo END

A Figura 53-B representa o desempenho por *benchmark* da organização GLOBAL. Observamos que houve desempenho crescente acentuado na maioria dos *benchmarks*, com um crescimento suave para swim, wave5 e mgrid. O *benchmark* swim apresentou melhor resultado com 99,23% de acerto e pior média de desempenho para cc1 com 90,59%, o único *benchmark* a ficar abaixo da taxa média de desempenho do previsor.

Para o agrupamento por *benchmark* da organização LG_AND, representado pela Figura 53-C, observamos que houve desempenho crescente suave (li, fpppp e cc1) ou estável na maioria dos casos, com um crescimento acentuado em algum momento para ijpeg. O *benchmark* que apresentou a pior média foi cc1 com 86,87%

e o melhor resultado foi swim com 99,64% de acerto. Além de cc1, li esteve abaixo da média com 94,46% de taxa de acerto.

Para a organização LG_OR, a Figura 53-D representa o desempenho no agrupamento por *benchmark*, observamos que estes tiveram desempenho crescente acentuado na maioria dos *benchmarks*, com um crescimento suave para swim, wave5 e mgrid. O melhor resultado apresentado foi para o *benchmark* swim com 99,23% de acerto e pior média de desempenho para cc1 com 90,59%. Além de cc1, outros dois *benchmarks* ficaram abaixo da média, li (94,51%) e ijpeg (91%).



A – LOCAL/1024/1 B – GLOBAL/1024/1

C – LG_AND/1024/1 D – LG_OR/1024/1

Figura 53 – Melhores resultados por benchmarks para o modelo END

Convém ressaltar que, no caso da associatividade 1, ocorre a interferência de diferentes desvios sobre a mesma linha da tabela e, neste caso, não ocorre a substituição dos dados do Perceptron (histórico e pesos). A previsão ocorre com o histórico e pesos afetados pelos diferentes desvios envolvidos. Para associatividades maiores, foi empregada a política de ocupação de células livres já descrita anteriormente, com possíveis substituições através do algoritmo LRU. Esta situação também ocorre no modelo DNE descrito posteriormente.

A Figura 54 apresenta o desempenho do modelo END para tabelas de tamanho 1024, com número de linhas da tabela aumentando de 64 a 1024 e associatividade reduzindo de 16 a 1, para todos os quatro tipos de organização. Observa-se que, com o aumento da associatividade para o mesmo tamanho da

tabela, não há aumento de desempenho em todos os casos. Definindo que o desempenho do previsor é definido pelo número de linhas da tabela. Isto mostra que, para um menor número de linhas da tabela, há um maior número de substituição de dados e, conseqüentemente, maior interferência entre desvios.

O ganho da tabela com maior número de linhas e menor associatividade para a que possui menor número de linhas e maior associatividade foi de 0,98% para a organização LOCAL, 2,20% para GLOBAL e 1,88% para LG_AND e LG_OR.

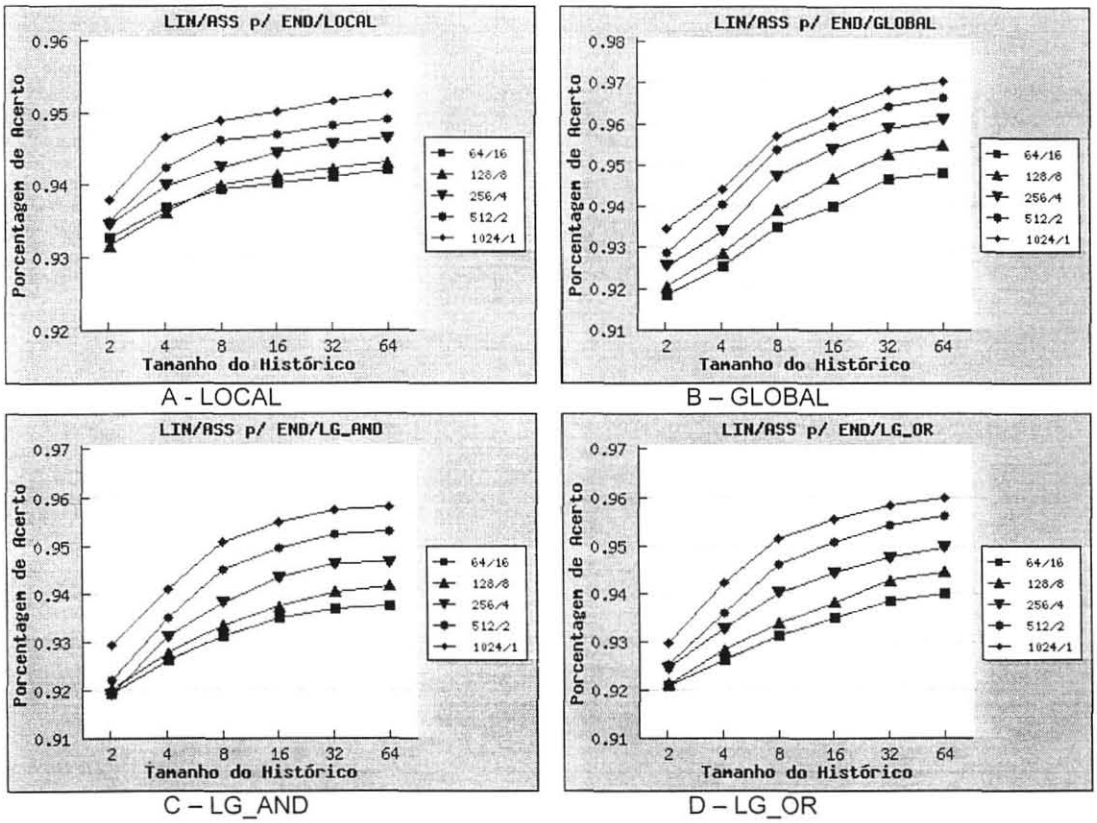


Figura 54 – Resultados para aumento associatividade da tabela para o modelo END

Nos casos em que o mesmo número de linhas da tabela é mantido e a associatividade é aumentada, ocorre o aumento do desempenho da previsão em todos os casos, com exceção para as organizações LOCAL e LG_AND com número de linhas igual a 64, cujo desempenho do agrupamento 64/1 foi maior que 64/2. O aumento do desempenho pelo aumento da associatividade é óbvio pelo fato que há aumento da capacidade da tabela. Os dois casos anômalos são explicados pelo fato de que as interferências entre desvios no agrupamento 64/2 ocasionou a queda de desempenho em relação ao agrupamento 64/1.

Em todas as organizações, a ordem decrescente de desempenho da relação LIN/ASS para tabelas de tamanho 1024 foi: 1024/1, 512/2, 256/4, 128/8, e 64/16. O mesmo ocorreu para as tabelas de tamanho 512 (512/1, 256/2, 128/4 e 64/8), 256 (256/1, 128/2 e 64/4) e 128 (128/1 e 64/2). Isto mostra que a diminuição do número de linhas e aumento de associatividade, de forma a manter o mesmo tamanho total da tabela (número de elementos), não manteve o desempenho do previsor. O desempenho da previsão é diretamente associado ao número de linhas da tabela.

7.4. Resultados de DNT

Devido ao número de variações para este modelo também é apresentado os melhores resultados para a relação LIN/ASS. A Figura 55 apresenta os resultados para diferentes números de linhas de tabela e associatividade fixa em 1 (não há variação de associatividade neste modelo) para todas as organizações deste modelo. Observa-se que há aumento de desempenho para o aumento das linhas da tabela. O melhor resultado obtido em todas as organizações foi para a relação LIN/ASS em 1024/1.

O gráfico por desvios para a organização LOCAL está representado na Figura 56-A. O desvio bgtz teve melhor resultado, com média de 65,05% e o pior resultado para o desvio blez, com 02,92% de taxa de acerto. A média deste previsor foi de 72,62%, e todos os desvios tiveram taxas médias de acerto abaixo desse valor. Com 78,49% de acerto ficaram os desvios para frente, acima da média, e com 44,65% os desvios para trás. Os desvios beq, bne, bc1f e bc1t apresentaram crescimento suave, com pequenas variações do desempenho, para o aumento do histórico. Os demais desvios apresentaram redução no desempenho.

A organização GLOBAL apresentou um desempenho médio de acerto de 73,80%. Para o agrupamento por desvios no gráfico da Figura 56-B, obteve-se para os desvios para frente um acerto de 79,78%, e para os desvios para trás um acerto de 46,46%, com o melhor desempenho para o desvio bgtz com média de 67,07% e o pior desempenho para blez com 3,42%. Todos os desvios tiveram taxa média de acerto abaixo da média do previsor. Tiveram crescimento do desempenho contínuo os desvios beq, bne blez e bltz, e redução de desempenho para bgez, os demais tiveram crescimento com oscilações para o aumento do histórico.

O ganho da tabela com maior número de linhas para a que possui menor número de linhas foi de 2,08% para a organização LOCAL, 1,26% para GLOBAL, 2,26% para LG_AND e 1,16% para LG_OR.

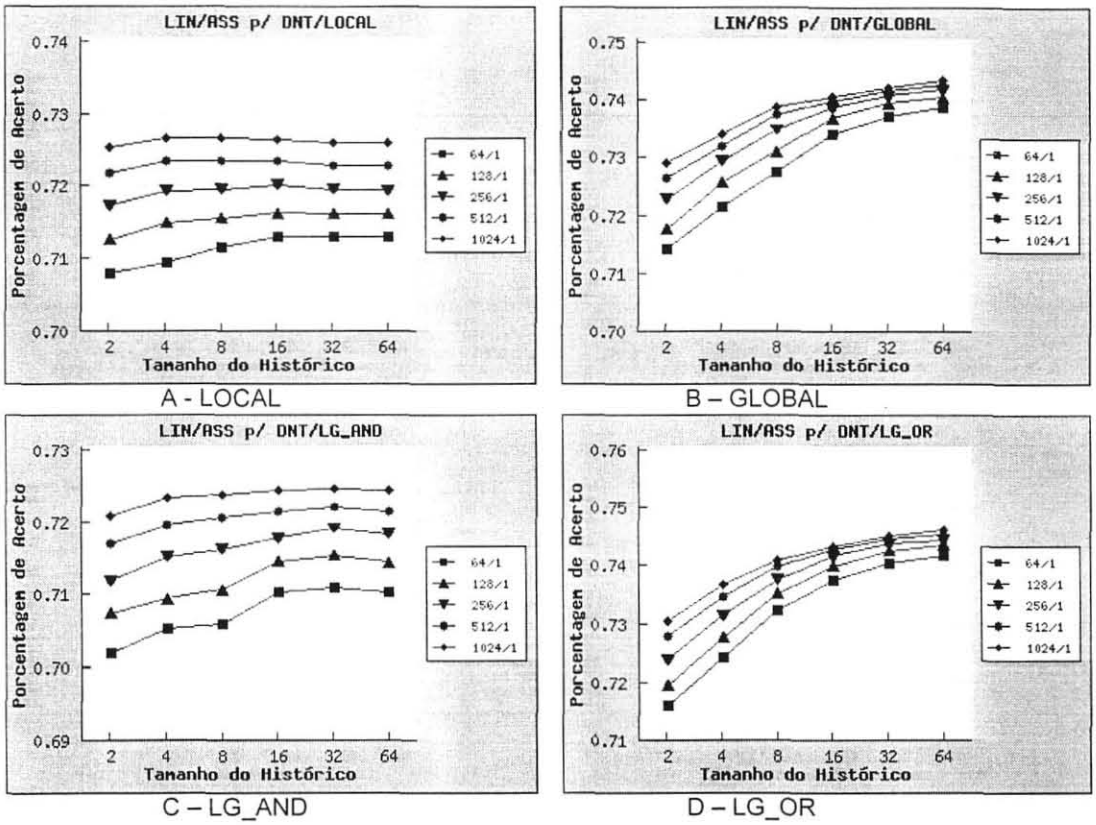
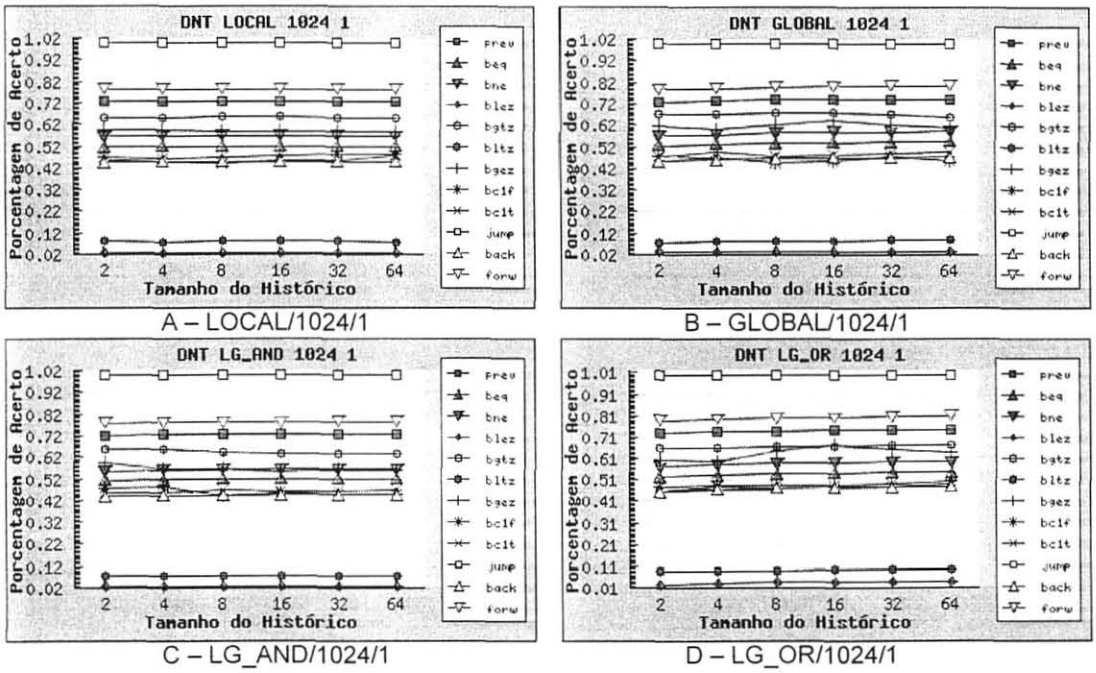


Figura 55 – Resultados para aumento de linhas da tabela para o modelo DNT

A organização LG_AND teve, como resultado médio de acerto, a taxa de 72,35%. O desvio bgtz teve melhor desempenho com 64,29% de acerto e o pior desempenho foi para blez com 2,76%. Somente o desvio bne teve aumento contínuo suave do desempenho, os desvios beq, bc1f e bc1t apresentaram oscilação no crescimento do desempenho, com o aumento do histórico. Os demais desvios apresentaram redução de desempenho com oscilação. Os desvios para frente tiveram um desempenho de 78.09% e os desvios para trás 44,21% de media de acerto. Todos os desvios ficaram abaixo da média do previsor. A Figura 56-C apresenta o gráfico por desvios para a organização LG_AND.

A Figura 56-D apresenta o gráfico por desvios para LG_OR. Este previsor atingiu a taxa média de acerto de 74,04%.Todos os desvios tiveram desempenho abaixo da média do previsor e crescimento do desempenho com o aumento do histórico. Os desvios beq, bne, blez e bc1f tiveram crescimento contínuo do

desempenho, os demais desvios tiveram crescimento com oscilação para o aumento do histórico. O menor desempenho ficou para o desvio blez com 3,28% e o maior ficou para bgtz com 66,61% da média de acerto em relação a todos *benchmarks*. Os desvios para frente tiveram desempenho de 79,95% de média de acerto enquanto que os desvios para trás atingiram 46,87% de acerto.



A – LOCAL/1024/1 B – GLOBAL/1024/1

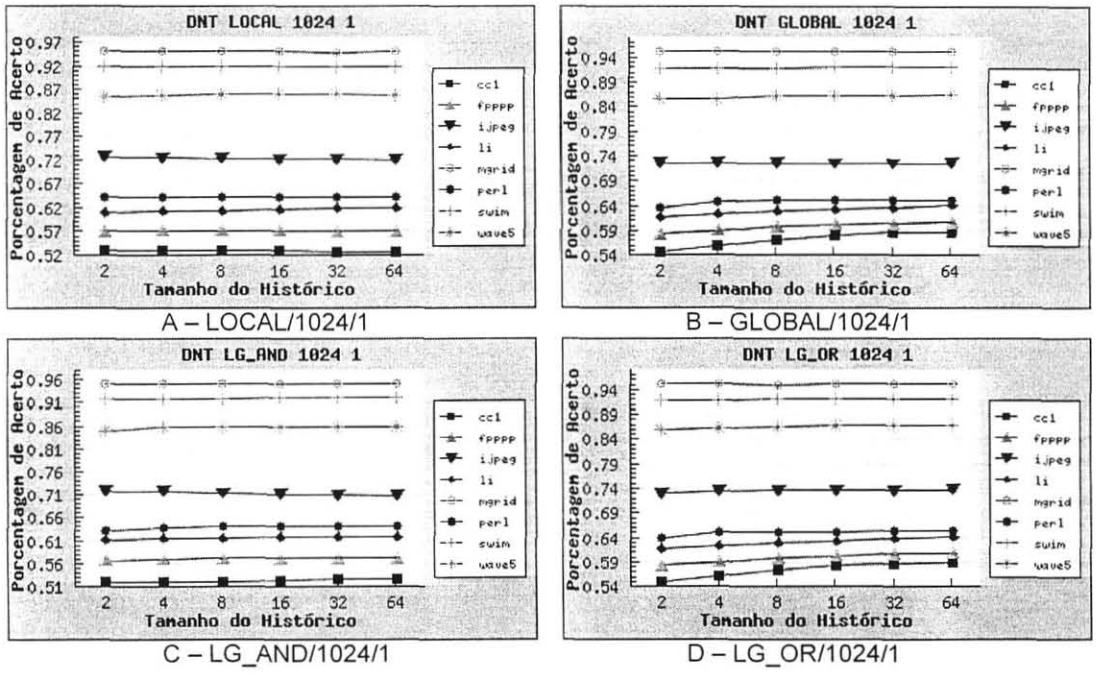
C – LG_AND/1024/1 D – LG_OR/1024/1

Figura 56 – Melhores resultados por desvios para o modelo DNT

O *benchmark* mgrid teve o melhor resultado com 95,02% de média de acerto e desempenho oscilante. O pior resultado foi para cc1 com 52,90% de taxa média de acerto, com redução do desempenho na variação do histórico. Os *benchmarks* cc1 e ijpeg apresentaram redução de desempenho suave com pequenas variações. Os demais *benchmarks* tiveram um crescimento pequeno ou estável com o aumento do histórico, sendo que apenas mgrid, swim e wave5 tiveram seu desempenho médio acima da média do previsor. A Figura 57-A apresenta estes resultados para a organização LOCAL.

Na Figura 57-B, representado o gráfico por *benchmark* para o previsor GLOBAL/1024/1, todos os *benchmarks* tiveram crescimento suave de desempenho. O melhor resultado foi de mgrid com média de 95,10%, que ficou acima da média juntamente com swim (91,96%) e wave5 (86,22%). O pior resultado foi para cc1 com 57,19% de taxa média de acerto.

O *benchmark* mgrid teve o melhor desempenho com 95,02% de média de acerto e houve redução de desempenho com variações. O *benchmark* ijpeg teve redução contínua de desempenho. Todos os demais *benchmarks* tiveram crescimento no desempenho, onde li teve crescimento contínuo e os demais apresentaram pequenas variações. O pior resultado foi para o *benchmark* cc1 com 52,28% de acerto. Ficaram acima da média, além de mgrid, os *benchmarks* swim com 91,93% e wave5 com 85,70%. A Figura 57-C apresenta os resultados para esta organização.



A – LOCAL/1024/1 B – GLOBAL/1024/1

C – LG_AND/1024/1 D – LG_OR/1024/1

Figura 57 – Melhores resultados por benchmarks para o modelo DNT

Todos os *benchmarks* apresentaram crescimento de forma suave com ou sem oscilação. Somente os *benchmarks* mgrid, swim e wave5 apresentaram taxas acima da média. O *benchmark* swim apresentou maior resultado com 91,91% de acerto e o menor resultado ficou para cc1 com média de 57,38% de taxa. A Figura 57-D apresenta o gráfico por *benchmark* para LG_OR/1024/1.

Na média geral, o *benchmark* que teve melhor desempenho foi mgrid e cc1 obteve pior desempenho neste modelo de previsor.

A melhor média de organização para este modelo de previsor foi a organização LG_OR com 73,67% de taxa média de acerto, seguida pela organização GLOBAL com 73,40% de taxa. Em seguida a organização LOCAL atingiu 71,90% de acerto e o pior desempenho ficou para a organização LG_AND

com 71,59% de média de acerto. O previsor LG_AND/64/1 com taxa de acerto de 70,75% obteve o pior desempenho e LG_OR/1024/1 teve o melhor desempenho com 74,04% de acerto neste modelo de previsor. A diferença entre a organização com maior resultado e a segunda colocada foi de 0,27 pontos percentuais, a diferença da segunda para a terceira colocada foi de 0,15 pontos percentuais e a diferença entre o terceiro e o quarto colocado foi de 0,31 pontos percentuais. Da organização com melhor desempenho para a de pior desempenho houve uma diferença de 2,08 pontos percentuais.

Através da média dos resultados obtidos para cada organização atinge-se a taxa de 72,64% de desempenho de acerto para o modelo de previsor DNT. Os *benchmarks* que tiveram melhor e o pior desempenho foram mgrid e cc1 para este modelo de previsor. O agrupamento ORG/LIN/ASS que apresentou pior desempenho foi LG_AND/64/1 com 70,75% de acerto.

Avaliando o desempenho das organizações deste modelo através da média das taxas de acerto dos *benchmarks* para cada agrupamento LIN/ASS observamos que as organizações LOCAL e LG_AND apresentaram crescimentos suaves e equivalentes, para o aumento do histórico, embora a organização LOCAL tenha apresentado desempenho ligeiramente superior. As organizações LG_OR e GLOBAL apresentaram crescimento acentuado no decorrer do crescimento do histórico, com a organização LG_OR apresentando melhor desempenho.

7.5. Resultados de DNE

Este modelo de previsor também apresenta diversas combinações de organização, número de linhas e associatividade de tabela, dessa forma, a análise será baseada nos melhores resultados.

A Figura 58 apresenta os resultados para aumento de linhas de tabela e associatividade fixa em 1 em todas as organizações do modelo DNE. Observa-se que sempre há aumento de desempenho com o aumento das linhas da tabela. O melhor resultado encontrado em todos as organizações foi para a relação LIN/ASS em 1024/1.

O ganho da tabela com maior número de linhas para a que possui menor número de linhas foi de 6,34% para a organização LOCAL, 2,67% para GLOBAL,

4,72% para LG_AND e 4,92% para LG_OR. A organização GLOBAL teve crescimento acentuado com os resultados das diferentes relações LIN/ASS se aproximando com o aumento do histórico. A organização LOCAL teve crescimento suave em todas as relações LIN/ASS e as demais organizações apresentaram maior crescimento para valores menores de linhas de tabela e menor crescimento no caso inverso.

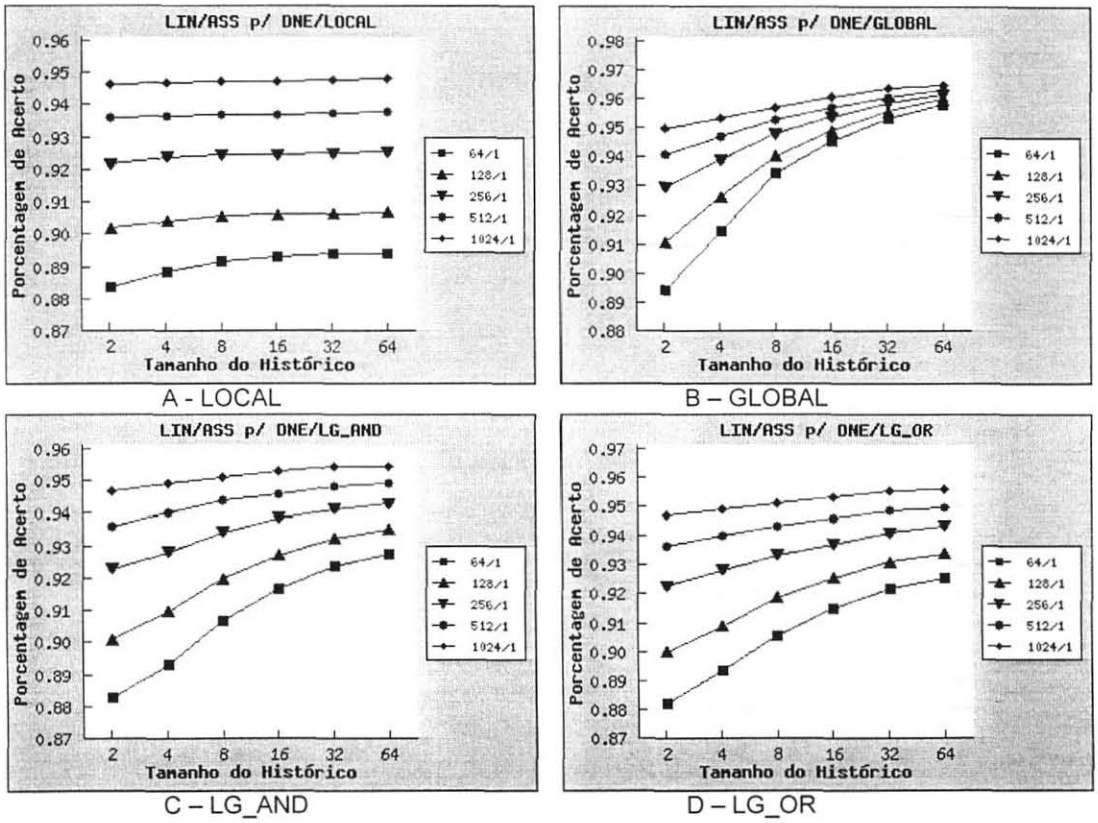
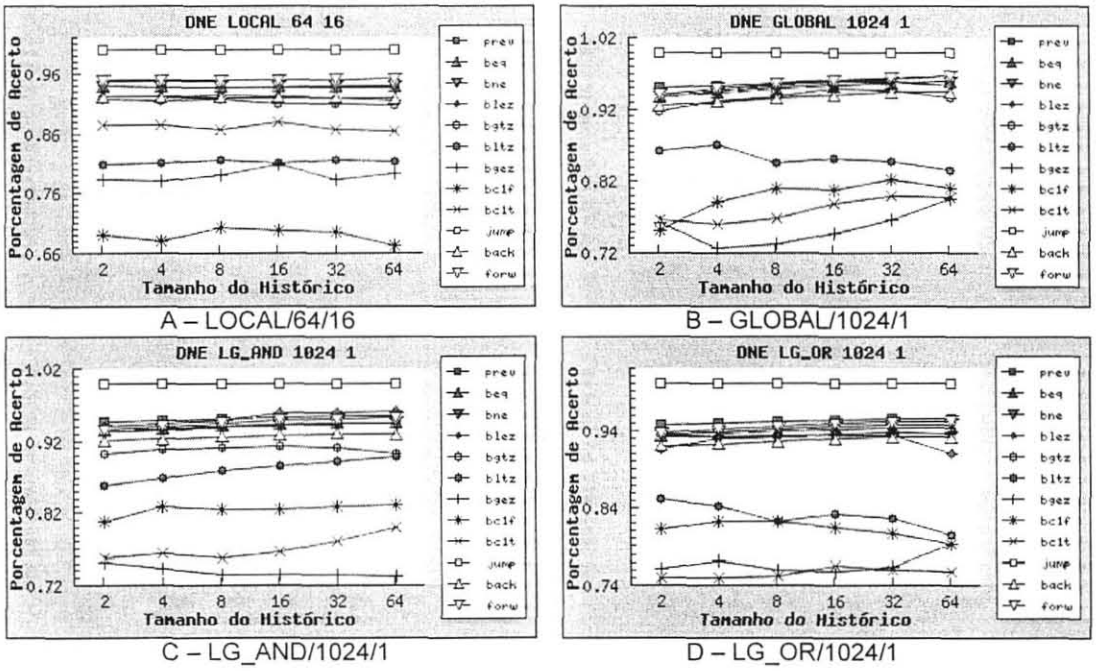


Figura 58 – Resultados para aumento de linhas da tabela para o modelo DNE

A seguir serão analisados os melhores casos para os agrupamentos por desvio e por *benchmark* para os quatro tipos de organização. Diferentemente das demais organizações, a organização LOCAL não teve melhor relação LIN/ASS em 1024/1. Isto se justifica pelo fato que, neste caso específico, a interferência entre os desvios foi maior do que o ocorrido nas outras organizações.

Para a organização LOCAL, o melhor resultado obtido foi para a relação LIN/ASS de 64/16, com média de acerto de 94,93%. O agrupamento por desvios, representado na Figura 59-A, apresentou desvios (beq, bltz e bgez) com aumento de desempenho com pequenas variações, e redução de desempenho para os demais desvios, sendo que bne teve redução contínua e os outros redução com variação,

para o aumento do histórico. A melhor média de resultados foi para bne com 98,78% de taxa e o pior resultado foi para bc1f com 68,88% de acerto, todos os desvios apresentaram desempenho abaixo da média do previsor. Os desvios para frente atingiram uma média de acerto de 94,93% e os desvios para trás o valor de 92,02% de taxa; os desvios para frente tiveram igual desempenho que a média do previsor.



A – LOCAL/64/16 B – GLOBAL/1024/1

C – LG_AND/1024/1 D – LG_OR/1024/1

Figura 59 – Melhores resultados por desvios para o modelo DNE

O previsor GLOBAL/1024/1 apresentou um desempenho médio de acerto em 95,82%, representando o melhor resultado para este modelo de previsor. Para o agrupamento por desvios, representado no gráfico da Figura 59-B, obteve-se para os desvios para frente um acerto de 95,61% e para os desvios para trás uma taxa de acerto de 93,63%, com o melhor desempenho para o desvio blez com taxa de 95,23% e o pior desempenho para bgez com 75,51%. Todos os desvios tiveram taxa média de acerto abaixo da média do previsor. Tiveram crescimento do desempenho contínuo, para o aumento do histórico, os desvios beq e bne e redução de desempenho para bltz, os demais tiveram crescimento com oscilações no decorrer do aumento do histórico.

O previsor LG_AND/1024/1 teve taxa média de acerto de 95,14%. O desvio blez teve melhor desempenho com 95,22% de acerto e o pior desempenho foi para bgez com 73,82%. Somente dois desvios (bgez e bgtz) tiveram redução suave do desempenho, o desvio bgtz apresentou oscilação, com o aumento do histórico. Para

os demais desvios, apresentaram crescimento contínuo os desvios beq, blez e bltz. Os desvios para frente tiveram um desempenho de 94,59% e os desvios para trás 92,62% de taxa media de acerto. Todos os desvios ficaram abaixo da média do previsor. O gráfico por desvios para a o previsor deste modelo e organização é apresentado na Figura 59-C.

A Figura 59-D apresenta o gráfico por desvio para o previsor DNE/LG_OR/1024/1. Este previsor atingiu a taxa média de acerto de 95,19%. Todos os desvios tiveram desempenho abaixo da média do previsor. Os desvios blez, bltz e bc1f tiveram redução do desempenho com oscilação, os demais desvios tiveram crescimento com oscilação, para o aumento do histórico, com exceção de bne que apresentou crescimento contínuo e o maior resultado com 94,26% de acerto. O menor resultado ficou para o desvio bc1t com 75,59% da média de acerto. Os desvios para frente tiveram desempenho de 94,60% enquanto que os desvios para trás atingiram 92,67% de acerto.

Para o agrupamento por *benchmark* para a organização LOCAL, apresentado na Figura 60-A, os *benchmarks* tiveram desempenho decrescente suave na maioria dos casos. Dois *benchmarks* (li e mgrid) apresentaram crescimento (acentuado a partir do histórico 16 para mgrid). O *benchmark* que apresentou melhor resultado foi swim com 99,91% de acerto e pior média para cc1 com 82,02%. Além de cc1, também ficou abaixo da média o *benchmark* ijpeg (91,50%).

Na Figura 60-B, representado o gráfico por *benchmark* para o previsor GLOBAL/1024/1, todos os *benchmarks* tiveram crescimento de desempenho; alguns (li, perl e mgrid) com crescimento mais acentuado em algum momento . O melhor resultado foi de swim com média de 99,91% e pior resultado para cc1 com 86,34% de média de acerto, que ficou abaixo da média juntamente com ijpeg com 91,57%.

Na organização LG_AND, o *benchmark* swim teve o melhor desempenho com 99,90% de média de acerto, que foi o único a manter estável o desempenho com o aumento de histórico. Todos os demais *benchmarks* tiveram crescimento no desempenho, onde mgrid, perl e wave5 tiveram crescimento contínuo e os demais apresentaram pequenas variações. O pior resultado foi para o *benchmark* cc1 com 84,87% de acerto, com valor abaixo da média juntamente com ijpeg com 91,04%. A Figura 60-C apresenta o gráfico por *benchmark* para LG_AND/1024/1.

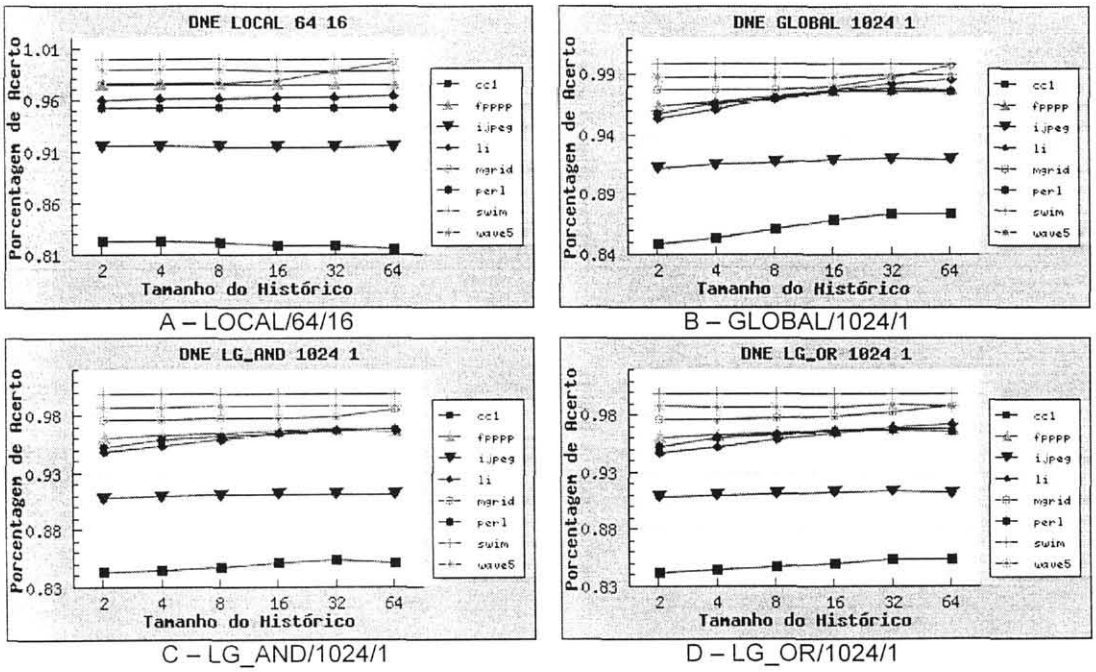


Figura 60 – Melhores resultados por benchmarks para o modelo DNE

Para a organização LG_OR, todos os *benchmarks* apresentaram crescimento de forma suave com alguma ou pouca oscilação. Somente os *benchmarks* cc1 e ijpeg apresentaram taxas abaixo da média. O *benchmark* swim apresentou maior resultado com 99,90% de acerto e o menor resultado ficou com cc1 com média de 84,85% de acerto. A Figura 60-D apresenta o gráfico por *benchmark* para LG_OR/1024/1.

A melhor média de organização para este modelo de previsor foi a organização GLOBAL com 94,34% de acerto, seguida por LG_AND com 93,18% de acerto. Em seguida a organização LG_OR atingiu 93,11% de acerto. O pior desempenho ficou para a média da organização LOCAL com 92,31% de acerto, especificamente o previsor LOCAL/64/2 com taxa de previsão de 88,31% deve o pior desempenho neste previsor. A diferença entre a organização de maior resultado e a segunda colocada foi de 1,16 pontos percentuais, a diferença da segunda para a terceira colocada foi de 0,07 pontos percentuais e a diferença entre o terceiro e quarto colocado foi de 0,08 pontos percentuais. Da organização com melhor desempenho para a de pior desempenho houve uma diferença de 2,03 pontos percentuais.

A média dos resultados obtidos em todas organizações é de 93,23% de desempenho de acerto para o modelo de previsor DNE. Os *benchmarks* que tiveram

melhor e o pior desempenho foram swim e cc1. Com pior desempenho ficou a organização LOCAL/64/2 com 88,31% de acerto.

Neste modelo, o *benchmark* com o melhor desempenho foi swim e o pior desempenho ficou para cc1. Avaliando o desempenho das organizações deste modelo através da média das taxas de acerto dos *benchmarks* para cada agrupamento LIN/ASS observamos que as organizações LG_OR e LG_AND apresentaram crescimentos equivalentes, mais suave para tabelas com número de linhas maiores que 256 e mais acentuado para tabelas menores que este valor. A organização LOCAL apresentou crescimento suave e pequeno no decorrer do crescimento do histórico. Já a tabela GLOBAL teve crescimento acentuado para tabelas com número de linhas menores e crescimento suave para tabelas com número de linhas maiores, com as taxas de acerto aumentando conforme o histórico aumenta.

A Figura 61 apresenta o desempenho do Modelo END para tabelas de tamanho 1024, com número de linhas da tabela aumentando de 64 a 1024 e associatividade reduzindo de 16 a 1, para todas as diferentes organizações.

O ganho da tabela com maior número de linhas e menor associatividade para a que possui menor número de linhas e maior associatividade foi de 0,60% para a organização LOCAL, 0,72% para GLOBAL, 0,41% para LG_AND e 0,46% para LG_OR.

Neste modelo, cada organização apresentou um conjunto diferentemente ordenado de desempenho para a relação LIN/ASS de forma a se manter o mesmo tamanho de tabela. A organização LOCAL teve como ordem decrescente de desempenho as associações 64/16, 1024/1, 128/8, 256/4 e 512/2; a organização GLOBAL a seqüência 1024/1, 512/2, 256/4, 64/16 e 128/8. LG_AND apresentou a seguinte ordem: 1024/1, 64/16, 512/2, 256/4 e 128/8. Finalmente LG_OR a seqüência: 1024/1, 64/16, 128/8, 512/2 e 256/4. Para as tabelas com 512 linhas também ocorreu algo semelhante. Já para as tabelas com 256 e 128 linhas ocorreu a ordem decrescente de desempenho conforme se reduzia pela metade o número de linhas e se dobrava a associatividade da tabela. Dessa forma, para tabelas maiores que 512 linhas não foi possível levantar qualquer relação do desempenho do previsor com seu número de linhas ou associatividade de tabela. Já para tabelas menores que este valor o comportamento foi semelhante ao modelo END.

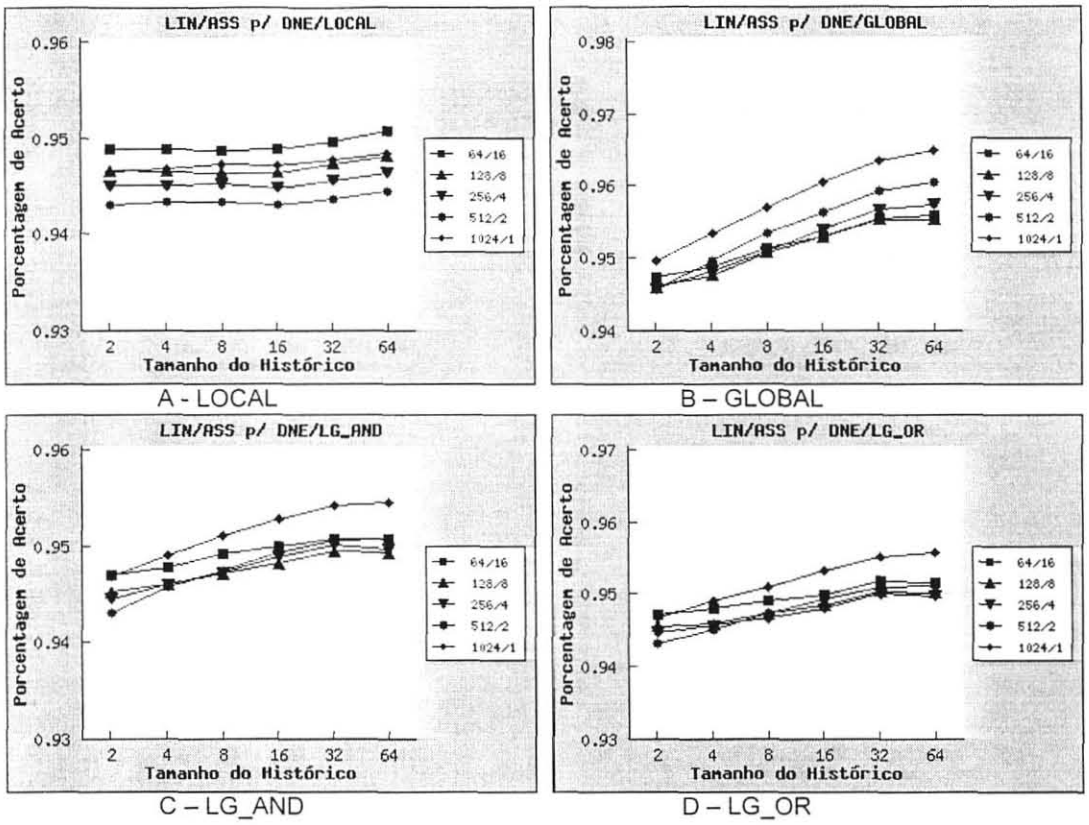


Figura 61 – Resultados para aumento associatividade da tabela para o modelo DNE

Observou-se também que o aumento da associatividade para o mesmo número de tabela não teve aumento crescente de desempenho nos casos das tabelas com número de linhas em 512 e 1024. Para o número de linhas em 256, a organização GLOBAL não teve aumento crescente de desempenho, em contradição às outras organizações. Para 128 linhas de tabela houve crescimento crescente com o aumento da associatividade.

7.6. Análise Geral dos Resultados

Analisando todos estes dados obtidos podemos levantar algumas informações a respeito de cada previsor, qual modelo de previsor apresentou melhor resultado e em que condições, e quais mecanismos implementados (organização, linhas e associatividade) auxiliaram no desempenho do previsor.

Na Tabela 4 temos de forma compacta os melhores resultados obtidos para cada modelo e organização de previsor, especificando qual foi a relação número de linhas e associatividade de tabela que definiu este melhor resultado. Também estão

especificados os melhores e piores casos alcançados para quais *benchmarks*, tipos de desvios e seu direcionamento e a pior relação ORG/LIN/ASS para cada modelo de previsor.

Podemos observar claramente que a ordem decrescente da média de desempenho para os modelos de previsor é DNE, END, UNI, DNT e TIP. A mesma ordem de desempenho ocorre para o melhor e pior caso para cada modelo de previsor. No melhor caso tem-se: DNE (95,82%), END (95,22%), UNI (83,40%), DNT (74,04%) e TIP (70,68%). E no pior caso: DNE (88,31%), END (88,23%), UNI (83,40%), DNT (70,75%) e TIP (68,16%). De forma geral, os modelos de previsor em dois níveis apresentaram melhor desempenho que os correspondentes em um nível.

O modelo DNE teve na média um acréscimo em torno de um ponto percentual sobre END e, no pior e melhor caso, a diferença ficou abaixo deste valor. Isto nos mostra que DNE e END tiveram desempenho equivalente. Estes modelos apresentaram desempenho acima de 91% de acerto, os resultados das previsões dos desvios para frente e para trás foram muito próximos e os piores resultados para os *benchmarks* ficaram próximos da média de seus modelos, indicando um bom desempenho destes modelos para ambos os direcionamentos dos desvios e na previsão sobre diferentes *benchmarks*.

Para o modelo DNT houve uma diferença em torno de 3 pontos percentuais sobre TIP na média e próximo a 4 pontos no melhor caso e 2 pontos percentuais no pior caso. O fato dos modelos DNT e TIP apresentarem os piores resultados sobre todos os modelos estudados se deve à interferência dos diferentes endereços de desvios, do mesmo tipo, sobre o previsor Perceptron. Ou seja, o previsor não consegue encontrar um padrão comportamental sobre o histórico por tipo de desvio. Dessa forma, embora DNT apresente ganho sobre TIP, no contexto global este ganho não apresenta vantagem, já que para os modelos de previsor por tipo o desempenho fica abaixo dos 75% de acerto na média.

Outra característica observada é que, para estes previsores (TIP e DNT), os desvios com piores desempenhos estão muito abaixo da média do previsor e abaixo dos piores casos para desvios nos outros modelos, o que justifica o fato da interferência sobre os desvios.

Tabela 4 – Resumo das porcentagens de acerto para todos os modelos de previsor

Melhor caso de MOD/ORG/LIN/ASS	Melhor caso:	Pior caso:	Pior caso de MOD/ORG/LIN/ASS	Comentários
UNI: 83,41%			Não há.	
ÚNICA: 83,41%	Forw: 83,25% beq: 76,73% mgrid: 97,02%	Back: 69,26% bltz: 51,40% perl: 66,76%		direção desvio tipo do desvio *benchmark*
TIP: 69,53%			LG_AND: 68,16%	
LOCAL: 69,50%	Forw: 76,01% bgtz: 84,56% mgrid: 94,98%	Back: 39,50% blez: 1,89% cc1: 48,97%		
GLOBAL: 69,79%	Forw: 76,42% bgtz: 84,81% mgrid: 95,01%	Back: 39,88% blez: 2,50% cc1: 49,75%		Idem
LG_AND: 68,16%	Forw: 74,42% bgtz: 83,545 mgrid: 94,74%	Back: 37,22% blez: 1,66% cc1: 47,26%		
LG_OR: 70,69%	Forw: 76,42% bgtz: 85,11% mgrid: 95,04%	Back: 39,88% blez: 2,61% cc1: 51,00%		
END: 92,30%			LOCAL/64/2: 88,23%	
LOCAL: 91,96%				
1024/1: 94,80%	Forw: 93,57% blez: 95,15% swim: 99,86%	Back: 92,09% bgez: 77,32% cc1: 85,02%		
GLOBAL: 93,34%				
1024/1: 95,63%	Forw: 93,47% blez: 98,16% swim: 99,23%	Back: 93,54% bgtz: 75,76% cc1: 90,59%		Idem
LG_AND: 91,93%				
1024/1: 94,87%	Forw: 92,83% blez: 97,28% swim: 99,64%	Back: 92,41% bgez: 76,05% cc1: 86,87%		
LG_OR: 91,96%				
1024/1: 95,96%	Forw: 93,31% blez: 94,65% swim: 99,23%	Back: 92,33% bltz: 76,45% cc1: 90,59%		
DNT: 72,64%			LG_AND/64/1: 70,75%	
LOCAL: 71,90%				
1024/1: 72,62%	Forw: 78,49% bgtz: 65,05% mgrid: 95,02%	Back: 44,65% blez: 2,92% cc1: 52,90%		
GLOBAL: 73,40%				
1024/1: 73,80%	Forw: 79,78% bgtz: 67,07% mgrid: 95,10%	Back: 46,46% blez: 3,42% cc1: 57,19%		
LG_AND: 71,59%				Idem
1024/1: 72,35%	Forw: 78,09% bgtz: 64,29% mgrid: 95,02%	Back: 44,21% blez: 2,76% cc1: 52,28%		
LG_OR: 73,67%				
1024/1: 74,04%	Forw: 79,95% bgtz: 66,61% mgrid: 91,91%	Back: 46,87% blez: 3,28% cc1: 57,38%		
DNE: 93,23%			LOCAL/64/2: 88,31%	
LOCAL: 92,31%				
64/16: 94,93%	Forw: 94,33% bne: 98,78% swim: 99,91%	Back: 92,02% bc1f: 68,88% cc1: 82,02%		
GLOBAL: 94,34%				
1024/1: 95,82%	Forw: 95,61% blez: 95,23% swim: 99,91%	Back: 93,63% bgez: 75,51% cc1: 86,34%		
LG_AND: 93,18%				Idem
1024/1: 95,14%	Forw: 94,59% blez: 95,22% swim: 99,90%	Back: 92,62% bgez: 73,82% cc1: 84,87%		
LG_OR: 93,11%				
1024/1: 95,19%	Forw: 94,60% bne: 94,26% swim: 99,90%	Back: 92,67% bc1t: 75,59% cc1: 84,85%		

Apesar do previsor UNI ser o modelo mais simples de previsor implementado, pois possui apenas um único Perceptron para avaliar todos os desvios de um programa, este apresentou desempenho intermediário no grupo analisado, com um desempenho acima de 80% de acerto.

A melhor relação LIN/ASS nos modelos que apresentam esta característica foi de 1024/1 em quase todas as organizações, com exceção para DNE/LOCAL cujo melhor resultado ficou para a relação 64/16.

O *benchmark* mgrid obteve melhor índice de previsão nos três modelos de pior desempenho, swim nos dois modelos de melhor desempenho, e cc1 obteve o pior índice de previsão em quase todos os modelos com exceção para UNI, cujo pior *benchmark* ficou para perl.

Os tipos de organização LG_AND e LG_OR apresentam desempenho intermediário entre as organizações LOCAL e GLOBAL nos modelos END e DNT. O comportamento inverso ocorre para os modelos TIP e DNT, onde as organizações LOCAL e GLOBAL apresentaram desempenho intermediário. Estes fatos são justificados pelos resultados obtidos pela previsão dos desvios para frente e para trás de cada modelo, LG_AND contribui para a previsão dos desvios não-tomados (back) e LG_OR contribui para a previsão dos desvios tomados (forw).

A Tabela 5 apresenta os ganhos entre os modelos para a média de desempenho alcançado pelos diferentes modelos de previsão. A diferença entre o modelo de menor e maior desempenho ficou próximo a 34%. Para os previsores de dois níveis em relação aos de um nível a diferença foi próximo a 4,5% para DNT e 1% para DNE. A diferença de ganho de UNI em relação aos outros previsores indica que seu desempenho está mais próximo ao desempenho dos previsores por endereço.

Tabela 5 – Porcentagem de ganho entre desempenho médio dos modelos

Ganhos entre Modelos

	UNI	TIP	END	DNT	DNE
UNI	0	+19,96%	-10,66%	+14,83%	-11,77%
TIP	-19,96%	0	-32,75%	-4,47%	-34,07%
END	+10,66%	+32,75%	0	+27,06%	-1,01
DNT	-14,83%	+4,47%	-27,06%	0	-28,34%
DNE	+11,77%	+34,07%	+1,01	+28,34%	0

A avaliação do custo de implementação em hardware dos previsores em todos modelos analisados, para o menor e maior histórico e tamanho de tabela de Perceptron, para históricos LOCAL e GLOBAL, e apresentados na Tabela 6, indicam

que a implementação dos mesmos é viável, já que o tamanho máximo alcançado corresponde a uma cache de dados equivalente a 83kb de tamanho, que representa um custo de hardware aceitável nos padrões atuais.

Tabela 6 – Tamanho de hardware dos previsores

Modelo	Tamanho do Histórico	Tamanho da Tabela	Histórico	Tamanho Bits	Tamanho Bytes
UNI	2	1	LOCAL	26	4
	64	1	LOCAL	584	73
TIP	2	8	LOCAL	194	25
	64	8	LOCAL	4224	528
	2	8	GLOBAL	208	26
	64	8	GLOBAL	4672	584
END	2	64	LOCAL	1538	193
	64	64	LOCAL	33344	4168
	2	1024	LOCAL	24578	3073
	64	1024	LOCAL	532544	66568
	2	64	GLOBAL	1664	208
	64	64	GLOBAL	37376	4672
	2	1024	GLOBAL	26624	3328
	64	1024	GLOBAL	598016	74752
DNT	2	64	LOCAL	1554	195
	64	64	LOCAL	33856	4232
	2	1024	LOCAL	24594	3075
	64	1024	LOCAL	533056	66632
	2	64	GLOBAL	1680	210
	64	64	GLOBAL	37888	4736
	2	1024	GLOBAL	26640	3330
	64	1024	GLOBAL	598528	74816
DNE	2	64	LOCAL	1666	209
	64	64	LOCAL	37440	4680
	2	1024	LOCAL	26626	3329
	64	1024	LOCAL	598080	74760
	2	64	GLOBAL	1792	224
	64	64	GLOBAL	41472	5184
	2	1024	GLOBAL	28672	3584
	64	1024	GLOBAL	663552	82944

As conclusões gerais sobre as informações obtidas e as possibilidades de continuidade deste trabalho são apresentadas no capítulo seguinte.

8. Conclusões e Trabalhos Futuros

Este trabalho contribuiu para o entendimento de redes neurais, especificamente o Perceptron, e sua aplicação sobre o mecanismo de previsão de desvios em arquitetura superescalar. Cinco modelos diferentes de previsor foram idealizados, simulados e avaliados, permitindo um estudo comparativo dos resultados dos diferentes modelos e uma análise do efeito de cada modelo sobre a previsão de desvios numa arquitetura superescalar através de simulações de *benchmarks*.

Cada organização diferenciada dos modelos de previsor foi analisada e avaliada os ganhos entre estas organizações diferentes de forma a possibilitar obter dados para a construção de um previsor que seleciona, em tempo de execução, a melhor organização de previsão para um determinado desvio, definindo um previsor seletivo.

O uso de associatividade nas tabelas do previsor permitiu avaliar que o aumento da associatividade com a redução do número de linhas da tabela não resultam em ganho de desempenho, e que há aumento de desempenho com o aumento da associatividade para o mesmo número de linhas da tabela.

Da mesma forma, foi avaliado o aumento do número de linhas para a mesma associatividade da tabela do previsor e verificou-se que há um ganho em desempenho com este aumento.

Os melhores resultados foram para programas de ponto flutuante e desvios para frente. As organizações LG_AND e LG_OR não apresentam contribuições representativas na previsão de desvios, ficando os melhores resultados para LOCAL e GLOBAL.

A implementação de previsores em dois níveis levou-nos a conclusão que tal arquitetura apresenta maior ganho sobre a equivalente em um nível, em previsores baseados no Perceptron, independente do modelo utilizado.

Das tarefas a que nos propomos, foram realizadas as implementações de diferentes modelos de previsor baseados no Perceptron e a avaliação do desempenho dos mesmos na previsão de desvios, principalmente, a implementação e avaliação de previsores em dois níveis.

Para continuidade deste trabalho apontamos a implementação de um previsor baseado no Perceptron seletivo, que dinamicamente, em tempo de execução, atribui o melhor mecanismo de previsão sobre determinado desvio. Também é possível o uso, no mecanismo de previsão, de múltiplos Perceptrons, constituindo uma rede MLP (Multi-Layer Perceptron) no previsor, e o uso de outras redes neurais, como back-propagation.

O modelo de previsor END com tamanho de hardware equivalente ao previsor baseado no Perceptron dos estudos de Jiménez [23, 24] apresentaram desempenho equivalentes, validando os resultados obtidos e indicando realmente que o previsor DNE apresenta ganho em relação ao previsor de um nível.

De uma forma geral, o presente trabalho mostrou que o uso do Perceptron em previsores de um nível possui os resultados equivalentes àqueles obtidos em trabalhos correlatos baseados no Perceptron e seu uso em previsores de dois níveis é atrativo apresentando ganhos nos resultados.

Referências Bibliográficas

[1] APACHE GROUP. **Apache HTTP Server Manual**, Apache HTTP Server Documentation Project, Jan. 2003. Disponível em: <http://www.apache.org> Acesso em: 15 Mai. 2003.

[2] ARAGON, J. L. et al. **Confidence Estimation for Branch Prediction Reversal**. In Proceedings of the 8th International Conference on High Performance Computing, Dec. 2001.

[3] AXMARK, D. et al. **MySQL Reference Manual**, Swedish Company MySQL AB, May. 2003. Disponível em: <http://www.mysql.com> Acesso em: 15 Mai. 2003.

[4] BAKKEN, S. S. et al. **PHP Manual**, PHP Documentation Group, Apr. 2003. Disponível em: <http://www.php.net> Acesso em: 15 Mai. 2003.

[5] BALL, T.; LARUS, J. R. **Branch Prediction for Free**. Technical Report #1137, Computer Sciences Department, University of Wisconsin, Madison, Feb. 1993.

[6] BRAY, B. K.; FLYNN, M. J. **Strategies for Branch Target Buffers**. ACM – Association for Computing Machinery, Jun. 1991, p. 42-50.

[7] BURGER, D.; AUSTIN, T. M. **The SimpleScalar Tool Set, Version 2.0**. Technical Report #1342, Computer Sciences Department, University of Wisconsin-Madison, Jun. 1997.

[8] CALDER, B. et al. **Evidence-based Static Branch Prediction using Machine Learning**. ACM Transactions on Programming Languages and Systems, 1997.

[9] CHOLEWO, T. J.; ZURADA, J. M. **Sequential Networks Construction for Time Series Prediction**. Proc. of the IEEE Joint Conference on Neural Networks, Computer, Houston, Texas, USA, Jun. 1997, p. 2034-2039.

[10] DE FALCO, I. et al. **Optimizing Neural Networks for Time Series Prediction**. Institute for Research on Parallel Information Systems, Third World Conference on Soft Computing (WSC3), Jun. 1998.

[11] DE WILDE, P. **Neural Network Models**: theory and projects, 2. ed. Springer. Chap. 3, 1997, p. 44-47, 53-67.

[12] ESPASA, R.; VALERO, M. **Exploiting Instruction and Data Level Paralellism,** IEEE Micro Journal. Vol. 17, N. 5, Sep. 1997, p. 20-27.

[13] ESSENREITER, R. **Time Series Prediction with Neural Nets**. Disponível em: <http://www-gpi.physik.uni-karlsruhe.de/pub/robert/Diplom/node17.html> Acesso em: 23 nov. 2001.

[14] FERN, A.; et al. **Dynamic Feature Selection for Hardware Prediction**. Technical Report TR-ECE 00-12, School of Electrical and Computer Engineering, Purdue University, 2000.

[15] FIGUEIREDO, M. F. **Redes Neurais Nebulosas Aplicadas em Problemas de Modelagem e Controle Autônomo**. Tese (Doutorado) – Campinas: DCA – FEEC – Unicamp, p. 8-22, 1997.

[16] FIGUEIREDO, M., GOMIDE, F. **Design of Fuzzy Systems Using Neurofuzzy Networks.** IEEE Transactions on Neural Networks, Vol. 10, no. 4, p.815-827, Jul. 1999.

[17] FRITZSCHE, H. **Programação Não-Linear**: Análise e Métodos. Editora da Universidade de São Paulo, 1978, p. 1-22.

[18] GONÇALVES, R. A. de L. **Arquiteturas Multi-Tarefas Simultâneas**: SEMPRE - Arquitetura SMT com Capacidade de Execução e Escalonamento de Processos. Tese (Doutorado) – Porto Alegre: PPGC do II/UFRGS, 2000.

[19] GONÇALVES, R. et al. **Evaluating the Effects of Branch Prediction Accuracy on the Performance of SMT Architectures.** Proc. of the 9th EUROMICRO - PDP 2001 - 9th Euromicro Workshop on Parallel and Distributed Processing. p. 355-362. Italy. Feb. 2001.

[20] HAYKIN, S. **Neural Networks**: a Comprehensive Foundation. Prentice Hall. New York. 1994, p.45-87,106-120.

[21] JAIN, A. K.; MAO, J. **Artificial Neural Networks**: A Tutorial. Proceedings of the IEEE Computer, Mar. 1996, p. 31-44

[22] JIMENEZ, D. A.; HANSON, H. L; Lin, C. **Boolean Formula-based Branch Predictions for Future Technologies.** Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, Sep. 2001.

[23] JIMÉNEZ, D. A.; LIN, C. **Dynamic Branch Prediction with Perceptrons**. Proceedings of the Seventh International Symposium on High Performance Computer Architecture (HPCA), Jan. 2001

[24] JIMÉNEZ, D. A.; LIN, C. **Perceptron Learning for Prediction the Behavior of Conditional Branches**. Technical Report TR2000-08, The University of Texas at Austin, USA, 2000.

[25] KHARDON, R.; ROTH, D.; SERVEDIO, R. **Efficiency versus Convergence of Boolean Kernels for On-Line Learning Algorithms**. Advances in Neural Information Processing Systems 14 (NIPS), 2001.

[26] KOSKELA, T. et al. **Time Series Prediction with Multilayer Perceptron, FIR and Elman Neural Networks**. Proceedings of the World Congress on Neural Networks, 1996, p. 491-496.

[27] KOUMPIS, K.; RENALS, S. **Transcription and Summarization of Voicemail Speech**. Proceedings of ICSLP, Vol. 2, Beijing, China, 2000, p. 688-691.

[28] KOVÁCS, Z. L. **Redes Neurais Artificiais:** Fundamentos e Aplicações. 2. ed. Collegium Cognition. Cap. 1-5, 1996, p. 13-90.

[29] KUVAYEV, L. **Using Neural Networks for Branch Prediction**. Seminar in Applying Learning to System Problems. May. 1996. Disponível em: http://citeseer.nj.nec.com/383643.html. Acesso em: 07 Set. 2003.

[30] LEE, J. K. F.; SMITH, A. J. **Branch Prediction Strategies and Branch Target Buffer Design**. IEEE Computer, Jan. 1984, p. 6-22.

[31] LEE, K.; ZHANG, B. **Learning Robot Behaviors by Evolving Genetic Programs**. Proceedings of the 26th International Conference on Industrial Electronics, Control and Instrumentation (IECON-2000).

[32] LIU, Z. et al. **Dynamic Image Sequence Analysis Using Fuzzy Measures**. IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics, Vol. 31, N. 4, Aug. 2001.

[33] MAJUMDAR, R.; WEITZ, D. **Branch Prediction using Neural Nets**. CS252 Final Report, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 2001.

[34] MCFARLING, S. **Combining Branch Predictors**. Technical Report TN-36m, Digital Western Laboratory, Jun. 1993.

[35] MILENKOVIC, M.; MILENKOVIC, A.; KULICK, J. **Demystifying Intel Branch Predictors**. Proceedings of the Workshop on Duplicating, Deconstructing, and Debunking (held in conjuction with 29th ISCA), Anchorage, Alaska, May 2002.

[36] MITRA, P.; MITRA, S.; PAL, S. K. **Staging of Cervical Cancer with Soft Computing**. IEEE Transactions on Biomedical Engineering, Vol. 47, 2000, p. 934-940.

[37] OLIVEIRA, L. S. et al. **A new approach to segment handwritten digits**, Proc. 7th IWFHR (International Workshop on Frontiers in Handwriting Recognition), Amsterdam, 2000, p. 577-582.

[38] PALACHARLA, S.; JOUPPI, N. P.; SMITH, J. E. **Complexity-Effective Superscalar Processors**. Proceedings of ISCA, Denver, USA, 1997.

[39] PERLEBERG, C. H.; SMITH, A. J. **Branch Target Buffer Design and Optimization**. IEEE Transactions on Computers, Vol. 2, N. 4, Apr. 1993, p. 396-412.

[40] PERSSON, J. **JpGraph Manual**, Johan Persson, Feb. 2003. Disponível em: <http://www.aditus.nu/jpgraph> Acesso em: 15 Mai. 2003.

[41] RAMIREZ, A.; LARRIBA-PEY, J. L.; VALERO, M. **The Effect of Code Reordering on Branch Prediction**. Proceedings of the Intl. Conference on Parallel Architectures and Compilation Techniques, Philadelphia (USA). p. 189-198. Oct. 2000.

[42] RIBAS, V.; FIGUEIREDO, M. e GONÇALVES, R. **Analyzing Branch Prediction Under Speculative Execution Using Perceptron**, XXIII Congresso da SBC - ENIA - Encontro Nacional de Inteligência Artificial, Campinas, Ago. 2003.

[43] ROTENBERG, E.; BENNETT, S.; SMITH, J. **Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching**. Proceedings of 29th International Symposium on Microarchiquecture, Dec. 1996, p. 24-35. Sep. 2000.

[44] RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence**: A Modern Approach. Prentice Hall Inc., Chap. 19, 1985, p. 563-597.

[45] SBERA, M.; VINTAN, L. N.; FLOREA, A. **Static and Dynamic Branch Prediction Using Neural Networks**, Acta Universitatis Cibiniensis, Technical Series, Computer Science and Automatic Control, p. 81-86, Vol. XLIII, Ed. Universitatii "L. Blaga" Sibiu, 2001.

[46] SHERWOOD, T.; CALDER, B. **Loop Termination Prediction**. Proceedings of the 3rd International Symposium on High Performance Computing (ISHPC2K), Oct. 2000.

[47] SMITH, J. E. **A Study of Branch Prediction Strategies**. IEEE, May 1981, p. 202-215.

[48] SMITH, J. E.; SOHI, G. S. **The Microarchitecture of SuperScalar Processors**. Proceedings of the IEEE, Dec. 1995, p. 1609-1624.

[49] SPEC. **The SPEC Benchmark homepage.** Disponível em: <http://www.spec.org> Acesso em: 14 jan. 2002.

[50] STALLINGS, W. **Computer Organization and Architecture**. 5. ed. Prentice Hall. Chap. 13 – Instruction-Level Paralellism and Superscalars Processors, 2000, p. 497-552.

[51] STEVEN, G. et al. **Dynamic Branch Prediction Using Neural Networks**, Proceedings of the International Euromicro Conference DSD'2001, Warsaw, Poland, Sep. 2001.

[52] TANENBAUM, A. S. **Structured Computer Organization**. 5. ed. Prentice Hall, 1999, p. 49-53, 270-291.

[53] THIESING, F. M.; MIDDELBERG, U.; VORNBERGER, O. **Parallel Back-Propagation for Prediction of Time Series**. First European PVM Users' Group Meeting, Rome, Oct. 1997.

[54] TOMASULO, R. M. **An Efficient Algorithm for Expliting Multiple Arithmetic Units**, IBM Journal, 1967, p. 25-32.

[55] UHT, A. K.; HALL, K. **Extraction of Massive Instruction-Level Paralellism**, Computer Architecture News, Jun. 1993.

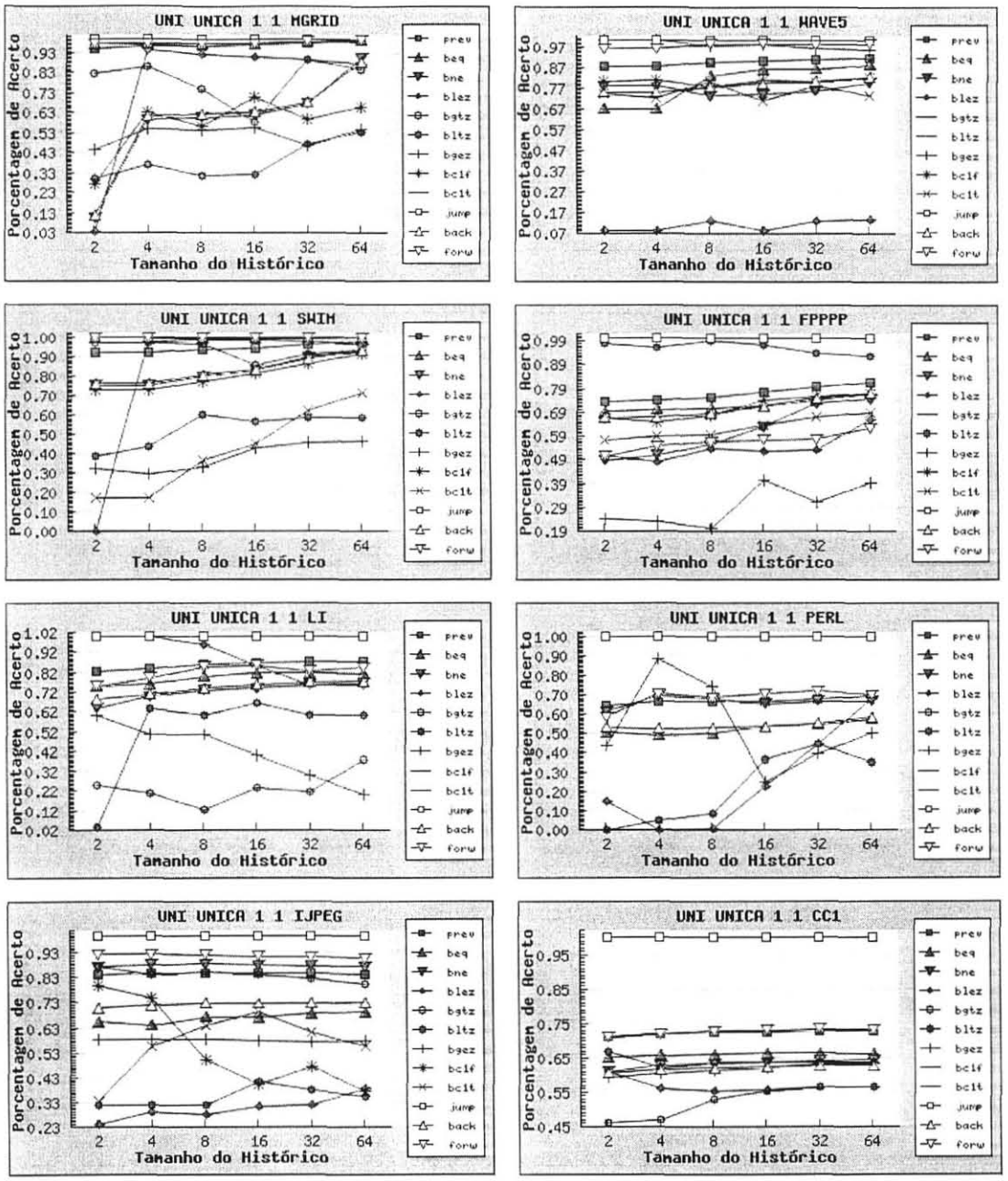
[56] YEH, T.; PATT, Y. N. **Alternative Implementation of Two-Level Adaptive Branch Prediction**. The 19th Annual International Symposium on Computer Architecture, Gold Coast, Australia, May. 1992, p. 124-134.

[57] YEH, T.; PATT, Y. N. **Two-Level Adaptive Training Branch Prediction**. The 24th ACM/IEEE International Symposium and Workshop on Microarchitecture, Nov. 1991, p. 51-61.

[58] ZILLES, C. B.; SOHI, G. S. **Understanding the Backward Slices of Performance Degrading Instructions**, Proceedings of the International Symposium on Computer Architecture, (Vancouver, British Columbia), p. 172-181, IEEE Computer Society and ACM SIGARCH, Jun. 2000.

APÊNDICES

Apêndice 1 – Gráficos por DSVs individualizados por MOD/ORG/LIN/ASS/PRG (Exemplo para UNI)

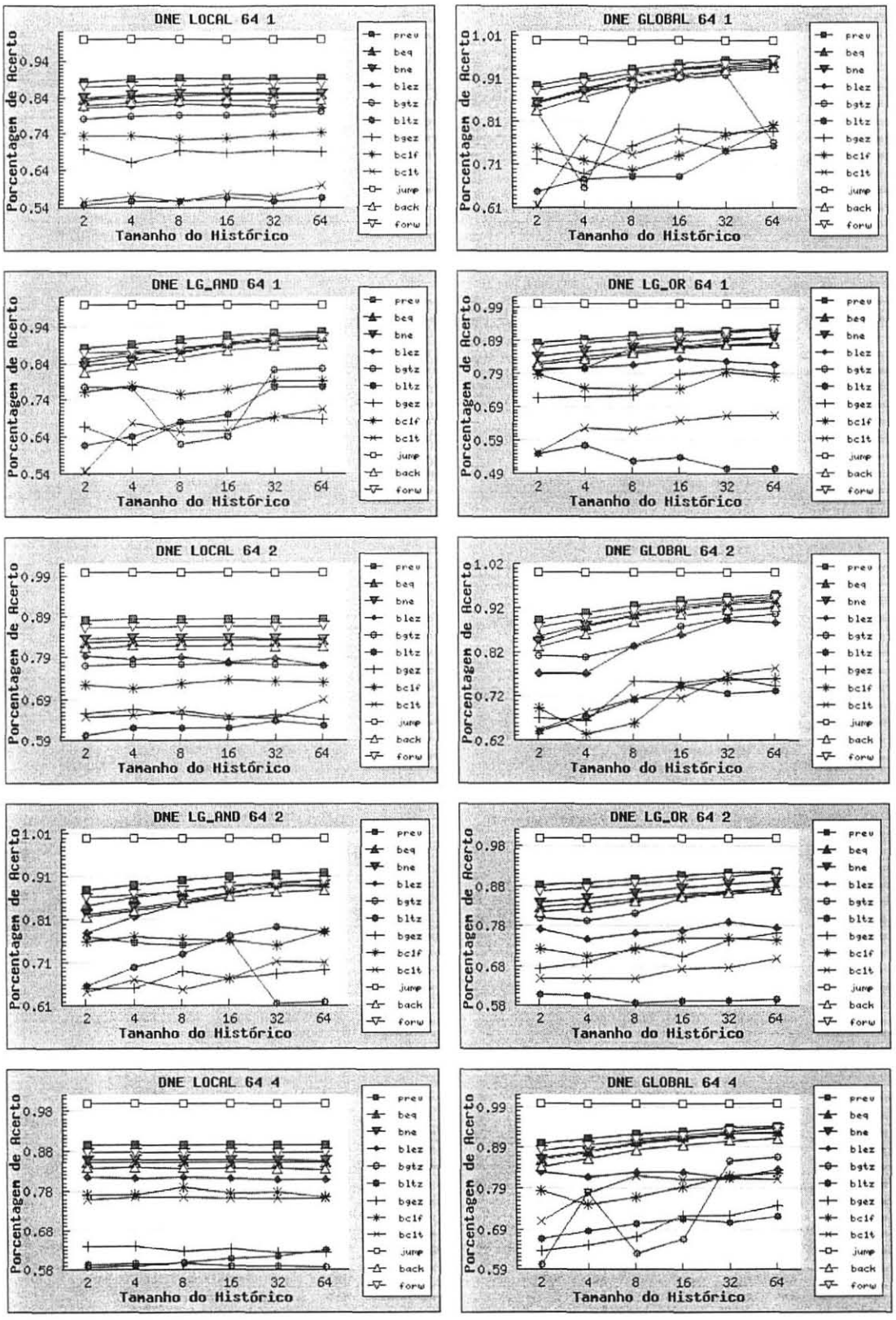


Obs.: Estamos apresentando apenas os gráficos por desvios individualizados por *benchmarks* para o modelo UNI devido ao número grande de imagens geradas para todos os modelos (1.160 no total).

Apêndice 1 - Folha 1

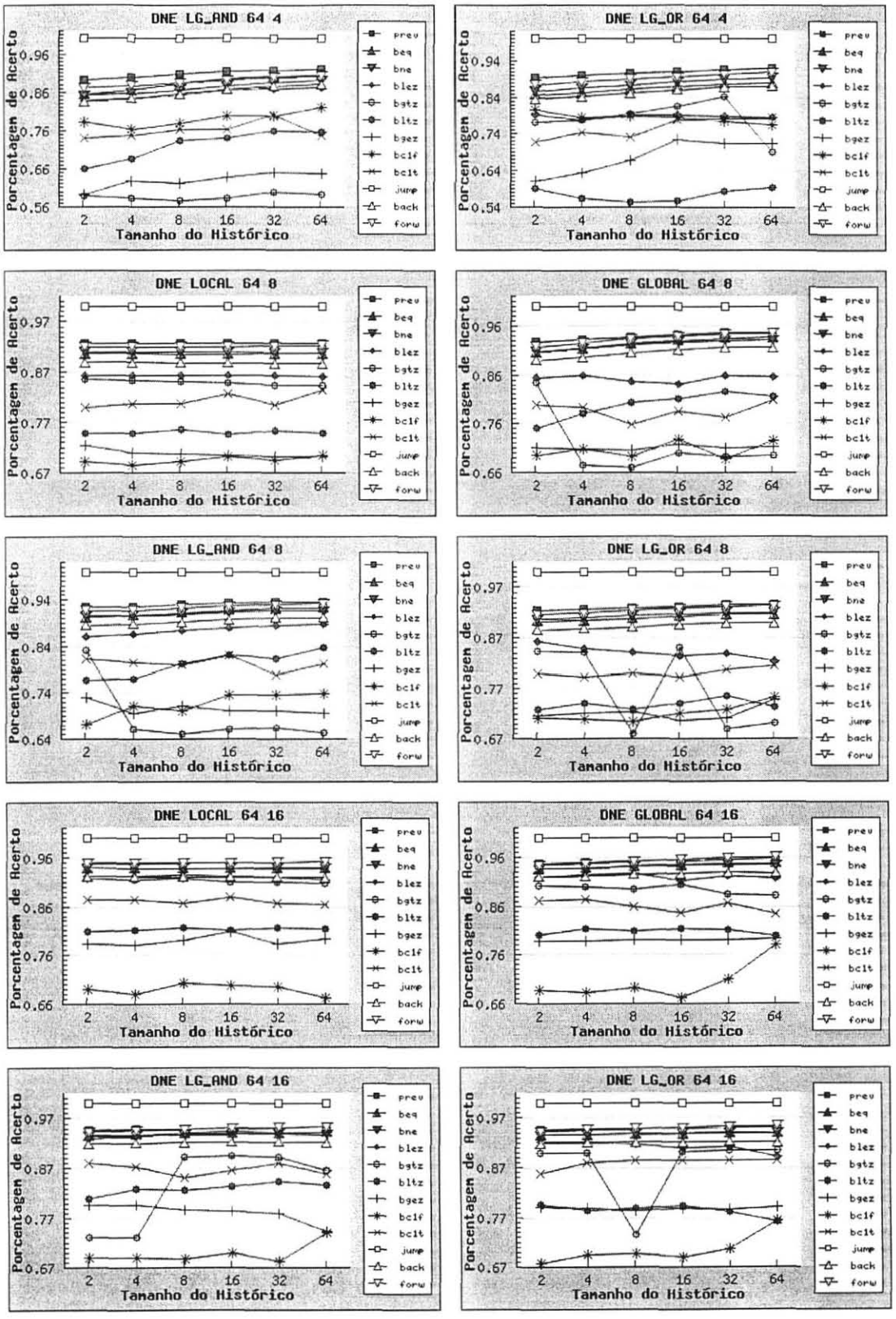
Gráficos por DSVs individualizados por MOD/ORG/LIN/ASS/PRG (Exemplo para UNI)

Apêndice 2 – Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



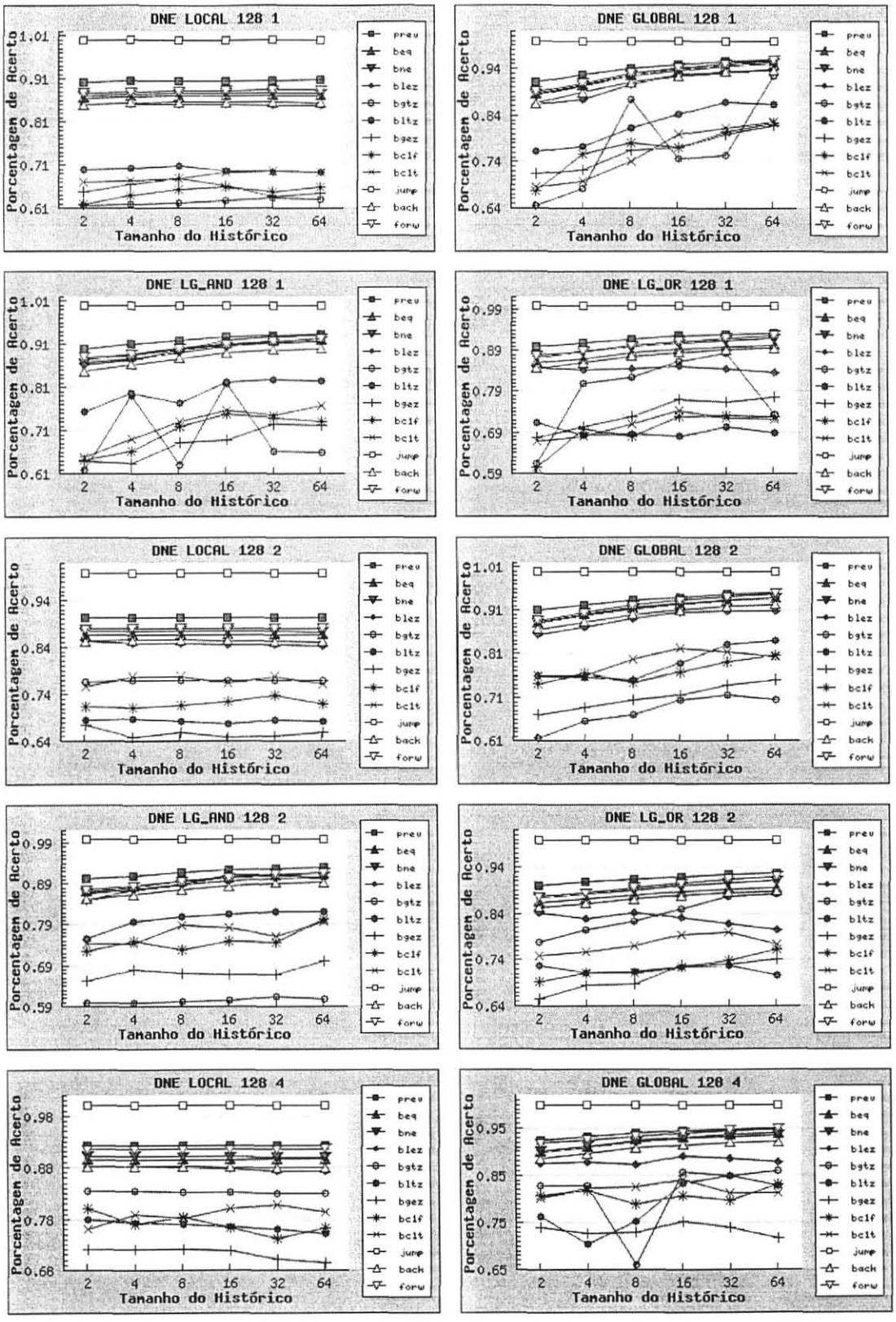
Apêndice 2 - Folha 1

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



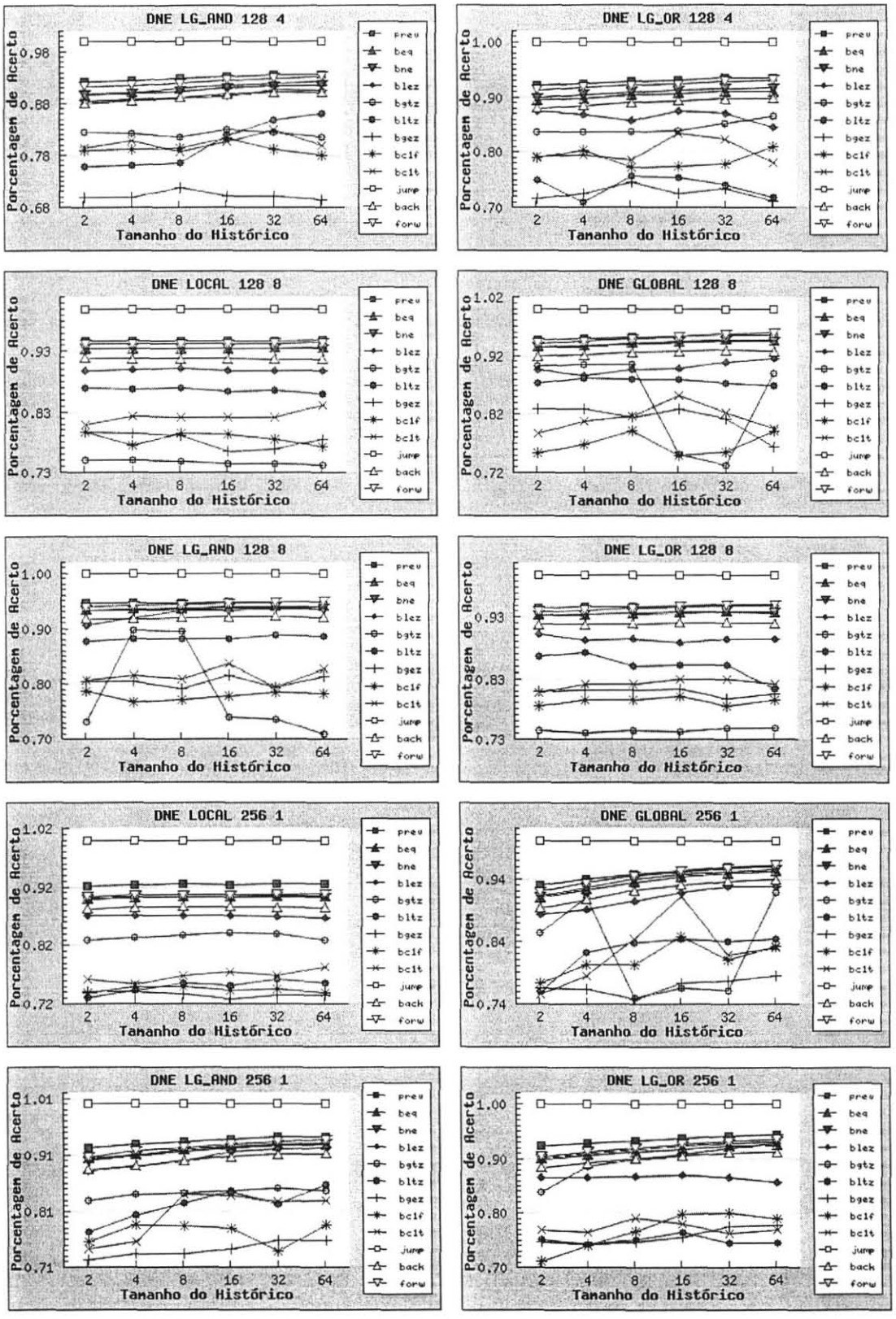
Apêndice 2 - Folha 2

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



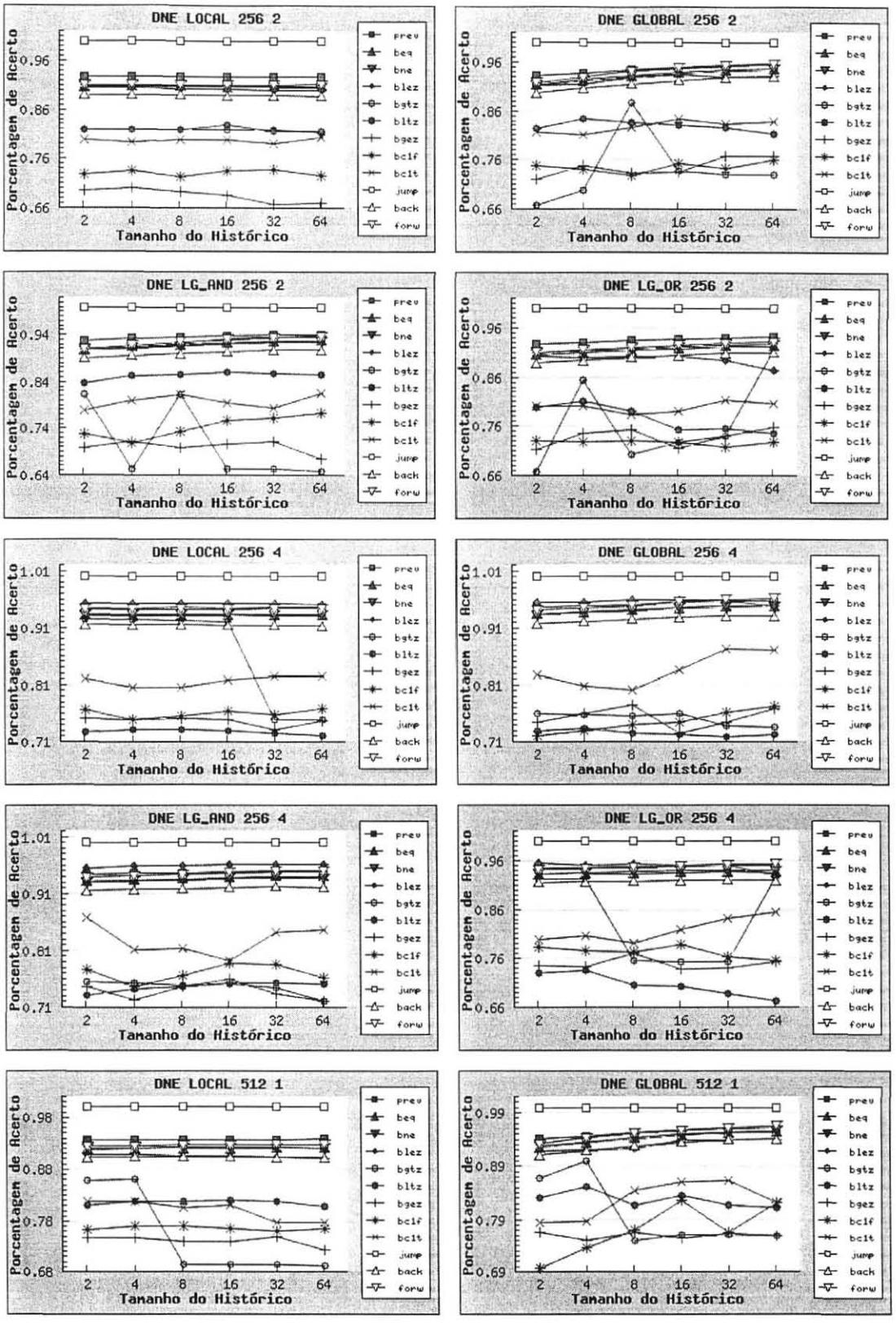
Apêndice 2 - Folha 3

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



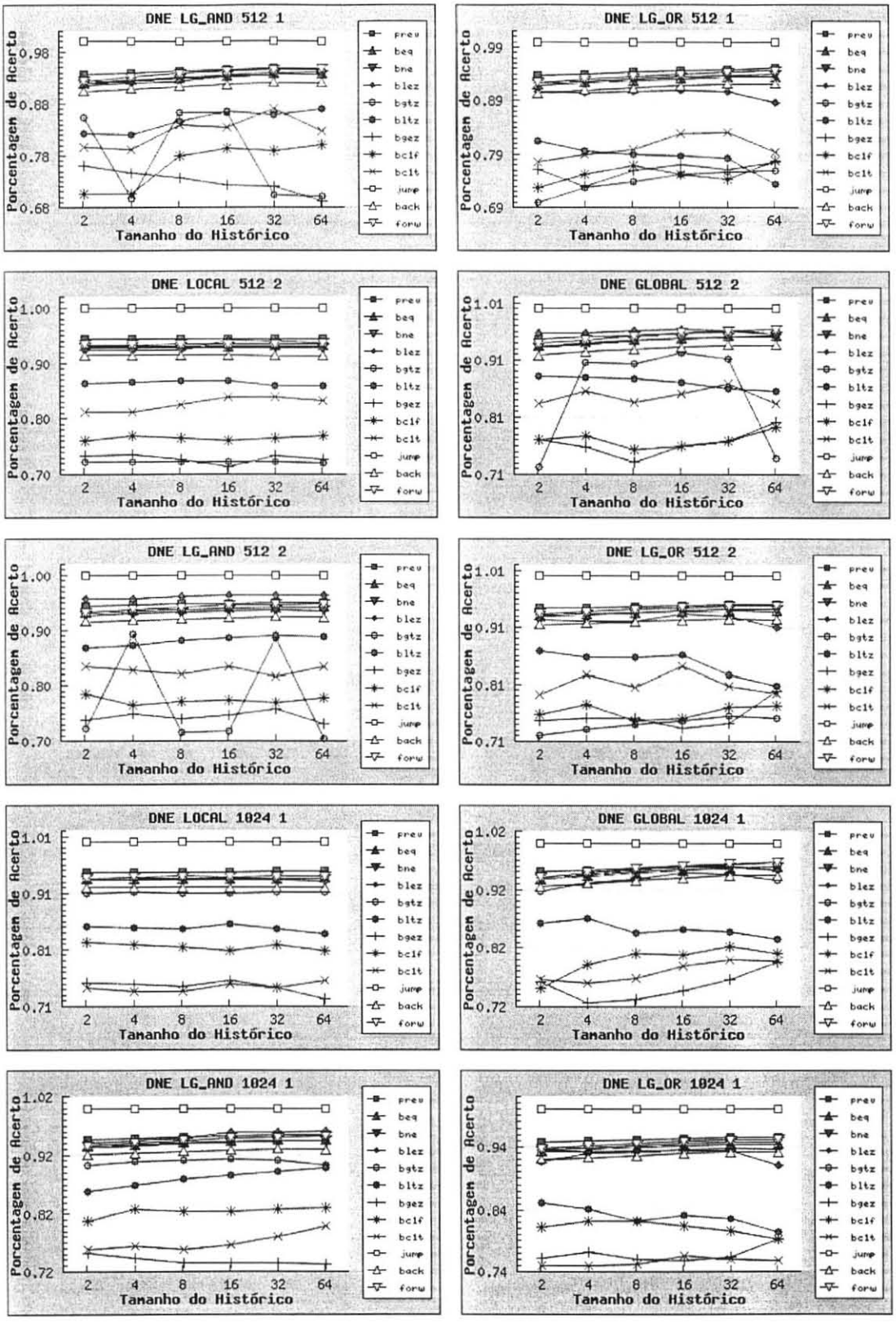
Apêndice 2 - Folha 4

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



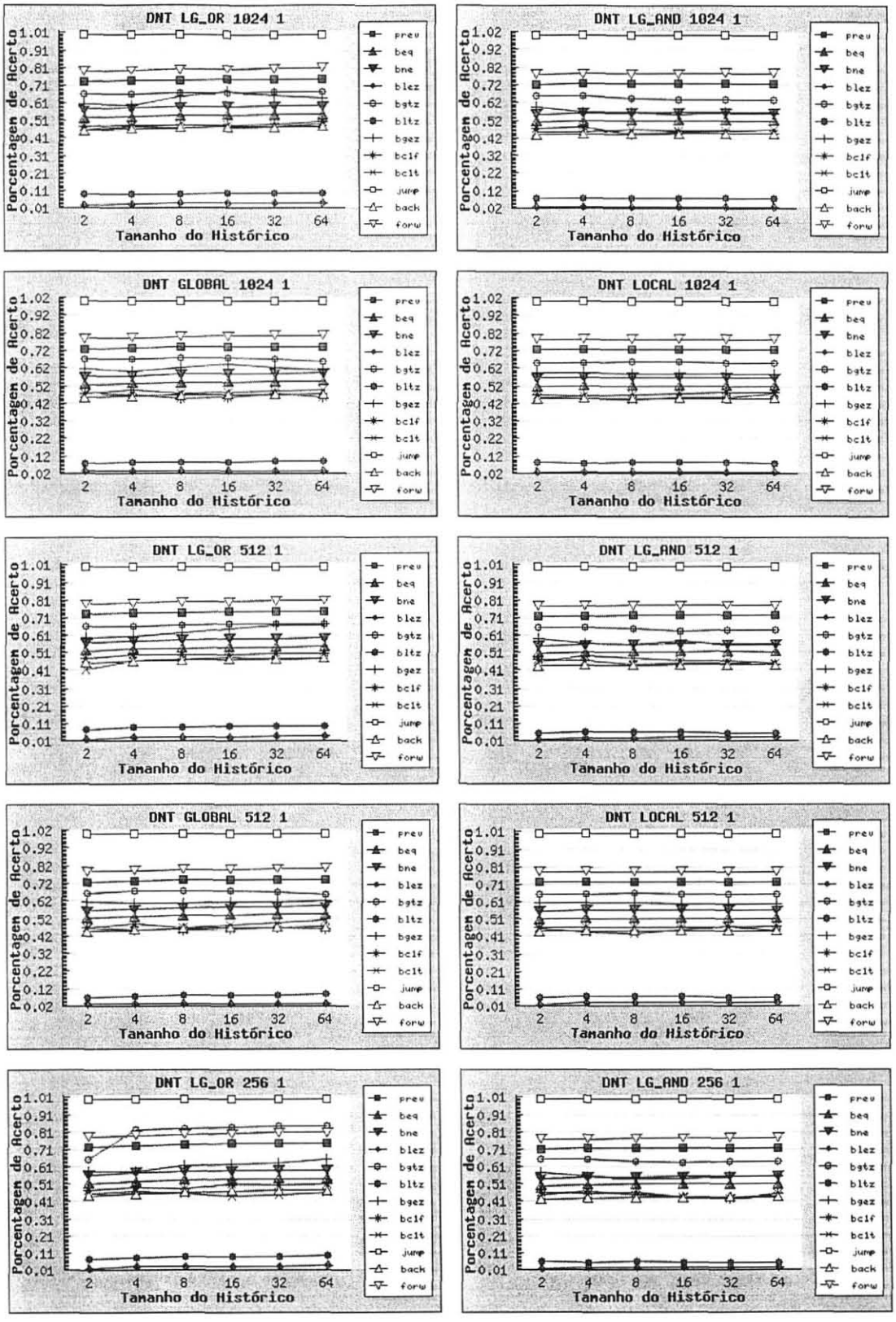
Apêndice 2 - Folha 5

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



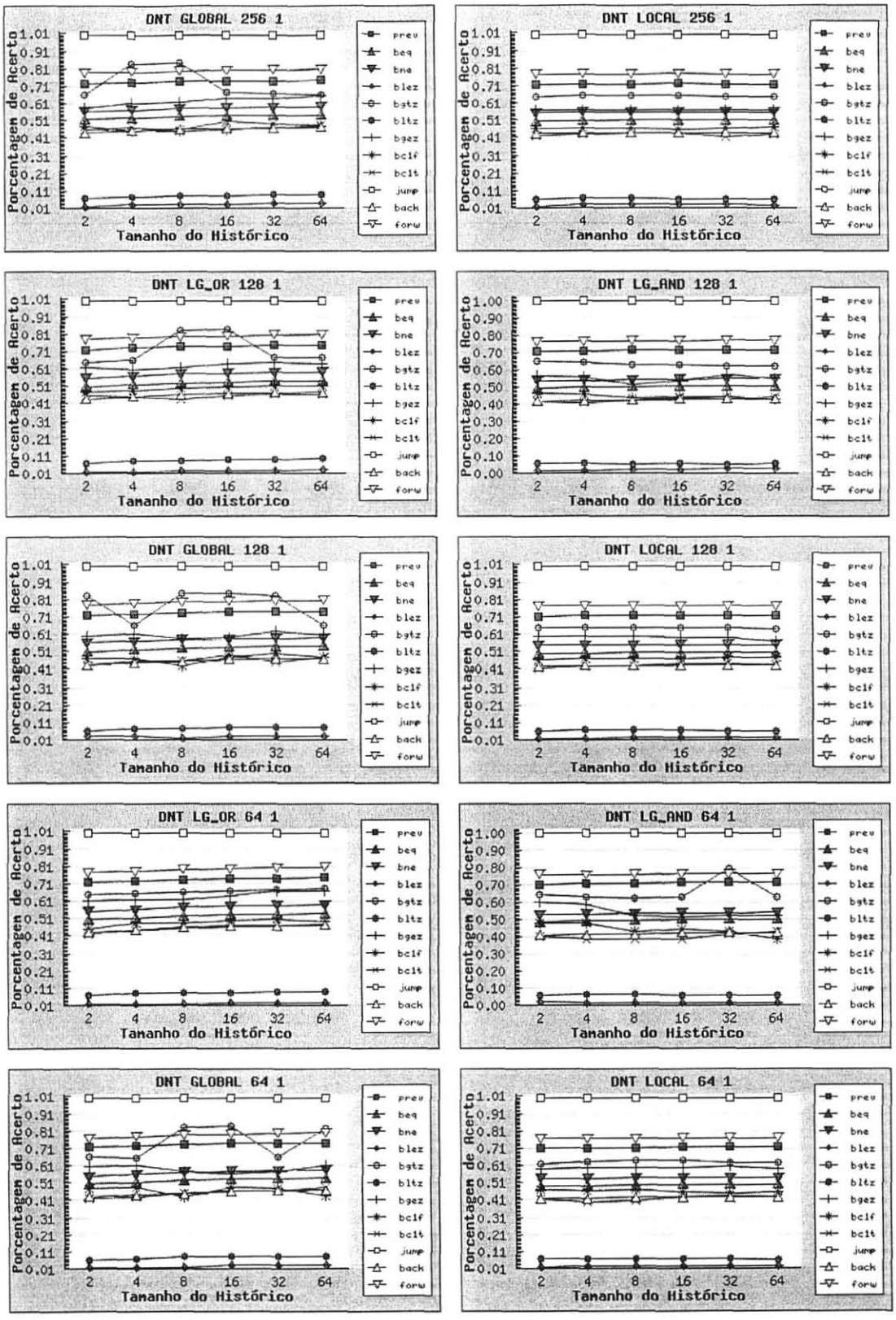
Apêndice 2 - Folha 6

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



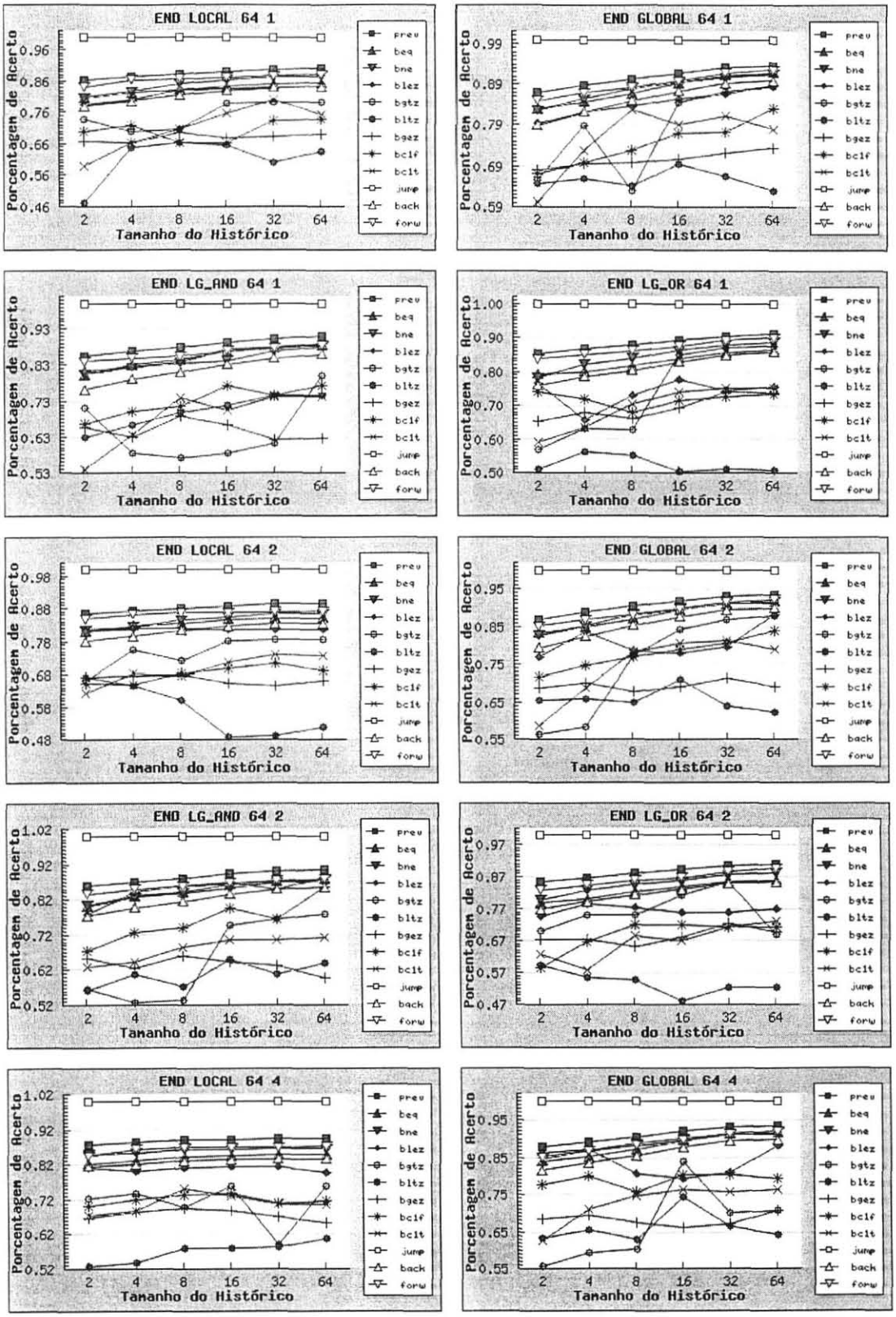
Apêndice 2 - Folha 7

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



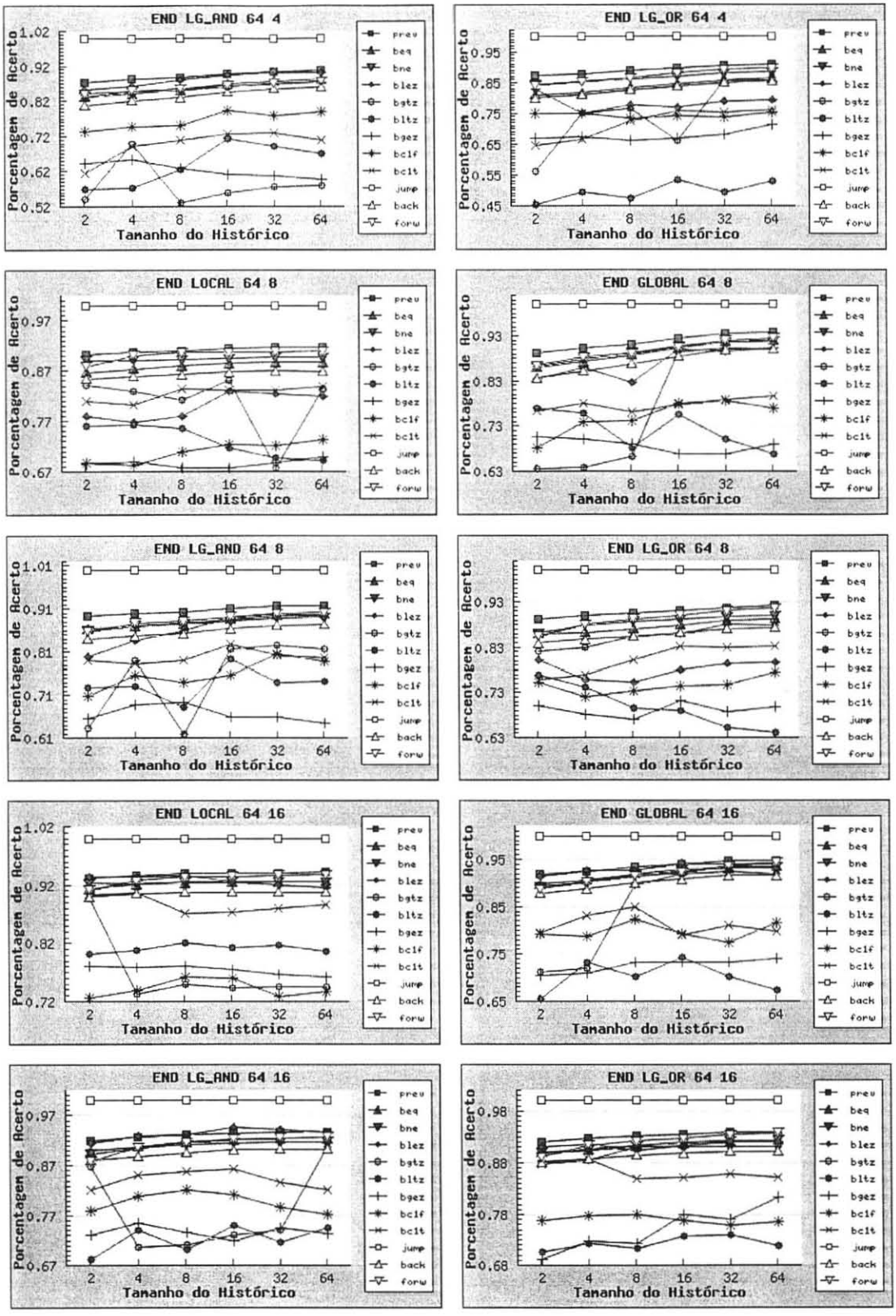
Apêndice 2 - Folha 8

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



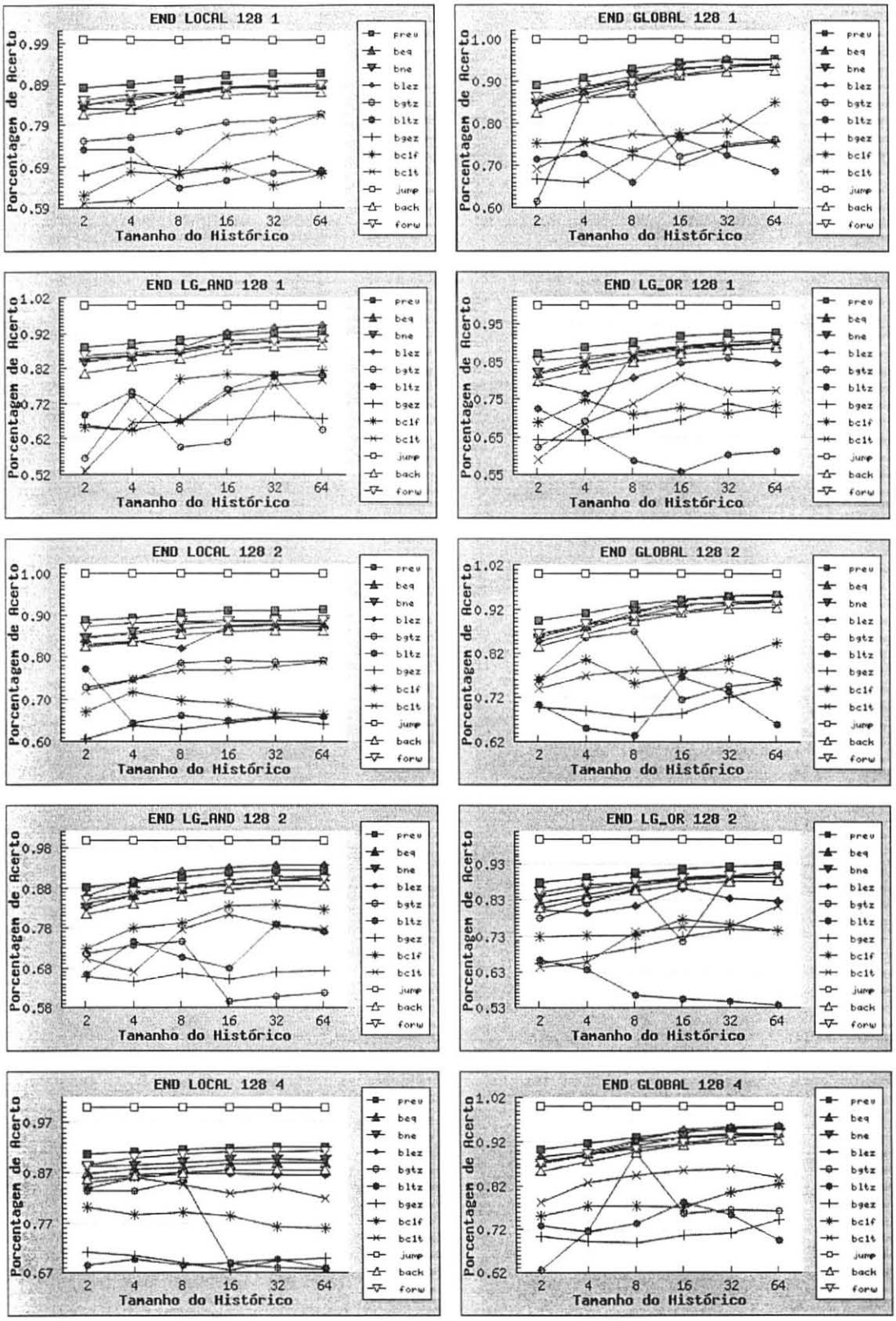
Apêndice 2 - Folha 9

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



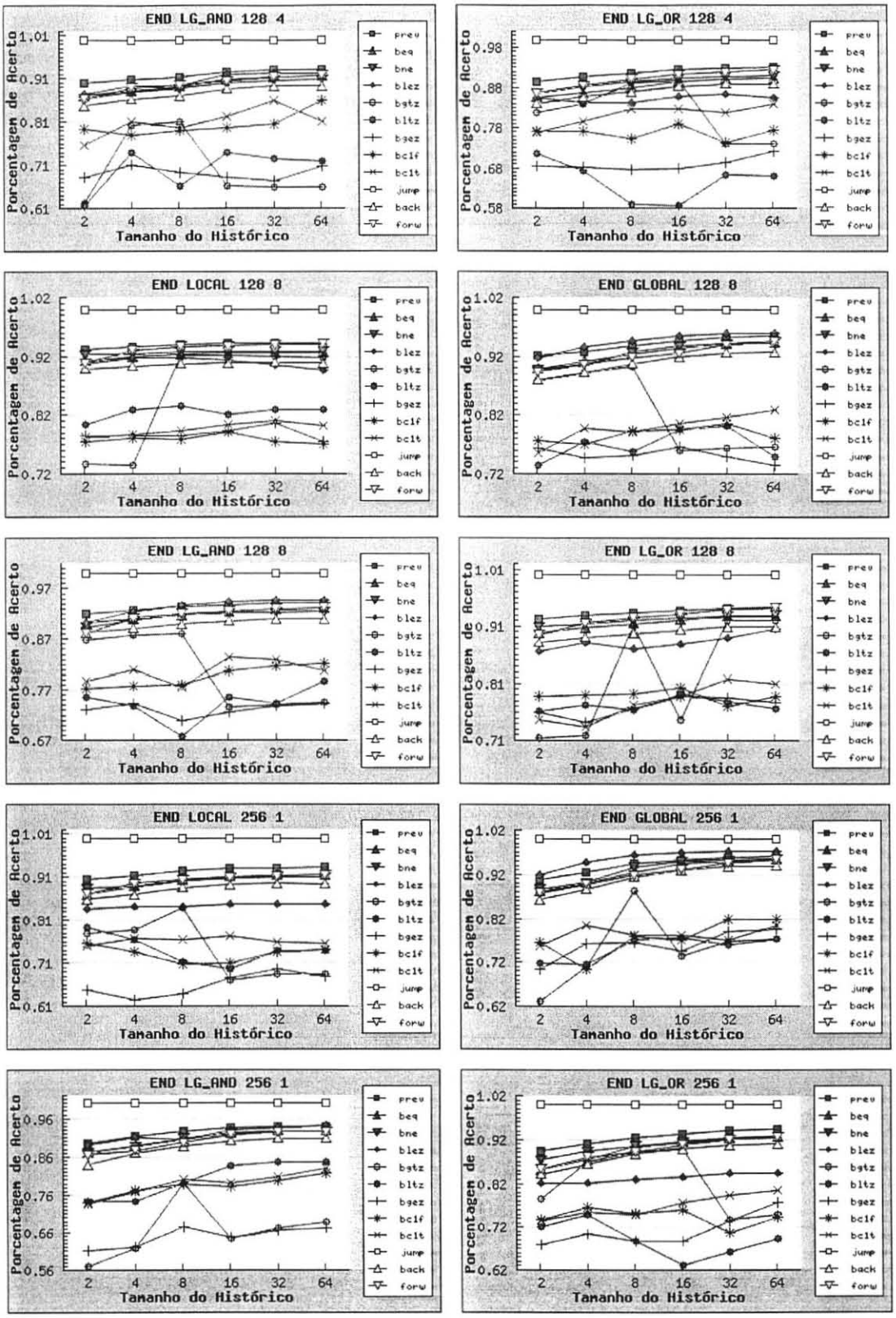
Apêndice 2 - Folha 10

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



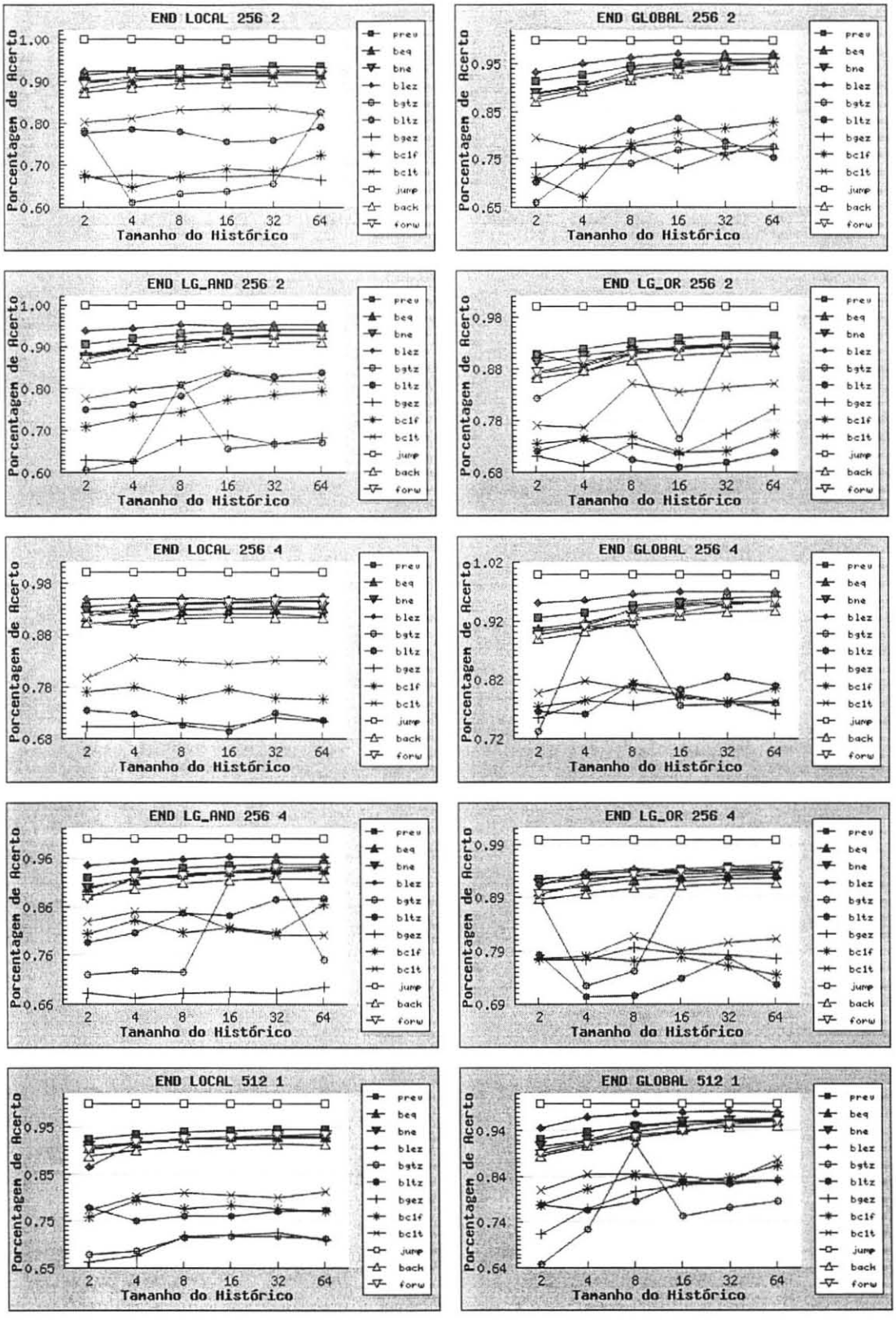
Apêndice 2 - Folha 11

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



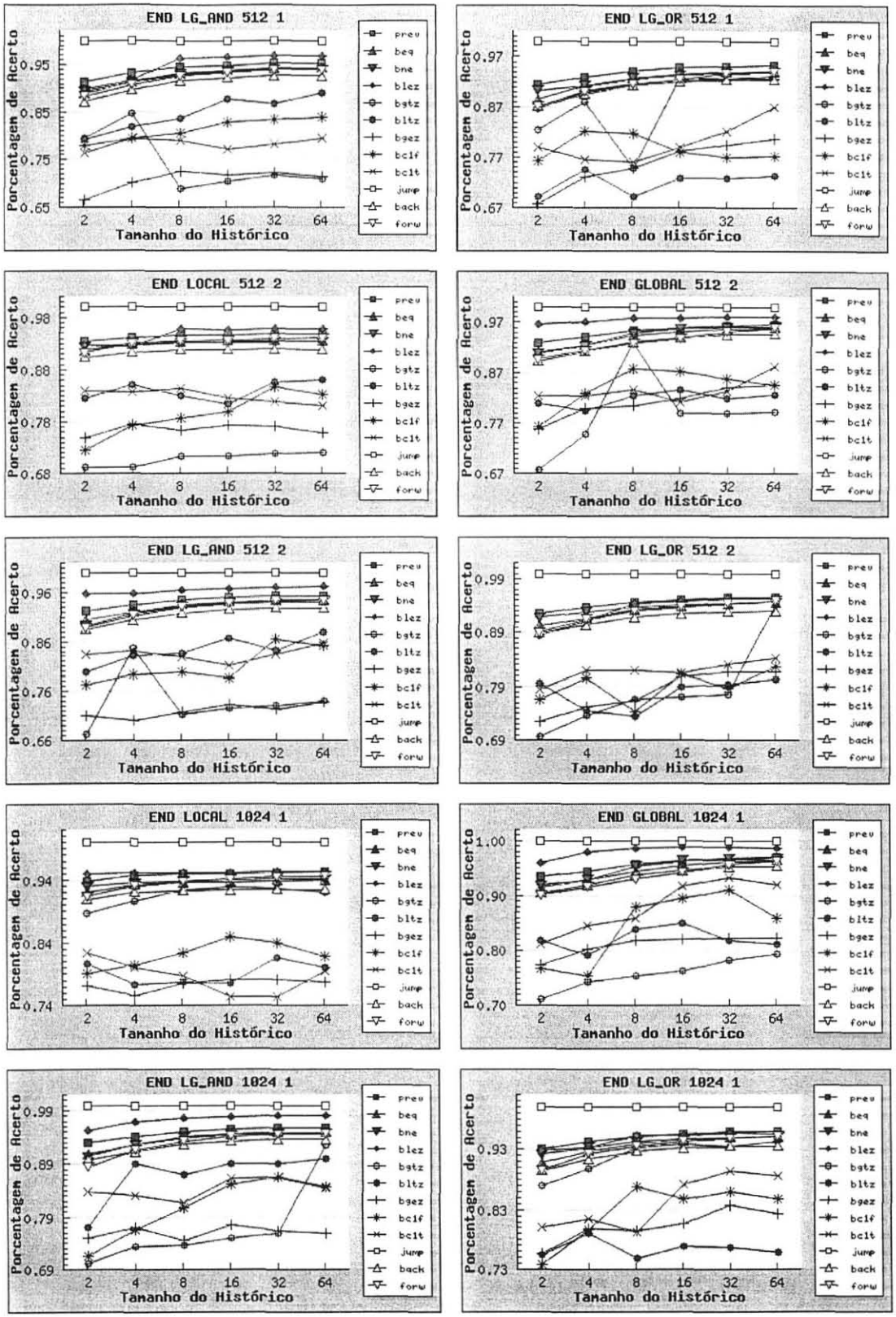
Apêndice 2 - Folha 12

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS



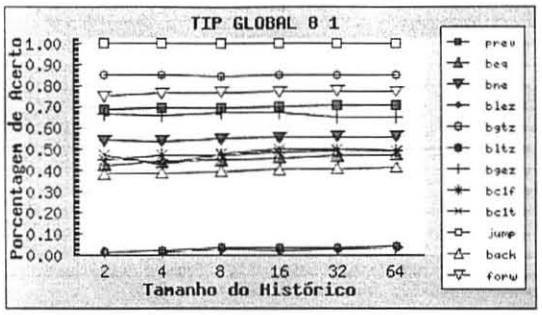
Apêndice 2 - Folha 13

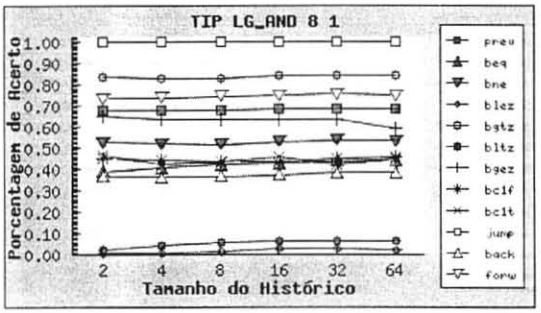
Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS

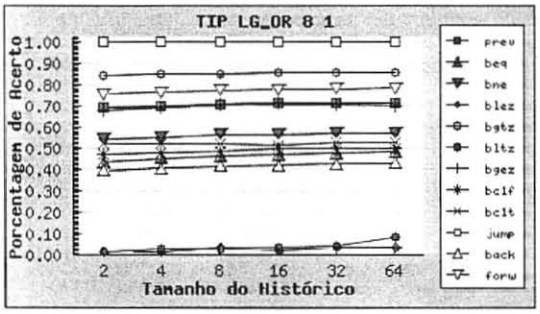


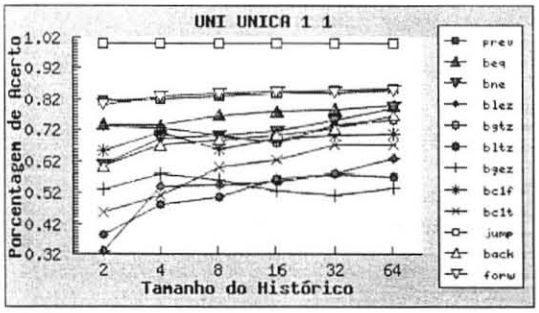
Apêndice 2 - Folha 14

Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS





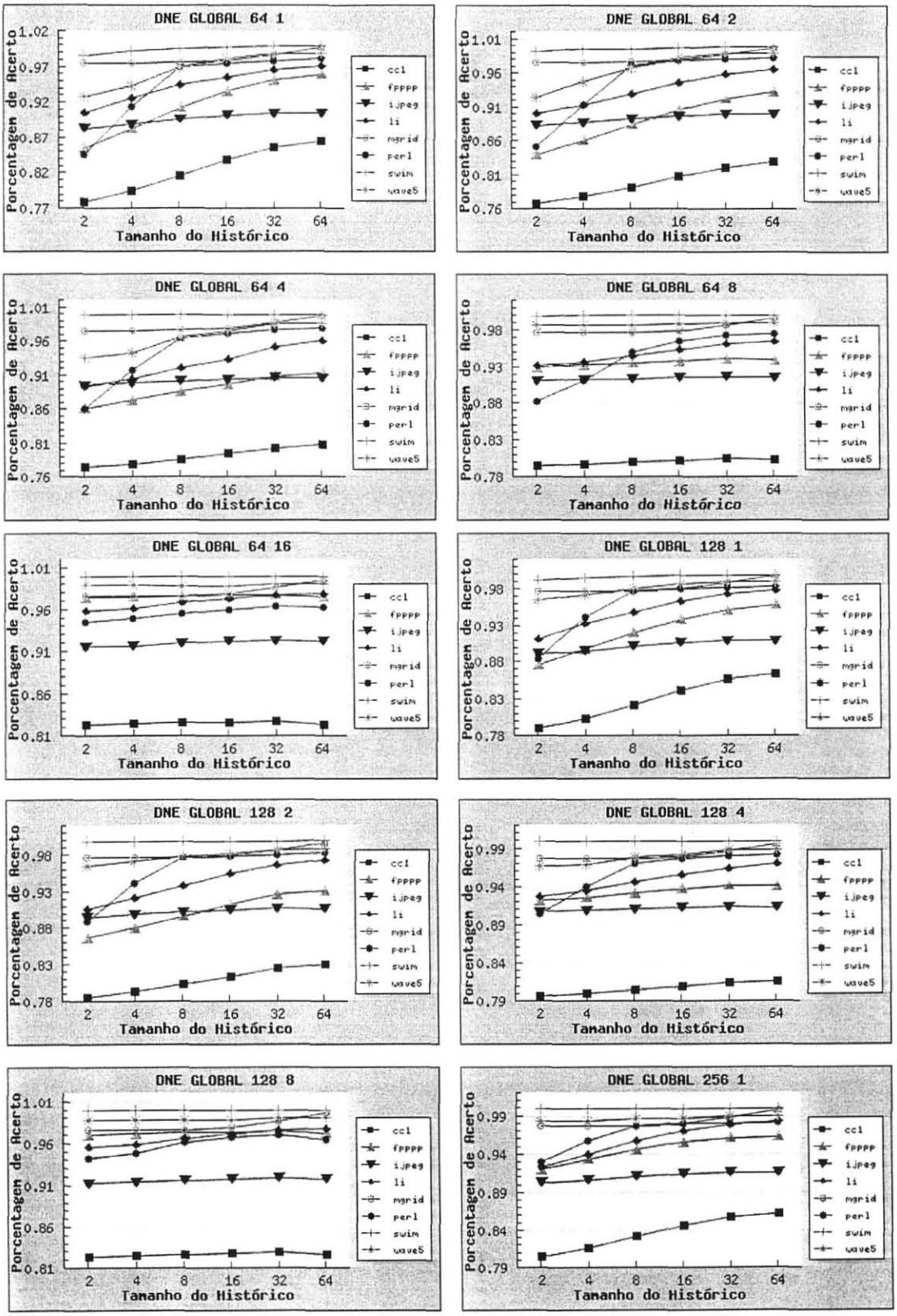




Apêndice 2 - Folha 15

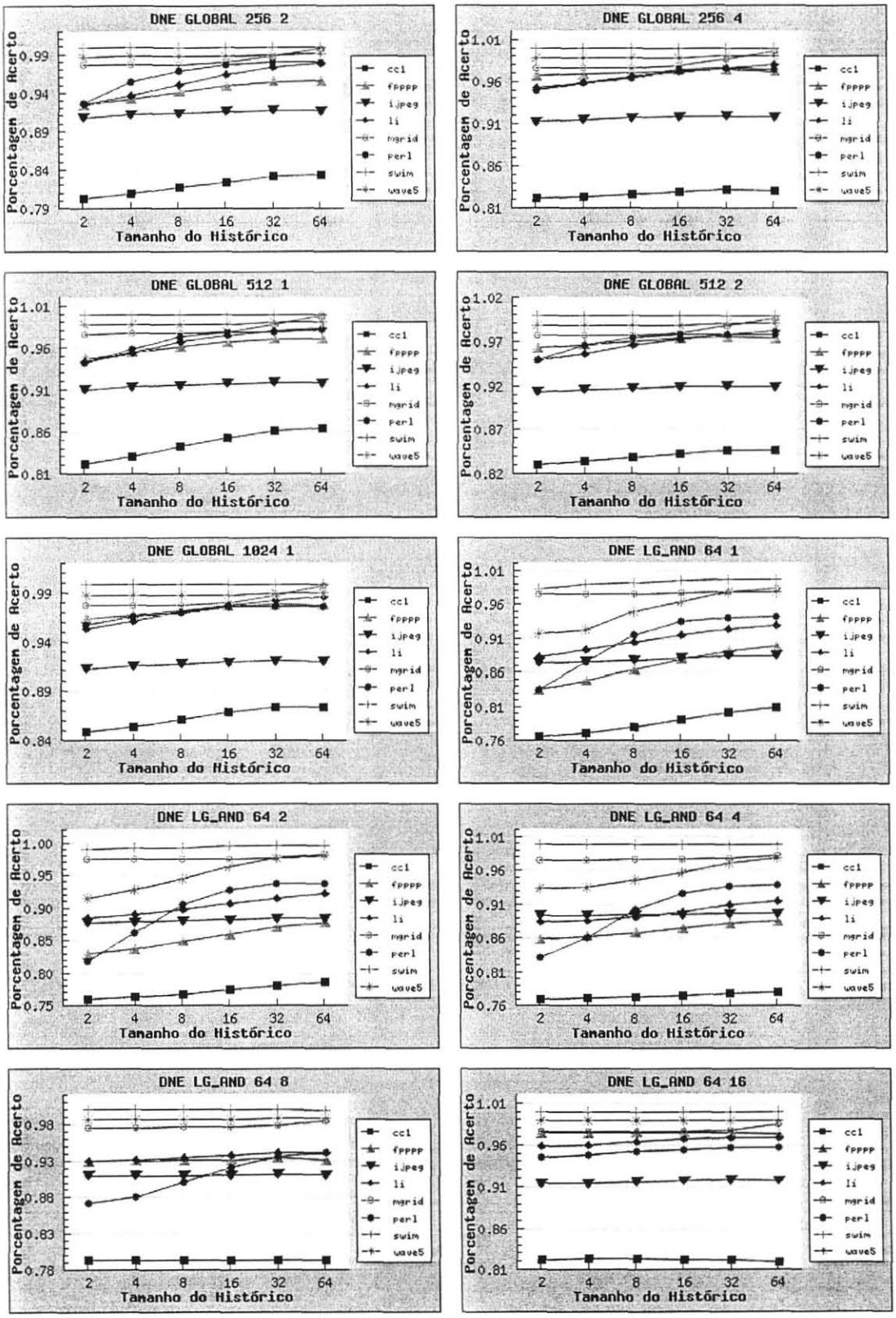
Gráficos por DSVs para a média dos oito PRGs individualizados por MOD/ORG/LIN/ASS

Apêndice 3 – Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



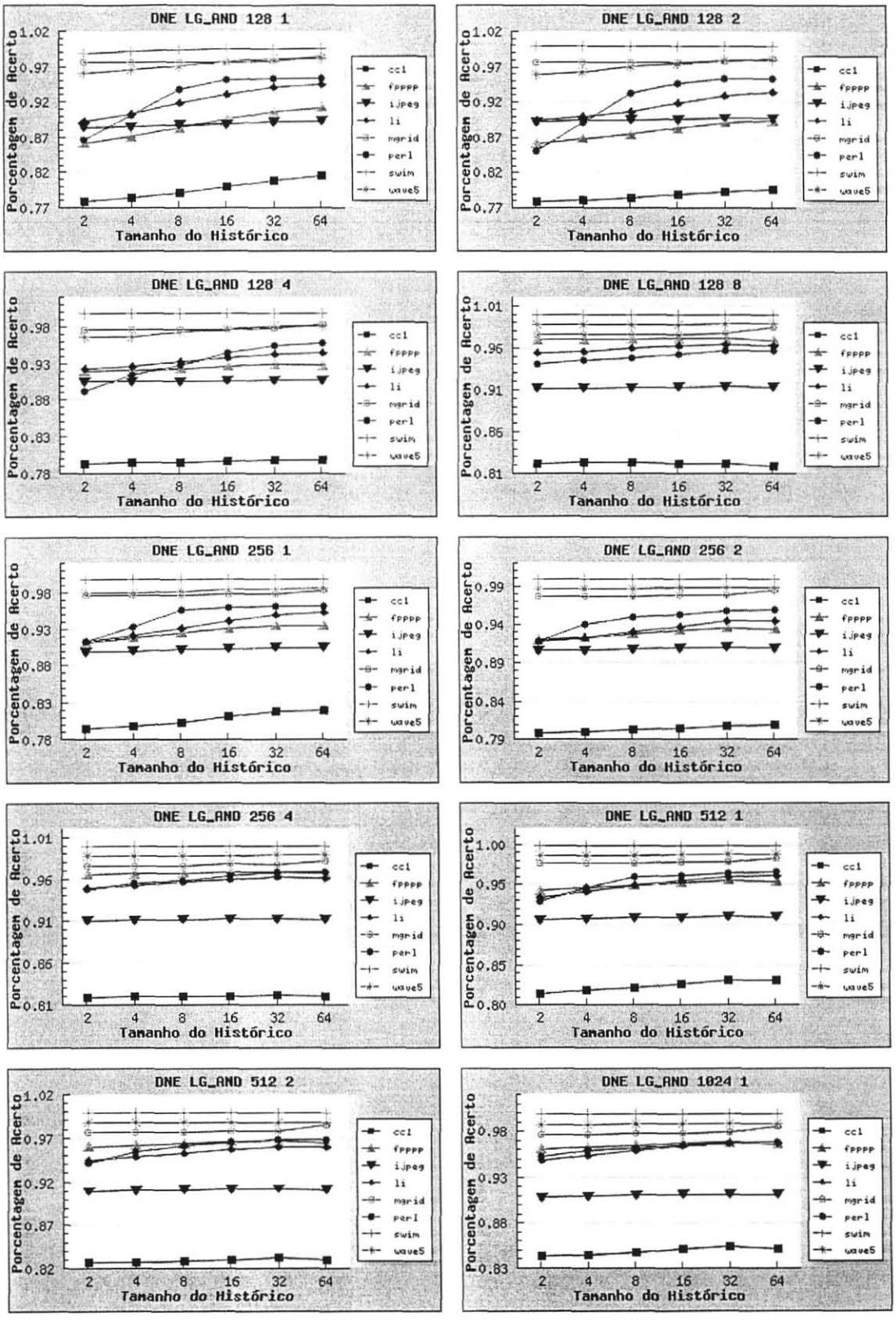
Apêndice 3 - Folha 1

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



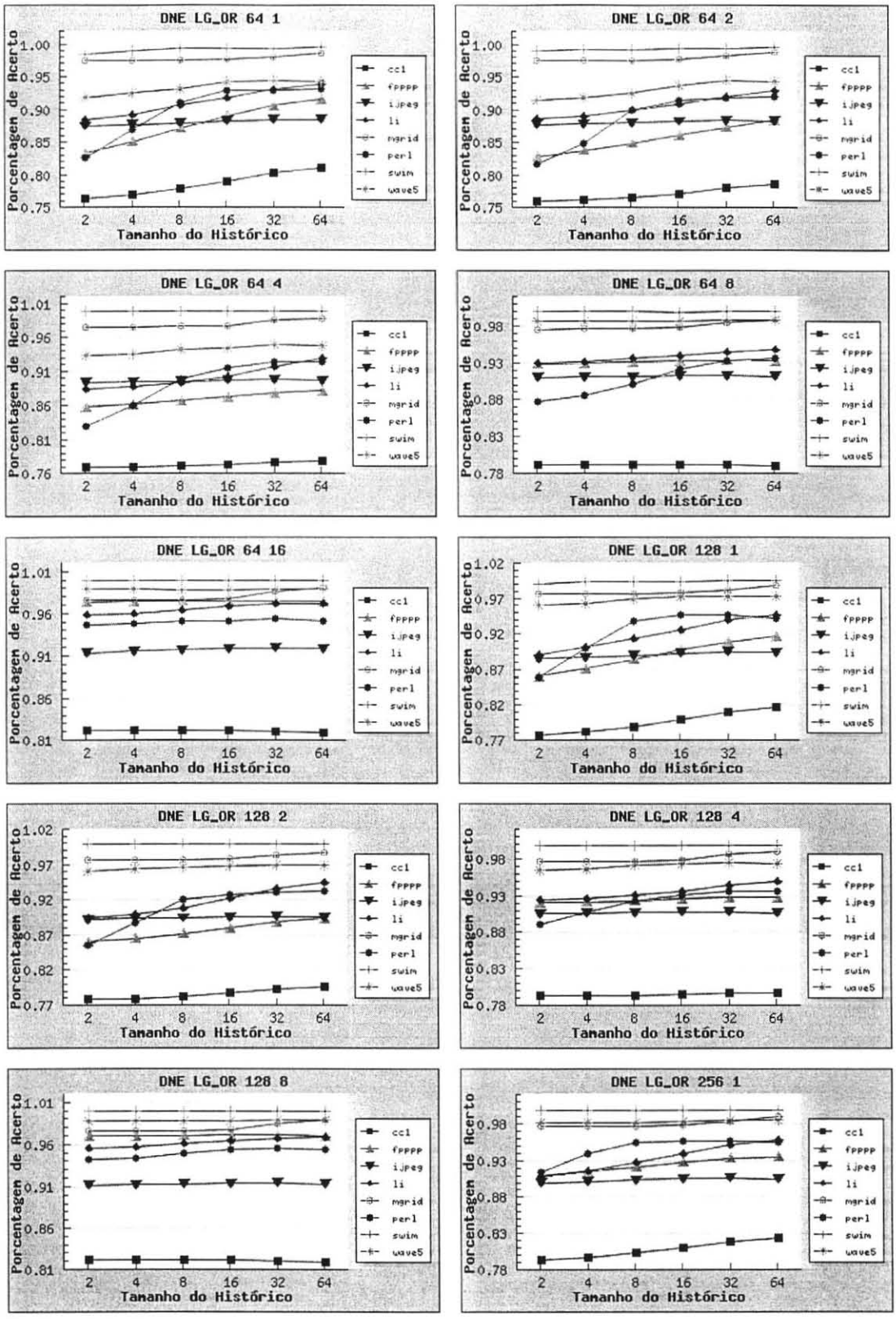
Apêndice 3 - Folha 2

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



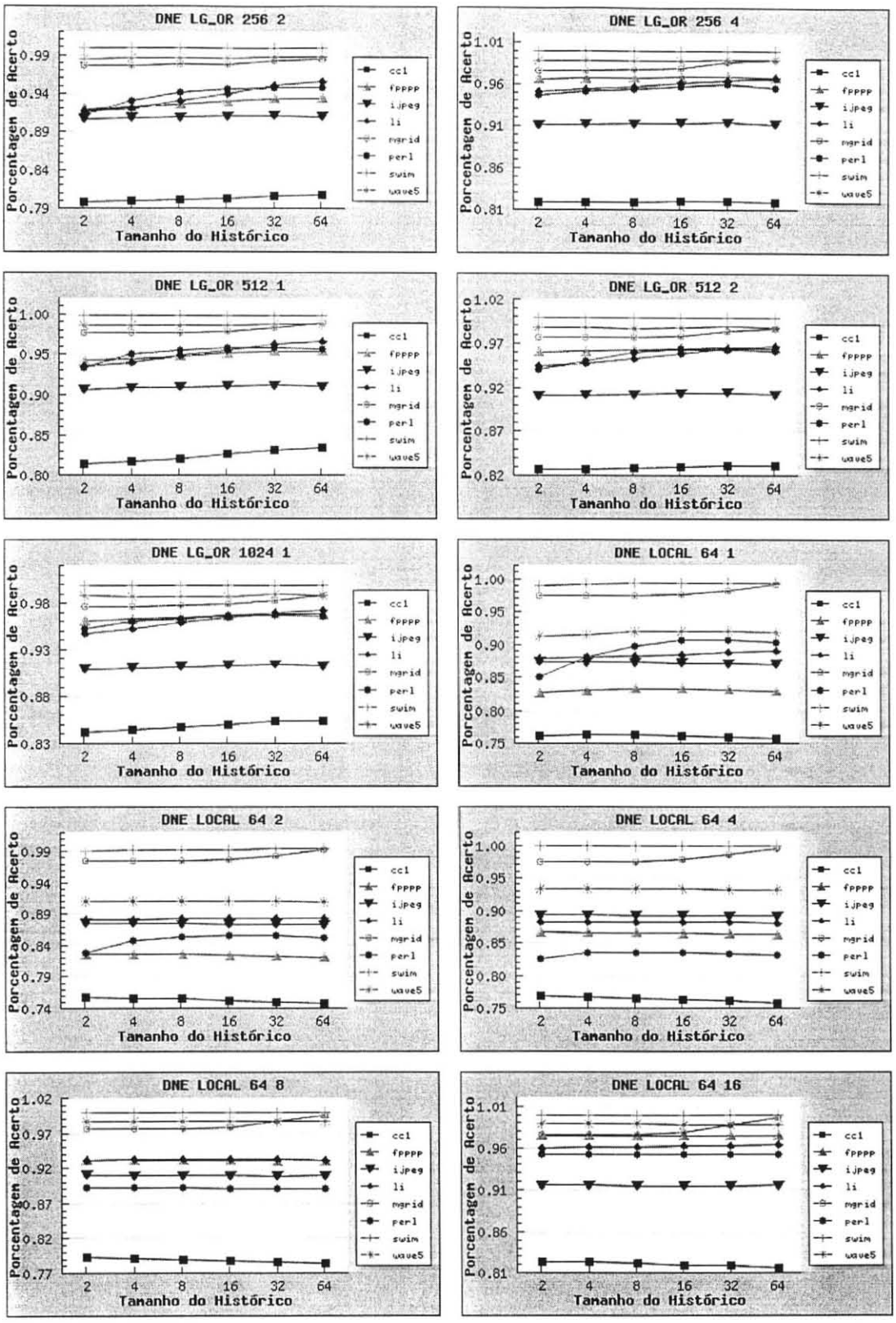
Apêndice 3 - Folha 3

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



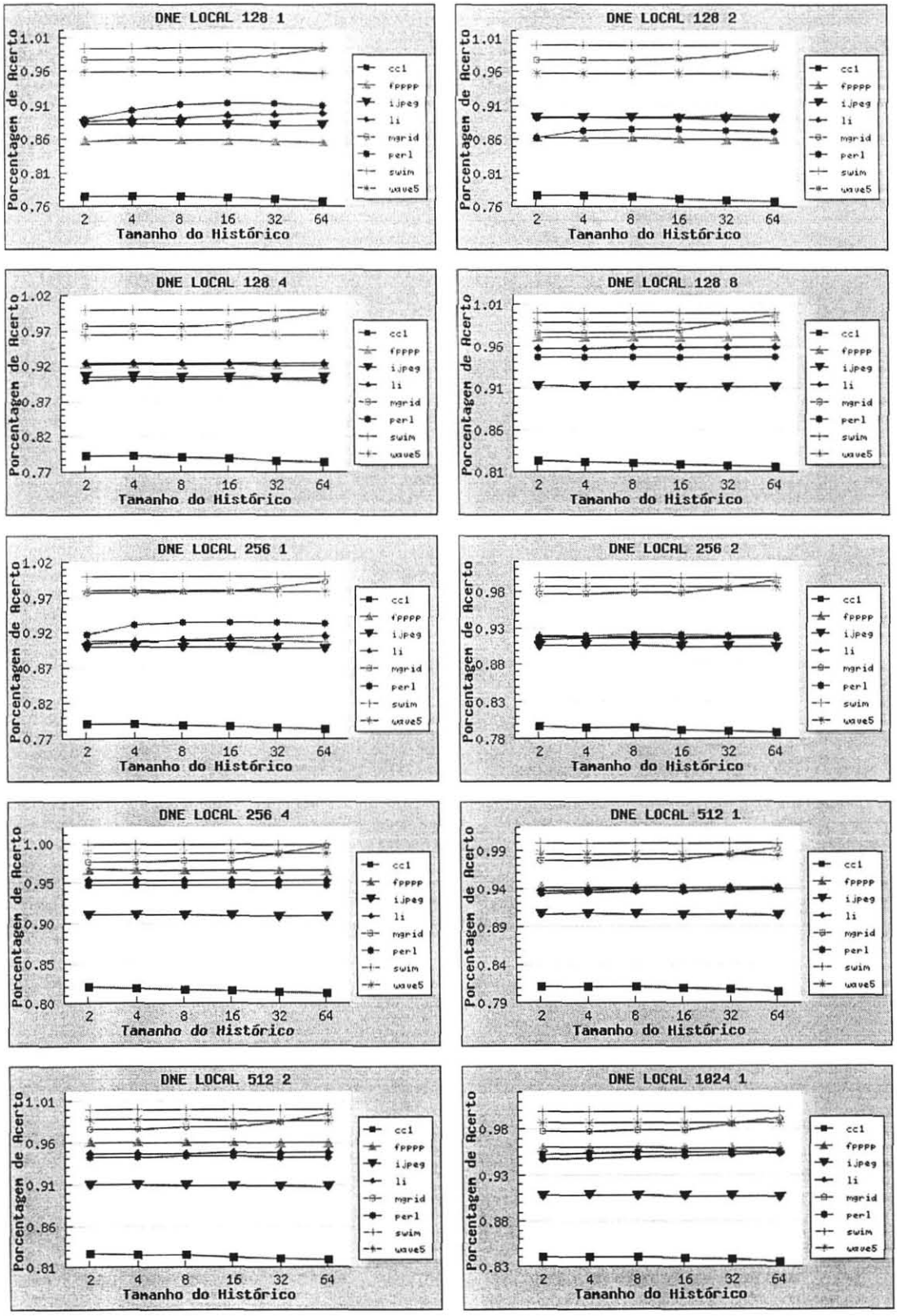
Apêndice 3 - Folha 4

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



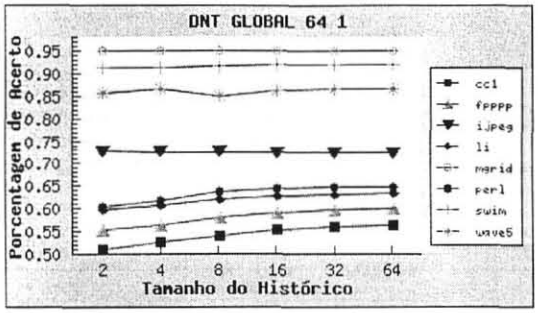
Apêndice 3 - Folha 5

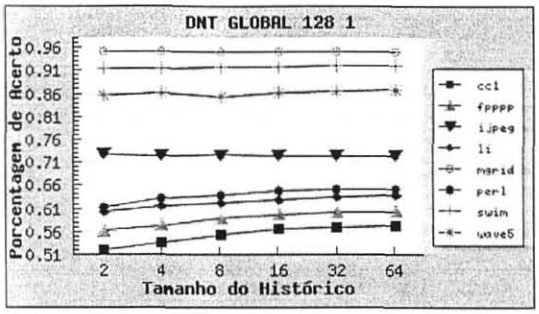
Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS

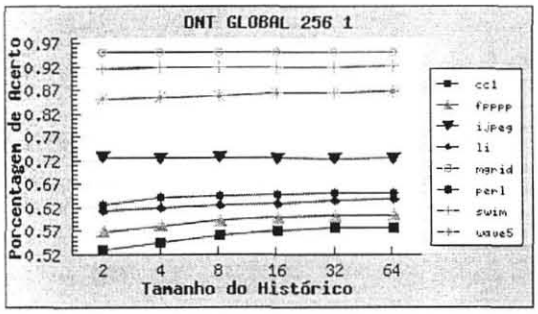


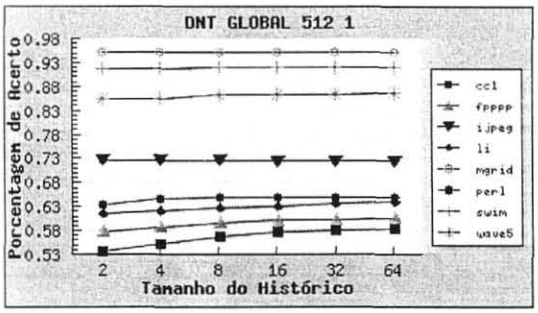
Apêndice 3 - Folha 6

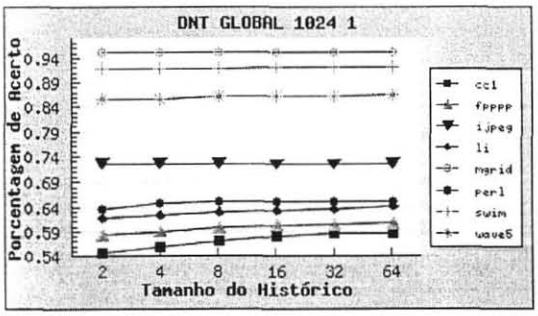
Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS

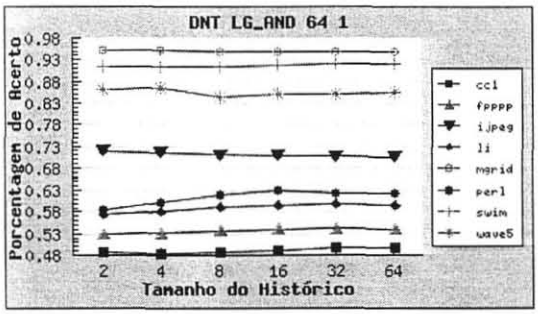


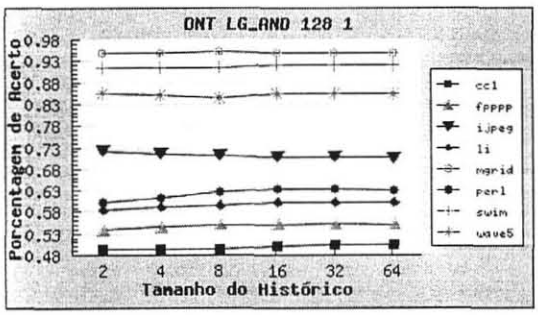


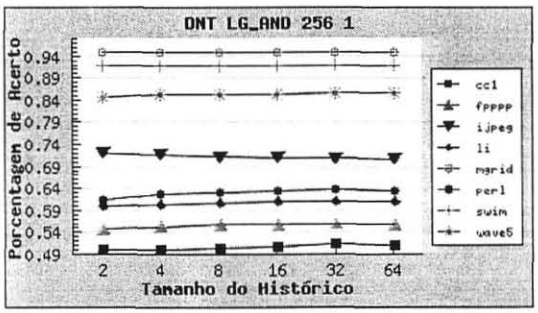


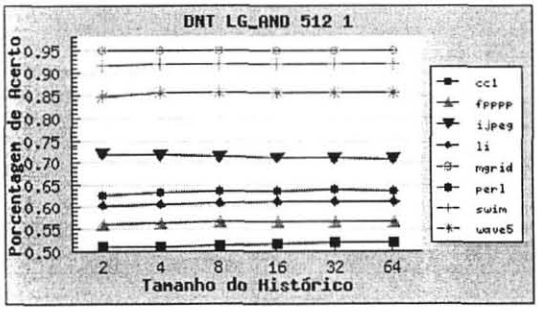


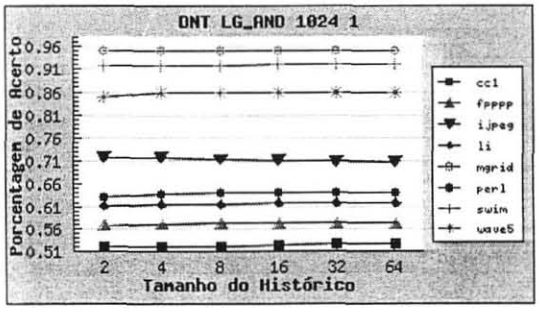






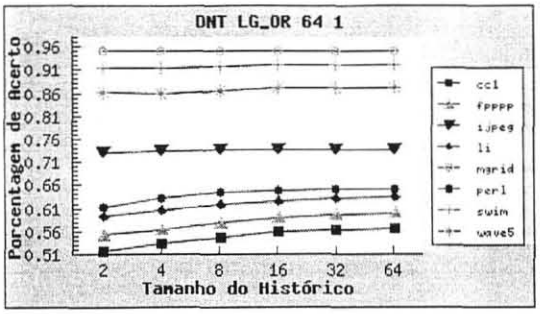


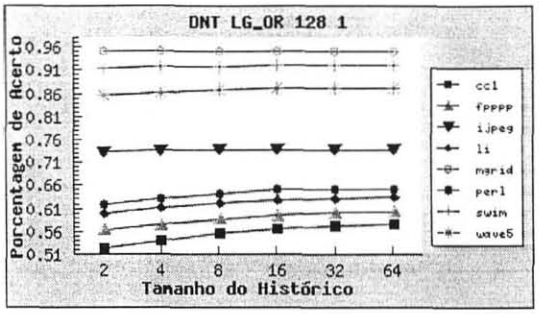


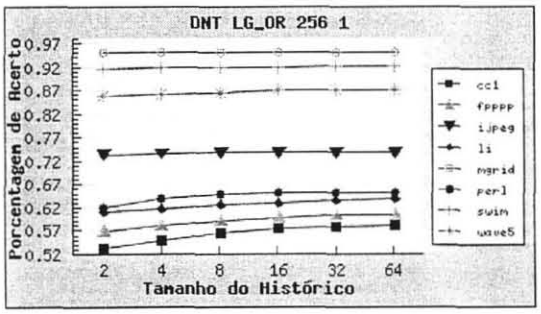


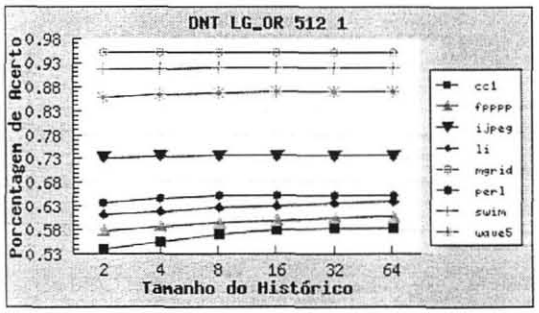
Apêndice 3 - Folha 7

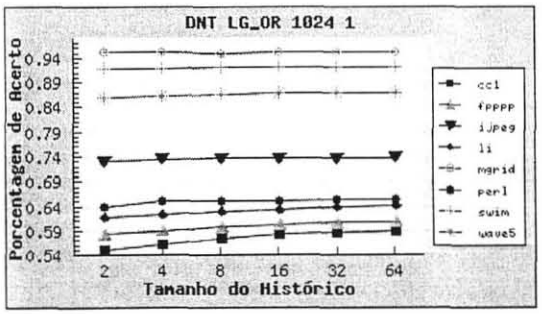
Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS

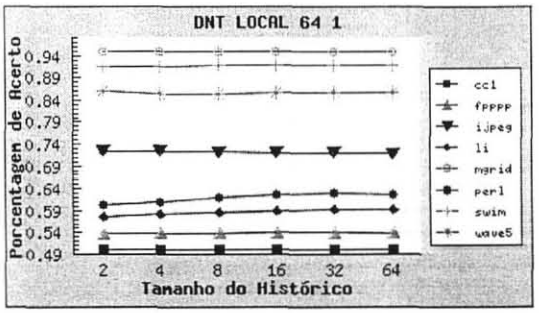


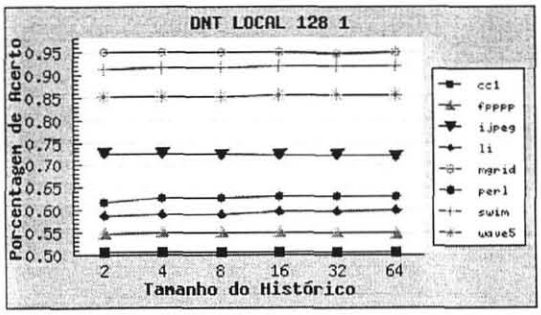






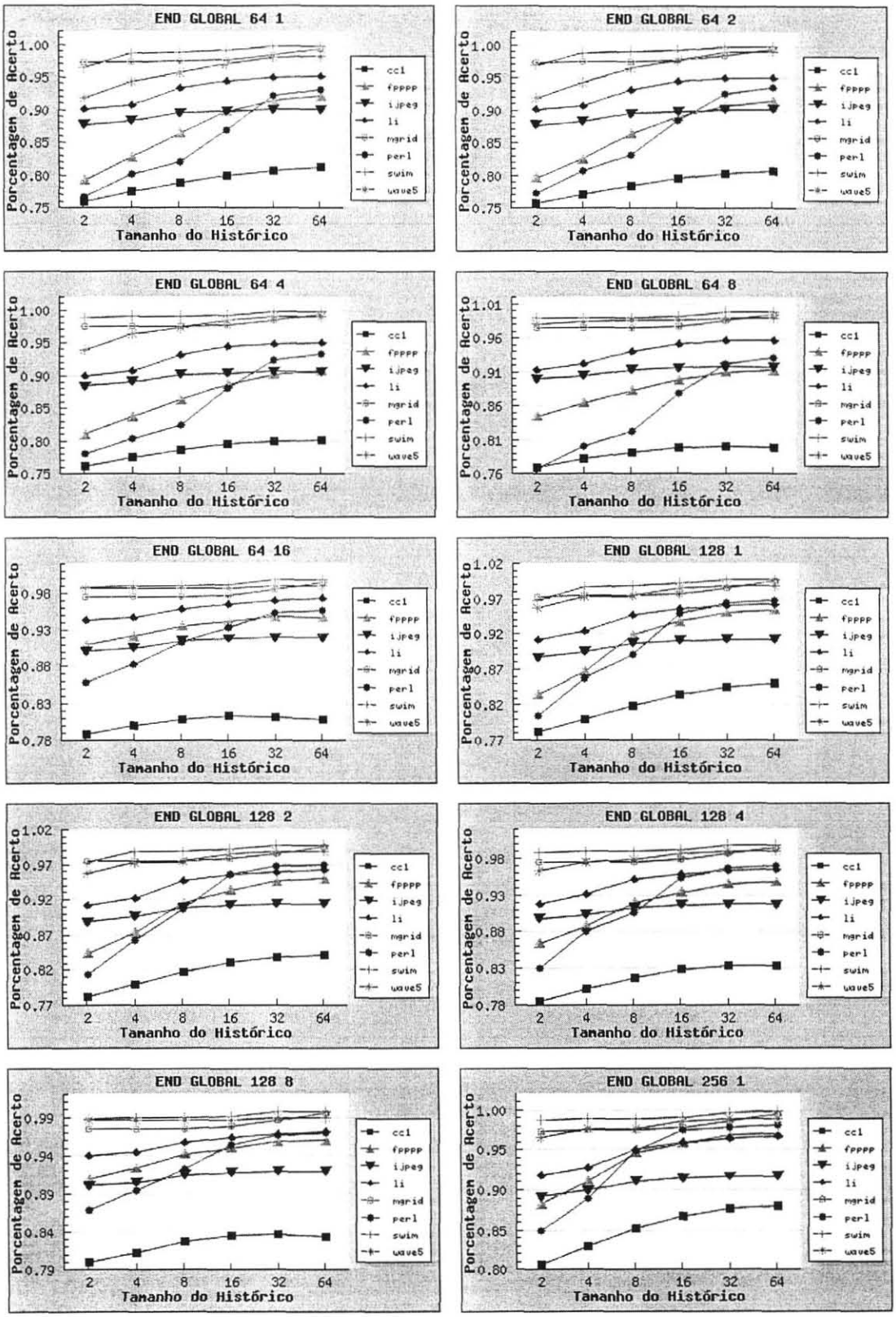






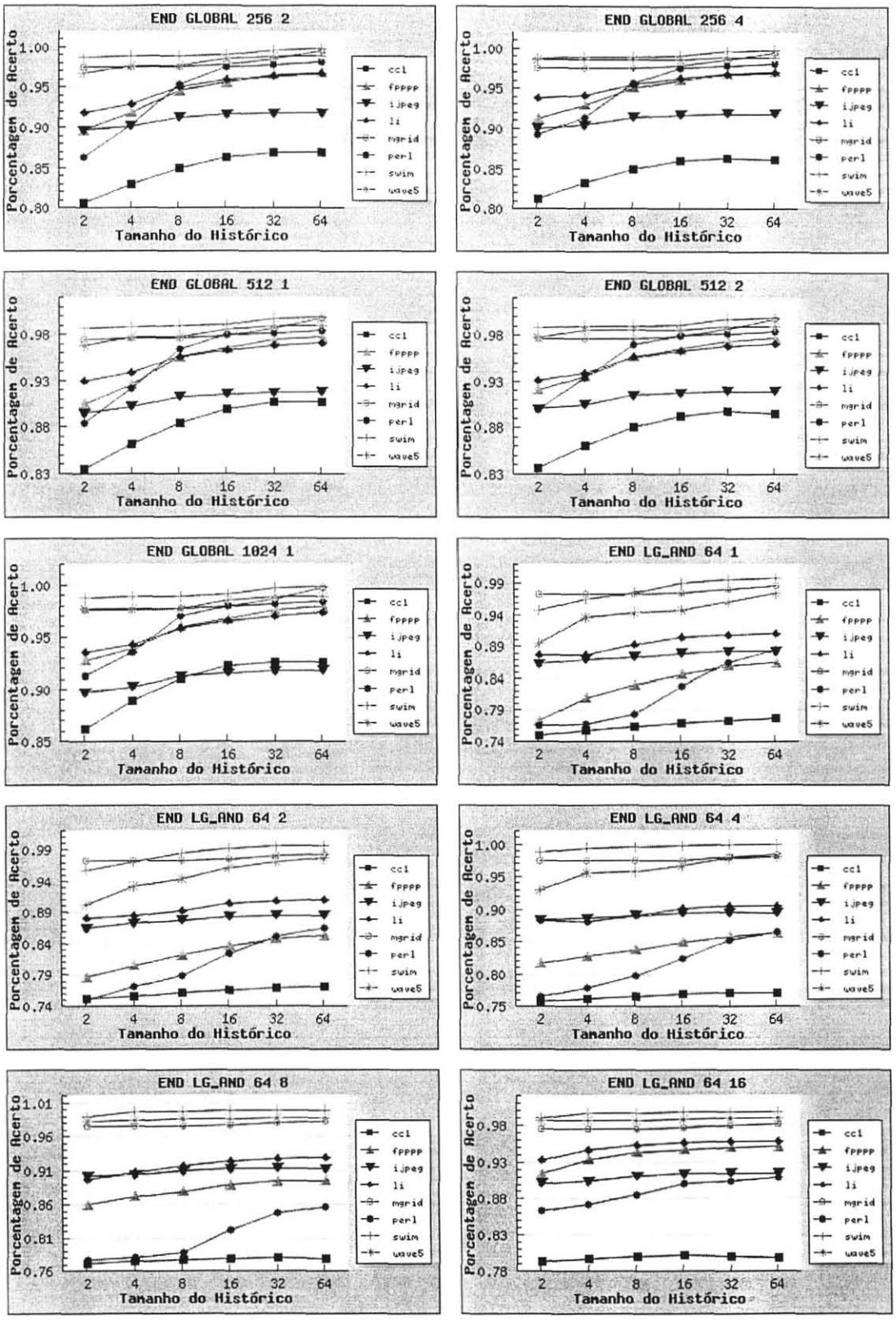
Apêndice 3 - Folha 8

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



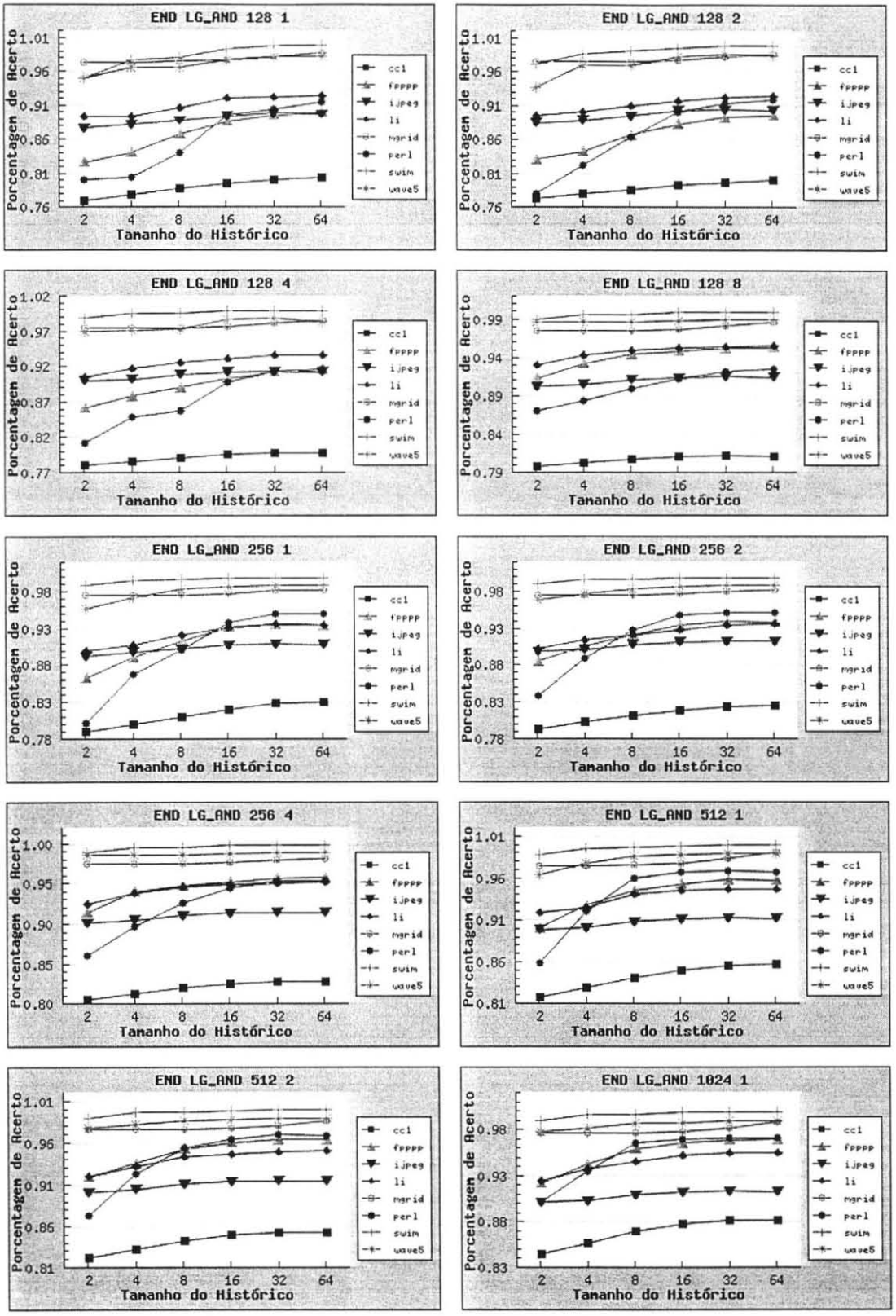
Apêndice 3 - Folha 9

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



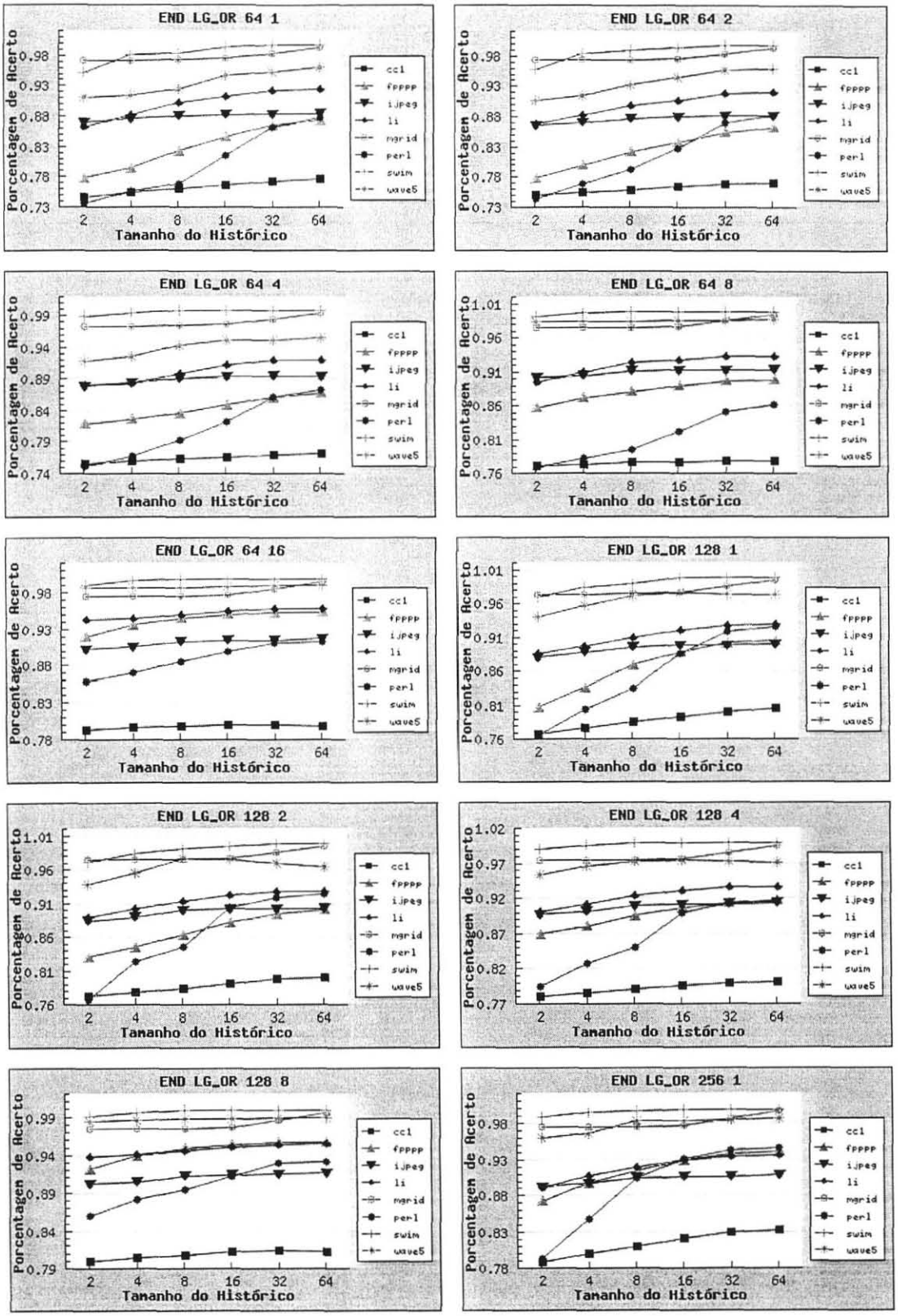
Apêndice 3 - Folha 10

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



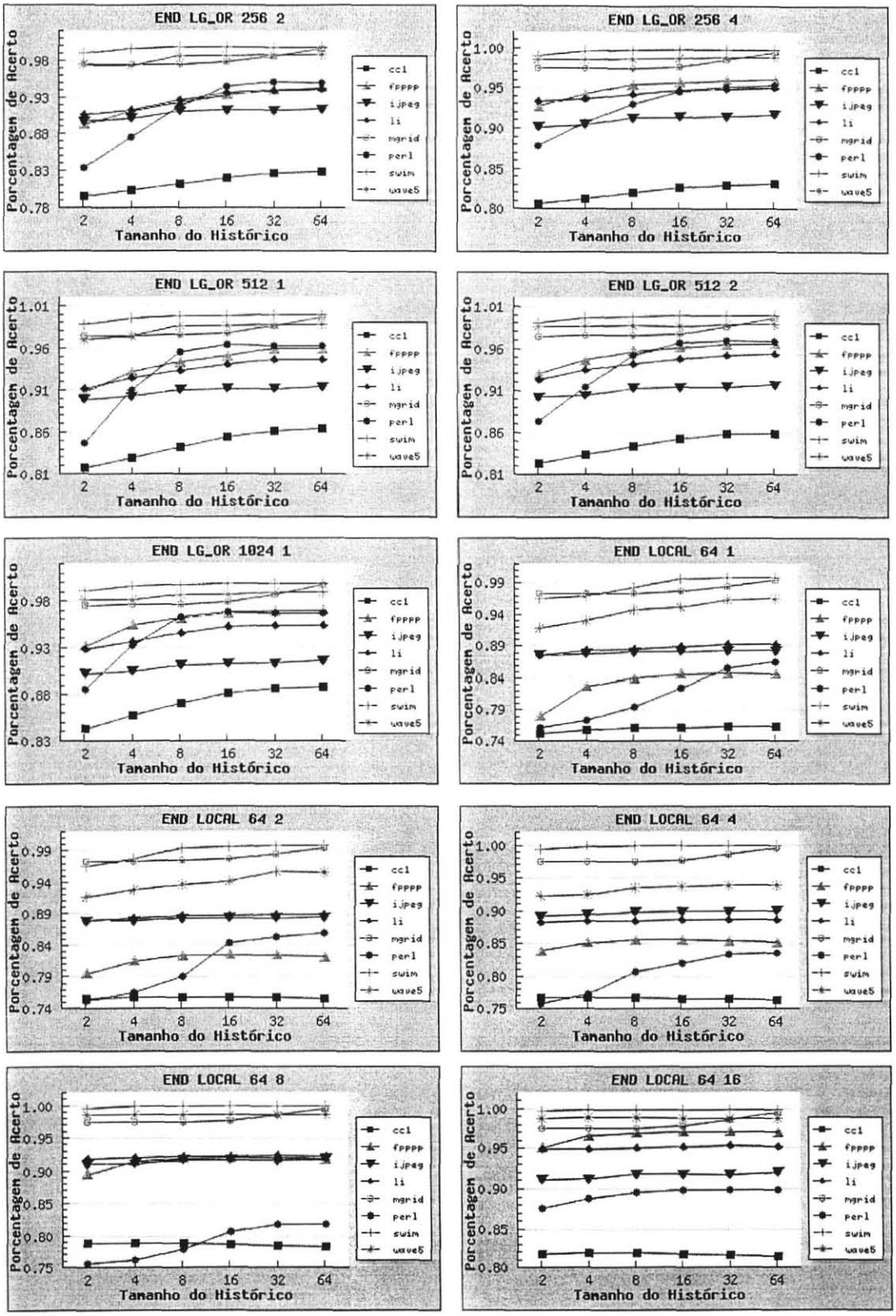
Apêndice 3 - Folha 11

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



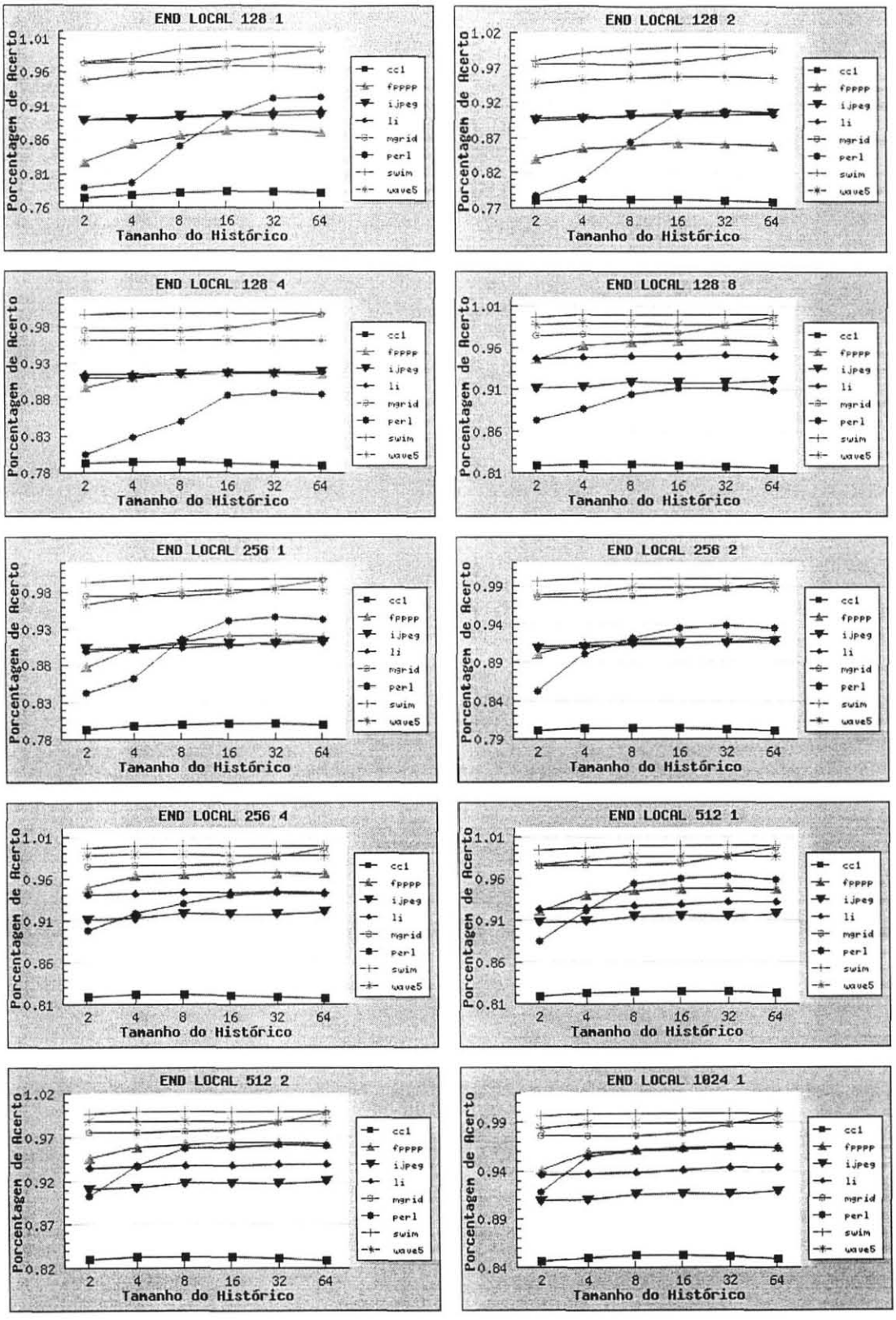
Apêndice 3 - Folha 12

Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS



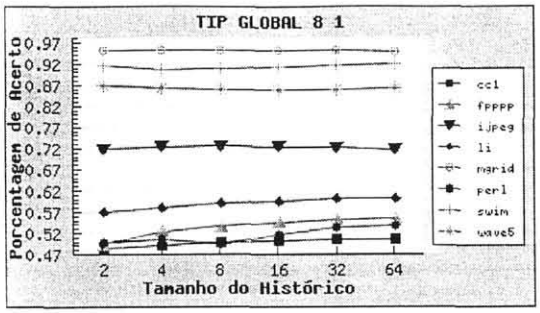
Apêndice 3 - Folha 13

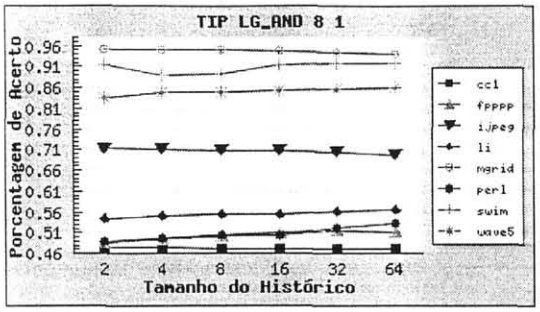
Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS

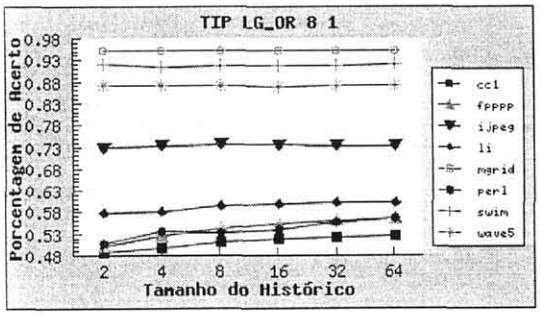


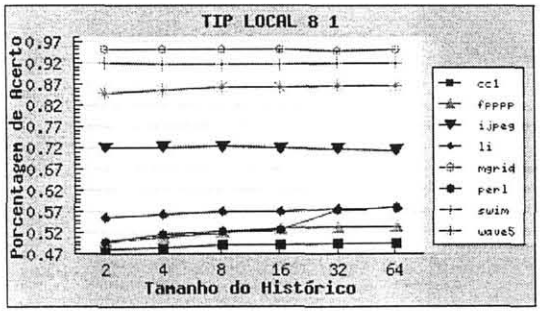
Apêndice 3 - Folha 14

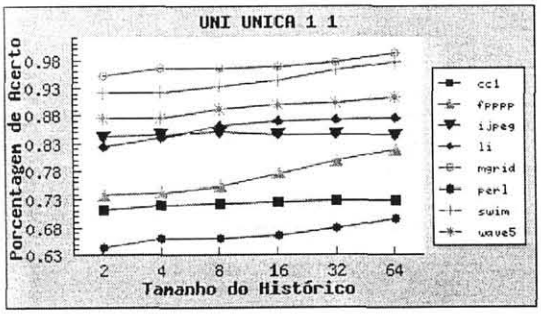
Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS





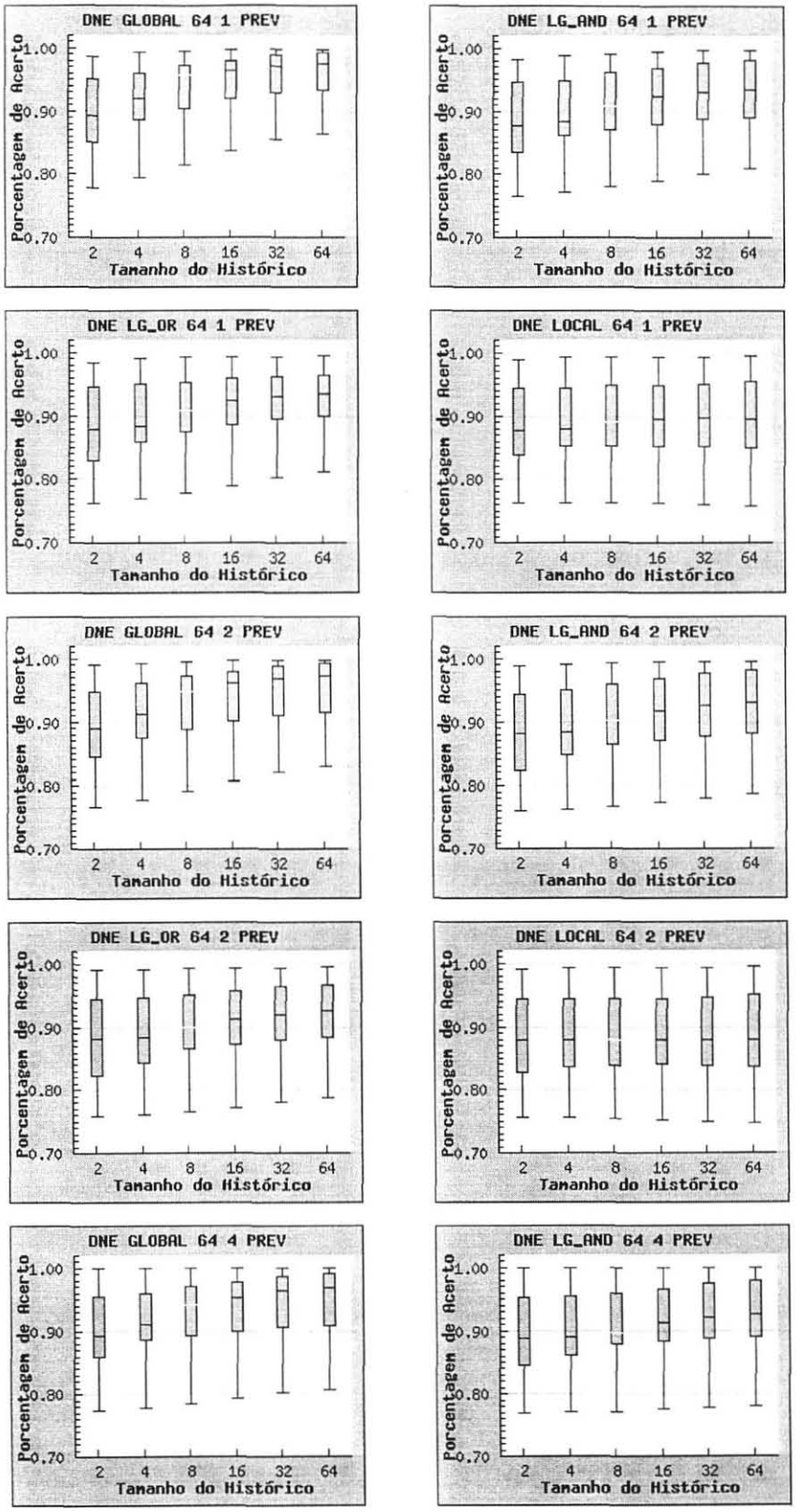






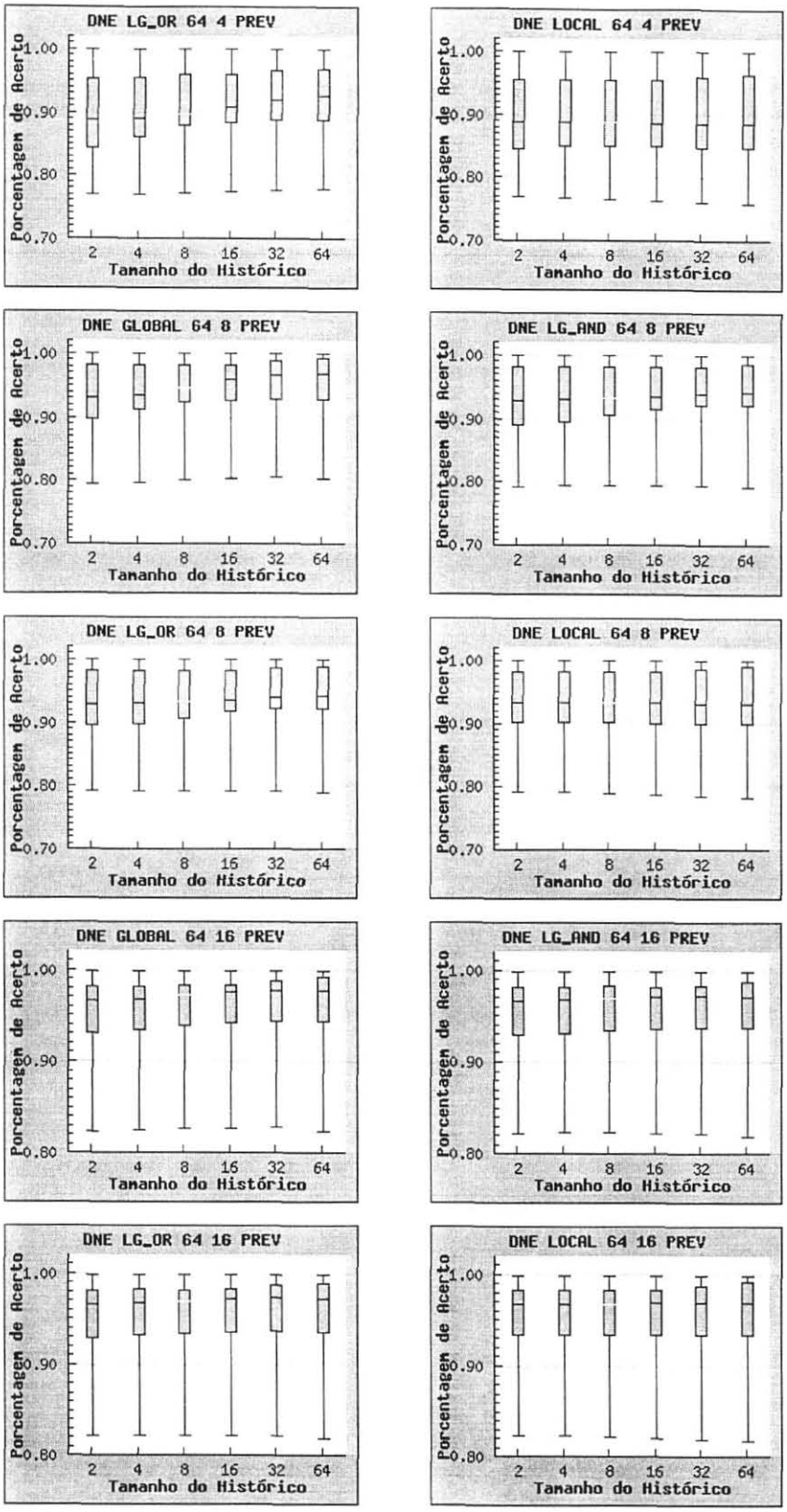
Gráficos por PRGs para prev individualizados por MOD/ORG/LIN/ASS

Apêndice 4 – Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS



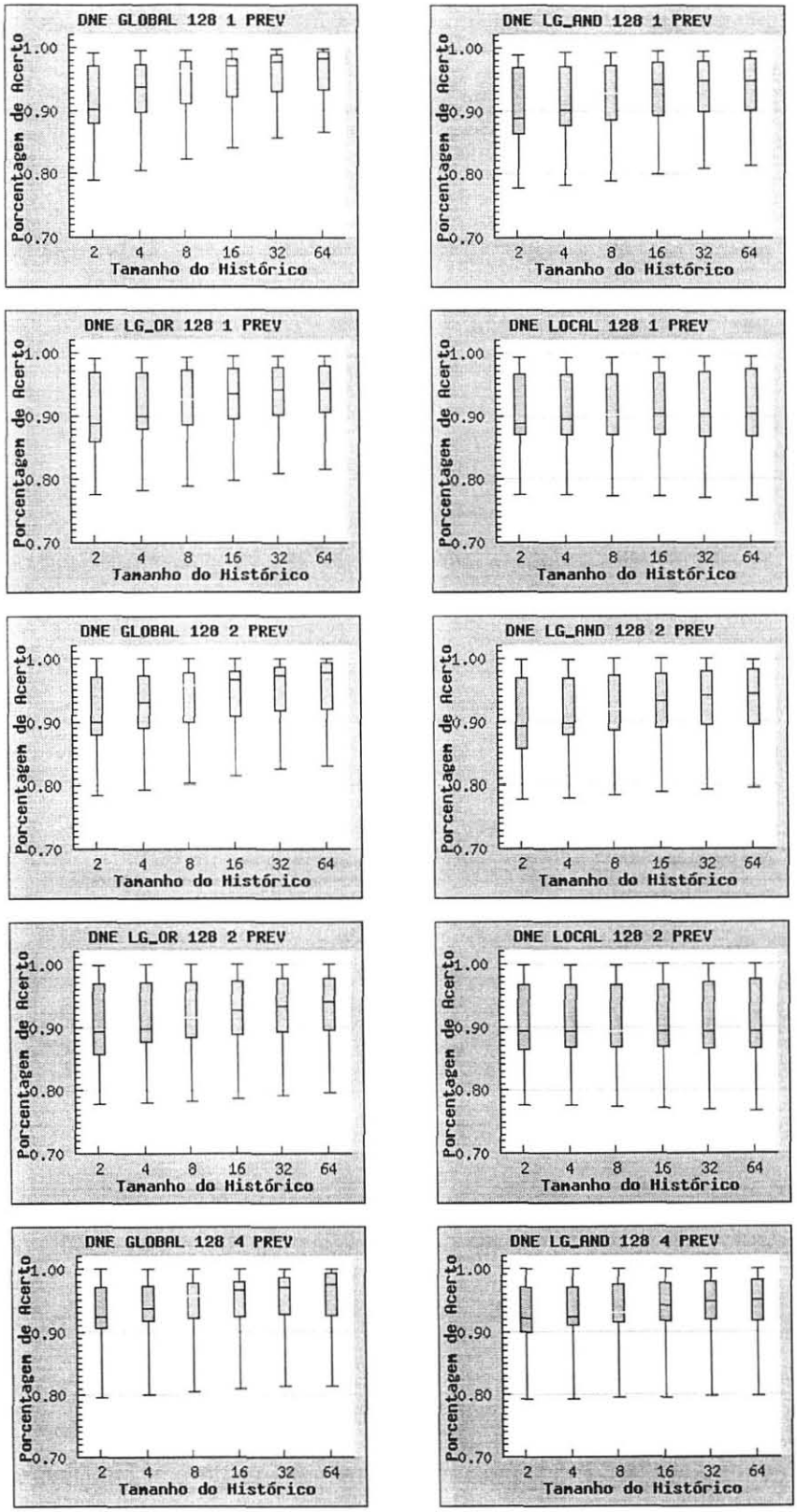
Apêndice 4 - Folha 1

Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS



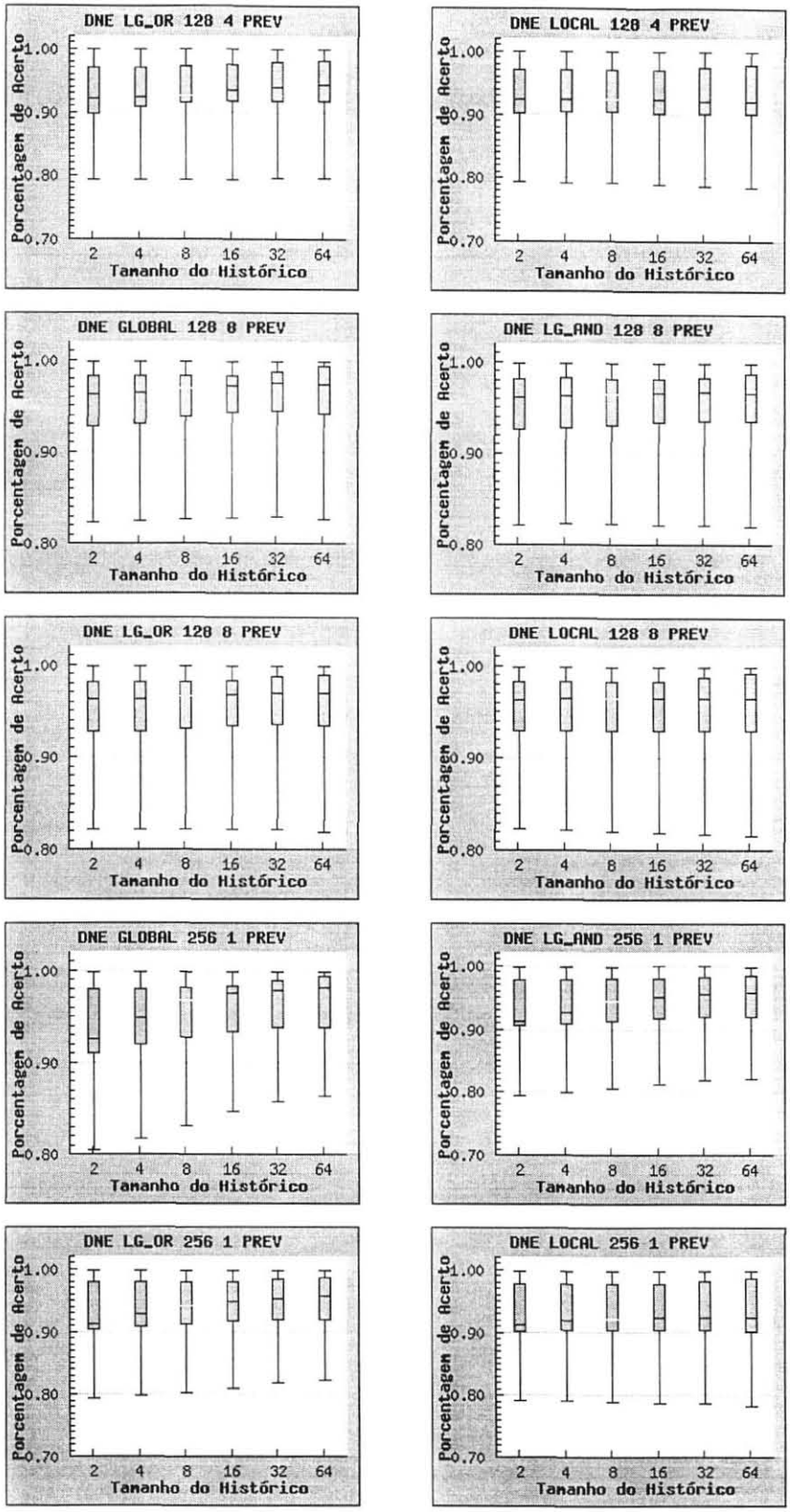
Apêndice 4 - Folha 2

Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS



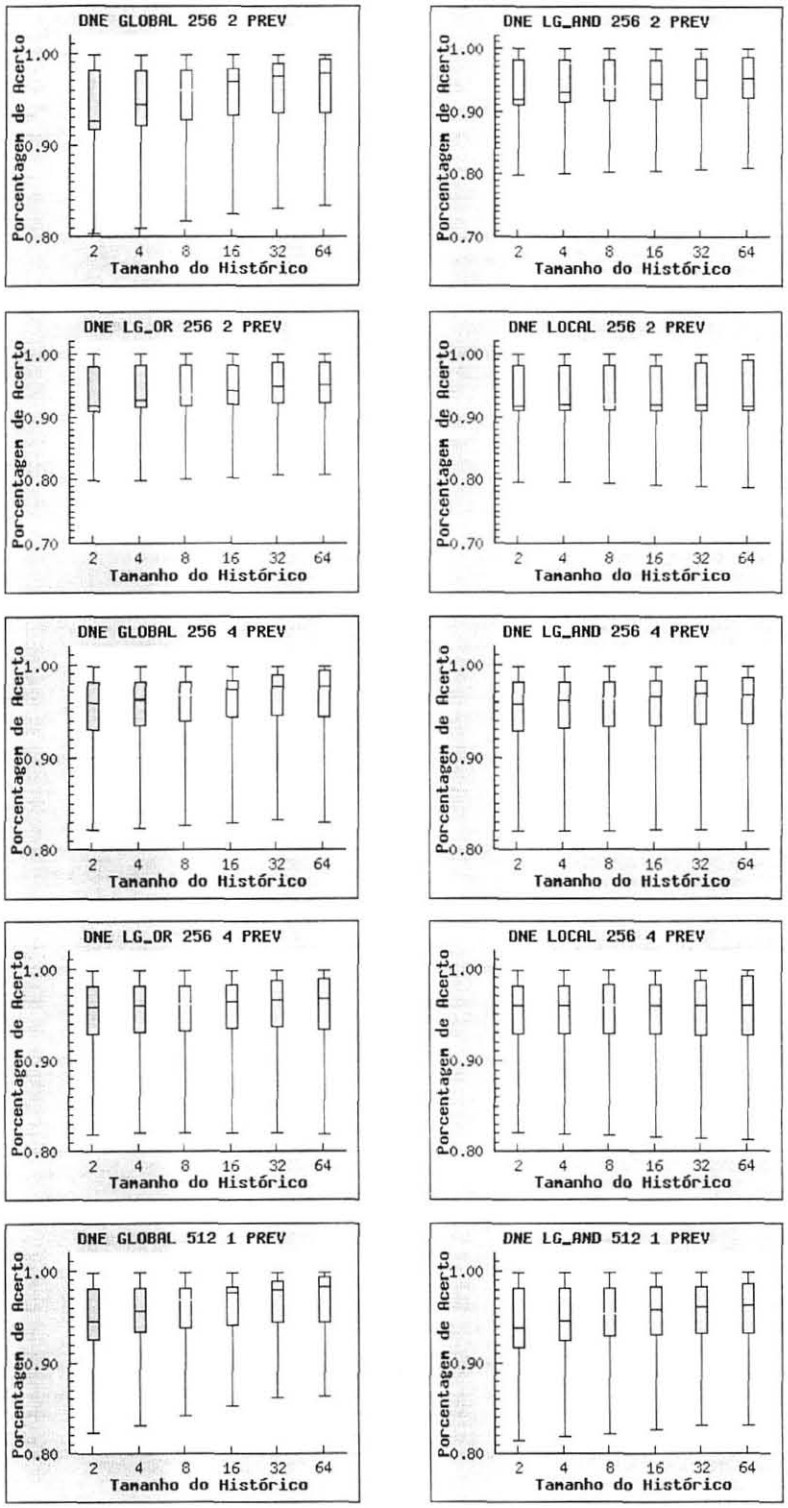
Apêndice 4 - Folha 3

Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS



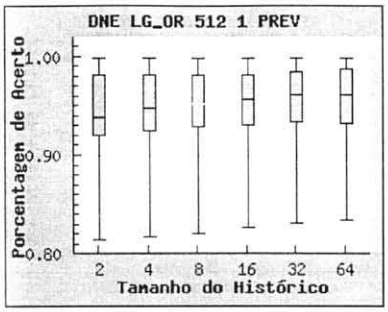
Apêndice 4 - Folha 4

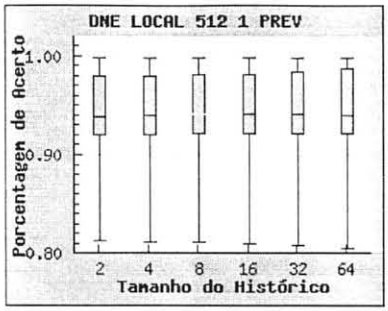
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

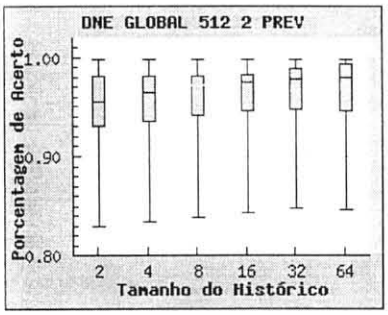


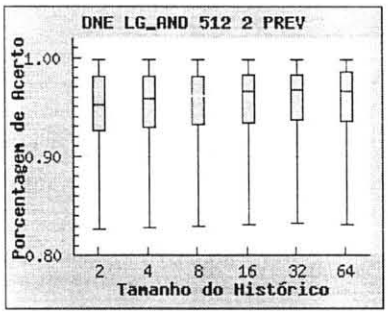
Apêndice 4 - Folha 5

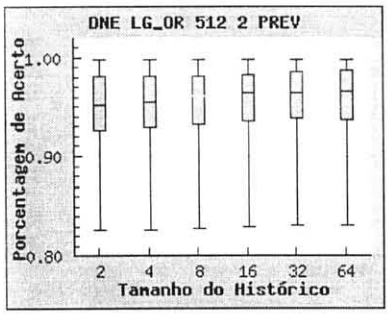
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

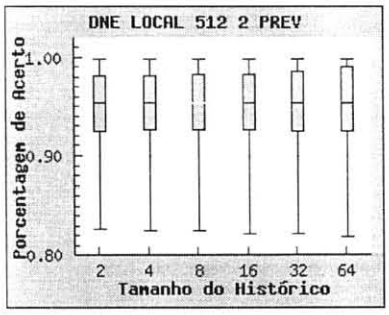


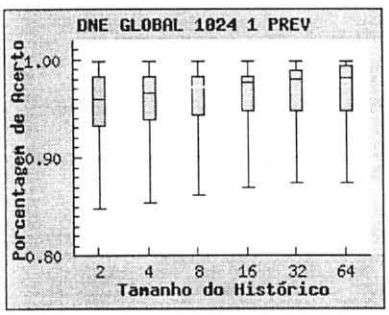


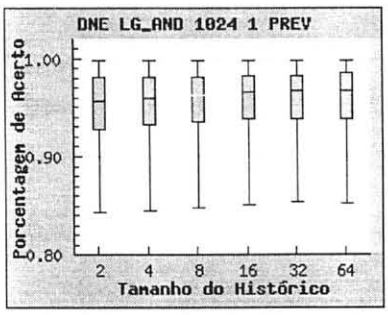


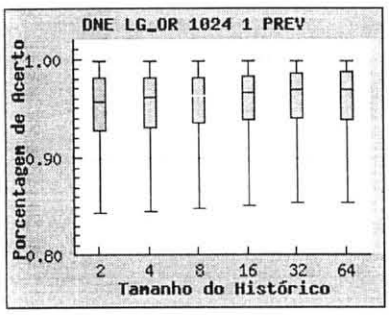


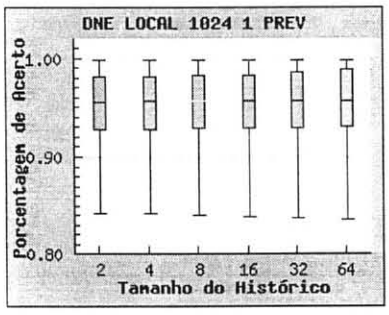






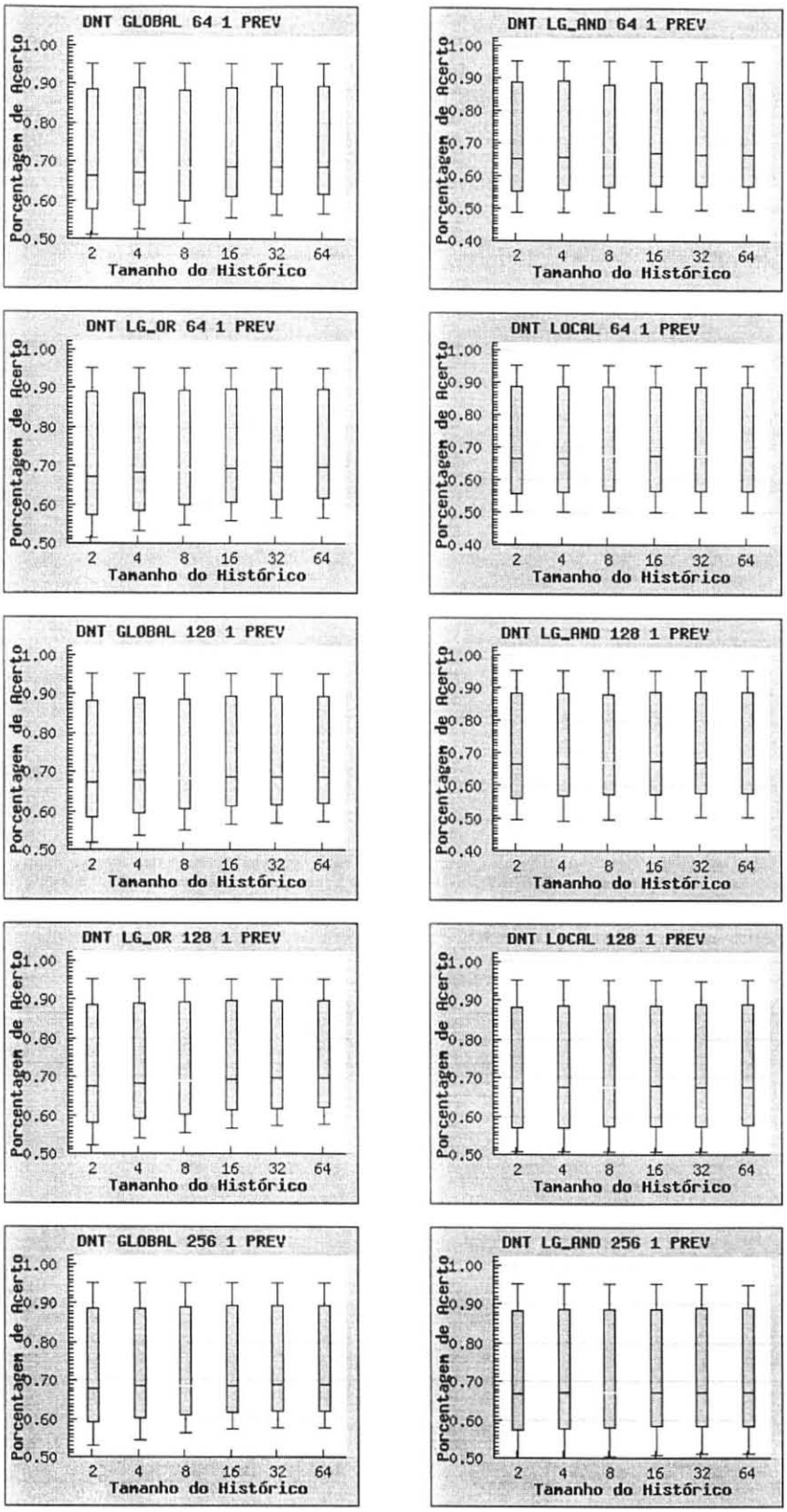






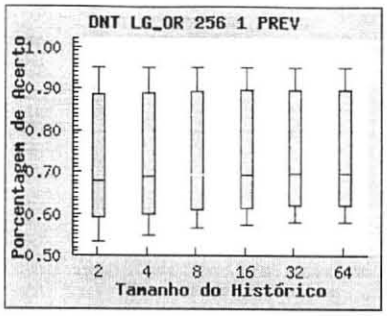
Apêndice 4 - Folha 6

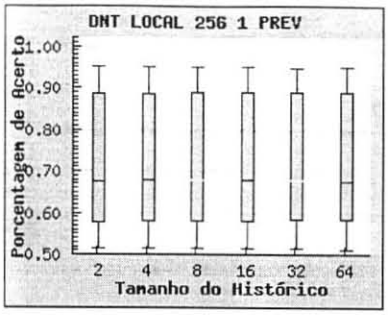
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

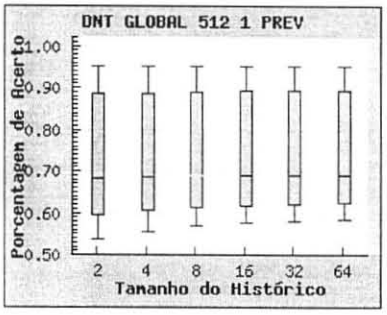


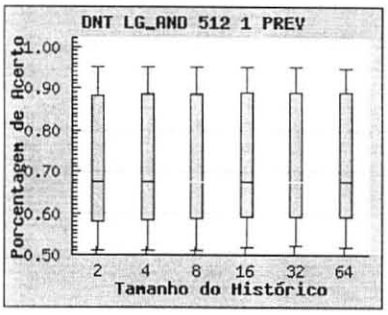
Apêndice 4 - Folha 7

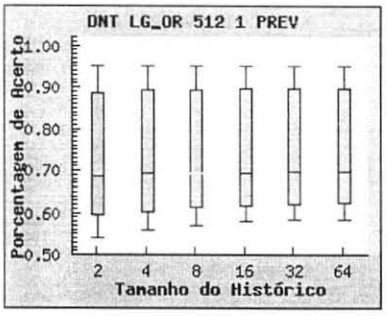
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

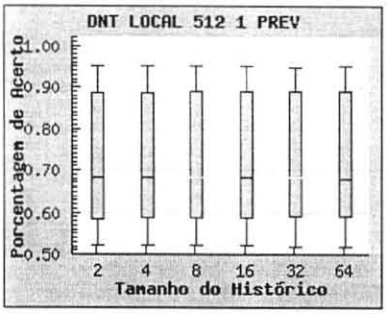


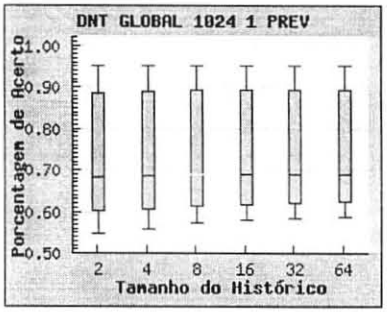


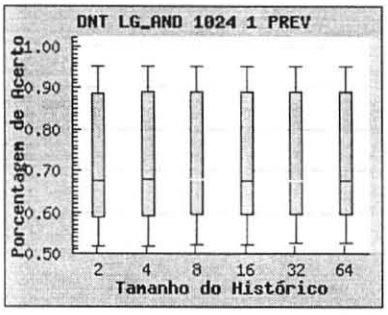


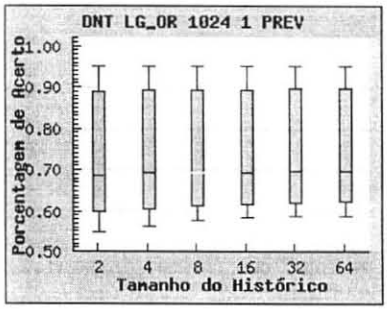


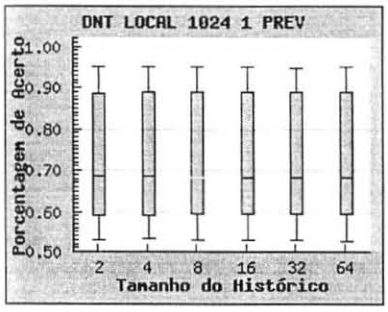






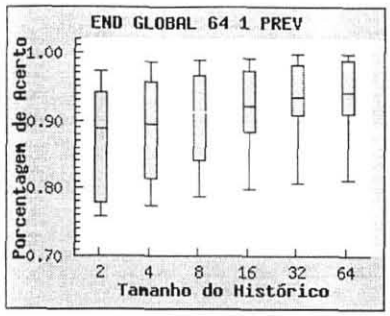


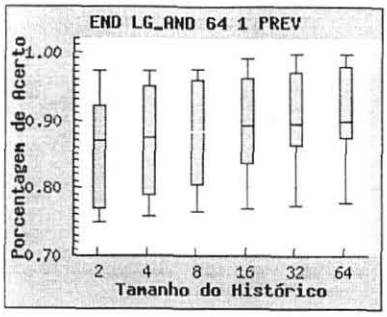


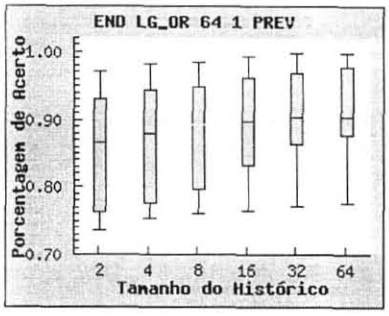


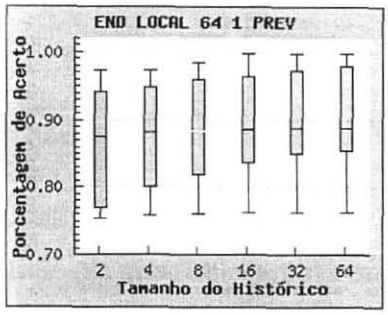
Apêndice 4 - Folha 8

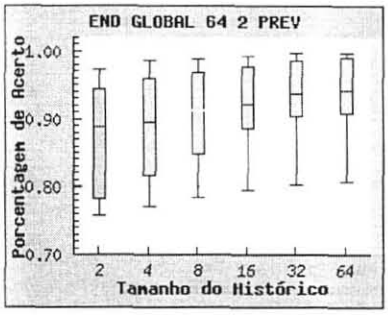
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

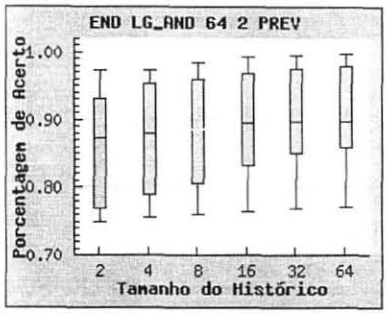


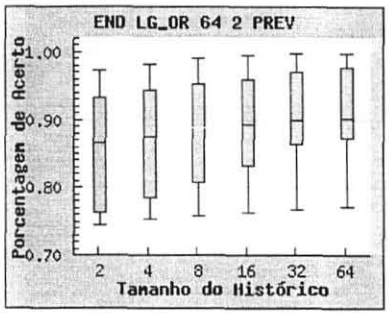


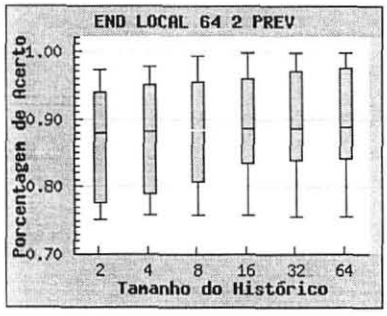


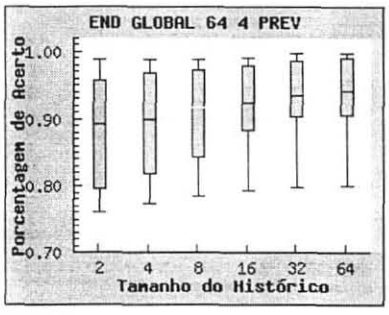


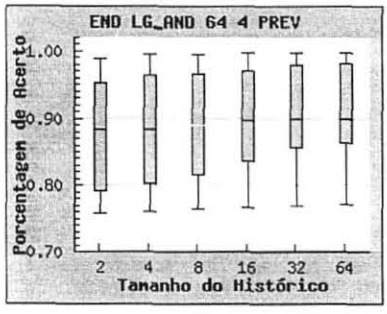






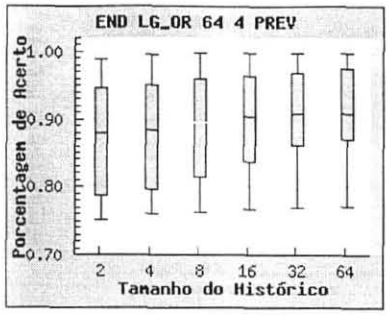


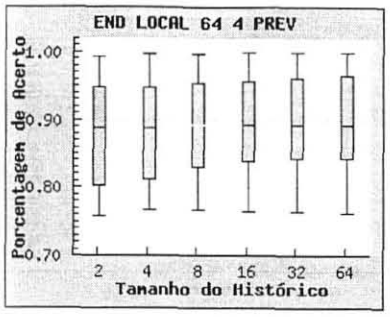


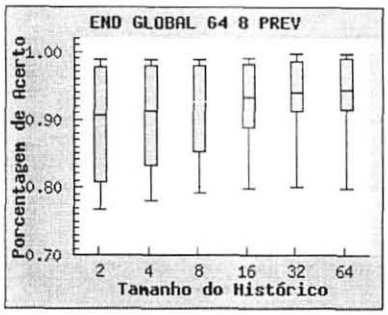


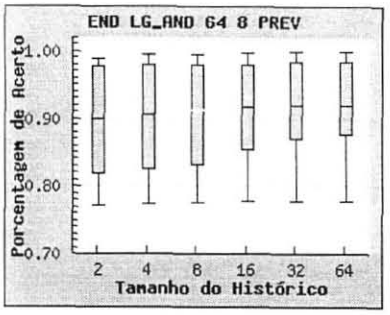
Apêndice 4 - Folha 9

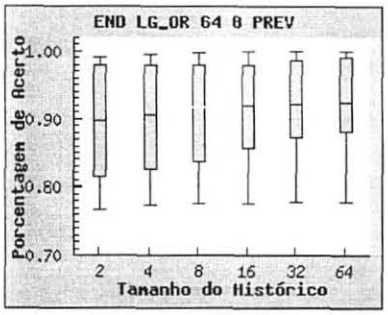
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

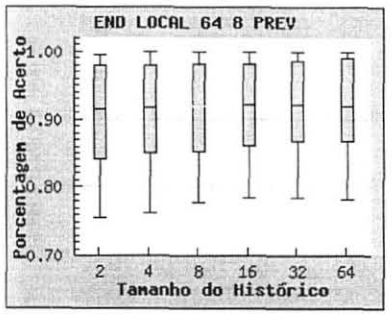


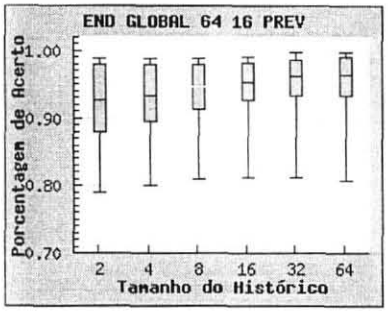


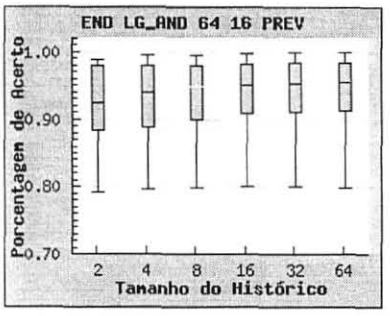


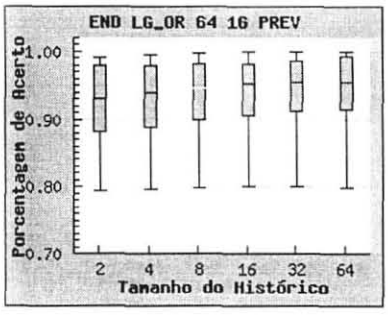


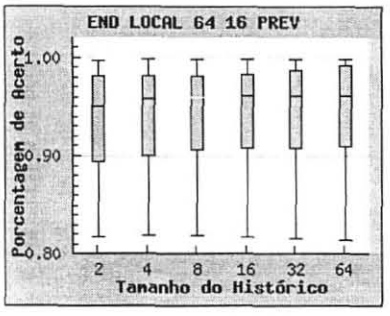






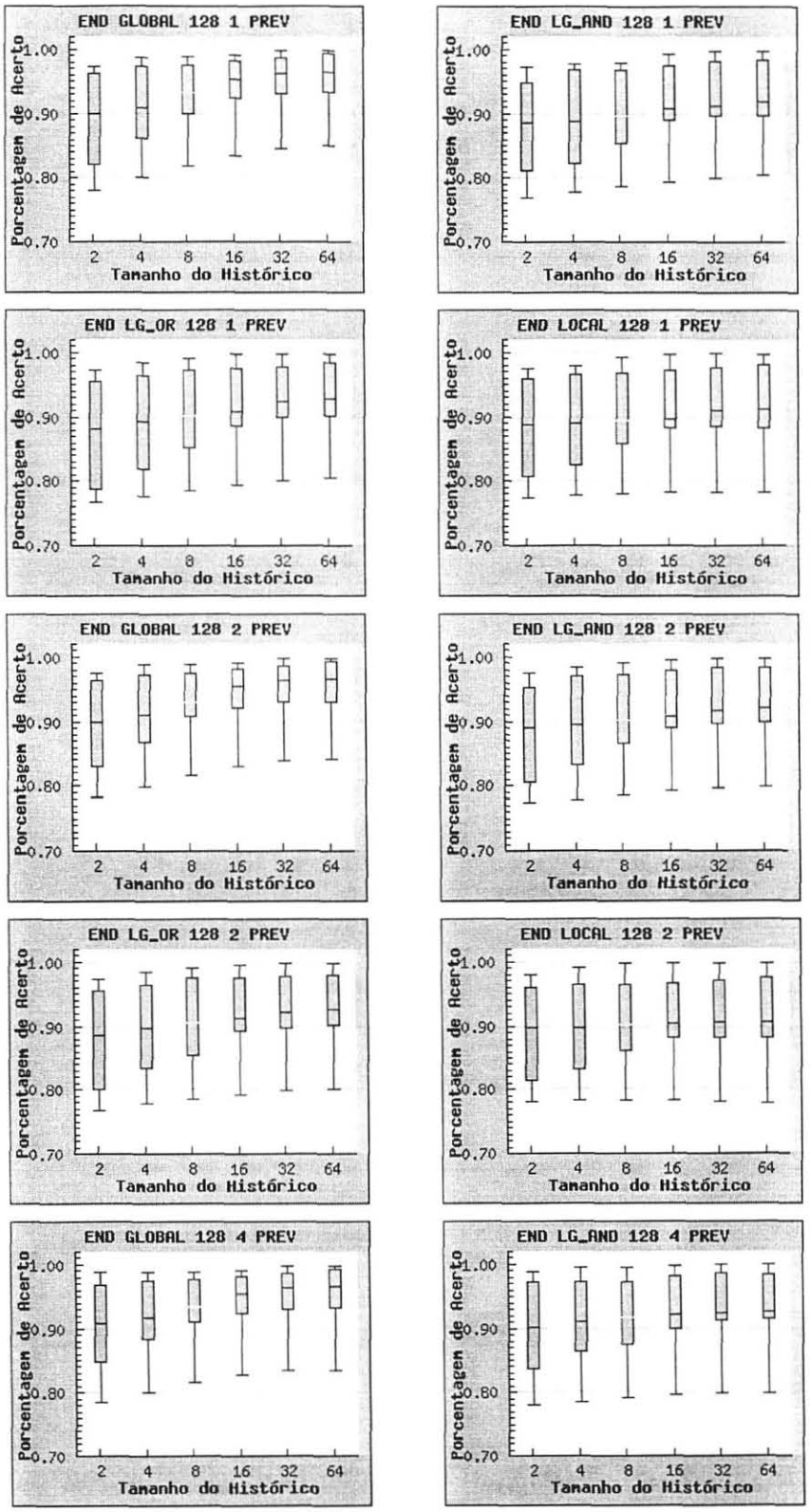






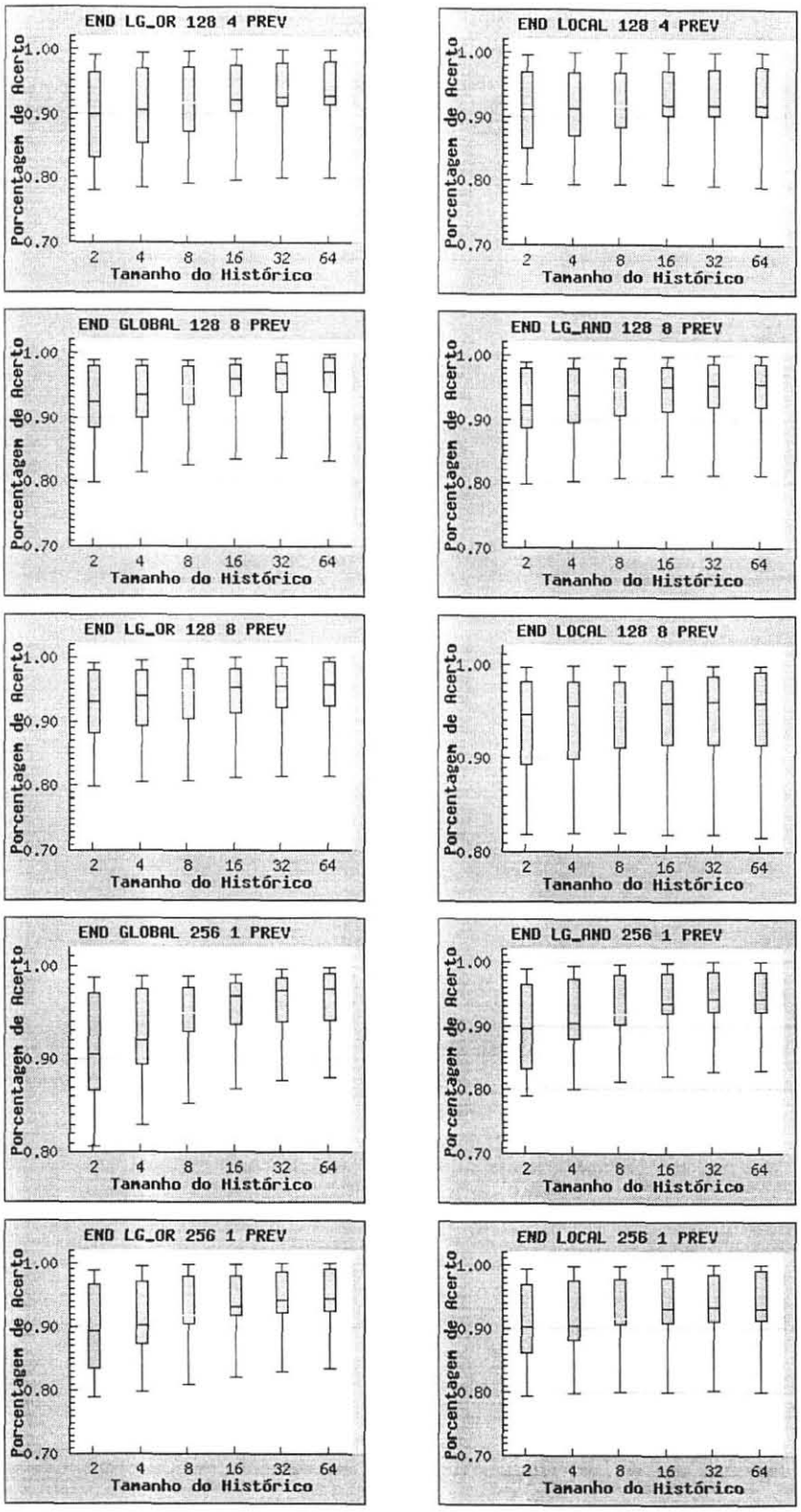
Apêndice 4 - Folha 10

Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS



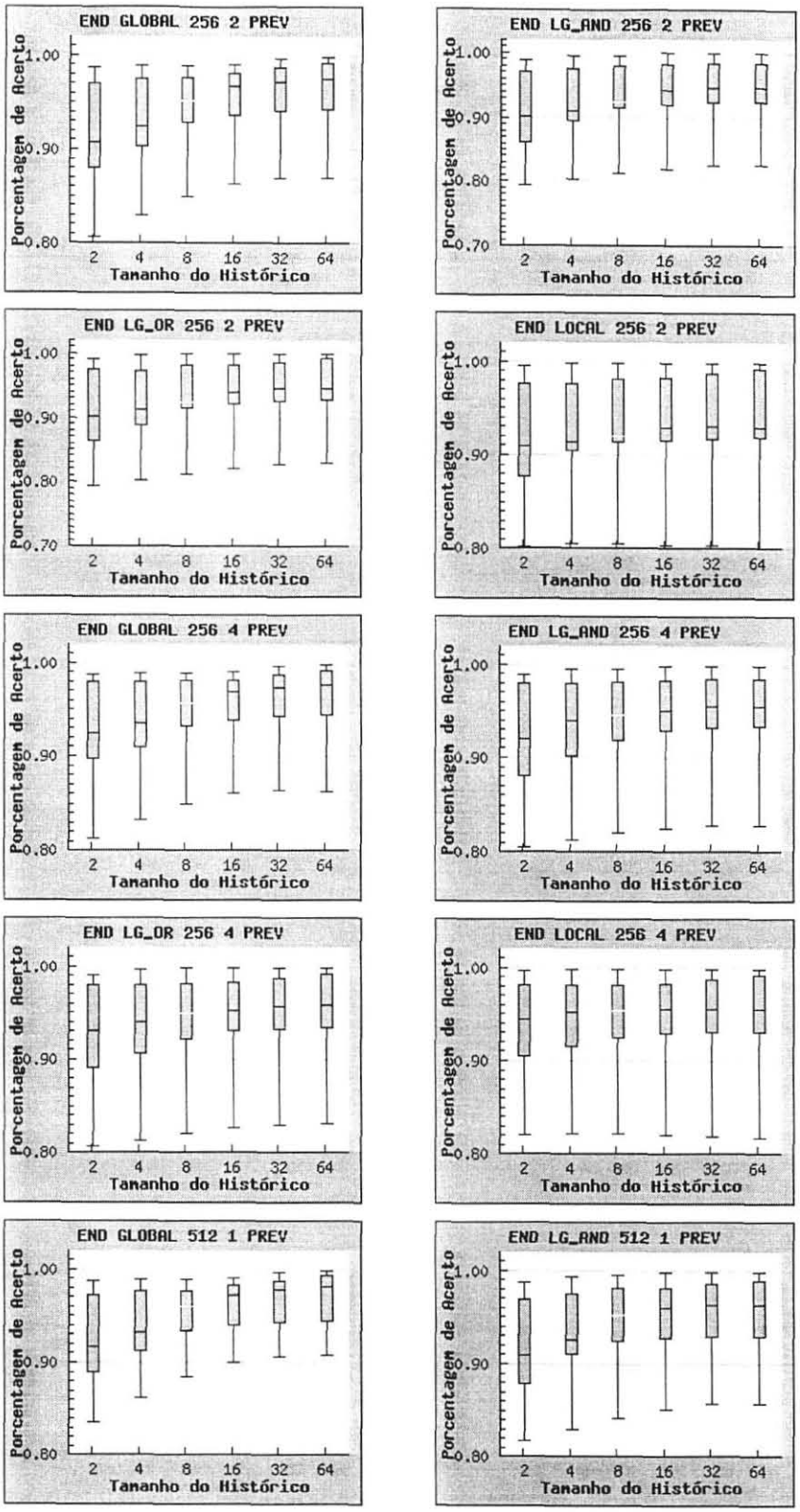
Apêndice 4 - Folha 11

Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS



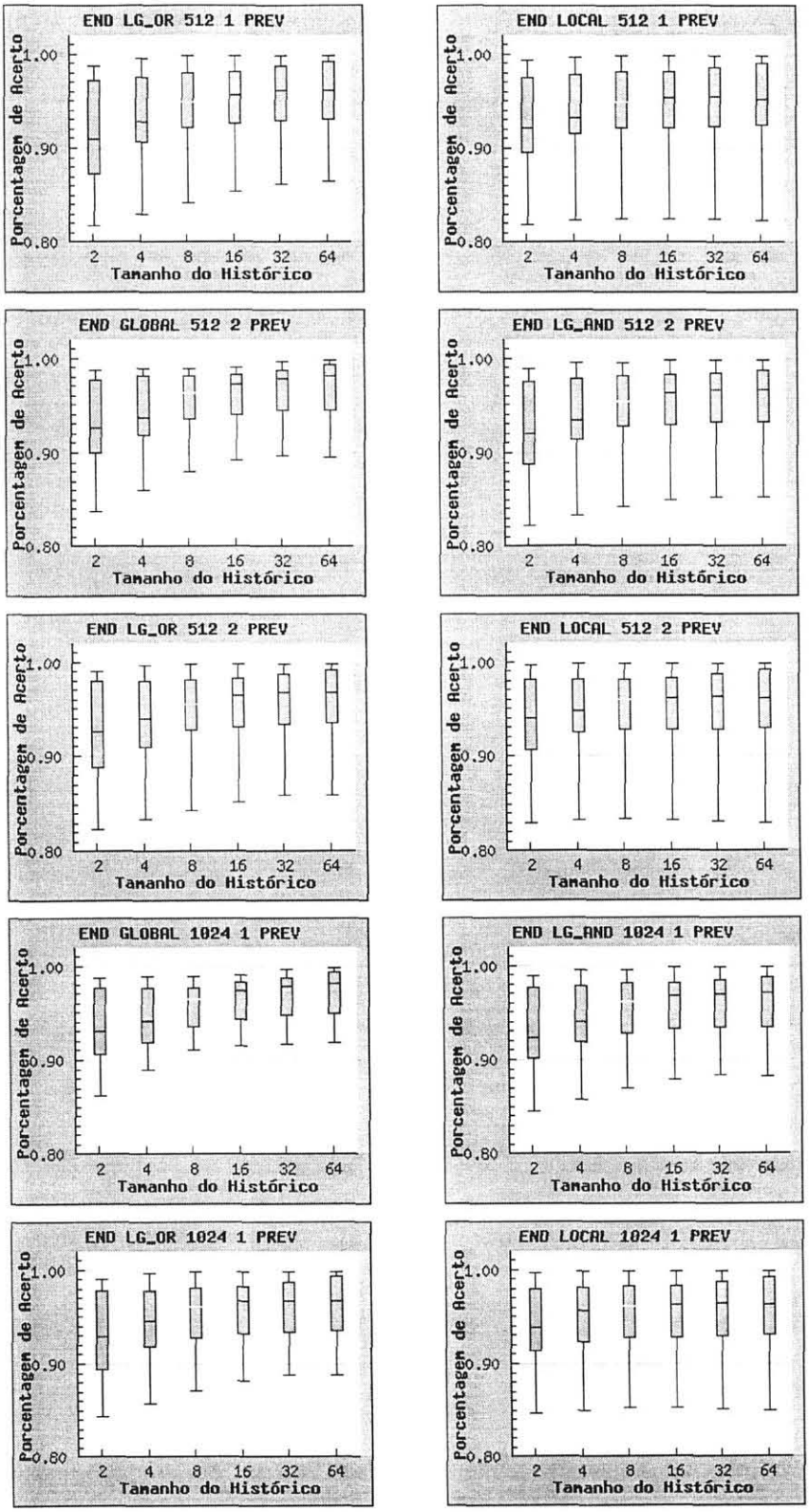
Apêndice 4 - Folha 12

Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS



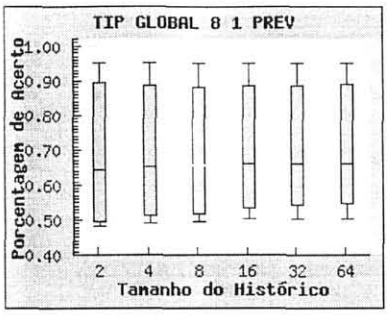
Apêndice 4 - Folha 13

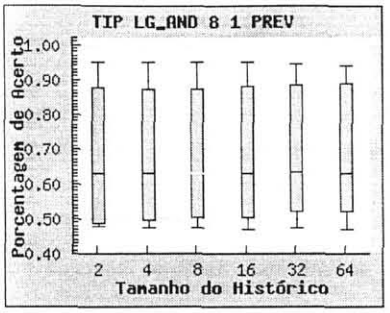
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

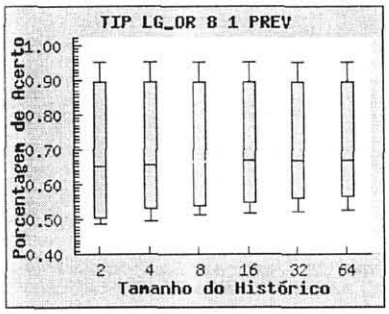


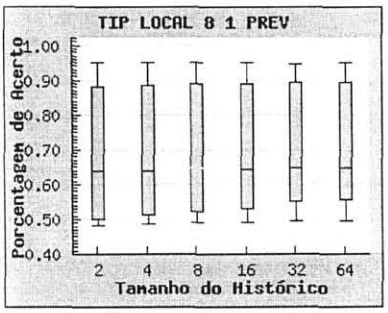
Apêndice 4 - Folha 14

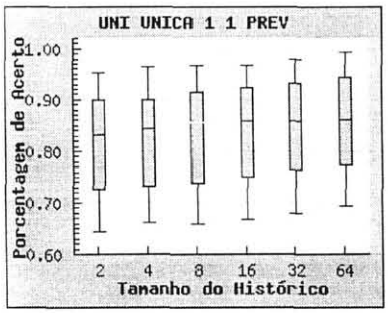
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS







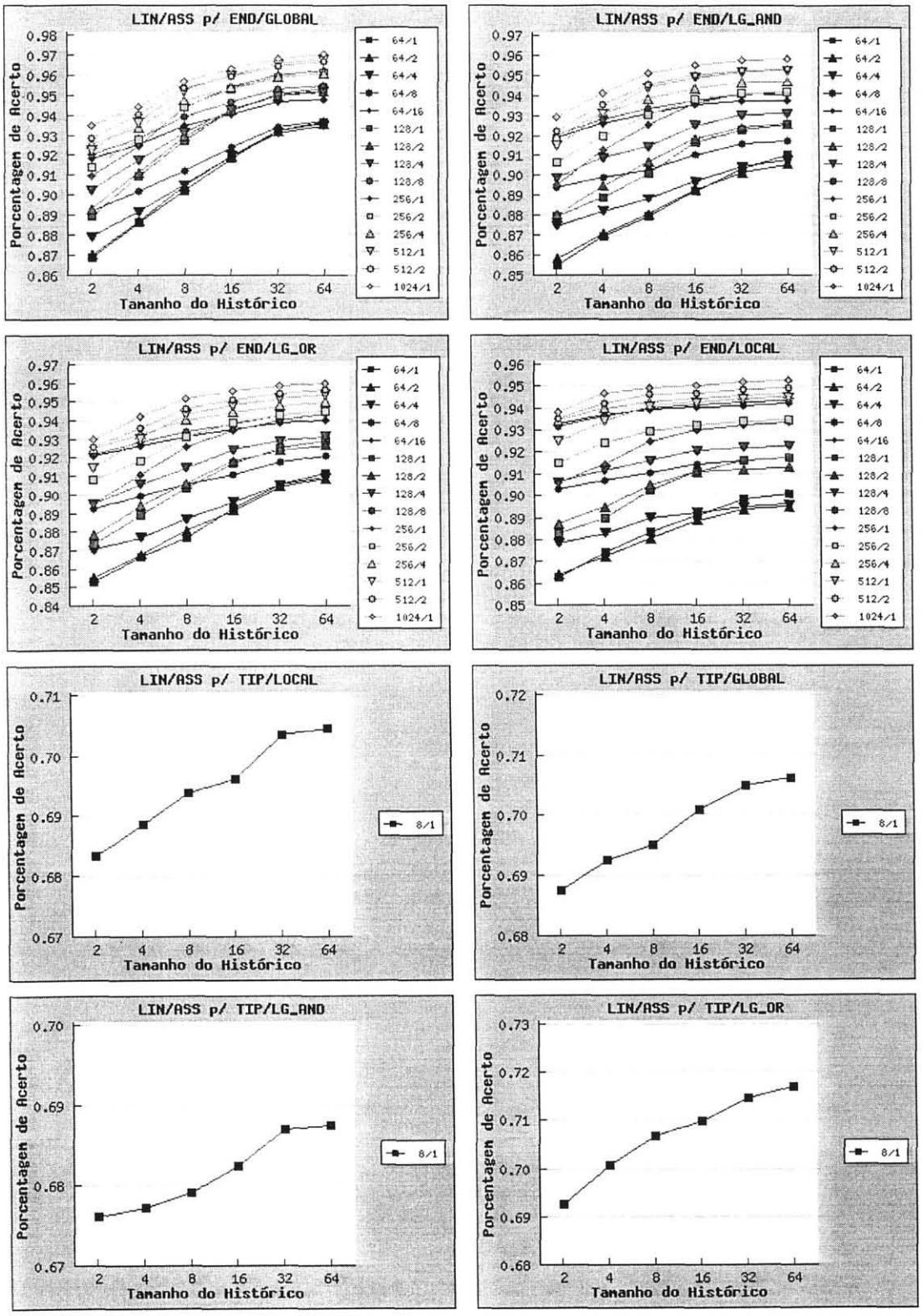




Apêndice 4 - Folha 15

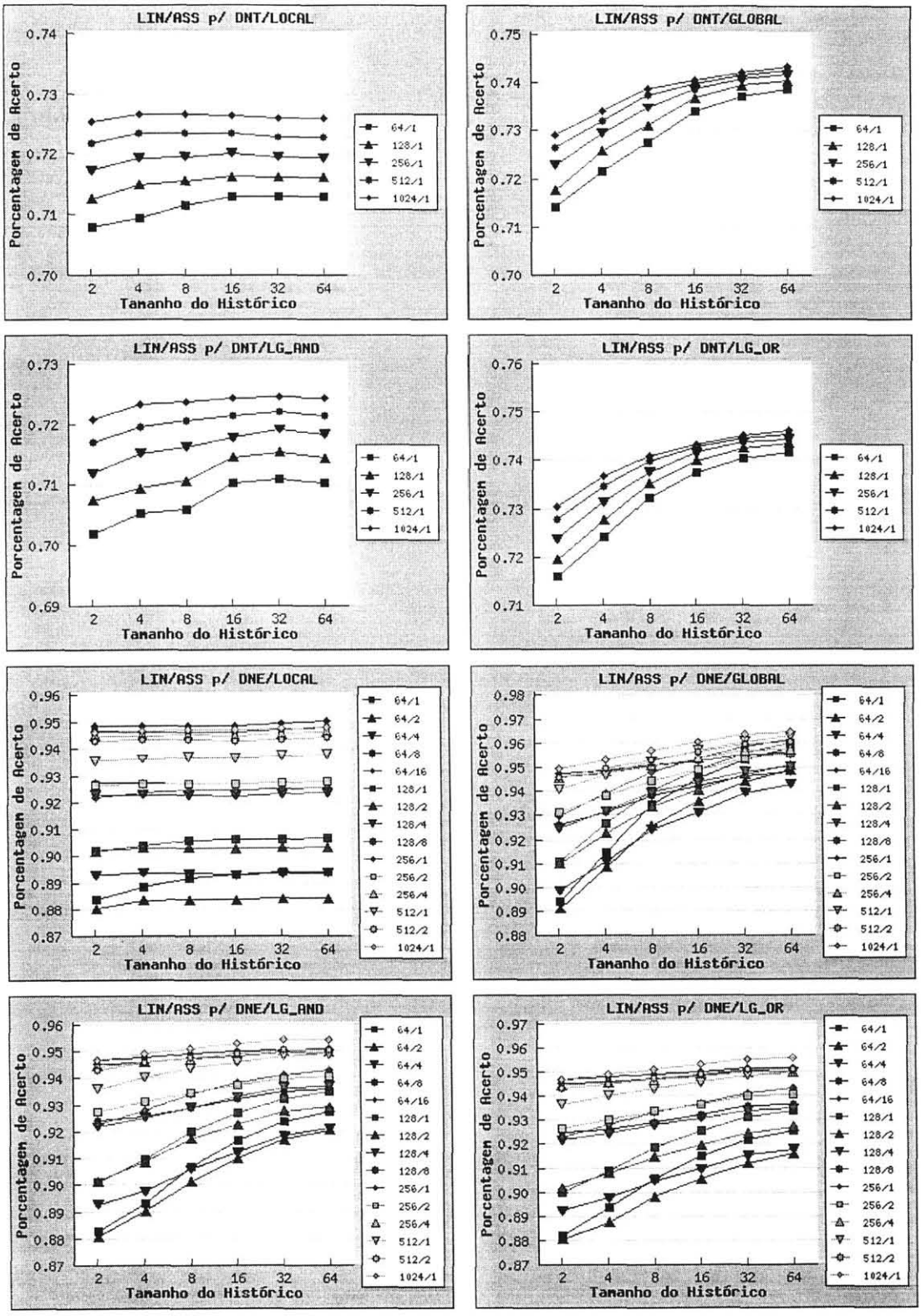
Gráficos do tipo Stock-Chart para a média de prev individualizados por MOD/ORG/LIN/ASS

Apêndice 5 – Gráficos por LIN/ASS individualizados por MOD/ORG



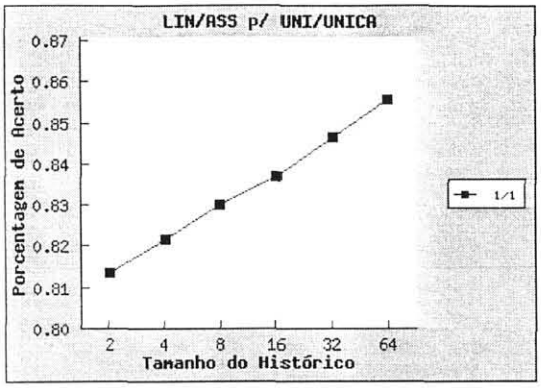
Apêndice 5 - Folha 1

Gráficos por LIN/ASS individualizados por MOD/ORG



Apêndice 5 - Folha 2

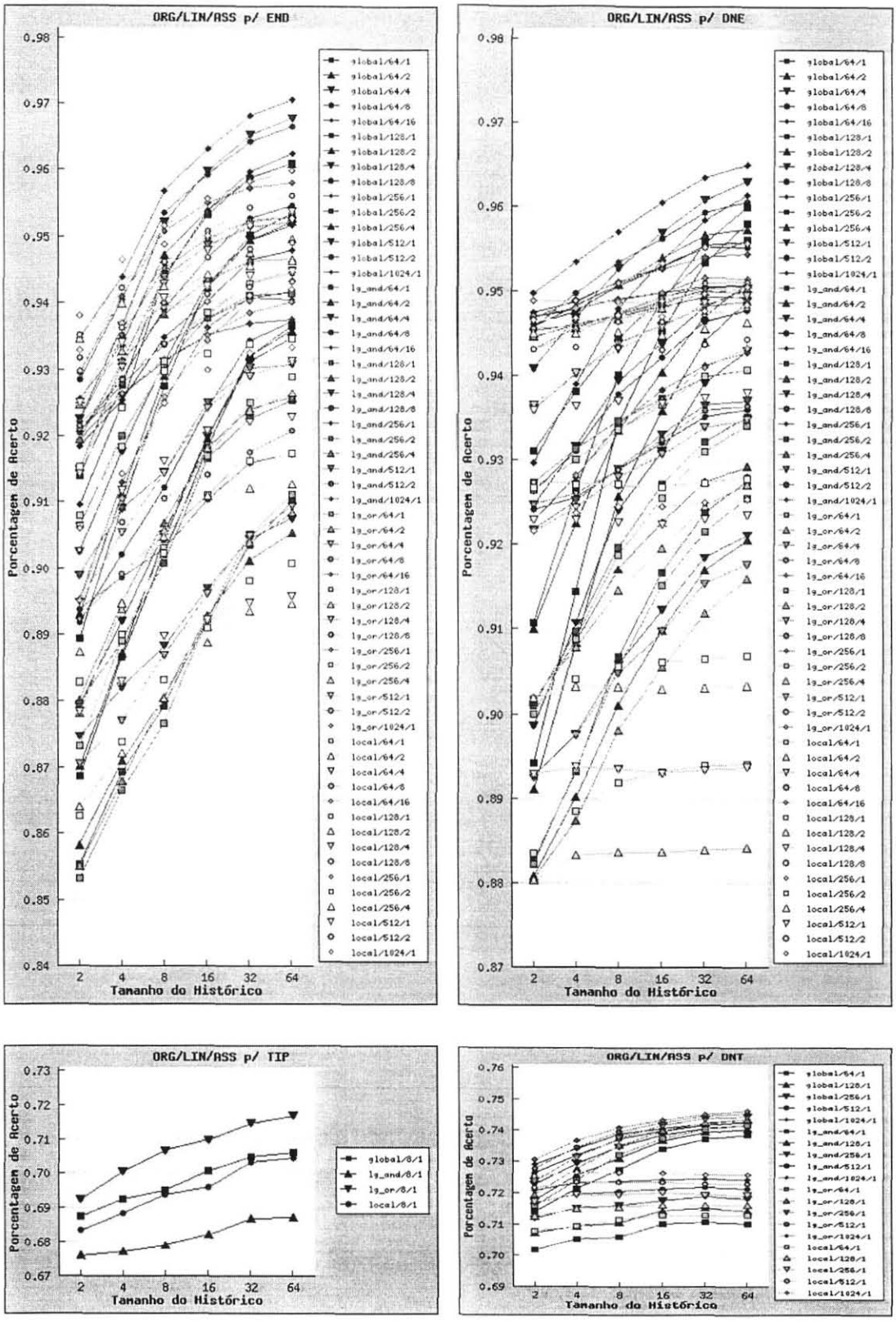
Gráficos por LIN/ASS individualizados por MOD/ORG



Apêndice 5 - Folha 3

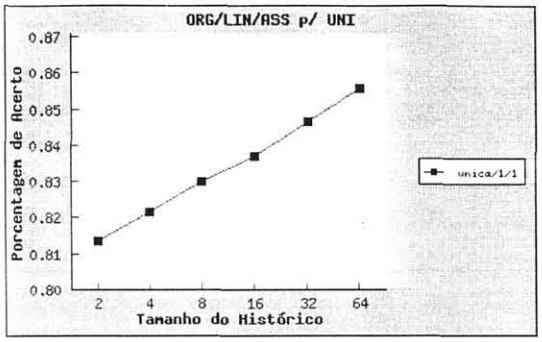
Gráficos por LIN/ASS individualizados por MOD/ORG

Apêndice 6 – Gráficos por ORG/LIN/ASS individualizados por MOD



Apêndice 6 - Folha 1

Gráficos por EST/LIN/ASS individualizados por MOD



Apêndice 6 - Folha 2

Gráficos por EST/LIN/ASS individualizados por MOD

Apêndice 7 – Gráfico único por MOD/ORG/LIN/ASS para todos os previsores

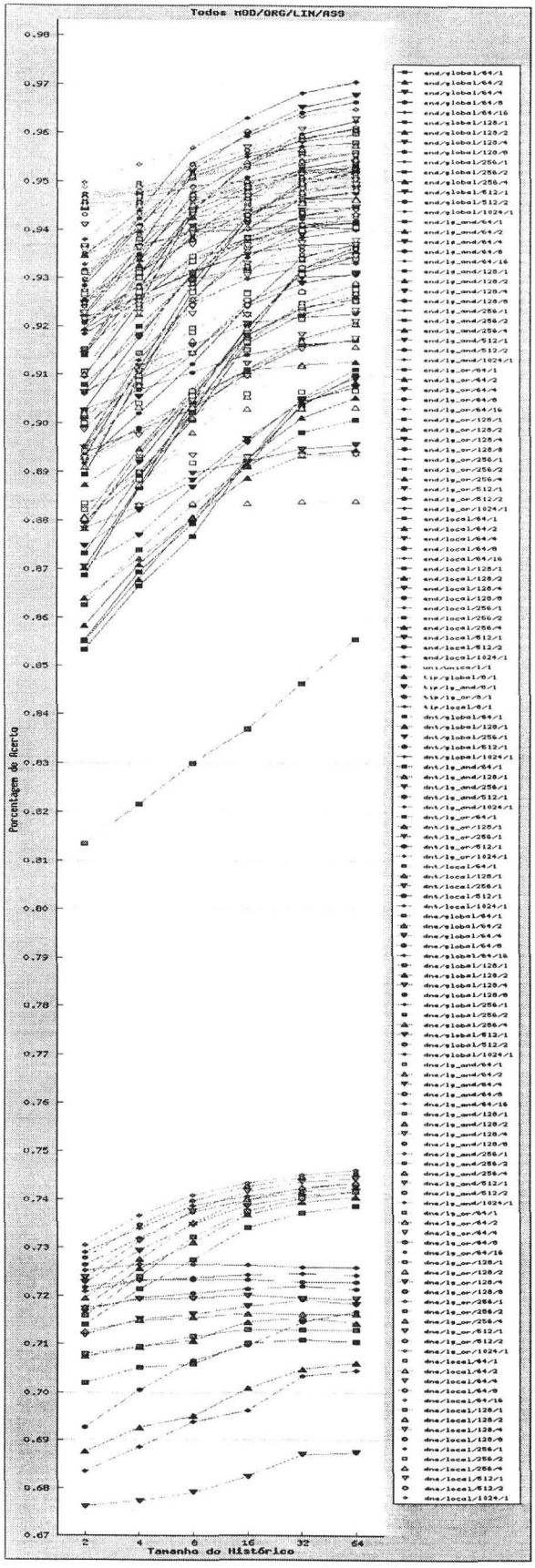


Gráfico único por MOD/ORG/LIN/ASS para todos os previsores