

WAGNER RODRIGO WEINERT

**COMPUTAÇÃO EVOLUCIONÁRIA PARA INDUÇÃO DE
REGRAS DE AUTÔMATOS CELULARES
MULTIDIMENSIONAIS**

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de “Doutor em Ciências” - Área de Concentração: Informática Industrial.

Orientador: Prof. Dr. Heitor Silvério Lopes

CURITIBA

2010

Ficha catalográfica elaborada pela Biblioteca da UTFPR, Campus Curitiba

W423 Weinert, Wagner Rodrigo
 Computação evolucionária para indução de regras de autômatos celulares multidimensionais / Wagner Rodrigo Weinert. — 2010.
 189 p. : il. ; 30 cm

 Orientador : Heitor Silvério Lopes
 Tese (Doutorado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 2010
 Bibliografia : 159-167

 1. Autômatos celulares. 2. Computação evolucionária. 3. Inteligência artificial. 4. Algoritmos paralelos. 5. Engenharia elétrica – Teses. I. Lopes, Heitor Silvério, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD (22. ed.) 621.3

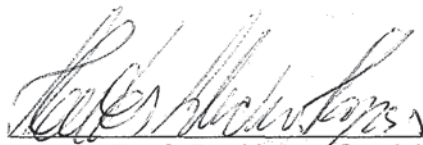
Título da Tese Nº 49:

“Computação Evolucionária para Indução de Regras de Autômatos Celulares Multidimensionais”

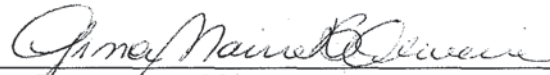
por

Wagner Rodrigo Weinert

Esta tese foi apresentada, às 08h30min do dia 08 de fevereiro de 2010, como requisito parcial para a obtenção do título de DOUTOR EM CIÊNCIAS – Área de Concentração: Informática Industrial, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:



Prof. Dr. Heitor Silvério Lopes
(Orientador – UTFPR)



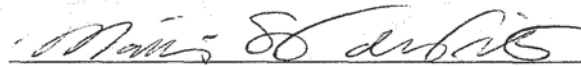
Prof. Dr. Gina Maira Barbosa de Oliveira
(UFU)



Prof. Dr. Leandro dos Santos Coelho
(PUC-PR)



Prof. Dr. Tânia Mezzadri Centeno
(UTFPR)



Prof. Dr. Mário Sérgio Teixeira de Freitas
(UTFPR)

Visto da coordenação:



Prof. Dr. Humberto Remígio Gamba
(Coordenador do CPGEI)

*Dedico esta Tese a minha esposa
Luciana V. Castilho-Weinert.*

AGRADECIMENTOS

Agradeço a DEUS por toda a força e espiritualidade, sempre iluminando meus caminhos e jamais me deixando sem amparo.

Agradeço a minha esposa Luciana, por seu amor, apoio e dedicação. Agradeço por sua compreensão em meus momentos de ausência e infinitas palavras de motivação.

Ao Prof. Dr. Heitor Silvério Lopes, como orientador, que tornou possível a realização deste trabalho e tanto se dedicou para o sucesso do mesmo.

Agradeço a meus pais Marcio e Sonia, minha irmã Elaine e meus familiares, por todo amor e dedicação. Nada seria possível sem vosso apoio.

Ao casal de grandes amigos Hugo e Michele, pelos momentos de descontração e pelo apoio em muitos momentos difíceis.

Aos meus colaboradores e amigos César e Maicon. A dedicação de ambos viabilizou a utilização de novas tecnologias no desenvolvimento de parte deste trabalho.

Aos meus amigos César, Rafael Betito e sua esposa Andréia, e tantos outros companheiros que me proporcionaram inúmeros momentos de alegria e descontração.

Ao pessoal do laboratório de Bio-Informática, pelas conversas e ajuda operacional.

Aos colegas do IFPR, Campus Paranaguá, pelo apoio.

A todas aquelas pessoas, que mesmo não tendo sido citadas aqui, contribuíram de diversas maneiras para a conclusão deste trabalho.

“O pessimista reclama do vento, o otimista espera
que ele mude. O sábio ajusta as velas.”

John Maxwell

RESUMO

WEINERT, Wagner Rodrigo. Computação evolucionária para indução de regras de autômatos celulares multidimensionais. 2010. 189p. Tese (Doutorado) - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2010.

Um autômato celular é um sistema dinâmico discreto que evolui pela iteração de regras. Os valores das variáveis do sistema mudam em função de seus valores correntes. Os autômatos celulares podem ser aplicados na resolução de diversos problemas. A tarefa de encontrar uma regra de transição que solucione um determinado problema pode ser generalizada como um problema de indução de regras para autômatos celulares. Várias abordagens baseadas em técnicas de computação evolucionária vêm sendo empregadas neste problema. No entanto, estas restringem-se a aplicações específicas. A principal contribuição deste trabalho é a proposição de uma metodologia genérica para indução de regras de autômatos celulares. Para alcançar este objetivo a pesquisa foi segmentada em quatro etapas. Na primeira etapa avaliou-se o desempenho de alguns parâmetros de previsão de comportamento calculados em função de regras de transição. Os resultados obtidos nesta etapa indicaram que os parâmetros de previsão de comportamento dinâmico devem ser utilizados de forma criteriosa. Este cuidado reside na possibilidade de se obter soluções válidas, porém, não satisfatórias. Ressalta-se também a necessidade da existência de parâmetros de referência que para a maioria dos problemas reais, não está disponível. Na segunda etapa apresentou-se um novo método para a previsão do comportamento dinâmico. Este método considera a regra de transição e a configuração inicial do autômato celular. Para a previsão utilizou-se como referência os padrões de comportamento dinâmico qualitativos descritos por Wolfram. O método mostrou-se eficiente para regras de comportamento nulo. Como o processo de simulação da dinâmica de um sistema pode ter um custo computacional elevado, desenvolveu-se uma terceira metodologia. Nesta metodologia implementou-se uma arquitetura baseada no conceito de *hardware/software co-design* com a finalidade de contornar problemas referentes a tempo de processamento. Esta arquitetura realiza a evolução de autômatos celulares utilizando lógica reconfigurável. A arquitetura diminuiu o tempo de processamento por centenas de vezes, mas algumas restrições do modelo, como número limitado de células lógicas e reprogramações do *hardware* inviabilizaram seu uso. Considerando-se as restrições impostas pela arquitetura implementada, iniciou-se a quarta etapa da pesquisa onde foi desenvolvida uma nova arquitetura paralela fundamentada no paradigma mestre-escravo. Neste paradigma um processo mestre implementa o algoritmo evolucionário e um conjunto de processos escravos dividem a tarefa de validação das regras obtidas. O sistema é executado em um *cluster* composto por 120 núcleos de processamento que se interligam por meio de uma rede *ethernet*. A estratégia co-evolucionária baseada em um modelo insular permitiu a busca por soluções que apresentam um melhor valor para função de *fitness*. O sistema genérico implementado sobre um ambiente paralelo foi capaz de solucionar os problemas abordados. Uma análise de distribuição de tarefas entre vários processadores enfatizou os benefícios do processamento paralelo. Os experimentos também indicaram um conjunto de parâmetros evolucionários de referência que podem ser utilizados para configurar o sistema. As contribuições deste trabalho foram tanto teóricas, com as avaliações realizadas sobre os parâmetros e os diferentes

métodos de previsão de comportamento dinâmico, quanto metodológicas, pois desenvolveu-se a proposta de duas arquiteturas de processamento distintas.

PALAVRAS-CHAVE: Autômatos Celulares. Computação Evolucionária. Comportamento Dinâmico. Lógica Reconfigurável. Computação Paralela.

ABSTRACT

WEINERT, Wagner Rodrigo. Evolutionary computation to rules induction of multidimensional cellular automata. 2010. 189p. Tese (Doutorado) - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2010.

A cellular automata is a discrete dynamic system that evolves through interactions of rules and can be applied to solve several complex problems. The task to find the transition rule to solve a problem can be generalized as a problem of rule induction for cellular automata. Several approaches, based on evolutionary computation techniques, have been proposed to solve this problem. However, there is no generic methodology capable of being applied to a large range of problems. The main contribution of this work is a generic methodology for rule induction for cellular automata. This research was done in four steps to achieve this objective. In the first step we evaluated the performance of some dynamic behavior forecasting parameters calculated as a function of a transition rule. The obtained results indicated that those parameters can be used in a careful way. This is due to the possibility of obtaining valid, but unsatisfactory solutions. We stress the importance of considering reference parameters, which for the majority of real problems, are not available. In the second research step we proposed a new method to forecast the dynamic behavior. This method considers the transition rule and the initial configuration of the cellular automata. We used the qualitative dynamic behavior patterns described by Wolfram as reference to the forecast. This method was efficient for null behavior rules. Since the process of dynamic simulation can have a high computational cost, we developed a third methodology: an architecture based on the concept of hardware/software co-design to accelerate the processing time. This architecture implements the evolution of cellular automata using reconfigurable logic and was able to decrease hundreds of times the processing time. In the fourth step we developed a new parallel architecture based on the master-slave paradigm. In this paradigm, the master process implements the evolutionary algorithm and a set of slave processes divide the task of validating the obtained rules. The system runs in a cluster with 120 processing cores connected by a local area network. The co-evolutionary strategy based on an insular model allowed the search for high quality solutions. The generic system implemented over a parallel environment was able to solve the problems proposed. A task distribution analysis among several processors emphasized the benefits of parallel processing. The experiments also indicated a set of reference parameters that can be used to configure the system. The contributions of this work were theoretical and methodological. The former refers to the evaluations done and the different methods for dynamic behavior forecasting parameters. The latter is about the development of two architectures for processing.

KEY-WORDS: Cellular Automata. Evolutionary Computation. Dynamic Behavior. Reconfigurable Logic. Parallel Computing.

LISTA DE FIGURAS

| | | |
|----|---|----|
| 1 | Fluxograma de um algoritmo genético. | 35 |
| 2 | Fluxograma de um algoritmo de programação genética. | 36 |
| 3 | Fluxograma de um algoritmo de programação de expressão genética. | 38 |
| 4 | Exemplo da estrutura do cromossomo, árvore de expressão e sua regra correspondente no GEPCLASS. | 40 |
| 5 | Processo de recombinação para problemas de classificação. | 43 |
| 6 | Processo de transposição IS para problemas de classificação. | 44 |
| 7 | Processo de transposição RIS para problemas de classificação. | 44 |
| 8 | Vizinhanças para um AC unidimensional. | 46 |
| 9 | (a) Vizinhança de Neumann. (b) Vizinhança de Moore. | 47 |
| 10 | Evolução de um AC unidimensional de raio 1. | 47 |
| 11 | Diagrama de padrões espaço-temporal do AC elementar 232. | 49 |
| 12 | Classificação dos comportamentos dinâmicos segundo Wolfram (Fonte: Wolfram (WOLFRAM, 2002) p. 55-56). O tempo $t = 0$ está representado no topo de cada figura. (a) Classe 1, (b) Classe 2, (c) Classe 3, (d) Classe 4. | 50 |
| 13 | Ajuste linear dos parâmetros de previsão de comportamento dinâmico. | 74 |
| 14 | Simulação da dinâmica das regras apresentadas na Tabela 7: (a) Regra_S, (b) Regra_DV, (c) Regra_PA, (d) Regra_AA, (e) Regra_Z_W e (f) Regra_A. | 77 |
| 15 | Simulação da dinâmica das regras apresentadas na Tabela 9: (a) Regra_2, (b) Regra_3, (c) Regra_4, (d) Regra_6, (e) Regra_8 e (f) Regra_10. | 77 |
| 16 | Regra elementar 160 de comportamento homogêneo. (a) AC, (b) Cópia do AC com estado da célula 135 alterado, (c) distância de Hamming ao longo de 30 iterações. | 82 |

| | | |
|----|--|-----|
| 17 | Regra elementar 76 de comportamento ponto fixo. (a) AC, (b) Cópia do AC com estado da célula 135 alterado, (c) distância de Hamming ao longo de 30 iterações. | 83 |
| 18 | Regra elementar 182 de comportamento caótico. (a) AC, (b) Cópia do AC com estado da célula 135 alterado, (c) distância de Hamming ao longo de 4000 iterações. | 84 |
| 19 | Regra elementar 124 de comportamento complexo. (a) AC - Iteração 0 até 200, (b) Cópia do AC - Iteração 0 até 200, com estado da célula 135 alterado, (c) AC - Iteração 800 até 1000, (b) Cópia do AC - Iteração 800 até 1000, (e) distância de Hamming ao longo de 4000 iterações. | 85 |
| 20 | Análise de regras complexas de Wuensche. (a) Regra: 2EFBDFE22DAA67F045 C14B0824A22E2B, (b) Regra: 3B469C0CE4F7FA96B93B4D3SB89ECC0E0. | 86 |
| 21 | Análise de comportamento das regras: (a) GKL, (b) MCH, (c) DAV e (d) DAS. | 87 |
| 22 | Análise de comportamento da regra 331133113311331111FF11FF11FF11FF. (a) AC, (b) Cópia do AC com estado da célula 109 alterado, (c) distância de Hamming ao longo de 200 iterações. | 88 |
| 23 | Diagrama espaço-temporal. | 89 |
| 24 | Evolução do erro médio quadrático. | 92 |
| 25 | Dinâmica completa. | 93 |
| 26 | Dinâmica parcial. | 93 |
| 27 | Arquitetura do modelo híbrido. | 96 |
| 28 | As 5 camadas do protocolo TCP/IP dentro da arquitetura híbrida proposta. | 98 |
| 29 | Diagrama de blocos do Módulo Autômato Celular. | 101 |
| 30 | Fluxograma de um gerador aleatório sequencial de ACs. | 104 |
| 31 | Código fonte da rotina de geração de números aleatórios do Mersenne Twister. | 105 |
| 32 | Fluxograma de um gerador aleatório paralelo de ACs. | 106 |
| 33 | Fluxograma referente a uma iteração sequencial de ACs. | 106 |
| 34 | Fluxograma referente a uma iteração paralela de ACs. | 107 |
| 35 | Diagrama de blocos do processo paralelo de iteração do ACs. | 108 |

| | | |
|----|---|-----|
| 36 | <i>Cluster</i> de processamento. | 112 |
| 37 | Arquitetura mestre-escravo. | 113 |
| 38 | Fluxograma do PIRGEP. | 114 |
| 39 | Codificação para indução de regras em problemas de classificação. | 118 |
| 40 | Exemplo de codificação para indução de regras de transição em ACs. | 121 |
| 41 | Classificação hierárquica para base <i>Abalone</i> | 124 |
| 42 | Classificação hierárquica para base <i>Balance Scala</i> | 127 |
| 43 | Classificação hierárquica para base <i>Car Evaluation</i> | 128 |
| 44 | Classificação hierárquica para base <i>Haberman</i> | 129 |
| 45 | Classificação hierárquica para base <i>Blood Transfusion</i> | 129 |
| 46 | Classificação hierárquica para base <i>Ljubljana Breast Cancer</i> | 130 |
| 47 | Classificação hierárquica para base <i>Wisconsin Breast Cancer</i> | 131 |
| 48 | Classificação hierárquica para base <i>Iris</i> | 131 |
| 49 | Classificação hierárquica para base <i>Nurse</i> | 132 |
| 50 | Classificação hierárquica para base <i>Letter Recognition</i> | 133 |
| 51 | Configuração inicial do AC de 149 células. | 134 |
| 52 | Simulação da dinâmica da regra 160 por 20 iterações. | 134 |
| 53 | Simulação da dinâmica da regra 118 por 20 iterações. | 136 |
| 54 | Simulação da dinâmica da regra 16 por 20 iterações. | 136 |
| 55 | Simulação da dinâmica da regra 95 por 20 iterações. | 137 |
| 56 | Simulação da dinâmica da regra 183 por 20 iterações. | 138 |
| 57 | Simulação da dinâmica da regra 124 por 4000 iterações. (a) Apresenta as primeiras 20 iterações e (b) as últimas 20. | 138 |
| 58 | Configurações inicial (a) e final (b) do AC bidimensional. | 139 |
| 59 | Relação entre o tempo de processamento, número de tarefas e número de processadores. | 141 |
| 60 | Simulação da dinâmica da regra GKL por 20 iterações. | 142 |

61 Gráficos de Pareto: (a) C4.5, (b) M/E e (c) M/E Ilhas. 146

LISTA DE TABELAS

| | | |
|----|--|-----|
| 1 | Exemplo de uma regra de transição para $k = 2$ e $r = 1$ | 48 |
| 2 | Regras de referência. | 69 |
| 3 | Cálculo dos parâmetros de previsão de comportamento dinâmico de ACs. | 70 |
| 4 | Matriz de correlação dos PPCD. | 71 |
| 5 | Tabela de transições. | 73 |
| 6 | Processo de ajuste linear. | 74 |
| 7 | Regras encontradas pelo GEPCLASS considerando os parâmetros individualmente. | 75 |
| 8 | Valores dos parâmetros para as regras apresentadas na Tabela 7. | 75 |
| 9 | Regras encontradas pelo GEPCLASS considerando todos os parâmetros. | 76 |
| 10 | Valores dos parâmetros para as regras apresentadas na Tabela 9. | 76 |
| 11 | Metodologia para previsão de comportamento dinâmico. | 90 |
| 12 | Avaliação da metodologia proposta. | 91 |
| 13 | Comparação entre diferentes abordagens. | 108 |
| 14 | Vizinhanças para uma regra de transição ainda desconhecida. | 120 |
| 15 | Determinação da regra de transição. | 122 |
| 16 | Características das bases de dados. | 123 |
| 17 | Base de dados <i>Abalone</i> | 127 |
| 18 | Base de dados <i>Balance Scala</i> | 127 |
| 19 | Base de dados <i>Car Evaluation</i> | 128 |
| 20 | Base de dados <i>Haberman</i> | 129 |
| 21 | Base de dados <i>Blood Transfusion</i> | 130 |
| 22 | Base de dados <i>Ljubljana Breast Cancer</i> | 130 |

| | | |
|----|---|-----|
| 23 | Base de dados <i>Wisconsin Breast Cancer</i> | 131 |
| 24 | Base de dados <i>Iris</i> | 132 |
| 25 | Base de dados <i>Nursery</i> | 133 |
| 26 | Base de dados <i>Letter Recognition</i> | 134 |
| 27 | Resultados para a regra 160. | 135 |
| 28 | Resultados para regra 118. | 136 |
| 29 | Resultados para regra 16. | 137 |
| 30 | Resultados para regra 95. | 137 |
| 31 | Resultados para regra 183. | 138 |
| 32 | Resultados para regra 124. | 139 |
| 33 | Experimentos, com 1 ilha, para definir valores para parâmetros do algoritmo evolucionário. | 143 |
| 34 | Experimentos, com 5 ilhas, para definir valores para parâmetros do algoritmo evolucionário. | 144 |

LISTA DE ABREVIATURAS E SIGLAS

| | |
|--------------|---|
| <i>A</i> | Atividade |
| <i>AA</i> | Atividade Absoluta |
| <i>AAm</i> | Atividade Absoluta Média |
| <i>AC</i> | Autômato Celular |
| <i>ACs</i> | Autômatos Celulares |
| <i>AE</i> | Árvore de Expressão |
| <i>AEs</i> | Árvores de Expressão |
| <i>AGs</i> | Algoritmos Genéticos |
| <i>AHDL</i> | <i>Altera Hardware Description Language</i> |
| <i>Am</i> | Atividade Média |
| <i>CE</i> | Computação Evolucionária |
| <i>CM</i> | Campo Médio |
| <i>CMm</i> | Campo Médio Médio |
| <i>CPU</i> | <i>Central Processing Unit</i> |
| <i>CPU</i> | <i>Central Processing Unit</i> |
| <i>DE</i> | Distância Euclidiana |
| <i>DHCP</i> | <i>Dynamic Host Configuration Protocol</i> |
| <i>DNS</i> | <i>Domain Name System</i> |
| <i>DP</i> | Desvio-Padrão |
| <i>DV</i> | Domínio da Vizinhança |
| <i>DVm</i> | Domínio da Vizinhança Médio |
| <i>FPGAs</i> | <i>Field Programmable Gate Arrays</i> |
| <i>FTP</i> | <i>File Transfer Protocol</i> |
| <i>HTTP</i> | <i>Hyper Text Transfer Protocol</i> |
| <i>IP</i> | <i>Internet Protocol</i> |
| <i>JTAG</i> | <i>Joint Test Action Group</i> |
| <i>MLS</i> | <i>Maximum Length Sequence</i> |

| | |
|--------------|--|
| NFS | <i>Network File System</i> |
| ORF | <i>Open Read Frame</i> |
| <i>PA</i> | Propagação da Atividade |
| <i>PAm</i> | Propagação da Atividade Média |
| PEG | Programação de Expressão Genética |
| PG | Programação Genética |
| PIRGEP | <i>Parallel Induction Rules with Gene Expression Programming</i> |
| POP | <i>Post Office Protocol</i> |
| PPCD | Parâmetros de Previsão de Comportamento Dinâmico |
| RAM | <i>Random Access Memory</i> |
| <i>S</i> | Sensitividade |
| <i>Sm</i> | Sensitividade Média |
| SMTP | <i>Simple Mail Transfer Protocol</i> |
| SSH | <i>Secure Shell</i> |
| SVA | Sistemas de Vida Artificial |
| TCP/IP | <i>Transmission Control Protocol/Internet Protocol</i> |
| VHDL | <i>VHSIC Hardware Description Language</i> |
| XML | <i>Extensible Markup Language</i> |
| <i>Z_Li</i> | Z de Li |
| <i>Z_Lim</i> | Z de Li Médio |
| <i>Z_W</i> | Z de Wuensche |
| <i>Z_Wm</i> | Z de Wuensche Médio |

SUMÁRIO

| | | |
|----------|---|-----------|
| 1 | INTRODUÇÃO | 27 |
| 1.1 | MOTIVAÇÃO E JUSTIFICATIVA | 27 |
| 1.2 | OBJETIVOS | 30 |
| 1.3 | ESTRUTURA DA TESE | 30 |
| 2 | FUNDAMENTAÇÃO TEÓRICA | 33 |
| 2.1 | COMPUTAÇÃO EVOLUCIONÁRIA | 33 |
| 2.1.1 | Algoritmos Genéticos | 33 |
| 2.1.2 | Programação Genética | 35 |
| 2.1.3 | Programação de Expressão Genética | 37 |
| 2.1.3.1 | GEPCLASS | 39 |
| 2.2 | AUTÔMATOS CELULARES | 44 |
| 2.2.1 | Definição | 45 |
| 2.2.1.1 | Notação | 45 |
| 2.2.1.2 | Vizinhança | 46 |
| 2.2.1.3 | Evolução Temporal do Autômato Celular | 47 |
| 2.2.2 | Representação do Comportamento Dinâmico | 48 |
| 2.2.3 | Parâmetros de Previsão de Comportamento Dinâmico | 52 |
| 2.2.4 | O Problema de Indução de Regras | 53 |
| 2.2.4.1 | O Problema de Classificação pela Regra da Maioria | 55 |
| 2.3 | COMPUTAÇÃO RECONFIGURÁVEL POR <i>HARDWARE</i> | 55 |
| 2.4 | PROCESSAMENTO PARALELO | 56 |
| 2.5 | TRABALHOS CORRELATOS | 57 |

| | | |
|----------|--|-----------|
| 3 | INDUÇÃO DE REGRAS UTILIZANDO PARÂMETROS DE PREVISÃO DE COM- PORTAMENTO DINÂMICO | 67 |
| 3.1 | INTRODUÇÃO | 67 |
| 3.2 | METODOLOGIA | 67 |
| 3.2.1 | Análise de Correlação dos PPCD | 68 |
| 3.2.2 | Indução de Regras de Transição | 71 |
| 3.2.2.1 | Implementação | 71 |
| 3.3 | RESULTADOS | 75 |
| 3.4 | DISCUSSÃO | 78 |
| 4 | NOVO MÉTODO PARA PREVISÃO DE COMPORTAMENTO DINÂMICO | 81 |
| 4.1 | INTRODUÇÃO | 81 |
| 4.2 | METODOLOGIA | 81 |
| 4.2.1 | Modelo de Wolfram para Identificação de Padrões de Comportamento | 81 |
| 4.2.2 | Abordagem Proposta para Previsão de Comportamento Dinâmico Nulo | 88 |
| 4.3 | RESULTADOS | 90 |
| 4.4 | DISCUSSÃO | 93 |
| 5 | <i>HARDWARE-SOFTWARE CO-DESIGN</i> PARA INDUÇÃO DE REGRAS | 95 |
| 5.1 | INTRODUÇÃO | 95 |
| 5.2 | METODOLOGIA | 96 |
| 5.2.1 | Algoritmo Evolucionário | 97 |
| 5.2.2 | Comunicação entre Computador e o Sistema Embarcado | 97 |
| 5.2.3 | Sistema Embarcado - FPGA | 99 |
| 5.2.3.1 | Processador NIOS II | 99 |
| 5.2.3.2 | Memória | 99 |
| 5.2.3.3 | Circuito Digital Dedicado | 101 |
| 5.2.4 | Paralelismo | 103 |

| | | |
|----------|--|------------|
| 5.2.4.1 | Geração de ACs Aleatórios | 104 |
| 5.2.4.2 | Processo de Atualização das Células do Autômato Celular . . | 106 |
| 5.3 | RESULTADOS | 108 |
| 5.4 | DISCUSSÃO | 109 |
| 6 | PIRGEP: UM MODELO COMPUTACIONAL PARALELO PARA INDUÇÃO DE REGRAS | 111 |
| 6.1 | INTRODUÇÃO | 111 |
| 6.2 | METODOLOGIA | 111 |
| 6.2.1 | Arquitetura Paralela | 111 |
| 6.2.2 | Modelagem do PIRGEP | 114 |
| 6.2.2.1 | Modelo para Problemas Genéricos de Classificação | 117 |
| 6.2.2.2 | Modelo para Indução de Regras de Autômatos Celulares . . . | 119 |
| 6.3 | RESULTADOS | 122 |
| 6.3.1 | Problemas de Classificação | 122 |
| 6.3.2 | Autômatos Celulares | 134 |
| 6.3.2.1 | Balanceamento de Carga na Arquitetura Paralela | 140 |
| 6.3.2.2 | Otimização dos Parâmetros Evolucionários | 141 |
| 6.4 | DISCUSSÃO | 144 |
| 7 | CONCLUSÕES E TRABALHOS FUTUROS | 153 |
| 7.1 | CONCLUSÕES | 153 |
| 7.2 | TRABALHOS FUTUROS | 156 |
| | REFERÊNCIAS | 159 |

1 INTRODUÇÃO

1.1 MOTIVAÇÃO E JUSTIFICATIVA

Os Sistemas de Vida Artificial (SVA) tentam simular os comportamentos e as características dos sistemas naturais. O comportamento ou dinâmica de um sistema emerge de um processo iterativo decorrente da atuação de regras simples sobre cada um dos elementos primitivos que formam o sistema. Assim, um SVA pode ser definido como uma coleção de unidades simples e básicas, onde a principal característica é a emergência de comportamentos complexos (LANGTON, 1995).

Segundo Luger (2004), os Autômatos Celulares (ACs) são famílias de máquinas de estado finito que exibem comportamentos emergentes através de suas iterações. O estudo de sociedades de máquinas de estado finito conectadas e analisadas como entidades emergentes foi denominado, por alguns autores, como programação evolucionária (FOGEL; OWENS; WALSH, 1996).

Em essência, um Autômato Celular (AC) é representado por um reticulado d -dimensional, uma vizinhança e uma regra de transição de estado que determina qual será o próximo estado do reticulado, a partir de seu estado atual (WOLFRAM, 1994).

O problema intitulado classificação pela regra da maioria (GACS; KURDYUMOV; LEVIN, 1978; ANDRE; BENNETT III; KOZA, 1996; JUILLÉ; POLLACK, 1998) é um problema largamente estudado pela teoria de ACs. Basicamente, o problema consiste na busca de uma regra de transição que conduza todas as células iniciais de um arranjo d -dimensional para o estado 0 ou estado 1 após t iterações. O estado final esperado para as células do arranjo depende da densidade apresentada pelo AC em sua configuração inicial. A densidade é obtida a partir da divisão do número de células com estado igual a 1 pelo número total de células do AC. Além deste problema, na Seção 2.5 são apresentados diversos trabalhos onde se aplica a teoria de ACs para resolução de problemas.

A dinâmica de um AC pode ser ilustrada por diagramas de padrões espaço-temporais, nos

quais a configuração de estados em um reticulado d -dimensional é determinada como uma função do tempo. A partir da análise destes diagramas é possível classificar o comportamento dinâmico de um sistema. Wolfram (1984) propôs um padrão de classificação de comportamento representado por 4 classes qualitativas. Posteriormente, Li e Packard (1990) refinaram a classificação de Wolfram em 6 classes.

Quando a regra de transição de estado é conhecida, simular o comportamento dinâmico de um AC para uma determinada aplicação é simples. No entanto, esta situação não ocorre para a maioria das aplicações de ACs. A tarefa de encontrar regras de transição de estado que simulem determinado comportamento dinâmico de um AC pode ser generalizada como um problema de indução de regras para ACs.

A maioria dos problemas modelados através de ACs apresentam um espaço de busca de soluções intratável para algoritmos de busca baseados em estratégias determinísticas, o que torna o problema de indução de regras para ACs difícil. Por exemplo, para um AC unidimensional de raio 5 tem-se um espaço de busca de $3,2 \times 10^{616}$ regras. Para um AC bidimensional de mesmo raio o espaço de busca é de $2^{26 \times 10^{35}}$ e para um AC tridimensional é de $2^{4 \times 10^{100}}$.

Esforços têm sido gastos na busca de novas abordagens eficientes e capazes de simular o comportamento desejado na tarefa de indução de regras para ACs. A maioria destas abordagens (RICHARDS; MEYER; PACKARD, 1990; ANDRE; BENNETT III; KOZA, 1996; JUILLÉ; POLLACK, 1998; MORALES; CRUTCHFIELD; MITCHELL, 2001; OLIVEIRA; OMAR; DE-OLIVEIRA, 2001; FERREIRA, 2002b; OLIVEIRA; ASAKURA; DE-OLIVEIRA, 2002; OLIVEIRA; BORTOT; DE-OLIVEIRA, 2002, 2007) utilizam técnicas baseadas em algoritmos evolucionários, como por exemplo: Algoritmos Genéticos (AGs) (GOLDBERG, 1989; HOLLAND, 1975; COLEY, 1997; SIVANANDAM; DEEPA, 2007) e Programação Genética (PG) (KOZA, 1992, 1994; LANGDON; POLI, 2002; POLI; LANGDON; MCPHEE, 2008). Estas técnicas evoluem uma população de indivíduos (no caso, regras) por n gerações. Em cada geração um conjunto de operadores genéticos atua sobre a população de indivíduos a fim de melhorar a qualidade do material genético. Esta qualidade é avaliada pela função de *fitness*, e normalmente o processo de evolução é encerrado quando uma solução satisfatória é encontrada ou quando algum critério de término é alcançado, por exemplo, um número máximo de gerações.

Em geral, no problema de indução de regras para ACs utiliza-se como função de *fitness* uma medida obtida através da análise do comportamento dinâmico do sistema, onde cada uma das possíveis soluções (regras) é submetida a um processo de simulação da dinâmica. A regra atua sobre uma configuração inicial de AC por t iterações, obtendo-se a configuração final que

é comparada à configuração esperada. Se ambas forem iguais então a regra é solução do problema, senão uma medida de similaridade quantifica a qualidade da regra. Nesta abordagem o cálculo do *fitness* tem um elevado custo computacional, o que por vezes, inviabiliza o processo. Uma importante contribuição para esta área é o desenvolvimento de novas metodologias que permitam reduzir o tempo de processamento deste processo. A utilização de um *hardware* dedicado, capaz de implementar o paralelismo intrínseco dos ACs, pode ser uma abordagem bastante interessante (WEINERT; BENITEZ; LOPES; LIMA, 2007).

Segundo Culik, Hurd e Yu (1990) a previsão do comportamento dinâmico de um AC não pode ser realizada com exatidão. Esta restrição é decorrente do enorme espaço de estados iniciais que um AC pode assumir. Pode ocorrer situações onde uma mesma regra de transição, aplicada a diferentes estados iniciais, apresente diferentes comportamentos dinâmicos ao longo das iterações. Muitas vezes, é necessário centenas de passos iterativos até que seja possível caracterizar o comportamento dinâmico de um AC. Uma outra importante contribuição para a área seria o desenvolvimento de uma metodologia onde fosse possível antecipar o conhecimento a respeito da dinâmica do sistema. Basicamente, reduzir o número de passos iterativos sem perder a precisão da previsão do comportamento dinâmico.

A literatura apresenta algumas métricas, conhecidas como Parâmetros de Previsão de Comportamento Dinâmico (PPCD), que podem auxiliar na tarefa de previsão do comportamento dinâmico de um AC, tais como: sensibilidade (BINDER, 1993), domínio da vizinhança, propagação da atividade e atividade absoluta (OLIVEIRA; OMAR; DE-OLIVEIRA, 2001), parâmetro Z (WUENSCHÉ, 1999), atividade (LANGTON, 1990), parâmetros de campo médio (LI, 1991), parâmetro Z de Li (LI, 1991). Oliveira, Asakura e De-Oliveira (2002) demonstraram que os parâmetros sensibilidade, atividade absoluta, domínio da vizinhança e propagação da atividade podem auxiliar um algoritmo genético co-evolucionário no problema de indução de regras para o problema de classificação pela regra da maioria. Neste trabalho a avaliação das regras é obtida em função de dois parâmetros. O primeiro parâmetro considera os valores obtidos pelo cálculo dos parâmetros de previsão de comportamento e o segundo parâmetro pela simulação da dinâmica do sistema. Atribui-se pesos diferentes a estes parâmetros. No entanto, ainda não foram reportados resultados onde se considera apenas os parâmetros de previsão de comportamento dinâmico no processo de avaliação das regras. Um estudo a respeito da utilização destes parâmetros pode revelar novos conhecimentos a respeito da utilização dos mesmos no processo de indução de regras para ACs.

Embora a literatura relate algumas técnicas aplicadas ao problema de indução de regras, uma restrição permanece constante, pois todas as abordagens são especializadas, ou seja, solu-

cionam um problema específico e não se apresenta uma solução genérica. Entende-se que o desenvolvimento de uma metodologia genérica é uma importante contribuição para esta área, onde se pode utilizar técnicas de computação evolucionária para agregar as diversas propriedades apresentadas pelos ACs para permitir a indução de regras em uma variedade de aplicações distintas. O problema de classificação pela regra da maioria apresenta-se como um excelente instrumento de estudo e consequente aprendizado para o desenvolvimento deste trabalho, uma vez que já foi explorado pela literatura.

1.2 OBJETIVOS

O objetivo geral deste trabalho é desenvolver métodos para indução de regras de automatos celulares multidimensionais.

Os objetivos específicos são:

- Desenvolver um método de computação evolucionária para indução de regras de ACs que apresentem comportamento nulo para o problema de classificação pela regra da maioria.
- A partir da análise de modelos descritos na literatura para previsão de comportamento dinâmico, apresentar uma nova metodologia para realização desta previsão.
- Apresentar um modelo híbrido de integração que permita conectar o sistema de simulação da dinâmica implementado em *hardware* (WEINERT; BENITEZ; LOPES; LIMA, 2007), com as devidas modificações e integração de novos módulos, a um sistema de indução de regras implementado em *software*.
- Implementar uma arquitetura insular paralela de processamento baseado no algoritmo evolucionário de Programação de Expressão Genética (PEG). O algoritmo destina-se tanto a resolução de problemas de classificação quanto ao processo de indução de regras de ACs multidimensionais.

1.3 ESTRUTURA DA TESE

O restante da tese está estruturado da seguinte forma. No capítulo 2 apresenta-se a fundamentação teórica. Descrevem-se conceitos e definições a respeito de ACs, algoritmos de computação evolucionária, lógica reconfigurável, computação paralela e são relatados alguns trabalhos relacionados a esta tese. O capítulo 3 apresenta um modelo evolucionário para o

problema de indução de regras de ACs utilizando PPCD. O capítulo 4 destina-se ao desenvolvimento metodológico de um novo método para previsão de comportamento dinâmico baseado num modelo de simulação parcial da dinâmica. O capítulo 5 apresenta um modelo híbrido de integração entre *hardware* e *software* para o problema de indução de regras de ACs. O capítulo 6 apresenta um modelo computacional paralelo para indução de regras. No capítulo 7 são apresentadas as conclusões e indicados novos caminhos que podem ser trilhados na continuidade deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 COMPUTAÇÃO EVOLUCIONÁRIA

A Computação Evolucionária (CE) tem sido aplicada em diversas áreas do conhecimento, desde ciências naturais e engenharia até biologia e ciência da computação. A CE abrange uma série de métodos computacionais inspirados na teoria da evolução das espécies, mais especificamente no conceito de seleção natural. Estes métodos manipulam soluções candidatas, também chamadas de indivíduos. Um conjunto de indivíduos constitui uma população. Estes indivíduos são apresentados a um ambiente, no caso o problema, onde os mais aptos sobreviverão. A evolução dos indivíduos, e consequente sobrevivência e adaptação dos mesmos, é realizada por meio de um processo iterativo de exploração do espaço de soluções através de métodos de seleção e operadores genéticos específicos. Cada processo iterativo constitui uma geração. Fazendo um paralelo com a evolução das espécies, uma nova geração acontece a cada nova reprodução, por meio dos descendentes. A aptidão de um indivíduo é mensurada por uma função de qualidade, conhecida na computação evolucionária como função de *fitness*. Esta função é particularizada para cada ambiente (EIBEN; SMITH, 2008). Diversos paradigmas e métodos computacionais implementam variações destes conceitos, tais como: Algoritmos Genéticos (AGs) (HOLLAND, 1975; GOLDBERG, 1989; COLEY, 1997; SIVANANDAM; DEEPA, 2007), Programação Genética (PG) (KOZA, 1992, 1994; LANGDON; POLI, 2002; POLI; LANGDON; MCPHEE, 2008) e Programação de Expressão Genética (PEG) (FERREIRA, 2002a, 2002b, 2006).

2.1.1 Algoritmos Genéticos

Os AGs constituem uma classe de métodos de busca de propósito geral, que não exploram o espaço de busca de um determinado problema por inteiro e, sim, realizam buscas orientadas pelas melhores soluções encontradas até determinado momento para encontrar uma solução satisfatória para o problema.

Os AGs são algoritmos de otimização, baseados nos mecanismos da seleção natural e da

genética. Eles empregam uma estratégia de busca paralela e estruturada que é voltada em direção ao reforço da busca em torno de pontos de maior aptidão, ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos).

Quando se opta por utilizar AGs com o objetivo de resolver um determinado problema, depara-se de início com uma tarefa importante, que é a codificação do problema. Cada variável é discretizada em um determinado intervalo e representada como um conjunto de *bits* (em sua forma original), sendo que o conjunto de variáveis codificado é denominado cromossomo. Um ou mais cromossomos são associados para formar um indivíduo. É importante ressaltar que, na maioria das vezes, o AG manipula os indivíduos (variáveis codificadas) e não as variáveis propriamente ditas. Uma codificação inadequada pode tornar o problema difícil ou mesmo impossível para o AG.

Uma população é formada por um conjunto de indivíduos. Pode-se gerar esta população aleatoriamente, com distribuição uniforme, ou levar em consideração algum conhecimento prévio a respeito do problema. Esta população de possíveis soluções caracteriza a exploração paralela do espaço de busca.

As possíveis soluções para um dado problema são representadas pelos indivíduos desta população. Logo, é necessário estabelecer uma maneira de avaliar a qualidade da solução, ou seja, o quão adaptado ao problema está cada indivíduo. Esta medida de avaliação é conhecida como *fitness*. O cálculo do *fitness* é um ponto crítico para o algoritmo, já que, em última análise, é a função de *fitness* que está sendo otimizada.

Com base no valor de *fitness* dos indivíduos, estes são selecionados de tal maneira a privilegiar as melhores soluções em detrimento das piores. Este procedimento imita o processo de seleção natural que guia a evolução das espécies. Os indivíduos são selecionados por métodos probabilísticos (ou mesmo determinísticos) de modo a poderem gerar descendentes, implementando, assim, o mecanismo da sobrevivência do mais apto.

Os operadores genéticos são, então, aplicados probabilisticamente sobre os indivíduos selecionados. O operador de recombinação, também conhecido como *crossover*, opera sobre dois indivíduos e combina partes de ambos para formar dois novos descendentes. O operador de mutação atua num determinado indivíduo alterando aleatoriamente um bit de sua composição.

O ciclo de avaliação do *fitness*, seleção dos indivíduos mais aptos, aplicação do operador de recombinação e operador de mutação é repetido por determinado número de vezes, esperando-se que a cada geração a qualidade média dos indivíduos aumente. Depois de uma certa quantidade de gerações é possível que tenham sido geradas soluções boas para o problema.

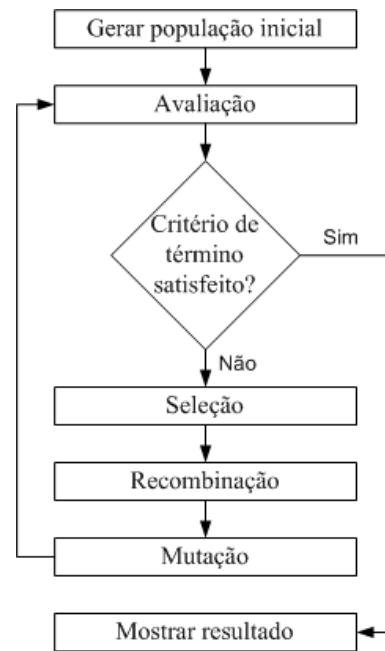


Figura 1: Fluxograma de um algoritmo genético.

A Figura 1 mostra o fluxograma de um algoritmo genético básico.

2.1.2 Programação Genética

A PG foi desenvolvida por John Koza (KOZA, 1992, 1994) e foi baseada nos trabalhos de John Holland sobre AGs (HOLLAND, 1975).

A idéia da PG é “ensinar” computadores a se programarem. Isto é, a partir de especificações de comportamento, o computador deve ser capaz de induzir um programa que satisfaça as especificações (KOZA, 1992). Isto é, a PG permite que os computadores resolvam problemas para os quais não foram previamente programados. A cada programa associa-se um valor de adequabilidade (*fitness*) que representa a capacidade deste programa em resolver um determinado problema.

Fundamentalmente, a PG mantém uma população de programas de computador, utiliza métodos de seleção baseados na capacidade de adequação (*fitness*) de cada programa e aplica operadores genéticos para modificá-los.

A especificação de comportamento é definida, normalmente, por um conjunto de valores de entrada e saída, denominados casos de *fitness*, representando o conjunto de treinamento. Com base neste conjunto, a PG visa evoluir um programa que produza, primeiramente, de forma não trivial, as saídas corretas para cada entrada fornecida (casos de *fitness*). Em segundo

lugar, o programa computa as saídas de tal forma que, se as entradas forem representativamente escolhidas, o programa terá capacidade de obter saídas corretas para entradas não cobertas pelo conjunto de treinamento (O'REILLY, 1995).

Pelo fato da PG manipular programas diretamente, uma estrutura de representação relativamente complexa e variável embasa este paradigma. Nos modelos tradicionais, esta estrutura é uma árvore de sintaxe abstrata composta por funções em seus nós internos e por terminais em seus nós folhas. A especificação do domínio do problema é estabelecida pela definição dos conjuntos de funções e de terminais (KOZA, 1992). Os conjuntos de funções e de terminais devem satisfazer as propriedades de fechamento e suficiência. A propriedade de fechamento estabelece que cada uma das funções do conjunto de funções seja capaz de aceitar, como seus argumentos, qualquer valor e tipo de dado que possa ser retornado por qualquer função do conjunto de funções ou assumido por qualquer terminal do conjunto de terminais. A propriedade da suficiência requer que o conjunto de terminais e o conjunto de funções sejam capazes de expressar a solução do problema.

O fluxograma básico da PG é mostrado na Figura 2.

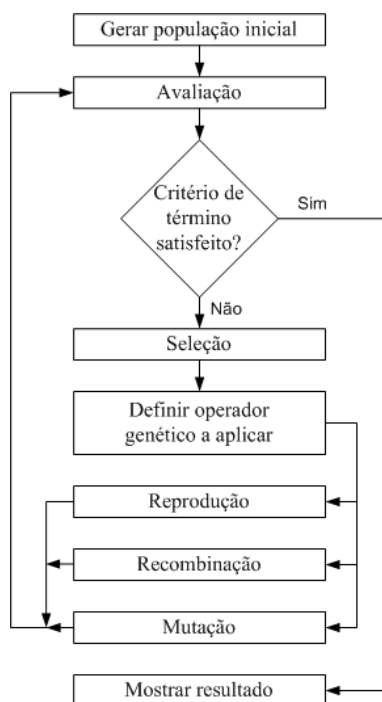


Figura 2: Fluxograma de um algoritmo de programação genética.

Em PG, normalmente quatro medidas explícitas de *fitness* podem ser utilizadas para quantificar a adaptação de um determinado indivíduo ao seu ambiente: *fitness raw*, *fitness* padronizado, *fitness* ajustado e *fitness* normalizado. Formalizações a respeito destas medidas podem ser en-

contradas em Koza (1992).

Uma das etapas da PG é a seleção dos indivíduos da população que estarão sujeitos à ação dos operadores genéticos tais como: recombinação e reprodução. Para realizar esta seleção a PG utiliza uma metodologia de seleção baseada no valor de *fitness* dos indivíduos. Existem diversos métodos de seleção, como por exemplo: roleta (GOLDBERG, 1989) e torneio (KOZA, 1992). A decisão de qual utilizar em determinada circunstância é uma tarefa importante dentro da PG (BANZHAF; NORDIN; KELLER; FRANCONI, 1998).

A PG implementa, em sua forma mais simples, dois operadores genéticos: o operador de reprodução e o de recombinação. O operador de reprodução seleciona um indivíduo dentro da população, de acordo com algum método de seleção, e o copia para a próxima geração sem que este sofra nenhuma alteração em seu material genético ou sua estrutura. O operador de recombinação cria diversidade na população, pela produção de uma descendência que consiste de partes retiradas de dois indivíduos previamente selecionados. A operação de recombinação começa com dois indivíduos-pais que trocam material genético originando dois novos indivíduos. Há também o operador de mutação, porém sua utilização não traz benefícios significativos à PG e normalmente não é utilizado (KOZA, 1992).

2.1.3 Programação de Expressão Genética

A PEG (FERREIRA, 2006) é um modelo de algoritmo evolucionário, que tem semelhanças com AGs e PG. A diferença fundamental entre os três algoritmos, AGs, PG e PEG, reside na natureza dos indivíduos. Nos AGs, os indivíduos são cadeias lineares normalmente de tamanho fixo (cromossomos) e na PG os indivíduos são entidades não-lineares de diferentes tamanhos e formas (árvores). Na PEG, os indivíduos são codificados em cadeias lineares de tamanho fixo (genoma ou cromossomo) que são expressas posteriormente como entidades não-lineares com diferentes tamanhos e formas (representações esquemáticas simples ou Árvores de Expressão (AEs)) (FERREIRA, 2006).

Na PEG, os cromossomos são entidades simples: lineares, compactas, relativamente pequenas e fáceis de manipular geneticamente (replicar, mutar, recombinar e transpor). As AEs são a expressão do respectivo cromossomo, e são as entidades sobre as quais opera a seleção, sendo selecionadas para se reproduzirem com modificação de acordo com sua aptidão. No processo de reprodução são os cromossomos dos indivíduos e não as AEs que são modificados e transmitidos à geração seguinte.

A organização estrutural dos cromossomos da PEG deve ser preservada visto que qualquer

modificação realizada sobre o genoma deverá sempre resultar em AEs ou programas sintaticamente corretos. O fluxograma básico da PEG é mostrado na Figura 3.

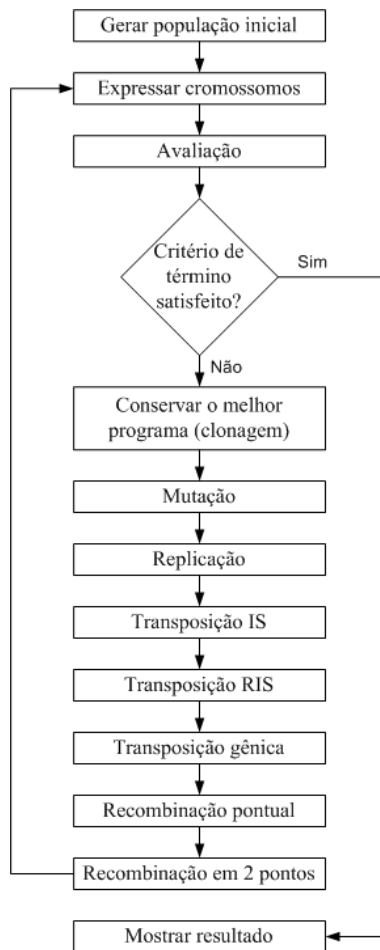


Figura 3: Fluxograma de um algoritmo de programação de expressão genética.

O primeiro passo a ser executado pelo algoritmo é a criação da população inicial. Este processo pode ser realizado de duas maneiras, aleatoriamente ou seguindo algum conhecimento prévio a respeito do problema. Em seguida, expressam-se os cromossomos através das AEs. Este processo transcorre pela aplicação da linguagem de transferência Karva (FERREIRA, 2006) sobre os cromossomos, das quais se extrai as soluções candidatas. O próximo passo é realizar a avaliação dos indivíduos, ou seja, o cálculo do *fitness*. Um indivíduo pode ser formado por mais de um cromossomo, entretanto todo o procedimento aqui explicado se refere a um indivíduo formado sempre por um único cromossomo, ou seja, indivíduo e cromossomo são vistos como sinônimos, para facilitar o entendimento. Terminado o processo de avaliação, testa-se a condição de término do algoritmo da seguinte maneira: se um critério de solução (por exemplo, medida de *fitness*) ou de parada (por exemplo, número de gerações) for alcançado encerra-se o processo, senão, aplica-se uma política de conservação do melhor programa e a

população de indivíduos é submetida a um processo de seleção que guia a aplicação de vários operadores genéticos introduzindo variação genética na população. Um indivíduo pode sofrer tanto a ação de um único operador como a de vários. Então, tem-se uma nova geração de programas que serão submetidos ao mesmo processo iterativo. Isto vai se repetir por um número determinado de gerações ou até que se encontre a melhor solução ou, pelo menos, uma solução satisfatória para o problema em questão.

Weinert e Lopes desenvolveram uma série de melhorias e adaptações no algoritmo da PEG originalmente definida por Ferreira (2001). Dentre estas melhorias destacam-se duas ferramentas de *software* conhecidas como EGIPSYS (WEINERT, 2004; LOPES; WEINERT, 2004; WEINERT; LOPES, 2007) e GEPCLASS (WEINERT, 2004; WEINERT; LOPES, 2006). O EGIPSYS é uma ferramenta genérica para solução de problemas de regressão simbólica e o GEPCLASS é utilizado para solução de problemas de classificação de dados.

A regressão simbólica objetiva encontrar uma função de identificação que seja capaz de mapear de forma simbólica um conjunto finito de amostras de pontos. Na regressão simbólica conhece-se apenas um conjunto de entradas e saídas (variáveis independentes e dependentes, respectivamente), mas não se conhece nada a respeito do modelo. O objetivo neste caso é encontrar uma função de transferência baseada nas variáveis independentes, que seja capaz de minimizar uma medida de erro entre os valores da variável dependente, calculadas pela função encontrada e o valor desejado.

Os problemas intitulados de classificação buscam encontrar uma regra ou um conjunto de regras que seja capaz de modelar um certo domínio de amostras conhecidas, para então futuramente classificar novos conjuntos de dados como pertencentes ou não a uma classe específica. O objetivo é, então, prever o valor de um atributo-meta, sendo fornecidos os valores de outros atributos, denominados atributos precursores. Frequentemente as regras são representadas na forma SE-ENTÃO. A parte SE é o antecedente e pode ser formada por combinações lógicas dos atributos precursores e seus respectivos valores. A parte ENTÃO é o conseqüente e prediz o valor para algum atributo-meta. Na seqüência descreve-se a metodologia implementada no GEPCLASS, uma vez que neste trabalho são utilizadas partes desta metodologia.

2.1.3.1 GEPCLASS

O GEPCLASS (WEINERT; LOPES, 2006) implementa uma estrutura, onde uma população de indivíduos evolui por n gerações. A cada geração um conjunto de operadores genéticos introduz diversidade genética e promove a melhoria do material genético segundo uma função de avaliação de qualidade conhecida como função de *fitness*.

No contexto adotado, um indivíduo é composto de um único cromossomo que, por sua vez é, composto de e genes. Cada gene subdivide-se em duas estruturas: cabeça e cauda. A cabeça do gene pode ser preenchida por elementos pertencentes a um conjunto de funções, normalmente: AND, OR, NOT, =, \neq , $>$ e $<$, e a cauda por elementos pertencentes a um conjunto de terminais. Os terminais codificam atributos e valores particulares ao problema codificado.

A PEG implementa o conceito de ORF (*Open Read Frame*) no processo de codificação. Segundo este conceito biológico nem todo material genético disponível é utilizado para codificar um gene. Assim, a ORF, ou região codificadora de um gene, pode ativar e desativar material genético durante o processo evolutivo. Logo, nem todo material codificado dentro dos genes é utilizado na solução do problema.

Todo cromossomo é transcrito para uma Árvore de Expressão (AE) de tamanho variável. O material genético utilizado neste processo de transcrição é o material ativo presente em cada gene. As regras de transcrição seguem os padrões expressos pela linguagem Karva (FERREIRA, 2006) onde cada gene é transcrito para uma sub-árvore. Em seguida, unem-se as sub-árvores através de uma função de ligação (AND ou OR) gerando, assim, a AE. A AE representa a solução candidata para o problema em questão e, a partir da avaliação desta árvore, obtém-se o *fitness* do indivíduo. Na Figura 4 apresenta-se o processo de transcrição do gene.

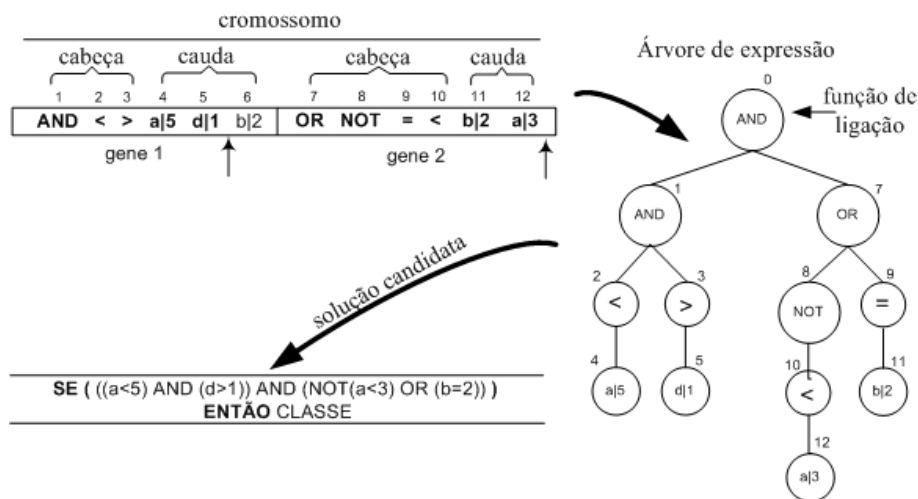


Figura 4: Exemplo da estrutura do cromossomo, árvore de expressão e sua regra correspondente no GEPCLASS.

No gene 1 a ORF termina na posição 5 e no gene 2 na posição 12, como indicado pelas setas. A ORF de um gene é determinada pelas funções e suas respectivas aridades (número de argumentos) codificadas na cabeça do gene. A leitura do gene segue de esquerda para di-

reita começando na posição mais à esquerda. Por exemplo: no gene 1, posição 1 tem-se o operador lógico AND cuja aridade é 2, ou seja, são necessários dois argumentos para que se obtenha uma resposta falsa ou verdadeira em relação a este operador. Assim, os dois operadores seguintes ao AND (< na posição 2 e > na posição 3) tornam-se argumentos do AND. Recursivamente determinam-se os argumentos de < e >. Este processo se repete até que não exista mais funções sem argumentos. O tamanho da cauda do gene é determinada em função do tamanho da cabeça (WEINERT; LOPES, 2006). Assim, garante-se que nunca faltarão argumentos para as funções codificadas na cabeça do gene. Deve-se sempre manter uma estrutura de precedência hierárquica entre os operadores lógicos, os operadores relacionais e os terminais, respectivamente.

No GEPCLASS, os cromossomos podem ser de tamanhos variáveis. Estes cromossomos podem ser formados por um ou mais genes, sendo estes de mesmo tamanho. Metade da população é gerada de maneira uniforme, respeitando uma proporcionalidade parametrizada por um intervalo de tamanho de cabeças definido pelo usuário. A outra metade é gerada aleatoriamente dentro deste mesmo intervalo. Esta abordagem está inspirada na técnica do *ramped-half-and-half* (KOZA, 1994).

Os métodos de seleção implementados no GEPCLASS são: método da roleta (GOLDBERG, 1989) e o torneio estocástico (KOZA, 1992).

A função de *fitness* utilizada é mostrada na Equação 1 e está baseada em Lopes (1996). Esta função combina dois índices: especificidade (E) (Equação 2) e sensibilidade (S) (Equação 3) que relacionam variáveis ligadas a conceitos definidos para regras de classificação.

$$fitness = E.S \quad (1)$$

$$E = \frac{vn}{vn + fp} \quad (2)$$

$$S = \frac{vp}{vp + fn} \quad (3)$$

onde

- vp : verdadeiro positivo - A regra prediz que a instância pertence a uma determinada classe e ela realmente pertence a esta classe.
- fp : falso positivo - A regra prediz que a instância pertence a uma determinada classe, mas

ela não pertence a esta classe.

- *vn*: verdadeiro negativo - A regra prediz que a instância não pertence a uma determinada classe e ela realmente não pertence.
- *fn*: falso negativo - A regra prediz que a instância não pertence a uma determinada classe, no entanto ela pertence a esta classe.

Muitas vezes ocorrem disparidades entre os valores de *fitness* dos indivíduos dentro de uma população. Isto pode prejudicar o processo evolutivo de busca da solução para um determinado problema, pois a pressão seletiva causada pela aplicação dos operadores de seleção baseados no valor de *fitness* dos indivíduos acaba normalmente selecionando os mesmos indivíduos. Isto leva o algoritmo a uma convergência prematura, ou seja, toda a população fica praticamente igual, sem atingir uma solução satisfatória e estagnando a evolução. Para contornar esta limitação foi implementado um procedimento de escalonamento linear para os valores de *fitness*, conforme aquele proposto por Goldberg (1989). Para AGs este escalonamento preserva o valor do *fitness* médio da população após sua aplicação. A intenção deste escalonamento é aproximar os indivíduos fazendo com que estes tenham mais chances de se manterem dentro da população no decorrer do processo evolutivo, evitando uma possível convergência prematura da população.

Os operadores genéticos sofreram modificações de modo a garantir a propriedade de fechamento, produzindo sempre regras válidas. A estrutura hierárquica das AEs também deve ser mantida após a aplicação dos operadores.

O operador de mutação atua em 3 níveis distintos, garantindo sempre o cumprimento da propriedade de fechamento. Este deve sempre substituir um operador lógico por outro operador lógico, um operador relacional por outro relacional e um terminal por outro terminal. Se a mutação ocorrer num terminal, será escolhido um novo atributo, bem como um valor pertencente ao domínio deste atributo. Quando o operador NOT for selecionado para sofrer mutação a operação não será realizada, uma vez que ele é o único operador lógico de aridade 1. A permutação do operador NOT (aridade 1) com o operador AND (aridade 2), por exemplo, poderia gerar uma regra inválida.

O operador de recombinação opera da mesma forma tanto para a recombinação entre pais de mesmo tamanho quanto para pais de tamanhos diferentes. Após a seleção dos pais seleciona-se o gene que será submetido à operação de recombinação e um ponto de corte pertencente à cabeça do menor pai, caso sejam de tamanhos diferentes. Este operador sempre troca material genético entre as cabeças dos genes. A Figura 5 apresenta o processo de recombinação.

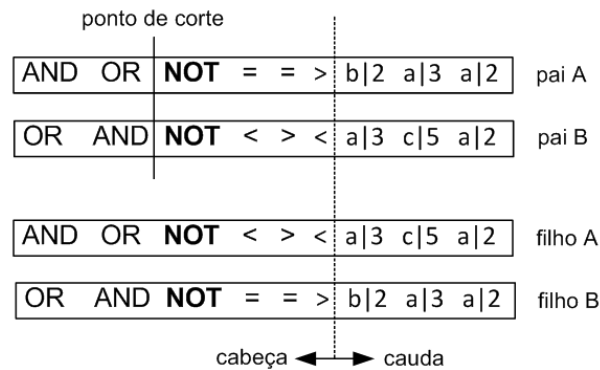


Figura 5: Processo de recombinação para problemas de classificação.

Dados dois pais formados por cromossomos de um único gene seleciona-se um ponto de corte (sobre uma função) na cabeça do gene do pai A e extrai-se a aridade da função presente neste ponto de corte (por exemplo, NOT - aridade 1). Percorre-se então o pai B, da esquerda para a direita, procurando-se por uma função de mesma aridade dentro da cabeça do gene. Encontrada tal função (NOT, posição 2) tem-se então o ponto de corte relacionado ao pai B. Caso não se encontre nenhuma função de mesma aridade, o operador de recombinação não atua. A partir da determinação dos pontos de corte, percorre-se ambos os pais até que a aridade de uma determinada função seja diferente entre o pai A e o pai B, ou até que se alcance o ponto limite entre a cabeça e a cauda do gene. Este procedimento fornecerá o segmento de material genético que será permutado entre os pais. No exemplo da Figura 5 selecionou-se o ponto de corte na posição 2 para o pai A, cuja aridade da referida função é 1. Ao se procurar por uma função de mesma aridade no pai B obteve-se como resultado a mesma posição. Transcorrida a análise, conclui-se que serão permutadas 4 posições entre as cabeças dos genes que seguem a seguinte seqüência de aridade: 1;2;2 e 2 em ambos os pais. Logo, esta troca não afetará a estrutura sintática que estes cromossomos representam.

O operador de transposição IS atua sobre um único cromossomo. Ele modifica o material genético presente na cauda do gene, ou seja, altera a posição dos terminais dentro do gene. Pode levar material genético de um gene para outro, como também pode transpor material dentro de um mesmo gene. Selecionados os pontos de envio e recebimento de informação no gene doador e no receptor respectivamente, define-se o tamanho da transposição. O segmento a ser transposto desloca para a esquerda o material genético tantas posições quanto o tamanho para a transposição a partir do ponto de recebimento de informação. Caso o gene doador e receptor sejam os mesmos (como mostrado na Figura 6) ocorre apenas uma permutação de posições dentro da cauda do gene, enquanto que, se estes forem diferentes, o gene receptor perde em sua cauda tantos terminais quanto for o tamanho da transposição.

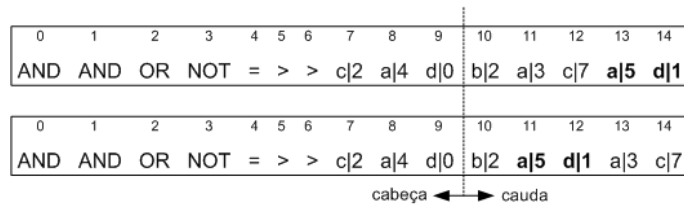


Figura 6: Processo de transposição IS para problemas de classificação.

O operador de transposição RIS também atua sobre um único cromossomo. O ponto de envio de informação sempre se encontra na cauda do gene, enquanto que o ponto de recebimento de informação sempre é o primeiro terminal presente neste mesmo gene. O segmento a ser transposto desloca para a esquerda o material genético, a partir do ponto de recebimento de informação, tantas posições quanto for o tamanho da transposição, ocorrendo apenas uma permutação de posições dentro da cauda do gene. A Figura 7 mostra este processo.

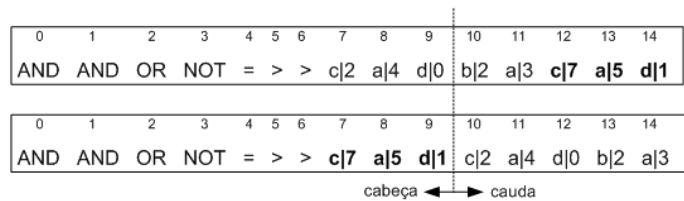


Figura 7: Processo de transposição RIS para problemas de classificação.

Detalhes de implementação dos operadores genéticos e suas propriedades de fechamento, assim como da função de *fitness* implementada para solucionar problemas de classificação de forma genérica são encontrados em Weinert (2004) e Weinert e Lopes (2006).

2.2 AUTÔMATOS CELULARES

Um AC é um sistema dinâmico discreto que evolui pela iteração de uma regra determinística simples, sendo que os valores das variáveis do sistema mudam em função de seus valores correntes (WUENSCHER; LESSER, 1992).

John Von Neumann e Stanislaw Ulam foram os idealizadores dos primeiros conceitos sobre ACs, com o objetivo de projetar mecanismos artificiais capazes de autorreprodução (NEUMANN; BURKS, 1966). Em 1970, o matemático John Conway apresentou a descrição de um AC binário bidimensional para um jogo de computador chamado *game of life* (jogo da vida). A evolução deste AC é orientada por uma regra de transição simples. O trabalho de Conway demonstrou que regras simples aplicadas repetidamente sobre estados aleatórios produzem

resultados que se assemelham aos apresentados por sistemas do mundo real (BERLEKAMP; CONWAY; GUY, 2001).

O comportamento dinâmico de um AC pode ser representado por sua evolução ao longo do tempo em um diagrama espaço-temporal. Wolfram é um dos pesquisadores mais importantes desta área. Dentre suas contribuições destaca-se uma classificação para os diferentes comportamentos dinâmicos apresentados pelos ACs, além de inúmeros trabalhos que estudam características destes comportamentos (WOLFRAM, 1983, 1983b, 1984, 1984b, 1988, 1994, 2002).

2.2.1 Definição

Um AC pode ser representado por um arranjo d -dimensional. Todas as células desta estrutura são idênticas e mantém uma relação de vizinhança determinada por um raio (r) predefinido. O tamanho (m) desta vizinhança é definido em função de r na forma: $m = 2r + 1$.

As condições de contorno permitem que células situadas nas extremidades do arranjo se conectem entre si. Todas as células do arranjo d -dimensional são atualizadas em paralelo por uma regra de transição de estados. Esta regra aplicada sobre cada célula determina o próximo estado da célula em função de seu estado atual e do estado atual das células que formam sua vizinhança.

2.2.1.1 Notação

Mitchell (1996) definiu a seguinte notação para ACs:

- Σ : conjunto de elementos possíveis para cada célula;
- k : número de estados do conjunto Σ ;
- i : índice que denota uma célula;
- S_i^t : estado de uma célula num dado tempo t onde $S_i^t \in \Sigma$;
- η_i^t : vizinhança da célula i , ou seja, o estado S_i^t da célula i , junto com os estados das células às quais a célula i está conectada;
- $\Phi(\eta_i)$: regra de transição que fornece o próximo estado S_i^{t+1} para cada célula i em função de η_i^t .

Como mencionado anteriormente, a cada iteração, todas as células do arranjo d -dimensional atualizam seus estados de forma síncrona, segundo a regra $\Phi(\eta_i)$. No entanto, existem modelos

nos quais as células não são atualizadas desta forma. Por exemplo, as células podem ser atualizadas seqüencialmente, de forma que na atualização da segunda célula do arranjo, já se considera a atualização da primeira célula, e assim sucessivamente (SWIECICKA; SEREDYNSKI, 2000). O modelo apresentado por Alonso-Sanz e Martin (2006) utiliza o conceito de memória, onde são considerados os três últimos estados das células que formam uma vizinhança de uma célula i no processo de atualização da mesma.

2.2.1.2 Vizinhança

A vizinhança de uma célula i no tempo t (S_i^t) é determinada em função de um parâmetro denominado raio (r). O tamanho da vizinhança (m) é determinado em função do raio pela fórmula: $m = 2r + 1$. Teoricamente qualquer tamanho de vizinhança pode ser utilizado mas, na prática, tamanhos pequenos facilitam a observação de como o comportamento global do sistema emerge a partir de interações locais (OLIVEIRA, 2003). O número de combinações de vizinhança (ncv) é obtido pela função: k^{2r+1} .

Um AC unidimensional pode ser definido como uma estrutura de dados matricial unidimensional C de tamanho a ($C_{1 \times a}$). Para cada célula c_i se considera as células à esquerda de c_i ($c_{i-1}, c_{i-2}, \dots, c_{i-a}$) e as células à direita de c_i ($c_{i+1}, c_{i+2}, \dots, c_{i+a}$) para formar uma vizinhança. Para raio igual a 1, a vizinhança da célula c_i é formada por um vizinho à esquerda (c_{i-1}) e um vizinho à direita (c_{i+1}) de c_i . Para raio igual a 2, a vizinhança da célula c_i é formada por dois vizinhos à esquerda (c_{i-2} e c_{i-1}) e dois vizinhos à direita (c_{i+2} e c_{i+1}) de c_i (Figura 8), e assim sucessivamente.

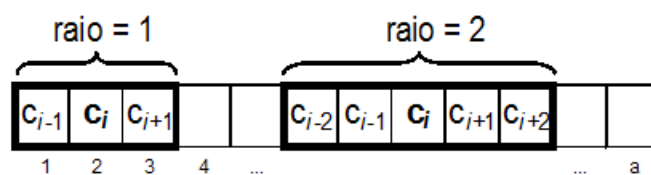


Figura 8: Vizinhanças para um AC unidimensional.

Outras maneiras de se determinar vizinhança são propostas na literatura. Li (1992) apresenta o conceito de vizinhança não-local, onde as células que compõem a vizinhança podem variar a cada atualização, sendo sorteadas a cada passo de tempo. Seredynski (1998) apresenta um modelo onde uma vizinhança diferente é definida para cada célula em função do problema e Jiang, Luo, Szeto e Yang (2001) descrevem um modelo onde a vizinhança é uma área multi-forme definida em função dos centros das células e suas vizinhas.

Para ACs bidimensionais costuma-se utilizar dois padrões de vizinhança denominadas Neumann e Moore (Figura 9). Na vizinhança de Neumann para cada célula $c_{i,j}$ de um arranjo bidimensional C_{axb} consideram-se as células: $c_{i-1,j}, c_{i,j-1}, c_{i,j}, c_{i,j+1}, c_{i+1,j}$ como vizinhança e na de Moore consideram-se as células: $c_{i-1,j-1}, c_{i-1,j}, c_{i-1,j+1}, c_{i,j-1}, c_{i,j}, c_{i,j+1}, c_{i+1,j-1}, c_{i+1,j}, c_{i+1,j+1}$.

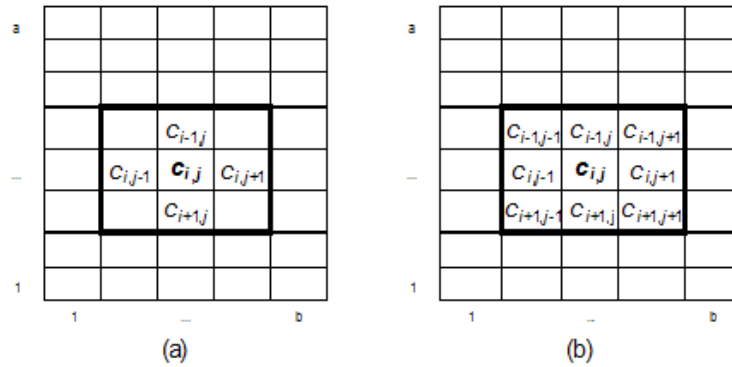


Figura 9: (a) Vizinhança de Neumann. (b) Vizinhança de Moore.

2.2.1.3 Evolução Temporal do Autômato Celular

A Figura 10 apresenta o processo de evolução temporal de um AC unidimensional de raio 1 formado por 12 células ($C_{1 \times 12}$). Destaca-se a evolução da célula c_1 passando da iteração 0 para a iteração 1.

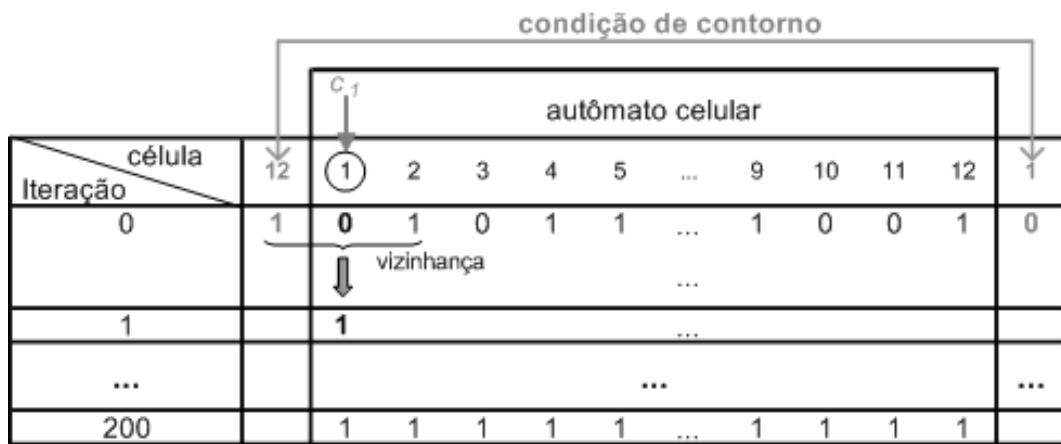


Figura 10: Evolução de um AC unidimensional de raio 1.

As células são aleatoriamente inicializadas na iteração 0 por elementos pertencentes ao conjunto binário Σ ($\Sigma = \{0, 1\}$): $c_1 = 0, c_2 = 1, c_3 = 0, c_4 = 1, c_5 = 1, \dots, c_9 = 1, c_{10} = 0, c_{11} = 0, c_{12} = 1$. Neste exemplo, considera-se $r = 1$, logo $m = 3$. Uma vez que a representação do

AC se dá em uma estrutura matricial linear, a primeira célula (c_1) e a última célula (c_{12}) não possuem vizinhos à esquerda e à direita respectivamente. Necessita-se, então, de uma condição de contorno que basicamente consiste em conectar a célula c_1 e a célula c_{12} , ou seja, transformar a estrutura matricial linear numa estrutura circular. Isto garante que a regra de transição $\Phi(\eta_i)$ possa ser aplicada a todas as células que compõem o AC.

A regra de transição $\Phi(\eta_i)$ é formada por k^{2r+1} possíveis transições. Então, uma regra onde $k = 2$ e $r = 1$ é composta por 8 transições (Tabela 1 - primeira coluna). Cada transição representa uma possível configuração de vizinhança. Como $m = 2r + 1$, cada vizinhança é composta pela combinação de 3 elementos pertencentes ao conjunto Σ (Tabela 1 - coluna central). O número de combinações possíveis sem repetição é exatamente igual ao número de transições. Visualizando cada vizinhança como um número binário, por exemplo 101, pode-se associar diretamente este número binário a seu respectivo valor decimal, neste caso 5. Para cada ocorrência de uma vizinhança existe um elemento de saída que também pertence ao conjunto Σ . Por exemplo, para vizinhança 101 este elemento é 1 (Tabela 1 - última coluna).

Tabela 1: Exemplo de uma regra de transição para $k = 2$ e $r = 1$.

| transição | vizinhança | regra |
|-----------|------------|----------|
| 0 | 000 | 1 |
| 1 | 001 | 0 |
| 2 | 010 | 1 |
| 3 | 011 | 0 |
| 4 | 100 | 1 |
| 5 | 101 | 1 |
| 6 | 110 | 0 |
| 7 | 111 | 1 |

Existem $k^{k^{2r+1}}$ diferentes regras de transição que podem ser aplicadas sobre um AC unidimensional. Assim, para $k = 2$ e $r = 1$, 256 regras podem ser utilizadas. O exemplo da Tabela 1 apresenta a regra 181. Esta representação decimal para a regra vem do conceito de ACs elementares proposto por Wolfram (WOLFRAM, 1983). O número binário 10110101 extraído da concatenação das transições 7, 6, 5, 4, 3, 2, 1 e 0 equivale ao número 181 na base decimal.

2.2.2 Representação do Comportamento Dinâmico

A dinâmica de um AC pode ser ilustrada por diagramas de padrões espaço-temporais, nos quais a configuração de estados em um arranjo d -dimensional é plotada como uma função do tempo. Em geral, as células em estado 0 são representadas pela cor branca e as em estado 1

pela cor preta. A Figura 11 apresenta um diagrama espaço-temporal de um AC de 12 células, evoluído por 20 iterações com a regra elementar 232.

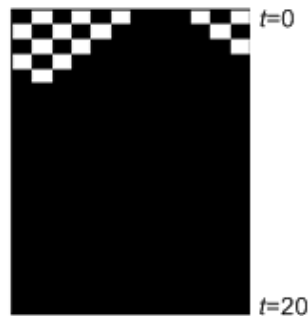


Figura 11: Diagrama de padrões espaço-temporal do AC elementar 232.

Segundo Wuensche e Lesser (1992), este gráfico ou diagrama de padrões espaço-temporais representa o comportamento local do sistema, uma vez que o comportamento global do sistema independe de qualquer estado inicial. O comportamento global pode ser obtido a partir de um conjunto de diagramas de padrões espaço-temporais sobrepostos, cada um construído a partir de um estado inicial diferente.

Algumas métricas estatísticas também podem auxiliar no processo de identificação do comportamento dinâmico do sistema. Dentre estas métricas pode-se citar: informação mútua, entropia e taxa de espalhamento de padrões de diferença (WOLFRAM, 1983b; LI; PACKARD, 1990). A análise do espaço de estados determinado por a^k a partir de grafos cujos nós representam todos os estados sucessivos pelos quais o arranjo pode passar também oferece parâmetros para determinação do comportamento dinâmico do sistema (WUENSCHER; LESSER, 1992; WUENSCHER, 1994, 1999). Em a^k , k representa o número de estados do alfabeto utilizado para codificar o AC e a o número de células do arranjo d -dimensional. Neste contexto, estado representa a configuração de todas as células que compõem o arranjo num determinado instante de tempo. Os sistemas que possuem um número finito de células apresentam um comportamento periódico de curta ou de longa duração. Então, o problema de previsão de comportamento consiste em prever o tamanho do período que caracteriza a repetição de um mesmo estado. É importante considerar que a condição inicial de um AC pode consistir num estado transiente.

Outras abordagens consideram a análise do espaço de regras. O tamanho da regra é normalmente inferior ao tamanho do arranjo e independe do mesmo. Por isto, a maioria dos estudos sobre a dinâmica dos ACs é feita sobre este espaço. A literatura apresenta algumas métricas que podem auxiliar na tarefa de previsão do comportamento dinâmico de um AC, tais como: sensibilidade (BINDER, 1993), domínio da vizinhança, propagação da atividade (λ) e atividade absoluta (OLIVEIRA; OMAR; DE-OLIVEIRA, 2001), parâmetro Z (WUENSCHER,

1999), atividade (LANGTON, 1990), parâmetros de campo médio (LI, 1991) e parâmetro Z de Li (LI, 1991).

Wolfram (1984) propôs um padrão de classificação de comportamento dinâmico. Ele demonstrou que os ACs podem exibir comportamentos extremamente complexos através do estudo de diagramas de padrões espaço-temporais. O comportamento dos ACs pode ser representado por 4 classes qualitativas:

- Classe 1 - A evolução do AC leva a estados homogêneos que, por exemplo, têm células com estado 1 (Figura 12 (a));
- Classe 2 - A evolução do AC leva a um conjunto de estruturas estáveis, periódicas e simples (Figura 12 (b));
- Classe 3 - A evolução do AC leva a padrões caóticos (Figura 12 (c));
- Classe 4 - A evolução do AC leva a estruturas complexas e, algumas vezes, de longa duração (Figura 12 (d)).

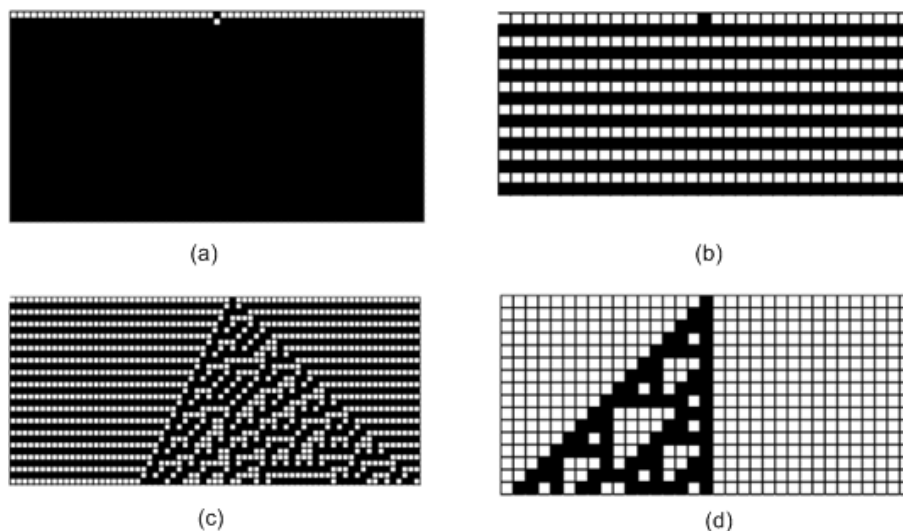


Figura 12: Classificação dos comportamentos dinâmicos segundo Wolfram (Fonte: Wolfram (WOLFRAM, 2002) p. 55-56). O tempo $t = 0$ está representado no topo de cada figura. (a) Classe 1, (b) Classe 2, (c) Classe 3, (d) Classe 4.

Os comportamentos representados pelas classes 1 e 2 são visualmente identificáveis. No entanto, a distinção entre os comportamentos referentes às classes 3 e 4 é difícil, visto que não existe uma fronteira bem definida entre estas classes. Wolfram (1994) propõe uma metodologia para classificação. Esta metodologia consiste da evolução de dois ACs idênticos, exceto pela

modificação de um bit aleatório no segundo AC. A cada iteração calcula-se a distância de Hamming entre as duas configurações. A distância de Hamming quantifica a similaridade entre duas seqüências binárias. Se a distância de Hamming tender a zero no decorrer das iterações o comportamento representado pela classe 1 é identificado. Se a distância de Hamming estabilizar em um valor constante, classe 2. Se a distância de Hamming oscilar sem um padrão definido, classe 3 e se a distância de Hamming variar durante algumas iterações e estabilizar em um valor constante, classe 4.

Li e Packard (1990) definiram uma extensão da classificação de Wolfram. No entanto, propuseram uma divisão mais específica representando as 4 classes de Wolfram em 6 classes:

- Regras nulas: Após t iterações todas as células do AC convergem para estado 0 ou 1;
- Regras ponto fixo: Após t iterações a aplicação de uma determinada regra mantém a configuração do AC constante. As regras nulas poderiam ser consideradas como regras ponto fixo, no entanto, possuem uma classe própria;
- Regras ciclo duplo: Após t iterações a cada duas aplicações da regra retorna-se a uma determinada configuração do AC;
- Regras periódicas: Após t iterações a cada n aplicações da regra retorna-se a uma determinada configuração do AC, com $n \geq 3$;
- Regras caóticas: Produzem dinâmicas não periódicas;
- Regras complexas: Podem produzir dinâmicas periódicas, no entanto, podem ser extremamente longas.

A evolução de um AC pode ser irreversível. Esta característica explica-se pelo fato de que diferentes estados iniciais podem evoluir para estruturas comuns no espaço de configuração dos ACs. Estas estruturas são chamadas de atratores e representam um subconjunto do espaço de configurações dos ACs. Evoluir em direção aos atratores a partir de estados iniciais aleatórios caracteriza uma conduta de auto-organização, onde estruturas podem surgir periodicamente. Uma vez que se conheça todas as configurações iniciais que resultam no mesmo atrator é possível representar seu comportamento “global” por um campo de bacias de atração (WUENSCHE, 1994). No entanto, esta forma de representação é pouco utilizada e difícil de ser obtida. Por exemplo, na construção de um campo de bacias de atração para um AC com 100 células, é necessário analisar 2^{100} diferentes configurações iniciais deste AC para obtenção dos atratores.

2.2.3 Parâmetros de Previsão de Comportamento Dinâmico

Os parâmetros de previsão de comportamento dinâmico (PPCD) fornecem medidas que salientam características inerentes ao comportamento dinâmico apresentado por uma determinada regra. Através do cálculo destes parâmetros supõe-se possível inferir conclusões a respeito desta dinâmica, não sendo necessário simulá-la.

Apresenta-se nesta sessão uma breve descrição dos seguintes PPCD: sensibilidade (BINDER, 1993), domínio da vizinhança, propagação da atividade e atividade absoluta (OLIVEIRA; OMAR; DE-OLIVEIRA, 2001), parâmetro Z (WUENSCHÉ, 1999), atividade (LANGTON, 1990), parâmetros de campo médio (LI, 1991) e parâmetro Z de Li (LI, 1991). A formalização matemática destes parâmetros pode ser encontrada em suas respectivas referências.

A sensibilidade mede o percentual de transições semelhantes que são mapeadas para estados diferentes (BINDER, 1993). Uma transição pode ser composta por u células associadas sequencialmente a uma única saída (s) na forma: $u_1, \dots, u_u \rightarrow s$. Considera-se transições semelhantes a $u_1, \dots, u_u \rightarrow s$ todas as transições que possuem uma única célula deste conjunto com valor diferente ao da transição original. A análise das transições semelhantes consiste em contabilizar todas as transições que não mantêm o valor de s mediante a modificação do valor de uma única célula u em relação à transição original.

O domínio da vizinhança é o parâmetro utilizado para quantificar as mudanças provocadas pela regra de transição do AC no estado da célula central, levando-se em consideração o estado que predomina na vizinhança.

O conceito de propagação da atividade baseia-se nos conceitos de domínio da vizinhança e sensibilidade. Segundo Oliveira, Omar e De-Oliveira (2001) a propagação da atividade é conceituada como sendo a possibilidade de transição de uma célula, seguindo o estado dominante das células vizinhas, e a possibilidade da transição ser sensível a uma mudança mínima de estado de sua vizinhança.

O parâmetro atividade foi definido por Langton (1990). Refere-se a uma estatística realizada sobre as transições de uma regra. Para ACs binários o cálculo deste parâmetro consiste no somatório dos valores de saída de cada uma das transições dividido pelo número de transições que compõe uma regra.

A atividade absoluta baseia-se no parâmetro atividade. Enquanto o parâmetro atividade quantifica as transições que mapeiam a célula central ao estado 1 independentemente dos estados apresentados pelas células da vizinhança, o parâmetro de atividade absoluta verifica as transições que levam a um estado diferente do estado atual da célula central ou de suas vizi-

nhas.

O estado do AC é a configuração do reticulado em um dado instante de tempo. Um estado inicial atribuído ao reticulado no instante t_0 dá origem a uma sucessão de estados nos instantes t_1, t_2, \dots, t_t pela iteração sucessiva da regra do AC. Um estado de um AC possui um único estado sucessor, determinado pela aplicação da regra, mas pode possuir um número arbitrário de predecessores, conhecidos como pré-imagens. O objetivo do parâmetro Z é quantificar a propagação de perturbações em uma regra (WUENSCHÉ; LESSER, 1992; WUENSCHÉ, 1994). Esta perturbação é calculada em função das pré-imagens no espaço de estados. O parâmetro Z indica se a quantidade de pré-imagens possíveis é alta (Z baixo) ou baixa (Z alto), para um reticulado arbitrário.

O parâmetro de campo médio é formado por p componentes, onde $p = m + 1$ e m representa o número de células que compõe a vizinhança. Cada componente é rotulada como componente cm_p e armazena o número de configurações (vizinhanças) que apresentam p ocorrências do estado 1 nas m células, e que mapeiam esta configuração para saída com valor 1.

O parâmetro Z de Li (LI, 1991) implementa o mesmo conceito proposto por Binder (1993) para o parâmetro sensibilidade. Apesar destes parâmetros terem sido obtidos em estudos paralelos ambos quantificam a mesma informação.

2.2.4 O Problema de Indução de Regras

No campo da matemática, o termo indução é utilizado para indicar um raciocínio segundo o qual se estende uma propriedade a todos os termos de um conjunto. Neste contexto, raciocínio é uma operação lógica discursiva e mental, onde são utilizadas uma ou mais proposições, para concluir, através de mecanismos de comparações e abstrações, quais são os dados que levam a respostas verdadeiras, falsas ou prováveis. Assim, conclusões são obtidas a partir de um conjunto de premissas.

Segundo Quinlan (1986), o problema de indução de regras consiste do desenvolvimento de regras de classificação que possam determinar a classe de qualquer objeto através dos valores de seus atributos. Os objetos são descritos em termos de uma coleção de atributos, sendo que cada atributo mede alguma característica importante do objeto. Freitas (2002) apresenta um algoritmo guloso para indução de regras. O algoritmo apresentado na seqüência, em pseudocódigo, é um procedimento genérico para gerar uma regra por indução.

regra induzida $\leftarrow \emptyset$;
enquanto *regra induzida pode ser melhorada pela adiço de nova condiço* **faça**
 escolher melhor condiço;
 regra induzida \leftarrow regra induzida pela melhor condiço;
fim

Neste algoritmo, o processo de induço se inicia com uma regra sem nenhuma condiço. No passo seguinte a melhor condiço (par de atributo-valor) segundo algum critrio de avaliaço,  adicionado  regra. Este laço se repete enquanto a adiço de condiço estiver melhorando a qualidade da regra gerada por induço. Esta abordagem pode encontrar uma regra rapidamente, no entanto, a adiço incremental de uma condiço caracteriza uma busca local e pode levar o algoritmo a uma convergncia prematura, no permitindo encontrar uma soluço que represente o timo global.

Neste trabalho o termo induço de regras  estendido para induço de regras de transiço, que  como so chamadas as regras que atualizam o estados das clulas de um AC ao longo das iteraçes.

O nmero de possveis regras de transiço para um AC unidimensional  dado por $k^{k^{2r+1}}$. Na maioria dos casos torna-se computacionalmente invivel determinar e avaliar todo o conjunto de regras at que se encontre uma regra que represente o comportamento dinmico desejado para o sistema, apesar deste mtodo garantir a soluço tima para o problema. Alm deste mtodo de busca exaustiva e do mtodo de busca gulosa, Martnez (2000) apresenta uma alternativa para o problema de induço de regras. Neste modelo so utilizados os conceitos de regras totalsticas e semitotalsticas. As regras totalsticas agrupam todos os vizinhos de acordo com a soma dos elementos que formam a vizinhança, sem levar em consideraço a ordem em que os valores se encontram. As regras semitotalsticas diferem das totalsticas unicamente pelo fato de que a clula central no faz parte da soma dos elementos. ACs de duas e trs dimenses tmbm podem utilizar estas regras (GONÇALVES, 2004).

Outra abordagem possvel  a utilizaço de tcnicas de inteligncia artificial, mais especificamente computao evolucionria, para encontrar uma regra de transiço que represente satisfatoriamente o comportamento esperado para um determinado sistema. Muitos esforços neste sentido podem ser encontrados na literatura. A Seço 2.5 apresenta a descriço de alguns trabalhos neste sentido.

A maioria dos mtodos de busca relacionados ao problema de induço de regras de transiço buscam encontrar uma nica regra que  aplicada sobre todas as clulas do AC ao longo das iteraçes, mas variaçes podem ser utilizadas. Por exemplo, Seiferas (1982) apresenta um

modelo denominado regra de transição estocástica, onde são associadas probabilidades às possíveis saídas de uma regra de transição. Também, Sipper (1996) apresenta regras de transição não-uniformes, onde diferentes células podem estar associadas a diferentes conjuntos de transições.

2.2.4.1 O Problema de Classificação pela Regra da Maioria

O problema de classificação pela regra da maioria pode ser modelado de diversas maneiras. Neste trabalho utiliza-se a abordagem apresentada por Juillé e Pollack (1998), na qual os estados das células são binários ($k = 2$), a estrutura matricial unidimensional do AC é composta por 149 células ($a = 149$) e o raio da vizinhança é igual a 3 ($r = 3$). Logo, o espaço de regras para este AC é de 2^{128} .

Este problema determina que todas as células alcancem o estado 0 ou estado 1 depois de t passos de tempo dependendo da densidade apresentada pelo AC em sua configuração inicial. Segundo Mitchell, Crutchfield e Hraber (1994), o parâmetro c denota um limiar para a tarefa de classificação. Ou seja, se o número de 1's no AC inicial dividido pelo número total de células do mesmo autômato for maior ou igual a c todas as células deverão estar no estado 1 depois de t passos, senão todas as células deverão estar no estado 0. Neste trabalho será utilizado $c = 0,5$, como foi feito em (JUILLÉ; POLLACK, 1998).

2.3 COMPUTAÇÃO RECONFIGURÁVEL POR *HARDWARE*

Ainda que o funcionamento básico de um AC seja simples, a pesquisa em ACs frequentemente leva à necessidade de muitas simulações computacionais, demandando, conseqüentemente, uma grande capacidade de processamento. Tradicionalmente, algoritmos desenvolvidos em linguagens de programação (*software*) são efetivamente executados em processadores / computadores de uso geral. Mais recentemente, a computação reconfigurável (HAUCK; DEHON, 2008) por *hardware* tem sido utilizada como uma alternativa viável onde se necessita de um grande poder de processamento com alto grau de paralelismo (LOPES; LIMA; MURATA, 2007).

Sistemas reconfiguráveis são sistemas que apresentam a característica de substituir parte de seu *software* por *hardware* para se adaptarem a tarefas específicas. Tais sistemas têm por objetivo obter alto desempenho com baixo custo, sendo uma alternativa à arquitetura de Von Neumann própria dos sistemas com microprocessadores. Nestes sistemas, as tarefas são executadas de forma seqüencial, com tempos de processamento que não são aceitáveis para muitas

aplicações (ITO; CARRO, 2000; COMPTON; HAUCK, 2002). Por outro lado, os sistemas de *hardware* reconfigurável apresentam grande potencial em termos de desempenho e adaptabilidade (PAGE, 1996; MIYAZAKI, 1998; ROSÁRIO; LIMA; FERASOLLI; PEROGARO, 2003; LIMA; ROSÁRIO, 2003). Assim, o uso da computação reconfigurável é particularmente interessante onde algoritmos simples são executados um grande número de vezes.

As FPGAs (*Field Programmable Gate Arrays*) são dispositivos que permitem a execução de algoritmos diretamente em *hardware*. Com estes dispositivos programáveis é possível executar os algoritmos explorando o paralelismo inerente à solução, executando-os mais rapidamente do que se os mesmos fossem executados sequencialmente por microprocessadores com arquitetura de Von Neumann (PAGE, 1996; MIYAZAKI, 1998; COMPTON; HAUCK, 2002; LIMA; ROSÁRIO, 2003).

A síntese dos circuitos lógicos em FPGA é feita através de sistemas CAD (*Computer Aided Design*), permitindo a utilização de diferentes interfaces de projeto. Diferentes módulos podem ser desenvolvidos com linguagens de alto nível de abstração, chamadas linguagens de descrição de *hardware*, a exemplo de VHDL (*VHSIC Hardware Description Language*) (PEDRONI, 2004) e AHDL (*Altera Hardware Description Language*). A ferramenta de *software* Quartus® II desenvolvida pela Altera utiliza a linguagem VHDL e provê um ambiente integrado possibilitando projeto, simulação, teste e documentação dos sistemas. Também é possível implementar blocos com representações gráficas, na forma de esquemáticos. A integração de blocos criados com diferentes linguagens permite a criação de projetos de forma flexível (LIMA; ROSÁRIO, 2003).

O uso de lógica reconfigurável é especialmente adequado a problemas intrinsecamente paralelos, onde os elementos implementados executem procedimentos simples. Esta característica torna a computação reconfigurável particularmente atraente para a simulação maciça de ACs.

2.4 PROCESSAMENTO PARALELO

O processamento paralelo caracteriza o processamento de diversas instruções no mesmo instante de tempo (DONGARRA et al., 2002). Este processamento pode ser realizado através de algoritmos paralelos. Nestes algoritmos diversos processos ou tarefas podem ser executados simultaneamente e podem se comunicar entre si de modo a trocar informações para a solução de um determinado problema (ROOSTA, 1999). O termo processo ou tarefa se refere ao fragmento distinto de código que é executado por cada processador. O conjunto de processadores envolvidos no processamento paralelo é chamado de arquitetura paralela. Este conjunto de

processadores pode estar fisicamente presente num único computador ou estar presente em v computadores heterogêneos ou não, interconectados por uma rede de comunicação que permita a transmissão de mensagens entre os mesmos.

Apenas tarefas independentes podem ser executadas em paralelo. Logo, identificar estas tarefas torna-se fundamental para o projeto de algoritmos paralelos eficientes.

As arquiteturas paralelas representam formas em que elementos computacionais são interligados para permitir a execução de algoritmos paralelos. Dentre as arquiteturas paralelas podem-se citar: o multiprocessamento e o processamento em *cluster*.

Os multiprocessadores são sistemas paralelos que contemplam mais de um processador por computador. Nestes sistemas pode-se identificar tanto diversos processadores fisicamente separados quanto diversos núcleos dentro de um mesmo processador.

O processamento em *cluster* agrupa diversos computadores interligados por meio de uma rede lógica dedicada somente à comunicação entre estas máquinas. Os *clusters*, geralmente, são homogêneos de modo a minimizar os problemas de balanceamento de carga entre os processadores. No, entanto, também é possível conectar diversas máquinas heterogêneas. O fato de interconectar computadores em uma rede dedicada, não faz deste conjunto de computadores um *cluster*. O termo *cluster* está diretamente relacionado à capacidade de se coordenar o processamento em um conjunto de computadores. Esta coordenação é realizada por um *software* específico. Existem diversas bibliotecas, ou modelos de programação que implementam estas soluções, como a MPI e a OpenMP. Este trabalho utiliza a biblioteca MPI (*Message Passing Interface*) (SNIR et al., 1997; GROPP; LUSK; THAKUR, 1999; GROPP; LUSK; SKJELLUM, 1999), na versão MPICH2 (Message Passing Interface Forum, 1998). Esta biblioteca cria um ambiente virtual que une as máquinas e define um conjunto de funções que permite a comunicação entre os computadores que compõem o *cluster*. Esta arquitetura incentiva a criação de um processo mestre que fica responsável por gerenciar todos os outros processos. É importante salientar que a responsabilidade de divisão das tarefas entre os processadores é totalmente do desenvolvedor. A biblioteca não oferece nenhuma funcionalidade neste sentido. O MPI implementa o paradigma de programação intitulado passagem de mensagens (ROOSTA, 1999).

2.5 TRABALHOS CORRELATOS

Esta Seção inicia com a apresentação de alguns trabalhos que abordam a indução de regras para ACs usando técnicas de computação evolucionária para o problema de classificação pela regra da maioria. Mais adiante, também são apresentados trabalhos que discorrem sobre

diversas aplicações de ACs.

O problema de classificação pela regra da maioria é considerado um problema clássico de difícil solução. Um agravante neste problema é o processo de avaliação das regras encontradas pelas diferentes técnicas de computação evolucionária. Normalmente, cada regra é submetida a um conjunto numeroso de configurações iniciais diferentes de um AC. Para cada configuração inicial aplica-se a regra durante t iterações e, então, computa-se a qualidade da regra em questão.

Segundo Andre, Bennett III e Koza (1996) é difícil programar ACs principalmente quando a computação requer comunicação e interação global de informações que estão distantes no espaço celular. Isto reforça a necessidade de uma metodologia mais genérica para o problema de indução de regras em ACs. Os autores propõem uma abordagem baseada em PG padrão com definição automática de funções (KOZA, 1994). O objetivo é encontrar uma função booleana de sete argumentos, onde cada argumento representa uma célula dentro da vizinhança. A função booleana é composta de um conjunto de sub-funções formada da combinação de diversos operadores lógicos. O algoritmo implementado se baseia num modelo paralelo insular. Este modelo é composto de um conjunto de diferentes populações que trocam material genético entre si ao longo das gerações. No total, o modelo é composto por 64 populações com 800 indivíduos cada. Os parâmetros utilizados são: 51 gerações, 3% de taxa de migração entre as populações, 89% de probabilidade de *crossover*, 10% de probabilidade de reprodução e 1% de probabilidade de mutação. A função de *fitness* se baseia na taxa de acerto obtida da evolução de 1000 ACs por 600 iterações. A melhor solução encontrada é uma regra que apresenta uma taxa de acerto de 82,326% obtida na simulação de 10^7 ACs. Este resultado mostrou-se superior aos resultados reportados até então.

Das, Crutchfield e Mitchell (1994) haviam apresentado uma solução baseada em AGs com um desempenho de 76,9%. Soluções obtidas manualmente também estavam disponíveis: Gacs, Kurdyumov e Levin (1978) com um desempenho de 81,6% obtida com a simulação de 10^6 ACs, Davis (1991) com um desempenho de 81,8% também obtida com a simulação de 10^6 ACs e Das, Crutchfield, Mitchell e Hanson (1995) com um desempenho de 82,18% obtida com a simulação de 10^7 .

Juillé e Pollack (1998) apresentam um algoritmo baseado no conceito de aprendizado co-evolucionário entre aprendizes e problemas. Neste trabalho, aprendizado co-evolucionário é definido como um ambiente onde uma população de aprendizes evolui com uma população de problemas que se modificam ao longo das gerações. A idéia central consiste em expor os aprendizes a problemas que estão além daqueles que eles sabem resolver. Uma medida de distância indica o grau de dificuldade de cada um dos problemas. Os autores apresentaram

2 regras (JP1 e JP2) que exibem uma significativa melhora sobre as regras conhecidas até o momento (descritas anteriormente). A regra JP1 apresenta um desempenho de 85,1% e JP2 de 86,0% obtidas na simulação de 400 ACs. Segundo os autores para obtenção dos resultados foram gastos uma semana de processamento em uma *Workstation*. Esta informação destaca o elevado custo computacional envolvido no processo de indução de regras.

Morales, Crutchfield e Mitchell (2001) apresentam uma abordagem baseada em AGs. Os autores trabalham com uma outra instância do problema de classificação pela regra da maioria, onde são considerados AC bidimensionais (21 x 21). Nesta abordagem as soluções candidatas (indivíduos) são codificadas em uma *string* de *bits*. Uma população de *strings* é gerada aleatoriamente e submetida ao processo de evolução durante muitas gerações. A função de *fitness* se baseia na taxa de acerto obtida da evolução de 100 ACs. As melhores regras encontradas apresentam um desempenho de aproximadamente 90%.

Ferreira (2002b) propõe a utilização de PEG. O algoritmo é executado diversas vezes. A cada execução armazenam-se as melhores soluções, que são então, utilizadas como integrantes de parte da população inicial na execução seguinte. A função de *fitness* se baseia na taxa de acerto obtida da evolução de 128 ACs (casos de *fitness*). Se o número de casos de *fitness* cobertos corretamente pela regra for superior a 64, atribui-se este número ao valor de *fitness*, caso contrário, atribui-se 1. O trabalho está utiliza as regras JP1 e JP2 de Juillé e Pollack (1998). Segundo a autora não se conheciam as expressões booleanas referentes a estas regras e pouco ou nenhum conhecimento poderia ser extraído a partir dos *bits* de saída apresentados nas tabelas de transição de cada uma das regras. Então, a autora utilizou o referido método de computação evolucionária para encontrar duas expressões booleanas equivalentes às regras JP1 e JP2. Este trabalho sugere a utilidade de PEG para o problema de indução de regras de ACs.

A pesquisadora Prof. Dr.^a Gina Maira Barbosa de Oliveira juntamente com seus colaboradores vem desenvolvendo pesquisas importantes envolvendo diferentes técnicas de computação evolucionária para o problema de classificação pela regra da maioria. A seguir listam-se alguns destes trabalhos:

- *Guidelines for dynamics-based parameterization of one-dimensional cellular automata rule spaces* (OLIVEIRA; DE-OLIVEIRA; OMAR, 2000a);
- *Evolving solutions of the density classification task in 1D cellular automata, guided by parameters that estimate their dynamic behavior* (OLIVEIRA; DE-OLIVEIRA; OMAR, 2000b);
- *Definition and application of a five-parameter characterization of one-dimensional cel-*

lular automata rule space (OLIVEIRA; OMAR; DE-OLIVEIRA, 2001);

- *Improving genetic search for cellular automata, using heuristics related to their dynamical behavior* (OLIVEIRA; DE-OLIVEIRA; OMAR, 2001);
- *Dynamic behaviour forecast as a driving force in the coevolution of one-dimensional cellular automata* (OLIVEIRA; ASAKURA; DE-OLIVEIRA, 2002);
- *Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task* (OLIVEIRA; BORTOT; DE-OLIVEIRA, 2002);
- *Coevolutionary search for one-dimensional cellular automata, based on parameters related to their dynamic behaviour* (OLIVEIRA; ASAKURA; DE-OLIVEIRA, 2003);
- *Multiobjective, heuristic evolutionary search in a cooperative environment leads to the best cellular automaton rule in the density classification task* (BORTOT; DE-OLIVEIRA; OLIVEIRA, 2004);
- *Using dynamic behavior prediction to guide an evolutionary search for designing two-dimensional cellular automata* (OLIVEIRA; SIQUEIRA, 2005);
- *Parameter characterization of two-dimensional cellular automata rule space* (OLIVEIRA; SIQUEIRA, 2006);
- *The best currently know class of dynamically equivalent cellular automata rules for density classification* (DE-OLIVEIRA; BORTOT; OLIVEIRA, 2006);
- *Heuristic search for cellular automata density classifiers with a multiobjective evolutionary algorithm* (OLIVEIRA; BORTOT; DE-OLIVEIRA, 2007);

Os trabalhos apresentam soluções para diferentes configurações de AC segundo o problema de classificação pela regra da maioria. A cada novo trabalho, diferentes características são exploradas. Abordagens multiobjetivo, em conjunto com técnicas de co-evolução e heurísticas baseadas em parâmetros de previsão de comportamento dinâmico são utilizadas.

Por exemplo, Oliveira, Bortot e De-Oliveira (2002) apresentam um algoritmo evolucionário multiobjetivo onde são consideradas informações obtidas a partir dos parâmetros de previsão de comportamento dinâmico: sensibilidade, atividade absoluta, domínio da vizinhança e propagação da atividade nos operadores de *crossover* e mutação. Neste trabalho as melhores regras apresentaram um desempenho de 78,94% quando foram utilizados 100 indivíduos na população e 81,16% para 200 indivíduos.

Em Bortot, De-Oliveira e Oliveira (2004) são apresentadas 5 regras para o problema de classificação pela regra da maioria segundo as configurações descritas na Seção 2.2.4.1. As taxas de acerto apresentam-se superiores a 85%.

Em Oliveira e Siqueira (2006) apresenta-se uma generalização para os PPCD: sensibilidade, domínio da vizinhança e propagação da atividade, a fim de aplicá-los no problema de classificação pela maioria em ACs bidimensionais. Um AG foi utilizado na tarefa de indução de regras. Os ACs foram organizados em um arranjo de 21 x 21 células e a vizinhança de Moore foi empregada. O melhor resultado alcançado se deu com a associação dos parâmetros domínio da vizinhança e propagação da atividade ao processo de simulação da dinâmica. A melhor regra retornou um desempenho de 70,62% obtido a partir da simulação de 10^4 ACs criados aleatoriamente com distribuição uniforme.

Recentemente, Oliveira, Bortot e De-Oliveira (2007) apresentam um algoritmo evolucionário multiobjetivo que encontrou 18 regras para o problema de classificação pela regra da maioria (Seção 2.2.4.1). As taxas de acerto apresentam-se entre 77,3% e 82,1%.

O cálculo da função de *fitness* pode consumir um elevado tempo de processamento no problema de classificação pela regra da maioria. Recentemente foi utilizado um *hardware* dedicado no cálculo do *fitness*, como uma alternativa de redução do tempo computacional (WEINERT; BENITEZ; LOPES; LIMA, 2007). Nesta implementação houve uma melhora significativa em relação ao custo computacional. No entanto, a metodologia utilizada se restringe ao problema de classificação pela regra da maioria e é difícil adaptá-la para outros problemas de indução de regras de ACs. Estas restrições motivam o desenvolvimento de métodos mais eficazes de avaliação do comportamento dinâmico apresentado pelos ACs.

A PEG é uma técnica relativamente recente e pouco explorada se comparada a suas precursoras: AGs e PG. Na tentativa de minizar o tempo de processamento e acelerar a convergência do algoritmo da PEG para boas soluções podem-se citar os seguintes trabalhos: Du, Ding e Jia (2008) introduziu a idéia de co-evolução de multi-populações. Neste trabalho os autores apresentam um método paralelo, distribuído e assíncrono baseado num modelo insular implementado no ambiente MPI. Neste modelo cada ilha executa uma instância do algoritmo da PEG, sendo as ilhas completamente interconectadas para permitir migração dos indivíduos entre as mesmas. Park, Grings, Santos e Soares (2008) propuseram uma arquitetura cliente-servidor, onde os clientes executam instâncias do GEP e o servidor executa um AG que tem por finalidade encontrar os melhores parâmetros para os PEGs executados nos clientes. Uma política de migração rege a troca de material genético entre os clientes.

Pode-se perceber que os trabalhos de Du, Ding e Jia (2008) e Park, Grings, Santos e Soares

(2008) visam acelerar a convergência do algoritmo, no entanto, se o custo de computação da função de *fitness* for elevado o tempo total de processamento também o será.

Um trabalho interessante é apresentado por Spears e Gordon (2000), embora este trabalho não esteja ligado diretamente ao problema de indução de regras de autômatos celulares. Basicamente, o trabalho apresenta um algoritmo evolucionário para criar máquinas de estado finito. No entanto, esta idéia pode ser adaptada ao problema de indução de regras. Pois, uma transição pode ser visualizada como uma máquina de estados finitos. Isto permitiria considerar o comportamento dinâmico das regras segundo a classificação de Wolfram (1984) no processo de avaliação das mesmas.

Os ACs vêm sendo utilizados na resolução de diversos problemas, tais como: criptografia (WOLFRAM, 1986; TOMASSINI; PERRENOUD, 2001; BENKINIOUAR; BENMOHAMED, 2004; OLIVEIRA; COELHO; MONTEIRO, 2004), compressão de textos (KHAN; CHOUDHURY; DIHIDAR; VERMA, 1999) escalonamento de tarefas (SWIECICKA; SEREDYNSKI, 2000; VIDICA; OLIVEIRA, 2006), propagação de epidemias (FU; MILNE, 2003), sincronização de tarefas (DAS; CRUTCHFIELD; MITCHELL; HANSON, 1995), processamento de imagens (ROSIN, 2005), classificação (ANDRE; BENNETT III; KOZA, 1996), jogos (AHAMED; ELGAZZAR, 2001), análise de séries temporais (RIGO; BARBOSA, 2006), previsão de comportamento de bolsa de valores (WEI; YING; FAN; WANG, 2003), estudo de atividades de terremotos (GEORGOUDAS; SIRAKOULIS; ANDREADIS, 2007), modelos presa-predador (CHEN; MYNETT, 2003) e outros sistemas complexos reais (LI, 1989; GUTOWITZ, 1995; CÂMARA; FERREIRA; CASTRO, 1996; BAR-YAM, 1997; BATTY, 2000).

Uma aplicação bastante explorada pela teoria de ACs é a simulação de sistemas biológicos, como por exemplo: simulação de infecção celular causada pelo vírus HIV (CORNE; FRISCO, 2008) e a predição de regiões transmembrânicas em proteínas (DIAO et al., 2007).

Os ACs permitem que sejam realizados acompanhamentos dinâmicos da evolução dos processos por eles modelados. Esta característica é particularmente interessante para simulação de sistemas biológicos. Os sistemas biológicos são complexos não apenas estruturalmente, mas também funcionalmente. Modelar estas estruturas é uma tarefa difícil. Os modelos gerados a partir de ACs permitem a compreensão do comportamento coletivo de diversos elementos simples com propriedades semelhantes que interagem em um espaço de possibilidades por meio de diferentes comportamentos dinâmicos. Esta definição pode ser estendida para sistemas vivos onde células semelhantes não possuem função quando estão isoladas. Porém, se estas atuam em conjunto, passam a ter função e emerge um sistema, cujo comportamento é complexo. A complexidade de um sistema relaciona-se com seu grau de estabilidade mediante perturbações,

de modo que a previsibilidade de seus comportamentos futuros torna-se muito difícil. Na sequência apresentam-se alguns trabalhos aonde são utilizados ACs para simulação de sistemas biológicos.

Kiera, Chenga, Testab e Carruptb (1996) modelaram reações enzimáticas com substrato em água através de ACs. As reações enzimáticas são simuladas como fenômenos dinâmicos. Esta abordagem apontou novas oportunidades para o estudo de um modelo genérico das reações enzimáticas aliado ao seu comportamento, que inclui a interação de ingredientes com o solvente.

Kansal et al. (2000) apresentaram um modelo baseado em ACs capaz de simular o crescimento de tumores cerebrais. Este modelo é capaz de simular o crescimento do tumor de Gompertzian. A composição prevista e as taxas de crescimento estão de acordo com um caso de teste obtido a partir da literatura médica disponível na época. O processo de simulação permitiu identificar uma importante propriedade dentro do processo de crescimento do tumor conhecida como competição clonal.

Laurio, Linaker e Narayanana (2002) mostraram como criar, sistematicamente, ACs para reconhecer padrões regulares tais como aqueles encontrados no PROSITE (HOFMAN; BUCHER; FALQUET; BAIROCH, 1999). O PROSITE é um banco de dados que armazena informações sobre proteínas. Estas informações contemplam: funcionalidade, classificação, padrões, entre outras. A partir de um padrão regular obtido no PROSITE determina-se uma regra utilizando conceitos de gramáticas regulares que é posteriormente convertida em uma regra de transição para o AC. Aplica-se a regra sobre o AC que representa a codificação de uma proteína, até que configuração do AC no tempo $t + 1$ seja igual à configuração no t . Então, computam-se os *matches* parciais e totais encontrados. Este trabalho também utiliza técnicas de paralelismo.

Características espaciais como a localização de células mortas podem afetar o espalhamento de infecções. Beauchemina, Samuelb e Tuszynskia (2005) utilizaram um modelo baseado num AC bidimensional para investigar a influência de heterogeneidades espaciais no espalhamento de infecções virais. Inicialmente, testou-se o modelo com dados clínicos imunológicos para influenza A.

Gangadhar (2005) apresenta uma técnica, intitulada Pelican, para realizar o alinhamento de sequências utilizando um modelo de AC bidimensional. Segundo o autor, Pelican é um caminho rápido para alinhar os *backbones* de duas estruturas de proteínas. O Pelican divide as estruturas das proteínas em matrizes de distância compostas por 5 aminoácidos. Então, utiliza a diferença destas matrizes para construir um AC bidimensional. A partir de um estado inicial desalinhado, o AC evolui por várias iterações de acordo com um conjunto de regras definidas gerando padrões emergentes que contribuem para o alinhamento das sequências.

Xiao et al. (2005) apresentam uma abordagem para visualização de sequências biológicas baseada em ACs. Este trabalho transforma uma sequência de aminoácidos numa codificação digital e utiliza uma regra de transição sobre esta codificação gerando uma imagem (diagrama espaço-temporal). Segundo os autores esta abordagem permite que características que estão originalmente escondidas ao longo da cadeia de aminoácidos possam ser claramente reveladas. A análise de diversas sequências pode permitir a criação de uma impressão digital para este conjunto de sequências. Neste problema o termo impressão digital pode ser substituído por *motifs*. Os *motifs* são regiões de tamanhos variados (formados pela mesma sequência de aminoácidos) que se repetem dentro da estrutura de uma proteína.

Xiao et al. (2006) também utilizam esta técnica de visualização de sequências biológicas baseada em ACs para predição da localização de uma proteína dentro da célula. Um algoritmo de detecção e classificação automática de imagens é aplicado sobre a imagem obtida pela técnica de visualização de sequências biológicas e classifica esta imagem como sendo pertencente a uma entre 12 classes que representam as subdivisões dentro da célula.

Malleta e Pillisb (2006) apresentaram um AC híbrido baseado em equações diferenciais de complexidade moderada para descrever, através de simulações dinâmicas, as interações entre um tumor que cresce próximo a uma fonte de nutrientes e o sistema imune do organismo hospedeiro.

Mizas et al. (2008) aplicaram AGs e ACs para reconstrução da evolução de sequências de DNA. Este trabalho é particularmente interessante devido a sua complexidade e contribuições para as ciências biológicas. Desenvolveu-se uma ferramenta de *software* que permite modelar uma sequência de DNA como um AC unidimensional, cujo alfabeto é capaz de representar as quatro bases do DNA (A, C, T e G). Um AG determina a regra de transição do AC capaz de representar o comportamento dinâmico desejado. Inicialmente, duas sequências de DNA são previamente identificadas em diferentes momentos da evolução. Então, o algoritmo encontra uma regra capaz de explicar as mudanças evolutivas ocorridas entre as duas sequências. Este sistema pode ser particularmente interessante para a identificação de mutações que ocorrem em todo o código genético em períodos distintos da evolução.

A vasta aplicabilidade dos ACs justifica o desenvolvimento de metodologias genéricas e adaptáveis a diferentes problemas que, em essência, são problemas de indução de regras de transição capazes de simular um determinado comportamento dinâmico.

Como o processo de simulação dos sistemas reflete o comportamento dinâmico dos ACs empregados, e normalmente são de elevado custo computacional, o estudo de mecanismos de previsibilidade de comportamentos futuros, como os realizados neste trabalho, são de grande

aplicabilidade para teoria de sistemas complexos.

3 INDUÇÃO DE REGRAS UTILIZANDO PARÂMETROS DE PREVISÃO DE COMPORTAMENTO DINÂMICO

3.1 INTRODUÇÃO

Este capítulo apresenta e analisa a eficiência dos parâmetros de previsão de comportamento dinâmico (PPCD) no problema de indução de regras para ACs unidimensionais. A proposta dos PPCD (Seção 2.2.3) é, a partir da regra de transição de estado, gerar informação a respeito do comportamento dinâmico do sistema.

O problema selecionado para este trabalho é o problema de classificação pela regra da maioria descrito na Seção 2.2.4.1. A escolha deste problema é devida à disponibilidade de um grande número de publicações que apresentam diversas técnicas computacionais para o problema e uma variedade de soluções satisfatórias (ANDRE; BENNETT III; KOZA, 1996; JUILLÉ; POLLACK, 1998; MORALES; CRUTCHFIELD; MITCHELL, 2001; OLIVEIRA; OMAR; DE-OLIVEIRA, 2001; FERREIRA, 2002b; OLIVEIRA; ASAKURA; DE-OLIVEIRA, 2002; OLIVEIRA; BORTOT; DE-OLIVEIRA, 2002, 2007).

Outra característica interessante deste problema é que as regras consideradas como soluções para o problema apresentam um comportamento classificado como nulo segundo a classificação de Li e Packard (1990).

Assim, o objetivo deste capítulo é propor um método computacional evolucionário para indução de regras de transição de ACs capazes de apresentar comportamento nulo para o problema de classificação pela regra da maioria. A idéia central é encontrar um conjunto de PPCD ou combinações dos mesmos que possa gerar resultados, no mínimo iguais aos já publicados, para o processo de indução de regras.

3.2 METODOLOGIA

O desenvolvimento desta metodologia divide-se em três etapas. Na primeira realiza-se uma análise de correlação entre os PPCD. A partir dos resultados obtidos define-se um subconjunto

de PPCD que são empregados na função de *fitness* do algoritmo evolucionário proposto.

3.2.1 Análise de Correlação dos PPCD

O espaço de regras do problema de classificação pela regra da maioria (Seção 2.2.4.1) é composto de 2^{128} regras, o que impossibilita o cálculo dos PPCD para todas as regras. Então, após uma busca pela literatura, 31 regras foram selecionadas para o processo de análise de correlação. A Tabela 2 apresenta estas regras em formato hexadecimal, seus autores e suas medidas de eficiência. A eficiência é calculada em função da simulação da evolução de 10^4 ACs aleatórios para cada regra, por 200 iterações. Duas situações são consideradas como acerto. Na primeira, se a densidade na configuração inicial do AC é menor do que 0,5 e, após 200 iterações, todas as células do arranjo convergem para valor igual a 0 e na segunda, a densidade na configuração inicial do AC é maior ou igual a 0,5 e, após 200 iterações, todas as células do arranjo convergem para valor igual a 1. A eficiência é obtida em função do número de ACs gerados aleatoriamente que atendem a estas duas condições, dividido por 10^4 .

A Tabela 3 apresenta os valores calculados dos PPCD: sensibilidade (S), domínio da vizinhança (DV), propagação da atividade (PA), atividade absoluta (AA), Z de Wuensche (Z_W), atividade (A), campo médio (CM) e Z de Li (Z_Li) para as regras apresentadas na Tabela 2. Também são apresentados o valor: médio ($Sm, DVm, PAm, AAm, Z_Wm, Am, CMm, Z_Lim$), desvio-padrão (DP), máximo e mínimo destes parâmetros, além da distância Euclidiana (DE) entre os valores de $S, DV, PA, AA, Z_W, A, CM, Z_Li$ e $Sm, DVm, PAm, AAm, Z_Wm, Am, CMm$ e Z_Lim .

A DE é definida pela Equação 4 e funciona como um indicativo de similaridade entre os valores dos parâmetros individuais de cada regra e os valores médios. Nesta formalização são consideradas 8 dimensões, uma para cada parâmetro. Quanto menor for o valor da DE mais similares serão o desempenho das regras sendo comparadas.

$$DE = \sqrt{\sum_{i=1}^8 (P_i - Pm_i)^2} \quad (4)$$

onde:

$$P_i = \{S, DV, PA, AA, Z_W, A, CM, Z_Li\};$$

O conjunto Pm_i representa os valores médios dos parâmetros pertencentes a P_i conforme Tabela 3.

Tabela 2: Regras de referência.

| Regra | Hexadecimal | Referência | Eficiência |
|-------|-----------------------------------|--|------------|
| GKL | 005F005F005F005F005FFF5F005FFF5F | Gacs, Kurdyumov e Levin (1978) | 81,60% |
| MHC | 0504058705000F77037755837BFFB77F | Mitchell, Hraber e Crutchfield (1993) | 76,90% |
| DAV | 002F035F001FCF1F002FFC5F001FFF1F | Davis (1991) | 81,80% |
| DAS | 070007FF0F000FFF0F0007FF0F310FFF | Das, Crutchfield, Mitchell e Hanson (1995) | 82,20% |
| ABK | 050055050500550555FF55FF55FF55FF | Andre, Bennett III e Koza (1996) | 82,30% |
| CRA | 00550055005500571F55FF57FF55FF57 | Cranny e Bossomaier (1999) | 82,50% |
| JP1 | 011430D7110F395705B4FF17F13DF957 | Juillé e Pollack (1998) | 85,10% |
| JP2 | 1451305C0050CE5F1711FF5F0F53CF5F | | 86,00% |
| BOO1 | 145500CC0F14021F1715FFCF0F17FF1F | Bortot, De-Oliveira e Oliveira (2004) | 86,16% |
| BOO2 | 070017070C0057DF07BFD707CCFF559F | | 85,97% |
| BOO3 | 070017070C0057D707BFD707CDDFF55D7 | | 85,39% |
| BOO4 | 002F131F010FF91F00ECFF1F013DF91F | | 85,38% |
| BOO5 | 1071307C0000286F17313F7FF33F2B7F | | 85,35% |
| R1 | 015E0012005500571F5FFFFF0F55CF5F | | 82,70% |
| R2 | 10111000531512531F15FF5FDF5DDF5F | | 82,60% |
| R3 | 00010355015511571F150F77FFF5FF57 | | 81,50% |
| R4 | 070447470700460705774757F777FF77 | | 82,70% |
| R5 | 015400550050045F055FFFDF5557FF5F | | 81,90% |
| R6 | 0445004C37770E3F044500CDF7773FFF | | 81,50% |
| R7 | 15005000350077071553775FF5F77F7F | | 81,70% |
| R8 | 000104171DDF555704DF441FDDDDFD557 | Oliveira, | 81,60% |
| R9 | 01000030011311370DFFBFFBDDFF11FF | Bortot e | 81,50% |
| R10 | 0001090703030B031F1F6F37FF776F77 | De-Oliveira | 79,20% |
| R11 | 0100050D1D9D155F05FD555FDDFF5557 | (2007) | 78,90% |
| R12 | 000103021111011317F5FFFDDFF11FF | | 78,70% |
| R13 | 0001017D2113C35F4B15DF75275B9FD7 | | 78,80% |
| R14 | 015500400054563F1057BF0FB7FFFB7F | | 77,80% |
| R15 | 0071023C00224D170379B53747BFFF7F | | 77,30% |
| R16 | 10041383005313DD3357CFED875F1FDF | | 78,70% |
| R17 | 040305502F06457D05013757D5F7FF7F | | 78,20% |
| R18 | 050470000006516D053FF5FF977FE77F | | 77,30% |

Pela análise da Tabela 3 é possível identificar uma relação direta entre os parâmetros S e Z_{Li} e os parâmetros A e CM . Os valores de S permanecem idênticos aos valores de Z_{Li} , bem como os valores de A aos de CM para todas as regras analisadas. Esta relação entre os parâmetros permite que os parâmetros Z_{Li} e CM sejam excluídos das análises futuras realizadas neste trabalho. Optou-se por excluir Z_{Li} e CM pelo fato de ambos apresentarem um nível de

Tabela 3: Cálculo dos parâmetros de previsão de comportamento dinâmico de ACs.

| Regra | <i>S</i> | <i>DV</i> | <i>PA</i> | <i>AA</i> | <i>Z_W</i> | <i>A</i> | <i>CM</i> | <i>Z_Li</i> | <i>DE</i> |
|-------|----------|-----------|-----------|-----------|------------|----------|-----------|-------------|-----------|
| GKL | 0,23 | 0,91 | 0,07 | 0,10 | 0,25 | 0,50 | 0,50 | 0,23 | 0,27 |
| MHC | 0,37 | 0,91 | 0,08 | 0,18 | 0,54 | 0,49 | 0,49 | 0,37 | 0,11 |
| DAV | 0,30 | 0,88 | 0,09 | 0,16 | 0,24 | 0,50 | 0,50 | 0,30 | 0,23 |
| DAS | 0,25 | 0,87 | 0,10 | 0,22 | 0,38 | 0,50 | 0,50 | 0,25 | 0,13 |
| ABK | 0,23 | 0,88 | 0,09 | 0,20 | 0,50 | 0,50 | 0,50 | 0,23 | 0,13 |
| CRA | 0,25 | 0,88 | 0,09 | 0,21 | 0,47 | 0,50 | 0,50 | 0,25 | 0,10 |
| JP1 | 0,40 | 0,85 | 0,11 | 0,25 | 0,48 | 0,51 | 0,51 | 0,40 | 0,13 |
| JP2 | 0,33 | 0,84 | 0,11 | 0,26 | 0,48 | 0,50 | 0,50 | 0,33 | 0,08 |
| BOO1 | 0,33 | 0,84 | 0,11 | 0,26 | 0,48 | 0,50 | 0,50 | 0,33 | 0,08 |
| BOO2 | 0,34 | 0,84 | 0,10 | 0,26 | 0,49 | 0,51 | 0,51 | 0,34 | 0,08 |
| BOO3 | 0,33 | 0,85 | 0,10 | 0,26 | 0,47 | 0,51 | 0,51 | 0,33 | 0,07 |
| BOO4 | 0,34 | 0,85 | 0,12 | 0,22 | 0,34 | 0,50 | 0,50 | 0,34 | 0,14 |
| BOO5 | 0,35 | 0,85 | 0,11 | 0,22 | 0,41 | 0,49 | 0,49 | 0,35 | 0,08 |
| R1 | 0,29 | 0,87 | 0,11 | 0,19 | 0,50 | 0,50 | 0,50 | 0,29 | 0,06 |
| R2 | 0,30 | 0,87 | 0,10 | 0,26 | 0,49 | 0,50 | 0,50 | 0,30 | 0,07 |
| R3 | 0,30 | 0,92 | 0,06 | 0,17 | 0,46 | 0,50 | 0,50 | 0,30 | 0,07 |
| R4 | 0,30 | 0,88 | 0,09 | 0,24 | 0,39 | 0,51 | 0,51 | 0,30 | 0,09 |
| R5 | 0,28 | 0,91 | 0,08 | 0,15 | 0,45 | 0,49 | 0,49 | 0,28 | 0,08 |
| R6 | 0,30 | 0,88 | 0,09 | 0,18 | 0,52 | 0,49 | 0,49 | 0,30 | 0,06 |
| R7 | 0,30 | 0,88 | 0,10 | 0,26 | 0,46 | 0,50 | 0,50 | 0,30 | 0,07 |
| R8 | 0,30 | 0,88 | 0,10 | 0,17 | 0,44 | 0,50 | 0,50 | 0,30 | 0,05 |
| R9 | 0,30 | 0,89 | 0,08 | 0,16 | 0,56 | 0,49 | 0,49 | 0,30 | 0,11 |
| R10 | 0,30 | 0,90 | 0,08 | 0,21 | 0,50 | 0,48 | 0,48 | 0,30 | 0,05 |
| R11 | 0,30 | 0,90 | 0,07 | 0,17 | 0,50 | 0,52 | 0,52 | 0,30 | 0,07 |
| R12 | 0,29 | 0,89 | 0,08 | 0,18 | 0,61 | 0,48 | 0,48 | 0,29 | 0,15 |
| R13 | 0,40 | 0,89 | 0,07 | 0,19 | 0,48 | 0,49 | 0,49 | 0,40 | 0,12 |
| R14 | 0,36 | 0,89 | 0,08 | 0,17 | 0,55 | 0,50 | 0,50 | 0,36 | 0,11 |
| R15 | 0,40 | 0,91 | 0,10 | 0,17 | 0,51 | 0,48 | 0,48 | 0,40 | 0,13 |
| R16 | 0,38 | 0,88 | 0,06 | 0,18 | 0,48 | 0,50 | 0,50 | 0,38 | 0,10 |
| R17 | 0,36 | 0,90 | 0,06 | 0,20 | 0,47 | 0,49 | 0,49 | 0,36 | 0,07 |
| R18 | 0,36 | 0,87 | 0,08 | 0,20 | 0,55 | 0,49 | 0,49 | 0,36 | 0,10 |

| | <i>Sm</i> | <i>DVm</i> | <i>PAm</i> | <i>AAm</i> | <i>Z_Wm</i> | <i>Am</i> | <i>CMm</i> | <i>Z_Lim</i> |
|--------|-----------|------------|------------|------------|-------------|-----------|------------|--------------|
| Média | 0,32 | 0,88 | 0,09 | 0,20 | 0,47 | 0,50 | 0,50 | 0,32 |
| DP | 0,05 | 0,02 | 0,02 | 0,04 | 0,08 | 0,01 | 0,01 | 0,05 |
| Máximo | 0,40 | 0,92 | 0,12 | 0,26 | 0,61 | 0,52 | 0,52 | 0,40 |
| Mínimo | 0,23 | 0,84 | 0,06 | 0,10 | 0,24 | 0,48 | 0,48 | 0,23 |

complexidade de cálculo superior aos parâmetros *S* e *A*, respectivamente.

Observando-se os valores médios, desvio-padrão, máximos e mínimos de cada parâmetro e a distância Euclidiana entre os valores de uma regra e os valores médios (Tabela 3), conclui-se que as regras apresentam valores de parâmetros bastante próximos, com exceção do parâmetro

Z_W que apresenta uma variação superior aos demais.

A Tabela 4 apresenta uma matriz de correlação de Pearson gerada a partir dos valores dos PPCD: S , DV , PA , AA , Z_W e A . Esta matriz permite identificar possíveis relações entre estes parâmetros e se baseia no cálculo do coeficiente de correlação de Pearson (r) (AHLGREN; JARNEVING; ROUSSEAU, 2003).

Segundo os resultados apresentados pela Tabela 4 não foi possível identificar nenhum “alto” nível de correlação entre os parâmetros: S , DV , PA , AA , Z_W e A . Vincent (2005) reputa a valores maiores de 0,9 uma definição de alta correlação. Isto sugere a hipótese de que todos estes parâmetros são importantes para definição de características que representem o comportamento dinâmico de um sistema.

Tabela 4: Matriz de correlação dos PPCD.

| | S | DV | PA | AA | Z_W | A |
|--------|------|-------|-------|-------|--------|-------|
| S | 1,00 | -0,16 | 0,02 | 0,21 | 0,30 | -0,14 |
| DV | | 1,00 | -0,75 | -0,76 | 0,06 | -0,39 |
| PA | | | 1,00 | 0,60 | -0,14 | 0,18 |
| AA | | | | 1,00 | 0,15 | 0,32 |
| Z_W | | | | | 1,00 | -0,30 |
| A | | | | | | 1,00 |

3.2.2 Indução de Regras de Transição

Nesta seção aborda-se as seguintes questões: Qual a real eficácia dos PPCD na tarefa de diferenciar comportamentos dinâmicos? É possível que regras com o mesmo valor para determinados parâmetros apresentem comportamentos diferentes?

O objetivo deste questionamento é avaliar se os PPCD podem ser eficientemente utilizados no processo de geração de regras e, assim, analisar se um conhecimento prévio a respeito do comportamento dinâmico desejado pode ser eficientemente utilizado no processo.

Para responder ao questionamento levantado desenvolveu-se um sistema de geração de regras com intuito de encontrar regras que apresentem comportamentos nulo.

3.2.2.1 Implementação

O sistema desenvolvido é fruto de uma adaptação realizada sobre o sistema GEPCLASS (WEINERT; LOPES, 2006). Este sistema foi implementado em linguagem C++ sobre a plataforma de desenvolvimento Borland® C++Builder® 6.0. Baseia-se no algoritmo evolucionário

conhecido como Programação de Expressão Genética (FERREIRA, 2001), e foi modificado para ser aplicado sobre o problema de classificação pela regra da maioria. Dentre as diversas instâncias para este problema adota-se a utilizada por Juillé e Pollack (1998), onde é considerado um AC unidimensional com 149 células e raio igual a 3.

Os parâmetros utilizados para configurar o GEPCLASS são:

- indivíduos: 50;
- gerações: 50;
- função de ligação: *AND*;
- conjunto de funções: *AND*, *OR*, *NOT*, *=* e *≠*;
- conjunto de terminais: *a*, *b*, *c*, *d*, *e*, *f*, *g*;
- número de genes por cromossomo: 3;
- tamanho da cabeça dos genes: 6-15;
- método de seleção: torneio estocástico;
- operadores genéticos: os mesmos definidos em Ferreira (2001) com as mesmas probabilidades.

Os terminais (*a*, *b*, *c*, *d*, *e*, *f*, *g*) podem assumir valor 0 ou 1 e representam todas as possíveis combinações de vizinhança do problema, mapeando, assim, todas as transições de uma determinada regra. O número de combinações é obtido em função da fórmula: k^{2r+1} , onde k representa o número de estados do alfabeto, no caso, 2 (valor 0 e valor 1), e r o raio, no caso, 3. Assim, o número de combinações ou de transições é 128. A definição destas 128 combinações binárias, cada uma contendo 7 bits (um bit para cada terminal), pode ser facilmente gerada a partir de um processo de conversão de base decimal para base binária de 7 bits. Ou seja, a conversão dos números decimais presentes no intervalo de 0 a 127 para seus respectivos números binários define as 128 transições (combinações de vizinhança), conforme Tabela 5.

Não existe nenhuma restrição quanto ao número de terminais que pode aparecer na regra. No exemplo, apresentado a seguir, são considerados apenas os terminais *a* e *b*. Os terminais excluídos da regra podem assumir quaisquer valores, uma vez que não exercem influência na decisão de qual estado (0 ou 1) será atribuído à regra.


```

se (( $a = 0$ ) AND ( $b = 1$ )) então
  Regra  $\leftarrow$  1;
senão
  Regra  $\leftarrow$  0;
fim

```

A concatenação de todas as 128 saídas (última coluna da Tabela 5) "00 ... 10" compõe a regra que é dita solução candidata ao problema de classificação pela regra da maioria.

Tabela 5: Tabela de transições.

| # | Transições | | | | | | | Regra |
|-----|------------|-----|-----|-----|-----|-----|-----|-------|
| | a | b | c | d | e | f | g | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| ... | | | | ... | | | | ... |
| 127 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 128 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Esta solução candidata é então submetida ao processo de avaliação ou cálculo do *fitness*. A função de *fitness* utilizada baseia-se nos PPCD: S , DV , PA , AA , Z_W e A . Como não existe um conjunto de valores ótimos para estes parâmetros adotou-se como valores de referência os valores médios: Sm , DVm , PAm , AAm , Z_Wm e Am apresentados na Tabela 3. A Equação 5 apresenta a função de *fitness* utilizada.

$$fitness = S_n * DV_n * PA_n * AA_n * Z_W_n * A_n \quad (5)$$

Onde os símbolos são os mesmos definidos na Seção 3.2.1 e o subscrito n indica que são normalizados entre 0 e 1, a partir de um processo de ajuste linear, conforme mostrado na Tabela 6, utilizando os valores médios. Quando todos assumirem valor 1 o *fitness* do indivíduo também será 1. A idéia deste ajuste é encontrar equações lineares que atribuam um valor de *fitness* elevado para as regras que possuam valores dos PPCD próximos aos valores médios e valores baixos à medida que se distanciarem destes.

A Figura 13 apresenta um gráfico onde é possível visualizar o processo de ajuste linear para os PPCD estudados.

Tabela 6: Processo de ajuste linear.

| Parâmetro Normalizado | Pseudocódigo |
|-----------------------|---|
| S_n | se ($S \leq S_m$) então $S_n \leftarrow 3,125 * S$; senão $S_n \leftarrow -1,470 * S + 1,470$; |
| DV_n | se ($DV \leq DV_m$) então $DV_n \leftarrow 1,136 * DV$; senão $DV_n \leftarrow -8,333 * DV + 8,333$; |
| PAn | se ($PA \leq PAm$) então $PAn \leftarrow 11,112 * PA$; senão $PAn \leftarrow -1,098 * PA + 1,098$; |
| AA_n | se ($AA \leq AA_m$) então $AA_n \leftarrow 5 * AA$; senão $AA_n \leftarrow -1,250 * AA + 1,250$; |
| Z_W_n | se ($Z_W \leq Z_W_m$) então $Z_W_n \leftarrow 2,127 * Z_W$; senão $Z_W_n \leftarrow -1,886 * Z_W + 1,886$; |
| An | se ($A \leq Am$) então $An \leftarrow 2 * A$; senão $An \leftarrow -2 * A + 2$; |

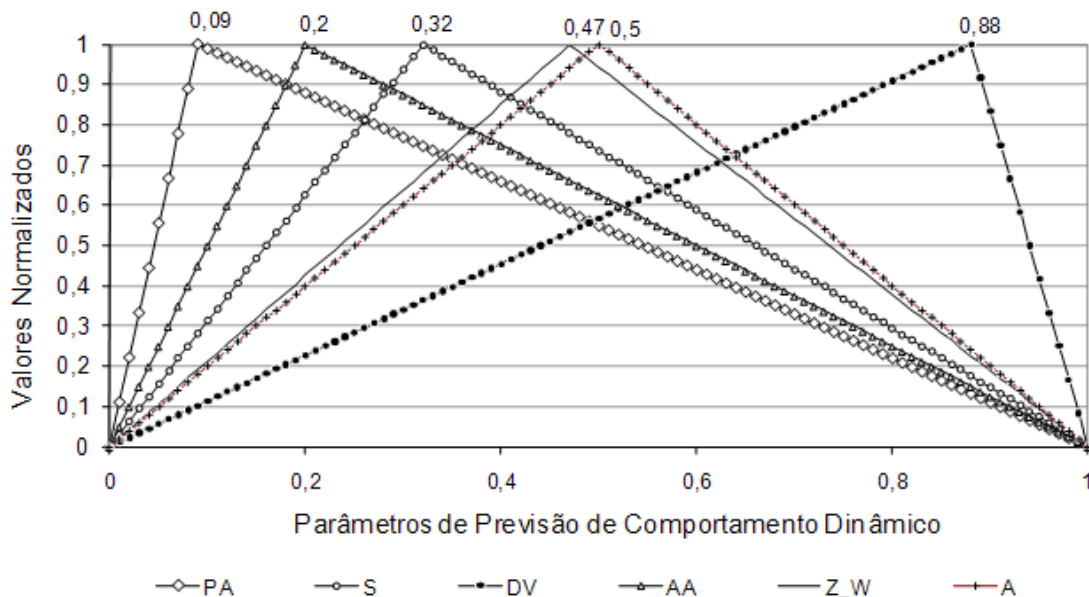


Figura 13: Ajuste linear dos parâmetros de previsão de comportamento dinâmico.

3.3 RESULTADOS

A Tabela 7 apresenta 6 regras encontradas pelo GEPCLASS para o problema de classificação pela regra da maioria considerando um AC de 149 células e raio igual a 3. Para facilitar a grafia as regras são apresentadas no formato hexadecimal com suas respectivas medidas de eficiência. Esta eficiência foi obtida pela simulação de 10^4 ACs aleatórios com distribuição uniforme evoluídos por 200 iterações.

Tabela 7: Regras encontradas pelo GEPCLASS considerando os parâmetros individualmente.

| Regra | Hexadecimal | Eficiência |
|----------|----------------------------------|------------|
| Regra_S | 0505A5A50F0FF0F00505A5A50F0FF0F0 | 0,00% |
| Regra_DV | 00110033555555550515053755555555 | 50,30% |
| Regra_PA | 0000000007770FFF0000000007770FFF | 50,33% |
| Regra_AA | 0CFF0CFF0CFF0CFF3FFF3FFF3FFF3FFF | 11,08% |
| Regra_ZW | FF33FF73FF33FF50FF33FF73FF33FF50 | 0,00% |
| Regra_A | AABB0031BBBB3333AABA0030BBBB3333 | 0,00% |

Cada uma das regras apresentadas na Tabela 7 refere-se à otimização de um determinado PPCD. Por exemplo, para a Regra_S buscou-se uma regra onde valor de S fosse igual ao valor de Sm . Uma vez que os outros parâmetros não foram considerados neste experimento, a função de *fitness* reduziu-se a: $fitness = Sn$. Adotou-se o mesmo procedimento para as demais regras. Tomando como referência resultados já publicados na literatura esperava-se atingir uma taxa de eficiência na ordem dos 80%, no entanto, as taxas não ultrapassaram os 51%.

A Tabela 8 apresenta os valores dos PPCD referentes às regras da Tabela 7. Em negrito encontram-se os valores otimizados para cada uma das regras encontradas.

Tabela 8: Valores dos parâmetros para as regras apresentadas na Tabela 7.

| Regra | S | DV | PA | AA | Z_W | A | DE |
|-----------|-------------|-------------|-------------|-------------|-------------|-------------|------|
| Regra_S | 0,32 | 0,62 | 0,19 | 0,40 | 0,69 | 0,44 | 0,41 |
| Regra_DV | 0,22 | 0,88 | 0,08 | 0,18 | 0,69 | 0,39 | 0,27 |
| Regra_PA | 0,20 | 0,87 | 0,09 | 0,18 | 0,66 | 0,33 | 0,28 |
| Regra_AA | 0,18 | 0,79 | 0,16 | 0,20 | 0,44 | 0,75 | 0,31 |
| Regra_Z_W | 0,19 | 0,38 | 0,33 | 0,67 | 0,47 | 0,73 | 0,77 |
| Regra_A | 0,24 | 0,52 | 0,27 | 0,45 | 0,55 | 0,50 | 0,49 |

A Tabela 9 apresenta outras 10 regras encontradas pelo GEPCLASS para o problema de classificação pela regra da maioria. Neste experimento buscou-se a otimização conjunta de todos os parâmetros através da função de *fitness* descrita pela Equação 5. A Tabela 10 apresenta os valores dos PPCD referentes às regras da Tabela 9.

Tabela 9: Regras encontradas pelo GEPCLASS considerando todos os parâmetros.

| Regra | Hexadecimal | Eficiência |
|----------|------------------------------------|------------|
| Regra_1 | 111110101111F3101F1F1F1F1FFF1F | 2,83% |
| Regra_2 | 000F000F555F555FCCCCCCCCFDDDFDDDF | 0,00% |
| Regra_3 | 37370113377F015F37370113377F015F | 49,96% |
| Regra_4 | 0AAA0FFF011105550AAA0FFF0BBB0FFF | 0,00% |
| Regra_5 | 00000F0FFFFFF0F1F00110F1FFFFFF0F1F | 1,83% |
| Regra_6 | 333300000000F0F3F3F3F3F3F3F3F3F | 49,33% |
| Regra_7 | 0A5F0A5F0A5FFFFFF005500550A553B77 | 0,00% |
| Regra_8 | 0033F0330033F0330033FFFF0033FFFF | 49,73% |
| Regra_9 | 0A0F0A0F000F000F3F3F3F3F3F3F3F3F | 1,62% |
| Regra_10 | 331133113311331111FF11FF11FF11FF | 49,33% |

Tabela 10: Valores dos parâmetros para as regras apresentadas na Tabela 9.

| Regra | <i>S</i> | <i>DV</i> | <i>PA</i> | <i>AA</i> | <i>Z_W</i> | <i>A</i> | <i>DE</i> |
|----------|----------|-----------|-----------|-----------|------------|----------|-----------|
| Regra_1 | 0,26 | 0,86 | 0,09 | 0,26 | 0,46 | 0,47 | 0,09 |
| Regra_2 | 0,24 | 0,82 | 0,13 | 0,22 | 0,49 | 0,58 | 0,14 |
| Regra_3 | 0,26 | 0,82 | 0,13 | 0,22 | 0,46 | 0,52 | 0,10 |
| Regra_4 | 0,27 | 0,80 | 0,13 | 0,22 | 0,49 | 0,52 | 0,11 |
| Regra_5 | 0,24 | 0,84 | 0,11 | 0,24 | 0,46 | 0,54 | 0,11 |
| Regra_6 | 0,25 | 0,85 | 0,10 | 0,25 | 0,50 | 0,50 | 0,10 |
| Regra_7 | 0,27 | 0,81 | 0,13 | 0,21 | 0,47 | 0,51 | 0,10 |
| Regra_8 | 0,23 | 0,85 | 0,10 | 0,20 | 0,45 | 0,50 | 0,10 |
| Regra_9 | 0,22 | 0,86 | 0,09 | 0,20 | 0,47 | 0,53 | 0,11 |
| Regra_10 | 0,25 | 0,85 | 0,10 | 0,20 | 0,50 | 0,50 | 0,08 |

A Figura 14 apresenta um exemplo de simulação da dinâmica para cada uma das regras apresentadas na Tabela 7 e a Figura 15 apresenta um exemplo de simulação da dinâmica para as regras: Regra_2, Regra_3, Regra_4, Regra_6, Regra_8 e Regra_10, apresentadas na Tabela 9. Na Figura 14 (b) e (c) e na Figura 15 (b), (d), (e) e (f) têm-se um comportamento dinâmico nulo. Nas demais figuras têm-se um comportamento dinâmico ponto fixo.

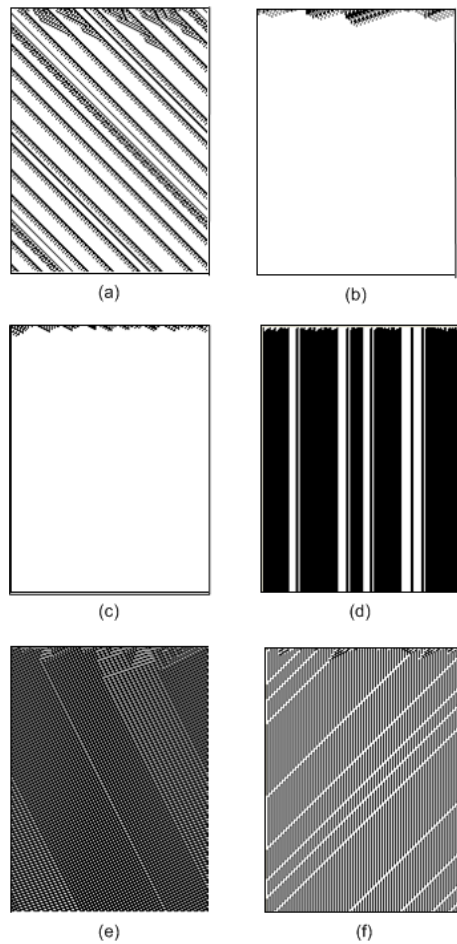


Figura 14: Simulação da dinâmica das regras apresentadas na Tabela 7: (a) Regra_S, (b) Regra_DV, (c) Regra_PA, (d) Regra_AA, (e) Regra_Z_W e (f) Regra_A.

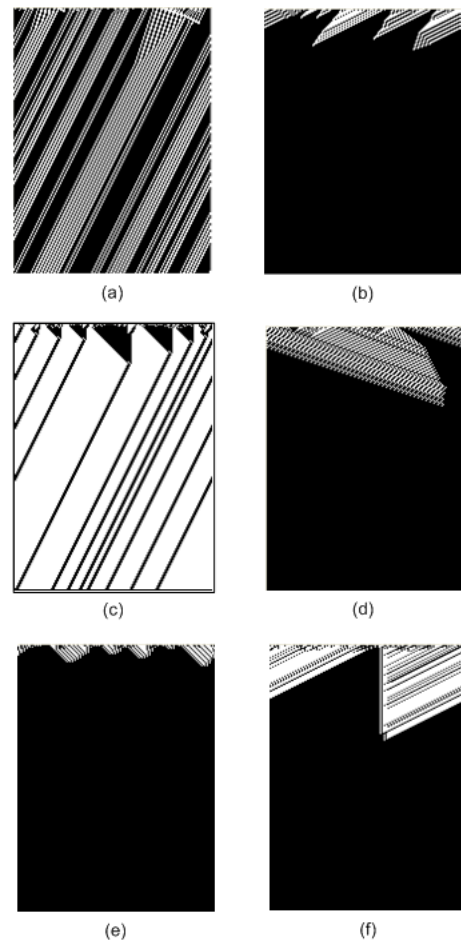


Figura 15: Simulação da dinâmica das regras apresentadas na Tabela 9: (a) Regra_2, (b) Regra_3, (c) Regra_4, (d) Regra_6, (e) Regra_8 e (f) Regra_10.

3.4 DISCUSSÃO

A Seção 3.2.1 apresentou uma análise de correlação realizadas sobre os PPCD. Percebeu-se que os parâmetros S e Z_{Li} e os parâmetros A e CM são numericamente equivalentes (Tabela 3) para uma mesma regra o que permitiu que fossem excluídos os parâmetros Z_{Li} e CM do processo subsequente de análise.

Uma vez que as 31 regras analisadas exibem comportamento nulo e não existem valores de referência para os parâmetros calculados sobre estas regras, adotou-se como valores de referência para regras que exibem comportamentos nulos os valores médios dos parâmetros (Sm , DVm , PAm , AAm , Z_{Wm} e Am). Considerou-se prudente a adoção dos valores médios, uma vez que a distância Euclidiana entre os parâmetros de cada uma das 31 regras e os valores médios destes mesmos parâmetros manteve-se pequena, num intervalo variando entre 0,05 e 0,27 (Tabela 3).

Então, implementou-se um sistema que tem a finalidade de encontrar regras que apresentem comportamento nulo. O algoritmo implementado neste sistema baseia-se na técnica de computação evolucionária conhecida como Programação de Expressão Genética e alterna sua função de *fitness* conforme o parâmetro de previsão que se deseja otimizar.

Inicialmente 6 regras foram encontradas conforme Tabela 7. Na primeira regra (Regra_S) o algoritmo tinha o objetivo de otimizar o valor de S não se preocupando com o valor dos demais parâmetros. Uma vez que o valor de S foi igual ao valor de Sm , cumpriu-se o objetivo. O mesmo valeu para as demais regras. Todas cumpriram o objetivo, conforme valores destacados em negrito na Tabela 8.

Apesar do sistema não ter apresentado dificuldades para encontrar cada uma das 6 regras, as medidas de eficiência (Tabela 7) e a simulação da dinâmica (Figura 14) indicam que as mesmas não são soluções para o problema de classificação pela maioria e que nem todas apresentam um comportamento nulo.

As taxas de eficiência para o problema de classificação pela regra da maioria foram muito abaixo das esperadas. A Regra_DV e a Regra_PA apresentaram uma eficiência na ordem dos 50%. No entanto, observando as simulações destas regras na Figura 14 e o valor do parâmetro atividade na Tabela 8 conclui-se que estas regras levam qualquer configuração inicial de AC para um estado onde todas as células possuem valor igual a 0 após 200 iterações. Isto não resolve o problema de classificação pela regra da maioria apesar de apresentar um comportamento nulo. A Regra_S, Regra_AA, Regra_Z_W e Regra_A apresentam valores de eficiência distantes dos esperados (Tabela 7) além de apresentarem comportamentos (Figura 14) classificados como ponto fixo.

A partir destes resultados conclui-se que os parâmetros quando considerados individualmente não são eficientes para realizar classificação de comportamento.

Realizou-se um último experimento utilizando todos os parâmetros, objetivando encontrar regras que apresentassem valores de parâmetros próximos aos utilizados como referência. Repetiu-se os experimentos por 10 vezes conforme Tabela 9 e Tabela 10. Diferentemente do experimento anterior o sistema apresentou bastante dificuldade para encontrar regras que satisfizessem os objetivos. Isto pode ter ocorrido em função do sistema de ajuste linear adotado para os PPCD (Figura 13). É provável que a consideração de intervalos de valores referenciais ao invés de um único valor para cada parâmetro, facilite o processo de otimização dos parâmetros.

A Regra_6, Regra_8 e Regra_10 foram as únicas que mantiveram seus valores de parâmetros dentro do intervalo máximo-mínimo estabelecido para os PPCD (Tabela 3). No entanto, estas regras também não resolvem eficientemente o problema de classificação pela regra da maioria, apesar de apresentarem um comportamento classificado como nulo (Figura 15). Observando as simulações destas regras na Figura 15 (d), (e) e (f), também conclui-se que elas levam qualquer configuração inicial de AC para um estado onde todas as células possuem valor igual a 1 após 200 iterações, independentemente da densidade do AC utilizado como configuração inicial. Como os AC iniciais foram criados aleatoriamente segundo uma distribuição uniforme, já era esperado que aproximadamente 50% dos AC tivessem um valor de densidade superior a 0,5. Visto que a Regra_6, Regra_7 e Regra_10 conduzem qualquer configuração inicial para uma configuração onde todas as células do AC assumem estado igual a 1, as taxas de eficiência na ordem dos 50% também já eram esperadas.

4 NOVO MÉTODO PARA PREVISÃO DE COMPORTAMENTO DINÂMICO

4.1 INTRODUÇÃO

Este capítulo propõe uma metodologia para a previsão do comportamento dinâmico de autômatos celulares unidimensionais. Para o desenvolvimento desta metodologia considera-se os padrões de comportamento qualitativos descritos por Wolfram (1984) e refinados por Li e Packard (1990) citados na Seção 2.2.2. Através da identificação destes padrões de comportamento, em uma simulação parcial da dinâmica do sistema, objetiva-se estabelecer uma previsão a respeito da dinâmica completa do sistema. O objetivo é reduzir o custo computacional da simulação de um AC. Para avaliar a metodologia utiliza-se regras aplicadas ao problema de classificação pela regra da maioria (Seção 2.2.4.1). A avaliação da metodologia se dá pela comparação do processo de simulação completa da dinâmica com o processo de simulação parcial proposto.

4.2 METODOLOGIA

Esta seção é organizado da seguinte maneira. Inicialmente, analisa-se o modelo de Wolfram para identificação de comportamento dinâmico. Posteriormente apresenta-se um novo método para realização de previsão de comportamento dinâmico.

4.2.1 Modelo de Wolfram para Identificação de Padrões de Comportamento

Wolfram apresenta um modelo de identificação de comportamento dinâmico baseado no cálculo da distância de Hamming (PETERSON; WELDON, 1972) entre dois ACs ao longo das iterações (Seção 2.2.2).

O procedimento ocorre da seguinte maneira. Primeiro gera-se aleatoriamente, segundo uma distribuição uniforme, um AC com a células (parâmetro AC na Equação 6). Em seguida, faz-se uma cópia deste AC alterando aleatoriamente o estado de uma célula (parâmetro $ACcp$ na Equação 6). Então computa-se a distância de Hamming (hd) entre os ACs segundo a Equação 6.

$$hd = \frac{\sum_{i=1}^a (|AC_i - ACcp_i|)}{a} \quad (6)$$

Se a distância de Hamming estabilizar em zero ao longo das iterações tem-se um comportamento homogêneo. Este comportamento homogêneo pode ser chamado de comportamento nulo (LI; PACKARD, 1990) se todas as células do AC estabilizarem seus estados em 1 ou 0. A Figura 16 (c) apresenta um gráfico onde se pode observar a variação dos valores de distância de Hamming por 30 iterações entre um AC formado por 200 células (Figura 16 (a)) e uma cópia deste AC (Figura 16 (b)). No AC replicado alterou-se o estado da célula 135. A regra utilizada na simulação desta dinâmica é a regra elementar 160 (WOLFRAM, 1983).

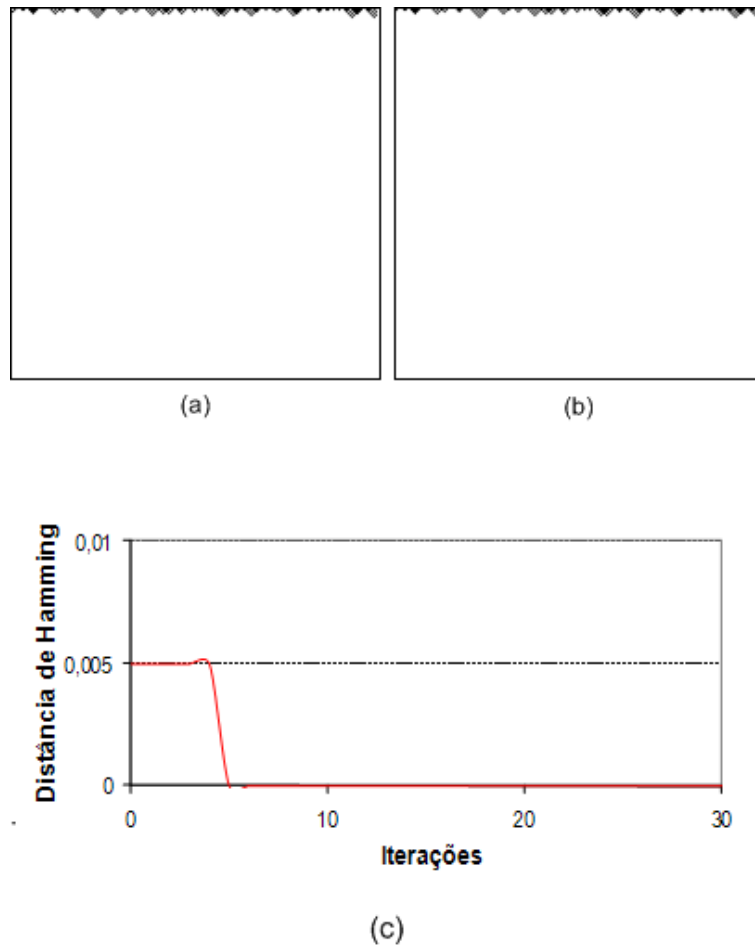


Figura 16: Regra elementar 160 de comportamento homogêneo. (a) AC, (b) Cópia do AC com estado da célula 135 alterado, (c) distância de Hamming ao longo de 30 iterações.

Se a distância de Hamming estabilizar em um valor, ou num conjunto de valores diferente de zero ao longo das iterações tem-se um comportamento estável ou periódico. A Figura 17 (c) apresenta um gráfico onde se pode observar a variação dos valores de distância de Hamming por 30 iterações. As configurações iniciais dos ACs utilizados (Figura 17 (a) e (b)) são as mesmas da Figura 16 (a) e (b). A regra utilizada na simulação desta dinâmica é a regra elementar 76 (WOLFRAM, 1983).

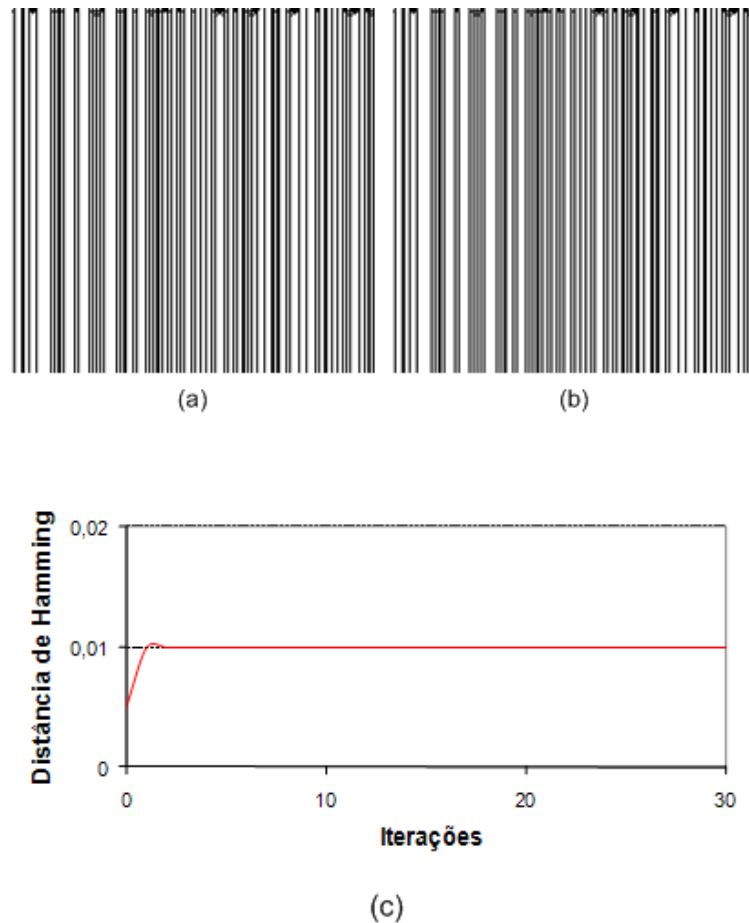


Figura 17: Regra elementar 76 de comportamento ponto fixo. (a) AC, (b) Cópia do AC com estado da célula 135 alterado, (c) distância de Hamming ao longo de 30 iterações.

Se a distância de Hamming não se estabilizar ao longo das iterações tem-se um comportamento caótico. A Figura 18 (c) apresenta um gráfico onde pode-se observar a variação dos valores de distância de Hamming por 4000 iterações. As configurações iniciais dos ACs utilizados são as mesmas da Figura 16. A regra utilizada na simulação desta dinâmica é a regra elementar 182 (WOLFRAM, 1983). É interessante salientar que mesmo após 4000 iterações o sistema continua instável.

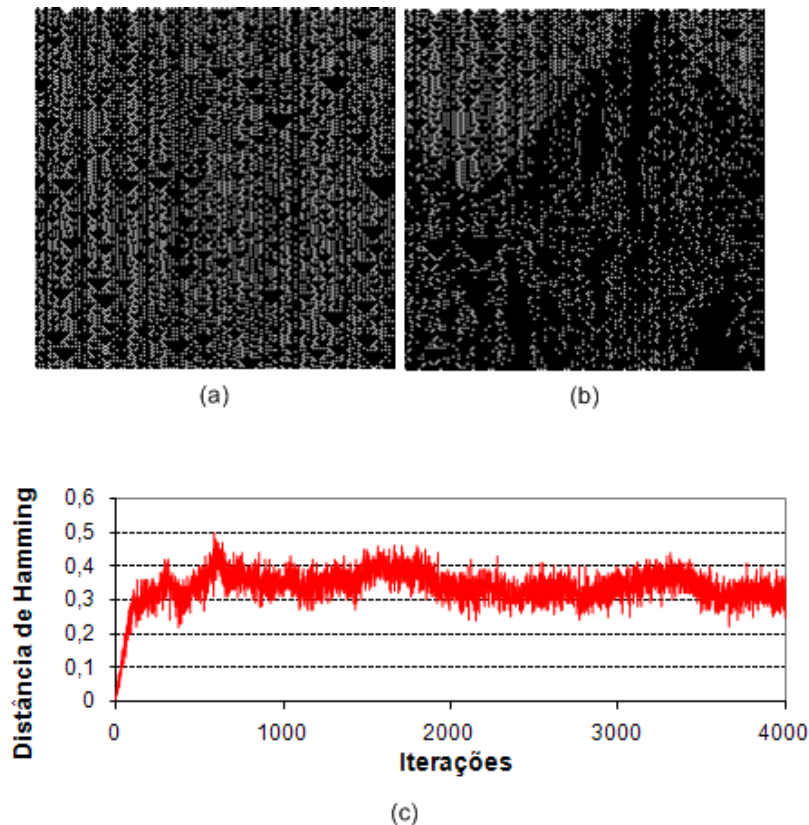


Figura 18: Regra elementar 182 de comportamento caótico. (a) AC, (b) Cópia do AC com estado da célula 135 alterado, (c) distância de Hamming ao longo de 4000 iterações.

Se a distância de Hamming se estabilizar após um longo período de instabilidade tem-se um comportamento complexo. A Figura 19 (e) apresenta um gráfico onde se pode observar a variação dos valores de distância de Hamming por 4000 iterações de um AC formado por 200 células. As simulações da Figura 19 (a) e Figura 19 (b) apresentam o diagrama espaço-temporal das primeiras 200 iterações. Observando apenas este intervalo tem-se a impressão de que este sistema apresenta um comportamento caótico. No entanto, a partir da iteração 800 (Figura 19 (c) e Figura 19 (d)) tem-se a confirmação do comportamento complexo. O sistema se estabilizou a partir da iteração 900. A regra utilizada na simulação desta dinâmica é a regra elementar 124 (WOLFRAM, 1983). As configurações iniciais dos ACs utilizados são as mesmas da Figura 16.

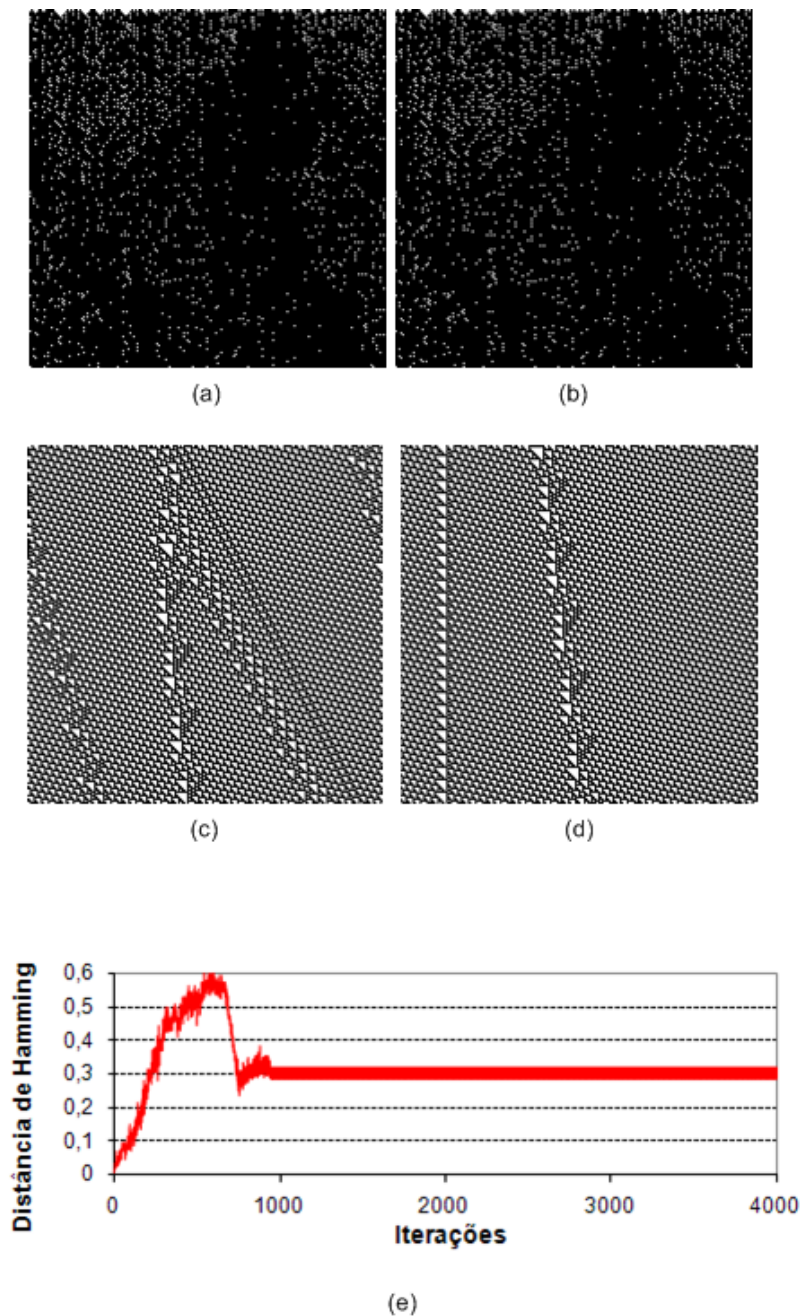
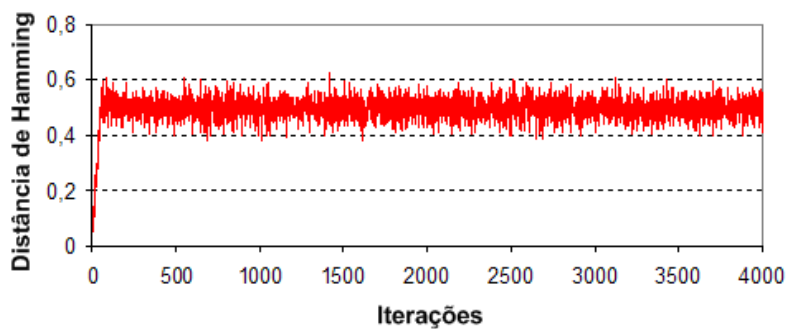


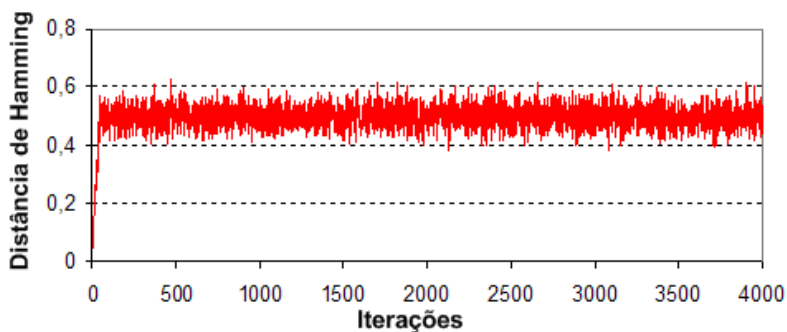
Figura 19: Regra elementar 124 de comportamento complexo. (a) AC - Iteração 0 até 200, (b) Cópia do AC - Iteração 0 até 200, com estado da célula 135 alterado, (c) AC - Iteração 800 até 1000, (d) Cópia do AC - Iteração 800 até 1000, (e) distância de Hamming ao longo de 4000 iterações.

O método desenvolvido por Wolfram pode exigir muitas iterações para detecção de um comportamento. Por exemplo, para duas regras, com $r = 3$ publicadas em (WUENSCH, 1994) de comportamento comprovadamente complexas o método baseado na distância de Hamming não foi capaz de identificar este comportamento em 4000 iterações. Esta análise pode ser visualizada na Figura 20 (a) e (b). Outra restrição a respeito desta metodologia é sua capacidade

de generalização. Ou seja, como apenas dois ACs foram utilizados no processo chega-se a uma conclusão referente ao comportamento local do sistema e não global.



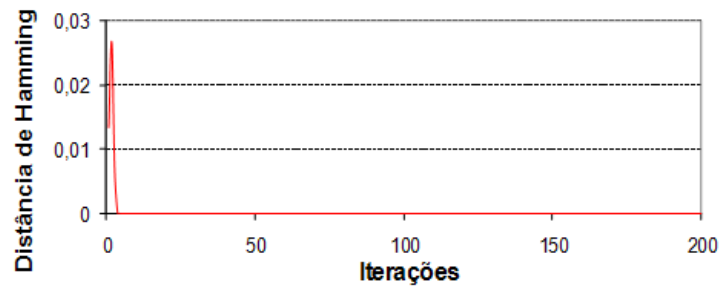
(a)



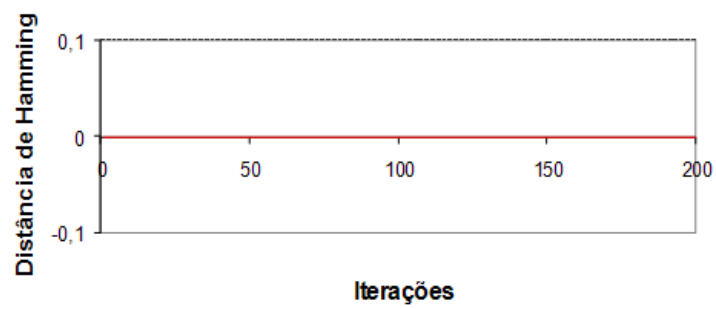
(b)

Figura 20: Análise de regras complexas de Wuensche. (a) Regra: 2EFBDFE22DAA67F045C14B0824A22E2B, (b) Regra: 3B469C0CE4F7FA96B93B4D3SB89ECC0E0.

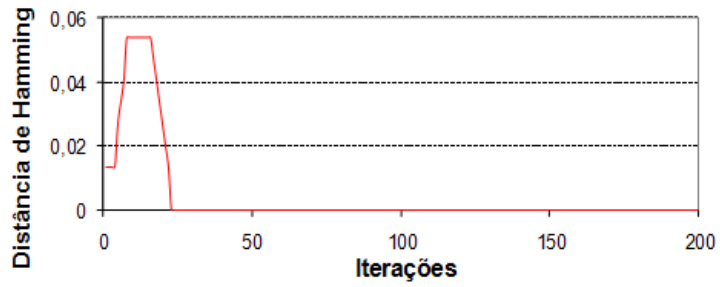
A Figura 21 apresenta uma análise de comportamento dinâmico segundo modelo de Wolfram para as regras GKL, MHC, DAV e DAS (Tabela 2). Gráficamente tem-se a confirmação de que estas regras apresentam um comportamento nulo.



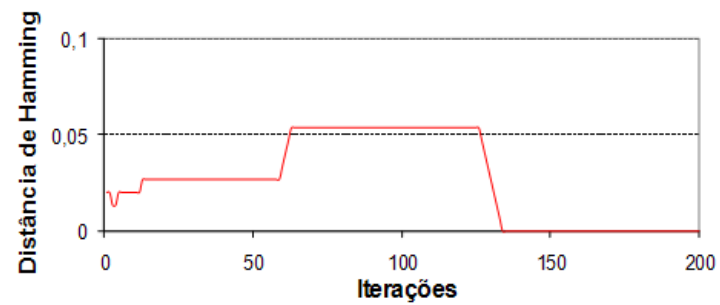
(a)



(b)



(c)



(d)

Figura 21: Análise de comportamento das regras: (a) GKL, (b) MCH, (c) DAV e (d) DAS.

A Figura 22 (a) apresenta uma simulação de dinâmica para a regra 331133113311331111FF11FF11FF11FF. Esta regra encontra-se na Tabela 9 e foi encontrada pelo sistema de indução de regras implementado no Capítulo 3. No AC cópia (Figura 22 (b)) alterou-se o estado da célula 109. Segundo esta simulação a regra apresenta comportamento periódico (Figura 22 (c)). Este comportamento caracteriza-se pela variação cíclica de valores de distância de Hamming ao longo das iterações.

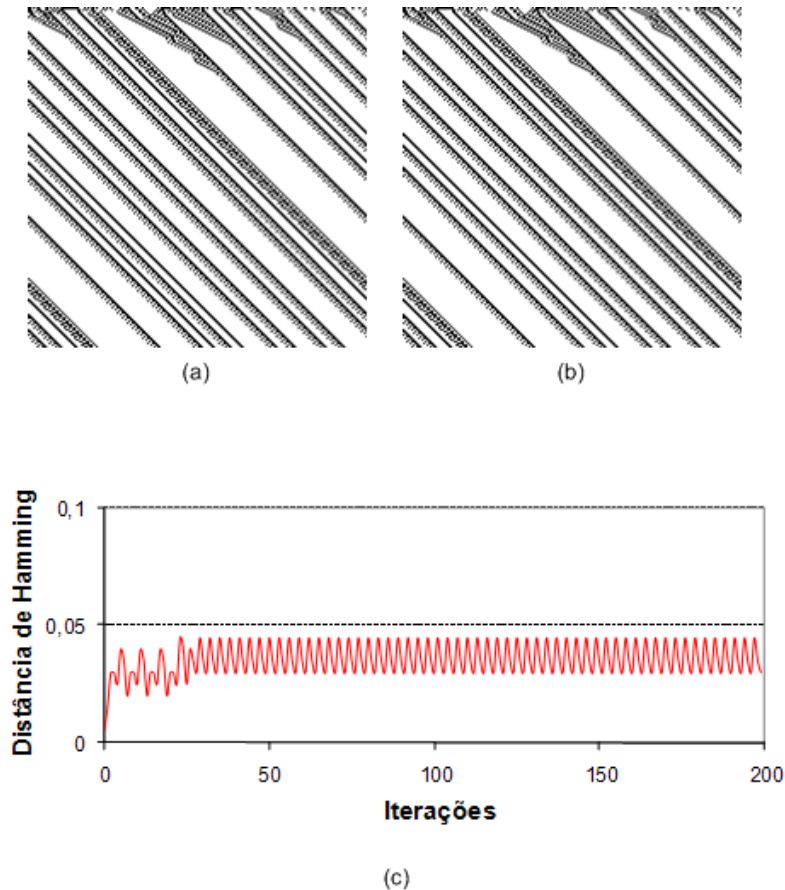


Figura 22: Análise de comportamento da regra 331133113311331111FF11FF11FF11FF. (a) AC, (b) Cópia do AC com estado da célula 109 alterado, (c) distância de Hamming ao longo de 200 iterações.

4.2.2 Abordagem Proposta para Previsão de Comportamento Dinâmico Nulo

Os métodos tradicionais de previsão de comportamento dinâmico realizam a simulação do sistema de forma completa, ou seja, aplicam a regra de transição sobre uma determinada configuração inicial de AC por t iterações, onde t representa um limite superior para o problema em questão, ou seja, aplicar a regra por mais iterações não mais alteraria o comportamento atual do sistema. Não existe um valor ideal para t , depende do problema. Por exemplo, alguns trabalhos relatados na literatura utilizam um t igual a 200 para o problema de classificação pela

regra da maioria com 149 células e raio igual a 3. O método proposto neste trabalho leva em consideração não somente a regra de transição, mas também considera a configuração inicial do sistema realizando uma análise parcial da dinâmica.

O método baseia-se numa análise estatística realizada sobre a regra de transição durante a simulação parcial do sistema. Esta análise consiste na contagem do número de vezes que determinada transição é utilizada em cada iteração. No decorrer das iterações é possível identificar transições dominantes. A partir destas transições dominantes é possível inferir o comportamento final do sistema. Para exemplificar o método utiliza-se uma instância reduzida do problema de classificação pela regra da maioria. Nesta instância o AC é composto por 12 células, raio igual a 1, configuração inicial: 010111010101 e regra de transição Φ ([000 \rightarrow 0 ; 001 \rightarrow 0; 010 \rightarrow 0; 011 \rightarrow 1; 100 \rightarrow 0; 101 \rightarrow 1; 110 \rightarrow 1; 111 \rightarrow 1]). A Figura 23 apresenta o diagrama espaço-temporal deste AC evoluído por 20 iterações. Os quadrados em preto representam as células com valor 1 e os quadrados em branco com valor 0.



Figura 23: Diagrama espaço-temporal.

Como a densidade inicial do AC (Figura 23) é 0,58, todas as células do AC inicial devem evoluir para 1 após t iterações. Neste exemplo considera-se $t = 20$, ou seja, 20 iterações são utilizadas para identificar a dinâmica completa do sistema. Pela observação da Figura 23 conclui-se que a regra realmente alcançou seu objetivo, ou seja, todas as células do arranjo unidimensional estão com valor igual a 1 na 20^a iteração.

A Tabela 11 apresenta o mecanismo de funcionamento do método proposto. Na primeira coluna são colocadas todas as transições que compõem a regra de transição Φ e nas colunas seguintes computa-se a frequência de utilização destas transições em cada uma das iterações no processo de simulação da dinâmica do sistema. Nesta Tabela são mostrados as 10 primeiras iterações e é possível observar que a transição 111 \rightarrow 1 tornou-se dominante após a quinta iteração.

Tabela 11: Metodologia para previsão de comportamento dinâmico.

| Transição | t_1 | t_2 | t_3 | t_4 | t_5 | t_6 | t_7 | t_8 | t_9 | t_{10} |
|-----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|
| 000 → 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 001 → 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 010 → 0 | 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 011 → 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 100 → 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 101 → 1 | 5 | 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 110 → 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 111 → 1 | 1 | 3 | 5 | 7 | 9 | 12 | 12 | 12 | 12 | 12 |
| Soma 0 | 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Soma 1 | 8 | 9 | 10 | 11 | 12 | 12 | 12 | 12 | 12 | 12 |

Para simplificar o processo de análise de frequência das transições adota-se para o problema de classificação pela regra da maioria a computação de duas variáveis ao longo das iterações. Estas variáveis são “Soma 0” e “Soma 1”. A primeira computa o somatório das frequências das transições que levam uma célula c_i^t para 0 no tempo $t + 1$ e a segunda computa o somatório das frequências das transições que levam uma célula c_i^t para 1 no tempo $t + 1$. Nesta simplificação a característica de dominância migra naturalmente para uma destas variáveis.

Observando-se os valores atribuídos a “Soma 0” e “Soma 1” ao longo das iterações é possível identificar que “Soma 1” tornou-se dominante a partir da quinta iteração não se alterando até a última iteração. Assumindo-se que este comportamento permanecerá constante até o final do período de simulação do sistema, infere-se o comportamento dinâmico do mesmo. Neste caso, assume-se que o sistema realmente terá todas suas células com valor 1 após 20 iterações, apesar de apenas as 10 primeiras iterações terem sido analisadas.

O período compreendido entre t_1 e t_4 na (Tabela 11) é chamado período transiente, não sendo ainda possível identificar um comportamento dominante no sistema. Assim, este período não contribui para previsão do comportamento dinâmico e pode ser ignorado.

4.3 RESULTADOS

A metodologia proposta foi avaliada sobre 31 regras consagradas na literatura como regras de comportamento nulo e soluções para o problema de classificação pela regra da maioria conforme a Tabela 2. Os resultados estão reportados na Tabela 12. Na primeira coluna desta Tabela encontram-se as regras avaliadas. Na segunda coluna encontram-se as medidas de desempenho obtidas para cada uma das regras em um processo de simulação da dinâmica completa. O desempenho médio, em percentagem, foi obtido pela simulação de 10000 ACs gerados aleato-

Tabela 12: Avaliação da metodologia proposta.

| Regra | Dinâmica Completa | Dinâmica Parcial | | | | | |
|-------|-------------------|------------------|--------|--------|---------|---------|---------|
| | | Exp 10 | Exp 50 | Exp 90 | Exp 160 | Exp 200 | Exp 400 |
| GKL | 81,20% | 64,36% | 55,99% | 79,93% | 81,05% | 81,07% | 81,20% |
| MHC | 77,24% | 60,45% | 54,19% | 76,88% | 77,23% | 77,24% | 77,24% |
| DAV | 81,65% | 67,46% | 60,64% | 81,65% | 81,65% | 81,65% | 81,65% |
| DAS | 81,48% | 70,65% | 59,51% | 81,48% | 81,48% | 81,48% | 81,48% |
| ABK | 81,84% | 77,55% | 71,28% | 81,62% | 81,84% | 81,84% | 81,84% |
| CRA | 84,18% | 79,59% | 71,52% | 84,18% | 84,18% | 84,18% | 84,18% |
| JP1 | 84,93% | 77,00% | 72,60% | 84,84% | 84,93% | 84,93% | 84,93% |
| JP2 | 85,97% | 75,17% | 83,85% | 85,89% | 85,97% | 85,97% | 85,97% |
| BOO1 | 86,19% | 75,84% | 84,43% | 86,06% | 86,19% | 86,19% | 86,19% |
| BOO2 | 85,79% | 73,46% | 82,45% | 84,24% | 85,54% | 85,72% | 85,80% |
| BOO3 | 85,46% | 71,51% | 82,92% | 83,86% | 85,22% | 85,43% | 85,47% |
| BOO4 | 85,34% | 77,36% | 70,42% | 85,12% | 85,34% | 85,34% | 85,34% |
| BOO5 | 85,25% | 74,95% | 69,89% | 85,25% | 85,25% | 85,25% | 85,25% |
| R1 | 82,63% | 74,79% | 67,56% | 82,63% | 82,63% | 82,63% | 82,63% |
| R2 | 81,81% | 73,87% | 68,30% | 81,59% | 81,81% | 81,81% | 81,81% |
| R3 | 81,81% | 73,62% | 64,58% | 81,81% | 81,81% | 81,81% | 81,81% |
| R4 | 82,66% | 72,24% | 67,57% | 82,64% | 82,66% | 82,66% | 82,66% |
| R5 | 81,28% | 68,09% | 61,20% | 81,26% | 81,28% | 81,28% | 81,28% |
| R6 | 80,97% | 60,61% | 48,23% | 73,26% | 80,97% | 80,97% | 80,97% |
| R7 | 81,24% | 67,71% | 71,45% | 81,33% | 81,29% | 81,31% | 81,29% |
| R8 | 80,58% | 70,55% | 63,76% | 80,58% | 80,58% | 80,58% | 80,58% |
| R9 | 80,62% | 71,46% | 72,72% | 78,79% | 80,62% | 80,62% | 80,62% |
| R10 | 77,85% | 59,97% | 69,92% | 72,34% | 77,70% | 77,85% | 77,85% |
| R11 | 78,47% | 66,88% | 60,72% | 78,47% | 78,47% | 78,47% | 78,47% |
| R12 | 78,06% | 64,74% | 69,02% | 76,53% | 78,06% | 78,06% | 78,06% |
| R13 | 78,23% | 65,77% | 58,20% | 78,23% | 78,23% | 78,23% | 78,23% |
| R14 | 77,66% | 53,67% | 65,28% | 76,08% | 77,66% | 77,66% | 77,66% |
| R15 | 77,05% | 49,59% | 41,53% | 68,49% | 77,01% | 77,05% | 77,05% |
| R16 | 77,88% | 62,12% | 58,03% | 77,86% | 77,88% | 77,88% | 77,88% |
| R17 | 78,07% | 51,18% | 62,61% | 77,48% | 78,07% | 78,07% | 78,07% |
| R18 | 78,52% | 62,44% | 65,52% | 76,88% | 78,52% | 78,52% | 78,52% |
| média | 81,35% | 68,21% | 66,32% | 80,23% | 81,33% | 81,35% | 81,35% |

riamente durante 400 iterações. Na literatura utiliza-se normalmente 200 iterações. No entanto, os experimentos demonstraram que nem sempre se consegue estabilizar o comportamento destas regras com este número de iterações (Figura 25). As demais colunas da Tabela apresentam o desempenho médio, em percentagem, de 6 diferentes experimentos contemplando o método de simulação de dinâmica parcial proposto neste trabalho. Os experimentos foram rotulados como: Exp 10, Exp 50, Exp 90, Exp 160, Exp 200 e Exp 400. Cada experimento considerou no processo de simulação da dinâmica parcial: 10, 50, 90, 160, 200 e 400 iterações, respectivamente. É importante destacar que nestes experimentos sempre se considerou como período transiente o

total de iterações subtraído de 10. Por exemplo, para um experimento de avaliação da dinâmica parcial realizado até a iteração 50, descartou-se as 40 primeiras iterações (período transitente) e aplicou-se o método de computação parcial da dinâmica entre a 40^a e a 50^a iterações. Também foram considerados 10000 ACs gerados aleatoriamente para simulação da dinâmica parcial.

Além dos resultados reportados para dinâmica parcial aplicada sobre as 31 regras na Tabela 12 foram realizados mais 36 experimentos. Estes experimentos preenchem as lacunas deixadas entre os experimentos Exp 10 e Exp 400. O que diferencia cada experimento é o acréscimo de 10 iterações a cada novo experimento. Assim, também executou-se o experimento Exp 20, Exp 30, Exp 40, Exp 60, até o Exp 390. Os resultados destes experimentos foram condensados nas Figuras 24, 25 e 26 que serão apresentadas na sequência.

O erro médio quadrático é uma medida de erro que representa a distância média entre os resultados obtidos e os resultados desejados. Neste caso, os resultados desejados são representados por um vetor de valores de 31 elementos que representam o desempenho das 31 regras na simulação de dinâmica completa, e os resultados obtidos são representados por outro vetor de valores de 31 elementos obtido do desempenho das 31 regras na simulação de dinâmica parcial. O gráfico mostrado na Figura 24 apresenta a evolução do erro médio quadrático ao longo dos 40 experimentos realizados para dinâmica parcial em relação a dinâmica completa.

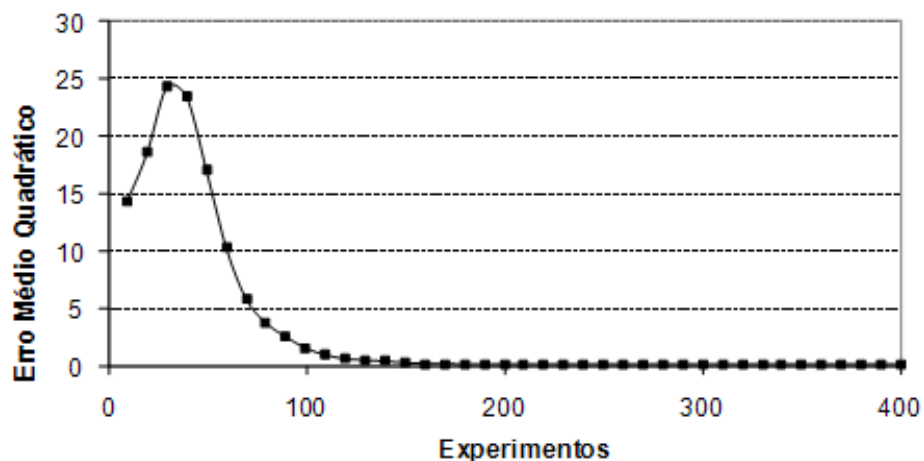


Figura 24: Evolução do erro médio quadrático.

As Figuras 25 e 26 apresentam o desempenho eficiência médio e o desvio-padrão para as 31 regras (Tabela 12) ao longo do processo iterativo, para dinâmica completa e parcial, respectivamente.

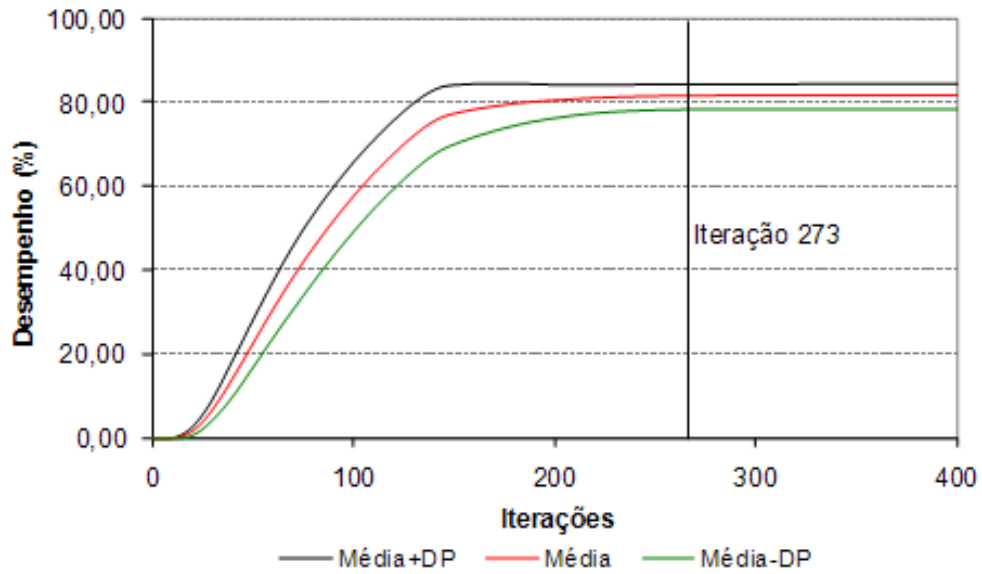


Figura 25: Dinâmica completa.

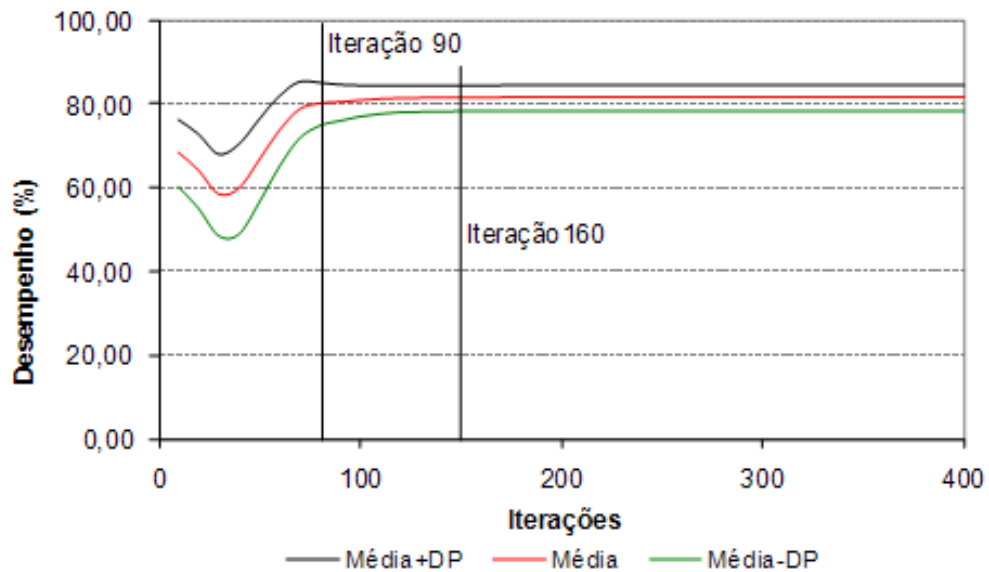


Figura 26: Dinâmica parcial.

4.4 DISCUSSÃO

A Figura 25 apresentou desempenho quando se utiliza simulação completa da dinâmica. Neste gráfico pode-se observar um crescimento contínuo das medidas de desempenho ao longo das iterações. Em média, precisou-se de 273 iterações para que as 31 regras avaliadas registrassem o seu melhor desempenho. A partir da iteração 273 não houve modificações das

medidas de desempenho. Observando a variação do desvio-padrão ao longo do processo iterativo evidenciam-se maiores distorções no período que marca o final do período transiente e início da estabilidade do sistema. Ou seja, a utilização de 200 iterações, como apresentado por alguns trabalhos da literatura, pode não reportar o melhor desempenho para a regra avaliada. E, a utilização de um número de iterações insuficiente para retirar o sistema do período transiente trará resultados ainda piores. Por exemplo, na iteração 90 o desempenho médio das regras foi de 51,27%, na iteração 160 de 78,00%, enquanto que, na iteração 273 o desempenho médio foi de 81,35%.

O gráfico da Figura 24 apresentou o erro médio quadrático entre a medida de desempenho obtida pela simulação completa do sistema e todas as medidas de desempenho obtidas pela simulação parcial do sistema ao longo de 400 iterações. Torna-se evidente que, a partir da iteração 160 do modelo de simulação parcial já não existem mais diferenças significativas entre as duas metodologias. Neste ponto o erro médio quadrático foi de 0,07 e a perda de desempenho de 0,02%.

A Figura 26 apresentou o desempenho para a metodologia proposta neste capítulo, chamada de simulação parcial da dinâmica, que comprova os resultados apresentados no gráfico da Figura 24. Na iteração 160 o desempenho médio foi de 81,33%. Observando as 90 primeiras iterações da dinâmica completa (Figura 25) e da dinâmica parcial (Figura 26) percebe-se uma disparidade bastante grande entre os resultados. Na dinâmica completa o desempenho médio foi de 51,27%, enquanto que na dinâmica parcial foi de 80,23%. Esta enorme diferença pode ser explicada pela maneira como se realiza a inferência do comportamento final do sistema que considera não somente a iteração atual, mas sim as últimas 10 iterações. Esta abordagem permitiu identificar uma linha de tendência do sistema que se comprova pelos resultados alcançados. Se pequenos erros forem aceitáveis, neste caso de aproximadamente 1,12%, 90 iterações no processo de dinâmica parcial podem substituir as 273 iterações da dinâmica completa para o problema de classificação pela regra da maioria.

5 *HARDWARE-SOFTWARE CO-DESIGN PARA INDUÇÃO DE REGRAS*

5.1 INTRODUÇÃO

Dentro do projeto de sistemas embarcados destaca-se uma abordagem de projeto conhecida como *Hardware-Software Co-design*. Esta abordagem tem por principal finalidade particionar problemas para que tarefas distintas sejam executadas em módulos de *hardware* e *software* específicos. Um *hardware* reconfigurável é programado por meio de um processo de reconfiguração de sua estrutura. Um algoritmo programado em um *hardware* reconfigurável também é conhecido como *configware* (BECKER; HARTENSTEIN, 2003). Os algoritmos heurísticos, incluindo algoritmos evolucionários são métodos para *Hardware-Software Partitioning* (ZHANG et al., 2008). Esta abordagem tem sido muito explorada em problemas que utilizam algoritmos evolucionários de elevado custo computacional (ZHANG et al., 2008; WANG; LI; DOU, 2008; NEDJAH; MOURELLE, 2005; CABRERA et al., 2004).

Em geral, no problema de indução de regras para ACs utiliza-se como função de *fitness* uma medida obtida pela análise da dinâmica do sistema, através da construção de um diagrama espaço-temporal. Isto torna o algoritmo implementado em *software* muito lento, chegando a dias de processamento, uma vez que não é possível aproveitar as características intrínsecas de paralelismo apresentadas pelos ACs, o que muitas vezes inviabiliza a implementação de tais abordagens. Recentemente Weinert, Benitez, Lopes e Lima (2007) propuseram um *hardware* reconfigurável para a simulação da dinâmica de ACs unidimensionais.

Este capítulo apresenta um modelo híbrido de integração que permite conectar o sistema de simulação da dinâmica implementado em *hardware* (WEINERT; BENITEZ; LOPES; LIMA, 2007), com as devidas modificações e integração de novos módulos, a um sistema de indução de regras implementado em *software*. A integração ocorre via protocolo *Ethernet* e o sistema de indução é construído a partir da adaptação do sistema GEPCLASS (Seção 2.1.3.1) (WEINERT; LOPES, 2006).

5.2 METODOLOGIA

A arquitetura do modelo híbrido (Figura 27) consiste da integração de dois componentes de processamento via interface *Ethernet* e protocolo HTTP (*Hyper Text Transfer Protocol*). O primeiro componente é representado por um computador microprocessado e o segundo por um *hardware* reconfigurável baseado no dispositivo FPGA STRATIX II EP2S60F672C5ES. No computador, Figura 27 (à esquerda), estão presentes os dispositivos comuns de *hardware* (processador, memória, barramentos, controladores e interfaces de comunicação) para realização de processamento sequencial de instruções e comunicação entre dispositivos. Também estão presentes em memória o sistema operacional e um sistema que implementa o algoritmo evolucionário responsável por evoluir regras para o problema de indução de regras de ACs. Na FPGA (Figura 27 à direita) encontram-se os dispositivos de comunicação *Ethernet* formados pela porta de conexão RJ45, pelo *chip* LAN91C111 e seus respectivos barramentos. Acoplou-se a esta FPGA dois componentes de *hardware* dedicado. Um responsável pelo cálculo da função de *fitness* (Módulo Autômato Celular) e outro pelo processamento de instruções de programas escritos em linguagem de alto nível, no caso, o Processador NIOS II. Um módulo de memória armazena os seguintes programas: sistema operacional, servidor *web* e *parsing*.

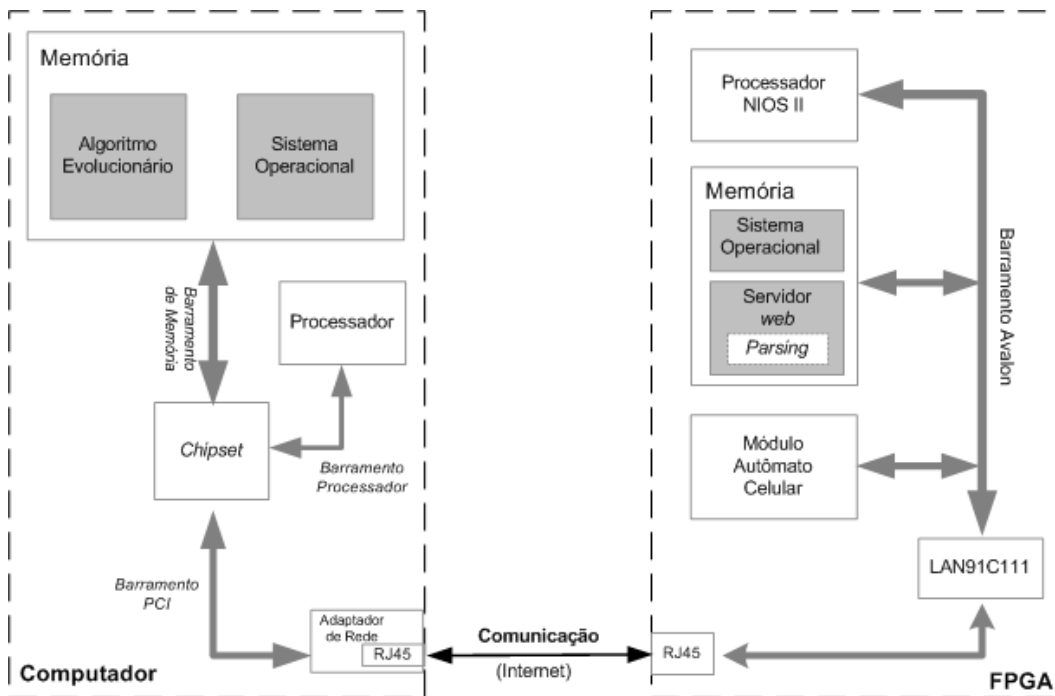


Figura 27: Arquitetura do modelo híbrido.

5.2.1 Algoritmo Evolucionário

O computador é responsável pela execução do sistema destinado a evolução de regras para o problema de classificação pela regra da maioria. O sistema GEPCLASS implementado na Seção 3.2.2.1 é adaptado para ser utilizado neste trabalho.

A adaptação ocorre no módulo responsável pelo cálculo da função de *fitness*. Este foi substituído por um módulo de comunicação que utiliza componentes específicos do Borland® C++Builder® 6.0 para implementar um *browser* simplificado. Através deste *browser*, que utiliza APIs (*Application Programming Interface*) do Internet Explorer, o sistema consegue se comunicar com a FPGA. Este processo de comunicação é descrito na Seção 5.2.2.

O computador também executa o sistema operacional, no caso Microsoft Windows XP, que funciona como uma interface entre a aplicação e o *hardware* permitindo que a mesma utilize todos os recursos de *hardware* de maneira transparente.

É importante ressaltar que uma arquitetura microprocessada não permite a execução real de instruções em paralelo, pois todos os processos são executados sequencialmente.

5.2.2 Comunicação entre Computador e o Sistema Embarcado

A comunicação entre o computador e a FPGA ocorre via interface *Ethernet* utilizando os protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*).

O TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede. Este conjunto pode ser visto como um modelo de camadas, onde cada camada é responsável por um grupo de tarefas, fornecendo um conjunto de serviços bem definidos para o protocolo da camada superior. As 5 camadas que compõem o protocolo TCP/IP são: Aplicação, Transporte, Rede, Enlace e Física (KUROSE; ROSS, 2007).

A camada de Aplicação disponibiliza os serviços ao usuário, esta camada funciona como uma interface de ligação entre os processos de comunicação de rede e as aplicações utilizadas pelo usuário. Nesta camada encontram-se as aplicações como DNS (*Domain Name System*), DHCP (*Dynamic Host Configuration Protocol*), FTP (*File Transfer Protocol*), HTTP (*Hyper-Text Transfer Protocol*), NFS (*Network File System*), SMTP (*Simple Mail Transfer Protocol*), SSH (*Secure Shell*), POP (*Post Office Protocol*), etc. O programa evolucionário que é executado dentro da arquitetura do computador utiliza chamadas disponibilizadas pelo Internet Explorer via HTTP para se comunicar com o servidor *web* que é executado na FPGA.

A camada de Transporte e a camada de Rede encontram-se implementadas nos sistemas

operacionais de ambas as arquiteturas. A camada de Transporte é responsável pela entrega confiável dos dados gerados pelas camadas superiores. Através desta camada é possível realizar a transmissão dos pacotes IP (*Internet Protocol*). Além disto, a camada é responsável pelo controle de fluxo, detecção e correção de erros durante a transmissão. A camada de Rede é responsável pela identificação lógica da interface na rede de computadores. Além disto, possui como uma de suas funcionalidades a realização do roteamento para transmissão dos dados entre os equipamentos.

A camada Enlace define a técnica de acesso ao meio, ou seja, nesta camada é que são definidas as normas de como os dispositivos que compõem a rede deverão acessar a rede para realizar as suas transmissões. A camada Física é responsável pela transmissão dos dados em si. Através dela é que os *bits* são transformados em sinais elétricos e transmitidos considerando-se que o meio utilizado seja por cabeamento ou através de ondas eletromagnéticas quando consideramos as redes sem fio. Na arquitetura utilizada tanto o adaptador de rede presente no computador quanto o *chip* LAN91C111 presente na FPGA possuem características das camadas Enlace e Física. Os conectores RJ45 possuem apenas características da camada Física.

A Figura 28 apresenta uma perspectiva da atuação de cada uma das camadas do modelo TCP/IP dentro da arquitetura híbrida proposta. Os módulos que se encontram destacados em cinza são todos implementados em *software*, enquanto que os demais são implementados em *hardware*. Destaca-se que tanto o adaptador de rede quanto o *chip* LAN91C111 utilizam tecnologias desenvolvidas em *software* combinadas a tecnologias de *hardware*.

Uma vez que a FPGA esteja configurada, operando e conectada à Internet pela interface de rede, a mesma recebe um endereço IP. Então o servidor *web* rodando no NIOS II passa a aguardar por solicitações enviadas a este endereço via protocolo de comunicação HTTP.

| Computador | Camadas TCP/IP | FPGA |
|-------------------|----------------|-----------------|
| Internet Explorer | Aplicação | Servidor web |
| Windows XP | Transporte | <i>μ</i> Clinux |
| | Rede | |
| Adaptador de Rede | Enlace | LAN91C111 |
| | Física | |
| RJ45 | | RJ45 |

Figura 28: As 5 camadas do protocolo TCP/IP dentro da arquitetura híbrida proposta.

5.2.3 Sistema Embarcado - FPGA

O FPGA consiste de um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e realizar roteamento para comunicação entre elas. Na arquitetura proposta estas células lógicas foram utilizadas para composição de 3 blocos: um processador; uma memória e um circuito digital dedicado (Figura 27 à direita). Para a implementação dos sistemas na FPGA utilizou-se o ambiente de desenvolvimento Quartus II da Altera na versão 6.0 e a implementação física utilizou o dispositivo Stratix II EP2S60F672C5ES.

5.2.3.1 Processador NIOS II

O NIOS II é um processador RISC 32 *bits* de alto desempenho. É equivalente a um microcontrolador que inclui uma CPU (*Central Processing Unit*) e uma combinação de periféricos e memórias em um único *chip*.

A Altera¹ disponibiliza uma versão padrão do processador NIOS II para as FPGAs STRATIX II que pode ser anexada a um projeto dentro do ambiente de desenvolvimento Quartus II e posteriormente gravado no dispositivo via interface JTAG (*Joint Test Action Group*). Nesta versão padrão encontram-se disponíveis todos recursos necessários para acesso às interfaces externas. O processador conecta-se a estas interfaces por meio do barramento Avalon.

A grande vantagem desta abordagem é que a presença de um processador dentro da FPGA permite a execução de código escrito em uma linguagem de programação de alto nível e comunicação com as interfaces de saída por intermédio de um sistema operacional. A desvantagem é que consome recursos (células lógicas) da FPGA, logo restam menos recursos para o desenvolvimento da aplicação específica desejada.

5.2.3.2 Memória

Nesta implementação, o processador NIOS II executa 3 módulos implementados em *software*: o sistema operacional, o servidor *web* e o *parser* que ficam armazenados em um módulo de memória.

Utilizou-se o sistema operacional *μClinux*². Este sistema é também conhecido como micro controlador Linux e foi especialmente desenvolvido para microprocessadores sem MMU

¹Disponível em <http://www.altera.com>

²Disponível em <http://uuu.enseirb.fr/kadionik/embedded/uclinux/nios-uclinux.html>

(*Memory Management Unit*). A falta de MMU é um fator comum em microprocessadores de baixo custo. Este sistema é uma solução de código-fonte aberto. Suporta diferentes plataformas de CPU (*Central Processing Unit*), incluindo ColdFire, Axix ETRAX, ARM, Atari 68k, NIOS e outros. Da mesma forma que o Linux, μ Clinux tem um forte suporte a redes incluindo uma pilha TCP/IP e suporte a uma grande variedade de protocolos de rede.

O módulo correspondente ao servidor *web* é escrito em linguagem C++. A própria Altera disponibiliza este servidor. Adaptações foram realizadas sobre este módulo a fim de que o mesmo realize as seguintes operações:

- 1. Receber uma solicitação via método GET.
- 2. Acionar o *parsing* para obtenção dos parâmetros de configuração.
- 3. Enviar os parâmetros ao Circuito Digital Dedicado (Módulo Autômato Celular - Seção 5.2.3.3) para o cálculo da função de *fitness*.
- 4. Receber o resultado do cálculo.
- 5. Formatar este resultado utilizando padrão XML (*Extensible Markup Language*).
- 6. Repassar o código XML para a aplicação solicitante que, no caso, é o algoritmo evolucionário que é executado no computador (Figura 27 à esquerda).

O módulo de *parsing* tem a tarefa de realizar uma segmentação sobre os dados de entrada para obter os parâmetros necessários para configuração do Módulo Autômato Celular. Esta segmentação consiste em subdividir o endereço passado ao servidor *web* via protocolo de comunicação HTTP no formato definido pelo método GET deste protocolo (KUROSE; ROSS, 2007) e alimentar os parâmetros de configuração a partir dos dados extraídos. Os parâmetros de configuração são:

- *endereco_ip*: endereço da interface de rede *Ethernet*;
- *aplicacao*: nome da aplicação que será executada;
- *parametro_1*: número de parâmetros;
- *parametro_2*: tamanho máximo de cada parâmetros;
- *parametro_3*: número de execuções da aplicação;
- *parametro_4*: número de ACs para evolução;

- *parametro_5*: regra no formato hexadecimal;

Os caracteres considerados como separadores entre os parâmetros são: /, ? e &. Na forma:

*http://endereço_ip/aplicacao?parametro_1¶metro_2¶metro_3
¶metro_4¶metro_5*

Um exemplo de solicitação via método GET seria:

http://200.17.96.202/fpga?5&42&1&10000&005f005f005f005fff5f005fff5f

5.2.3.3 Circuito Digital Dedicado

Após a realização da decodificação do cromossomo pelo *parser* (Seção 5.2.3.2) computa-se o *fitness* da regra de transição em questão através de um circuito digital dedicado.

Basicamente, a função de *fitness* consiste da evolução de 10000 ACs, iniciados aleatoriamente, por 200 iterações. Ao final do processo deseja-se saber quantos destes ACs atingiram o comportamento desejado. Neste trabalho, o comportamento desejado é aquele apresentado pelo problema de classificação pela regra da maioria descrito na Seção 2.2.4.1.

O cálculo da função de *fitness* é realizado por uma aplicação construída em *hardware* reconfigurável dentro da FPGA. O diagrama de blocos desta aplicação é apresentado na Figura 29. Esta aplicação foi desenvolvida em VHDL com a utilização do ambiente Quartus II.

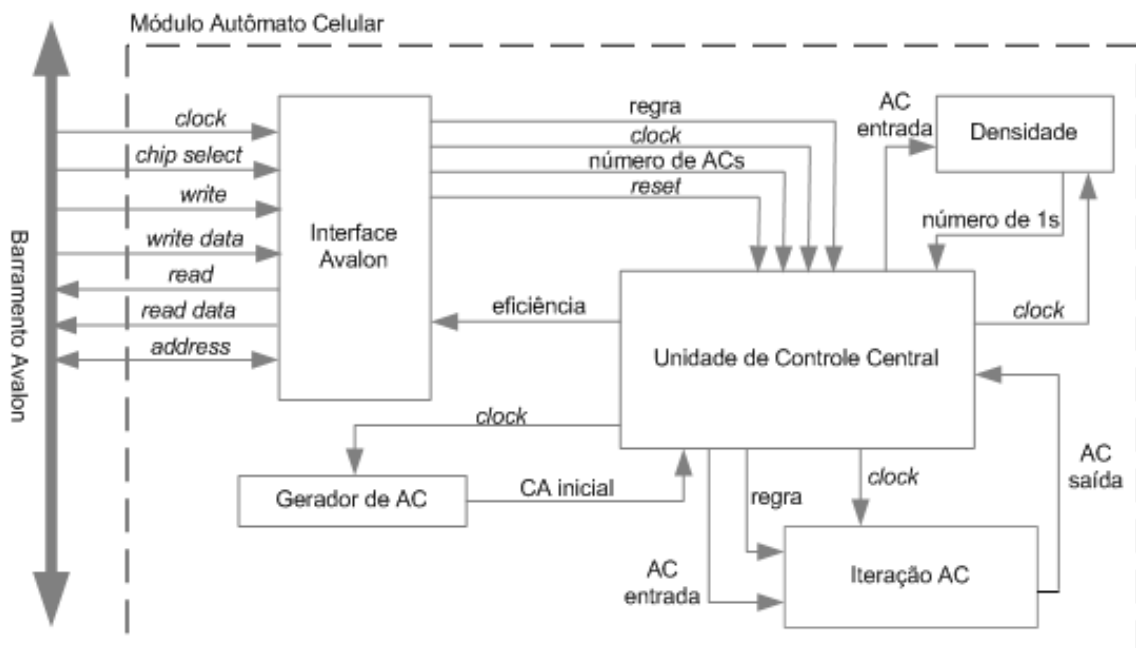


Figura 29: Diagrama de blocos do Módulo Autômato Celular.

Diferente do processador NIOS II que já possui uma interface de comunicação com o barramento Avalon, a aplicação necessita implementar esta interface. É através desta interface que a aplicação se comunica com o barramento e com o processador que está executando o servidor *web*, bem como o módulo responsável pelo *parser*. A interface Avalon possui 7 canais de comunicação com o barramento, como mostrado na Figura 29. Os 7 canais são:

- *clock*: pulsos que sincronizam o funcionamento do *hardware*;
- *chipsselect*: permite que a aplicação baseada em *hardware* se comunique com o barramento Avalon;
- *write*: sinal que libera a leitura de dados no barramento;
- *write data*: canal para o Módulo Autômato Celular realizar leitura de dados no barramento Avalon;
- *read*: sinal que libera a escrita de dados no barramento;
- *read data*: canal para o Módulo Autômato Celular realizar escrita de dados no barramento Avalon;
- *address*: identifica o endereço dos dados que são enviados à interface. A interface possui uma memória interna onde armazena os parâmetros de configuração do sistema. O *address* identifica em quais posições desta memória serão armazenados estes parâmetros.

O *parser* envia para a aplicação dois dados: a regra de transição de estados representada por uma *string* de 128 *bits* e o número de ACs que devem ser simulados no cálculo da função de *fitness*.

A Unidade de Controle Central mostrada na Figura 29 recebe os dados e os armazena em registradores. Uma vez que a simulação é iniciada, dois contadores registram o número de simulações realizadas e o número de acertos. Um acerto significa que a configuração de AC atingida foi igual à configuração esperada. Esta última informação é utilizada para calcular o valor da função de *fitness* da regra de transição enviada.

Necessita-se gerar um grande número de ACs com configuração inicial aleatória para que se realize o processo de simulação do comportamento. O bloco Gerador de AC (Figura 29) é responsável por esta tarefa. Ele está baseado no gerador pseudo-aleatório MLS³ (*Maximum Length Sequence*) de 16 *bits*.

³Disponível em <http://www.ph.ed.ac.uk/jonathan/thesis/node83.html>

A implementação física do gerador de ACs abrange a exploração de um recurso importante de sistemas reconfiguráveis: a possibilidade de paralelismo maciço como será apresentado na Seção 5.2.4.1.

Após o gerador de ACs criar uma nova configuração de AC, a Unidade de Controle Central (Figura 29) encaminha este AC para o bloco Densidade. Este bloco calcula a densidade do AC contando o número de *bits* “1” presentes. Aqui, novamente, ocorre o paralelismo. Então, o número de *bits* “1” é armazenado, para posterior utilização, como densidade inicial do AC.

A simulação da AC é realizada pelo bloco Iteração AC (Figura 29), com base na regra de transição (enviado pelo *parser*) e nos ACs iniciais. A regra de transição é aplicada em paralelo, como será apresentado na Seção 5.2.4.2, a todos os *bits* do AC, de tal forma que, com um único pulso de *clock*, é possível atualizar a configuração do AC. A nova configuração é enviada para a Unidade Central de Controle. Se o número predefinido de iterações é atingido (no nosso caso, 200), a configuração atual do AC é enviada para o bloco Densidade. Caso contrário, ele é reenviado para o bloco de Iteração do AC. No final dessas iterações, a densidade final do AC é obtida e do bloco Unidade de Controle Central decide se a simulação atual alcançou o comportamento esperado ou não, pela seguinte regra:

```

contador_acertos=0;
se (((densidade_inicial ≥ 75)E(densidade_final = 149))OU((densidade_inicial <
75)E(densidade_final = 0))) então
    comportamento_esperado_foi_atingido;
    contador_acertos=contador_acertos+1;
senão
    comportamento_esperado_nao_foi_atingido;
fim

```

Se o comportamento esperado foi atingido, o contador de acertos é incrementado. Todo o processo é repetido até que o número programado de simulações seja atingido (no nosso caso, 10000). Finalmente, a porcentagem de acertos (*fitness*) é calculada e enviada para o servidor *web* através da Interface Avalon.

5.2.4 Paralelismo

A utilização de um *hardware* dedicado como a FPGA Stratix II justifica-se pelo elevado nível de paralelismo que se consegue explorar em aplicações específicas.

O problema de indução de regras em ACs da forma como foi tratado neste trabalho destaca a exploração de dois pontos onde tarefas repetitivas e independentes podem ser executadas simultaneamente trazendo grande aumento de desempenho para aplicação como um todo.

As duas seções subseqüentes apresentam, mais detalhadamente, duas tarefas paralelizadas neste trabalho. Além destas, outra tarefa paralelizada é a computação da densidade do AC, que basicamente, consiste da soma de todos os valores que compõem as células do AC. Em uma solução implementada em *software* cada *bit* deve ser analisado e um contador incrementado. Por outro lado, em uma solução implementada em *hardware* reconfigurável, todos os *bits* são somados em um único pulso de *clock*.

5.2.4.1 Geração de ACs Aleatórios

Para facilitar a compreensão dos benefícios que a lógica reconfigurável trouxe para a tarefa de geração de ACs aleatórios compara-se duas abordagens: a implementada neste capítulo baseada no gerador pseudo-aleatório MLS e outra hipotética, baseada no algoritmo Mersenne Twister (MATSUMOTO; NISHIMURA, 1998) implementada em *software*. Como o tamanho dos ACs utilizados é de 149 *bits*, se uma solução de *software* fosse empregada aqui, seria necessário gerar números aleatórios para cada *bit* e decidir se a referida célula do AC receberia valor “0” ou “1”. Em nossa abordagem, usando lógica reconfigurável, 149 *bits* aleatórios são gerados em paralelo com um único pulso de *clock*, economizando tempo de processamento e acelerando a simulação.

Em um modelo hipotético e sequencial de processamento, para cada AC gerado executam-se os procedimentos referentes ao fluxograma apresentado na Figura 30.

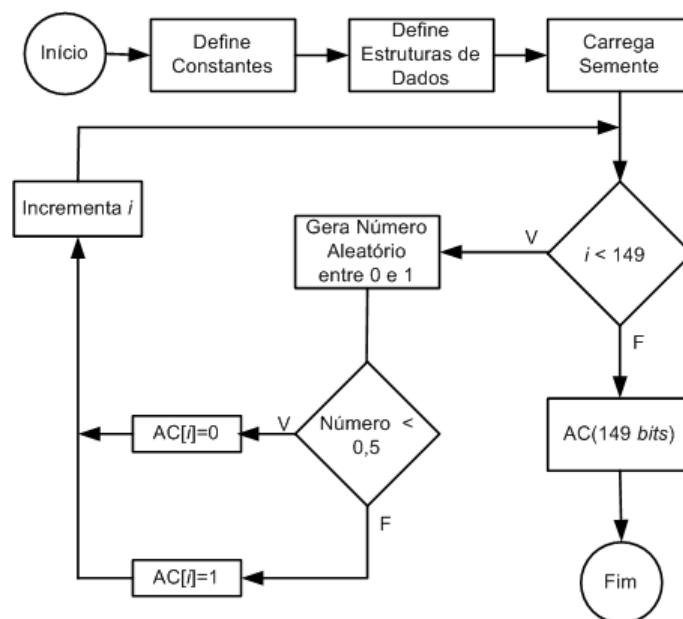


Figura 30: Fluxograma de um gerador aleatório sequencial de ACs.

O processo de geração de um AC aleatório, de maneira sequencial, consiste de:

1. Definição de valores para constantes que serão utilizadas pelo gerador.
2. Definição das estruturas de dados que serão utilizadas pelo gerador.
3. Carregar a semente inicial do gerador.
4. Iniciar um ciclo que se repete por 149 vezes.
 - (a) Gerar um número aleatório.
 - (b) Testar o valor do número gerado.
 - (c) Atualizar o contador de controle do ciclo.
5. Disponibilizar o AC.

É difícil mensurar o número exato de ciclos de *clock* necessários para a geração aleatória de um AC nesta abordagem. No entanto, observando o código fonte escrito em linguagem C referente ao bloco Gera Número Aleatório entre 0 e 1 (Figura 30) fica evidente que o número de operações a serem executadas pelo processador é enorme.

```
float myrnd() {
    unsigned long y;
    static unsigned long mag01[2]={0x0, MATRIX_A};

    if (mti >= N) {
        int kk;

        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1];
        mti = 0;
    }

    y = mt[mti++];
    y ^= TEMPERING_SHIFT_U(y);
    y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
    y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
    y ^= TEMPERING_SHIFT_L(y);

    return ( (float)y / (unsigned long)0xffffffff );
}
```

Figura 31: Código fonte da rotina de geração de números aleatórios do Mersenne Twister.

Deixando de lado as variáveis e atentando-se apenas para operações como: atribuição de valores a variáveis, somas, subtrações, exponencial, operações relacionais, etc., pode-se assumir que centenas de ciclos de *clock* serão gastos neste procedimento. Ressaltando-se, ainda, que o algoritmo apresentado na Figura 31 é executado 149 vezes para cada AC, estes ciclos podem facilmente chegar a milhares.

A Figura 32 representa o fluxograma do gerador paralelo de ACs implementado na FPGA. Este gerador representa uma expansão do bloco Gerador de AC apresentado na Figura 29. A abordagem paralela adotada para geração aleatória do AC utiliza apenas 3 ciclos de *clock* para geração de um AC. Isto é possível devido a 149 replicações do gerador MLS. No primeiro pulso de *clock* todos os geradores recebem um valor de semente. No segundo, todos os geradores são ativados e retornam um *bit* de valor 0 ou 1 e no terceiro pulso, realiza-se a concatenação dos resultados e gera-se um AC de 149 *bits*.

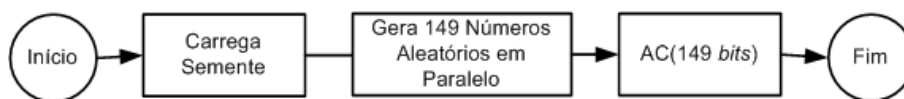


Figura 32: Fluxograma de um gerador aleatório paralelo de ACs.

5.2.4.2 Processo de Atualização das Células do Autômato Celular

A Seção 2.2.1.3 apresentou como funciona o processo de evolução de um AC ao longo de t iterações. Este processo definido de maneira sequencial, pode ser visualizado no fluxograma da Figura 33.

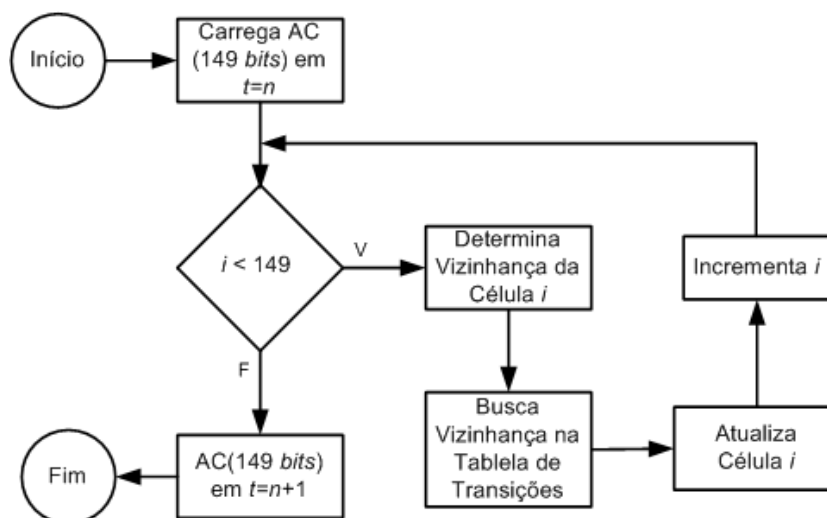


Figura 33: Fluxograma referente a uma iteração sequencial de ACs.

O processo de evolução de um AC de uma iteração t para uma iteração $t+1$ consiste de:

1. Carregar o AC na iteração t .
2. Iniciar um ciclo de atualização das células que se repete por 149 vezes.
 - (a) Determinar a vizinhança da célula.
 - (b) Encontrar a transição referente à vizinhança.
 - (c) Atualizar a célula.
 - (d) Atualizar o contador de controle do ciclo.
3. Disponibilizar o AC na iteração $t+1$.

Novamente não é possível determinar o número de ciclos de *clock* que serão gastos durante um processo de atualização das células do AC. Porém, observando-se o fluxograma da Figura 33 é possível estimar o custo computacional da operação, uma vez que serão executadas várias operações de busca e comparação em diferentes estruturas de dados. Assim, torna-se plausível considerar que centenas de ciclos de *clock* serão despendidos no processo, ainda sem considerar as 149 repetições necessárias para atualização de todas as células que constituem o AC.

A Figura 34 apresenta o fluxograma referente à solução paralela adotada para o procedimento de atualização das células do AC. Do mesmo modo como ocorreu na solução adotada para geração aleatória do AC 3 ciclos de *clock* serão gastos em todo procedimento. No primeiro ciclo de *clock* determinam-se as vizinhanças de todas as 149 células. No segundo *clock* determinam-se as transições referentes a cada umas das 149 vizinhanças e no último *clock* atualizam-se todas as células do AC.

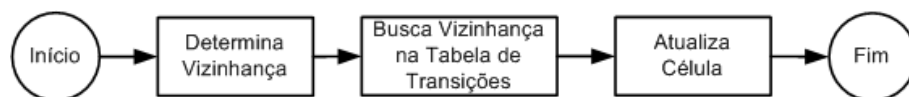


Figura 34: Fluxograma referente a uma iteração paralela de ACs.

O diagrama em blocos apresentado na Figura 35 apresenta uma visão interna do processo de paralelização dos procedimentos descritos na Figura 34. Para simplificar o desenho são apresentados apenas um bloco Busca Vizinhança na Tabela de Transições e um bloco Atualiza Célula, mas na arquitetura real existem 149 blocos Busca Vizinhança na Tabela de Transições, cada um conectado a seu respectivo bloco Atualiza Célula.

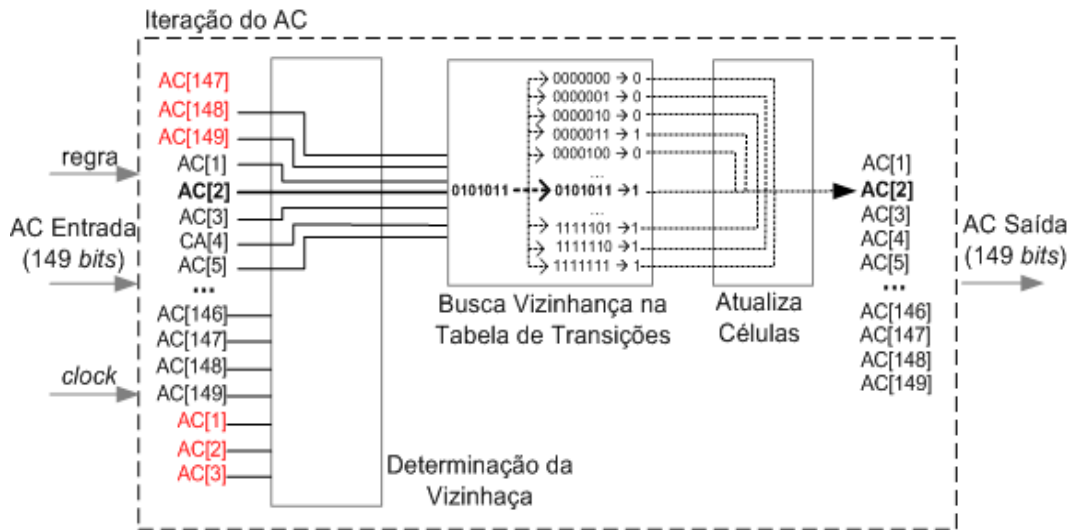


Figura 35: Diagrama de blocos do processo paralelo de iteração dos ACs.

5.3 RESULTADOS

Uma simples comparação entre dois sistemas foi realizada para avaliar a eficiência em termos de tempo de processamento da arquitetura híbrida proposta. O primeiro sistema é totalmente executado em *software* e o segundo compreende a abordagem proposta neste capítulo, ou seja, a integração entre *hardware* e *software*. O computador utilizado para ambos os sistemas apresenta a seguinte configuração de *hardware*: um processador de 2,8 GHz e 1 GB de RAM. É importante salientar que a FPGA trabalha a 25MHz, sendo, portanto um *clock* 112 vezes mais lento do que o do computador.

O sistema operacional utilizado no computador foi o Windows XP, e as aplicações foram implementadas em linguagem C++ sobre a plataforma de desenvolvimento C++Builder® 6.0 da Borland® sem a utilização de otimização de código. Os parâmetros de configuração do algoritmo evolucionário e do problema de classificação pela regra da maioria foram exatamente os mesmos para ambas as abordagens. Foram evoluídos 50 indivíduos por 50 gerações, onde utilizou-se a simulação de 10000 ACs por 200 iterações como função de *fitness*. Os resultados obtidos encontram-se reportados na Tabela 13.

Tabela 13: Comparação entre diferentes abordagens.

| Abordagem | Arquitetura | Tempo de Processamento |
|------------------------------------|-------------|------------------------|
| Somente <i>Software</i> | PC | 283615 segundos |
| <i>Hardware-Software</i> (Híbrido) | PC+FPGA | 1115 segundos |

5.4 DISCUSSÃO

A arquitetura de Von Neumann própria dos sistemas com microprocessadores não é capaz de implementar um modelo de processamento paralelo real. Por outro lado, sistemas de *hardware* reconfigurável apresentam grande potencial em termos de desempenho e adaptabilidade, oferecendo mecanismos de paralelismo real. No entanto, estes mecanismos somente podem ser utilizados em aplicações que possuam rotinas paralelizáveis, ou seja, rotinas que não possuem nenhuma relação de dependência e podem ocorrer ao mesmo tempo.

Um AC é um exemplo de aplicação que possui rotinas totalmente independentes, ou seja, paralelizáveis. Outra característica interessante dos ACs é que são estruturas simples e podem ser facilmente construídas e replicadas pela manipulação de células lógicas no FPGA.

As Seções 5.2.4.1 e 5.2.4.2 enfatizam o potencial ganho de desempenho referente à abordagem paralela aplicada sobre a tarefa de geração aleatória de AC e de atualização das células do AC. Foi notória a diminuição de pulsos de *clock* necessários para o desenvolvimento destas tarefas. A comparação entre diferentes plataformas de *hardware* não é completamente justa, pois em um microcomputador existem vários processos rodando e disputando os recursos de *hardware* disponíveis, mas é importante enfatizar a diferença dramática alcançada no tempo de processamento das aplicações conforme apresentado na Tabela 13.

A principal limitação desta abordagem encontra-se relacionada a inflexibilidade de adaptação para diferentes problemas e até mesmo diferentes configurações de um mesmo problema. A maioria das alterações no modelo demandam um alto custo de implementação e reprogramação da FPGA. Outro fator limitador é o número de células lógicas disponíveis na FPGA. Dependendo da complexidade da aplicação que se deseja implementar não existem células lógicas suficientes para contemplar a programação exigida.

6 PIRGEP: UM MODELO COMPUTACIONAL PARALELO PARA INDUÇÃO DE REGRAS

6.1 INTRODUÇÃO

Este capítulo apresenta uma arquitetura paralela de processamento baseado no algoritmo de PEG. O algoritmo foi genericamente codificado para ser aplicado tanto sobre problemas de classificação quanto problemas de indução de regras de ACs e foi batizado de PIRGEP (*Parallel Induction Rules with Gene Expression Programming*).

6.2 METODOLOGIA

6.2.1 Arquitetura Paralela

O PIRGEP foi totalmente desenvolvido em linguagem C padrão ANSI sobre a plataforma Linux. O paralelismo foi implementado através da biblioteca MPICH2 (GROPP; LUSK; THAKUR, 1999)¹ que codifica os conceitos do MPI (SNIR et al., 1997). Esta biblioteca cria um ambiente virtual composto por diversas unidades de processamento. Um conjunto de funções de comunicação e sincronismo permite que tais unidades de processamento comuniquem-se entre si. Esta biblioteca implementa o paradigma de programação paralela chamado passagem de mensagem (ROOSTA, 1999), em que a comunicação entre os processadores acontece por meio de mensagens enviadas uns aos outros, de forma restrita ou aberta a todos que estiverem conectados. Os processadores não compartilham memória, assim toda e qualquer informação deve ser transmitida pelo ambiente MPI.

Este modelo de paralelismo adequa-se perfeitamente a uma estrutura de processamento composta por diversos microcomputadores ligados em rede. Os microcomputadores não necessariamente precisam compartilhar um mesmo padrão de configuração de *hardware*. No entanto, a padronização facilita o processo de instalação e manutenção do ambiente, bem como facilita o balanceamento de carga entre as unidades de processamento. A esta estrutura de processamento atribui-se o nome *cluster*. A interconexão entre os microcomputadores que compõem o *cluster*

¹Disponível em: <http://www.mcs.anl.gov/research/projects/mpich2/>

se dá através de uma rede *Ethernet*.

O *cluster* utilizado neste trabalho é formado por 30 microcomputadores *Pentium Quad Core EM64T* de 2.4 GHz com 2 GB de memória RAM (*Random Access Memory*) e um servidor idêntico com 4 GB de memória RAM, mostrado pela Figura 36. Somente o servidor possui uma interface para comunicação externa, ou seja, acesso a Internet. Os demais microcomputadores estão interconectados em uma rede *Ethernet* dedicada. Isto diminuiu o tráfego de pacotes aumentando assim, a eficiência do processo de comunicação entre as unidades de processamento que compõem o *cluster*. Todos os microcomputadores utilizam o sistema operacional Open Suse Linux 10.0 e a versão 1.0.6 do MPCHI2.0.



Figura 36: *Cluster* de processamento.

Dentro do ambiente MPI todas as tarefas são subdivididas em processos. Cada processo é então executado por uma unidade de processamento. Uma unidade de processamento pode executar mais de um processo. No entanto, o aumento do número de processos por unidade de processamento deve ser feito com parcimônia, pois pode levar a queda de desempenho ocasionado por um balanço de carga mal dimensionado.

O PIRGEP implementa uma arquitetura lógica de dimensionamento dinâmico onde um processo mestre coordena n processos escravos. Um conjunto formado por um processo mestre e n processos escravos definem uma ilha de processamento. O PIRGEP também tem a propriedade de trabalhar com múltiplas ilhas que cooperam entre si para a solução do problema, conforme apresentado na Figura 37.

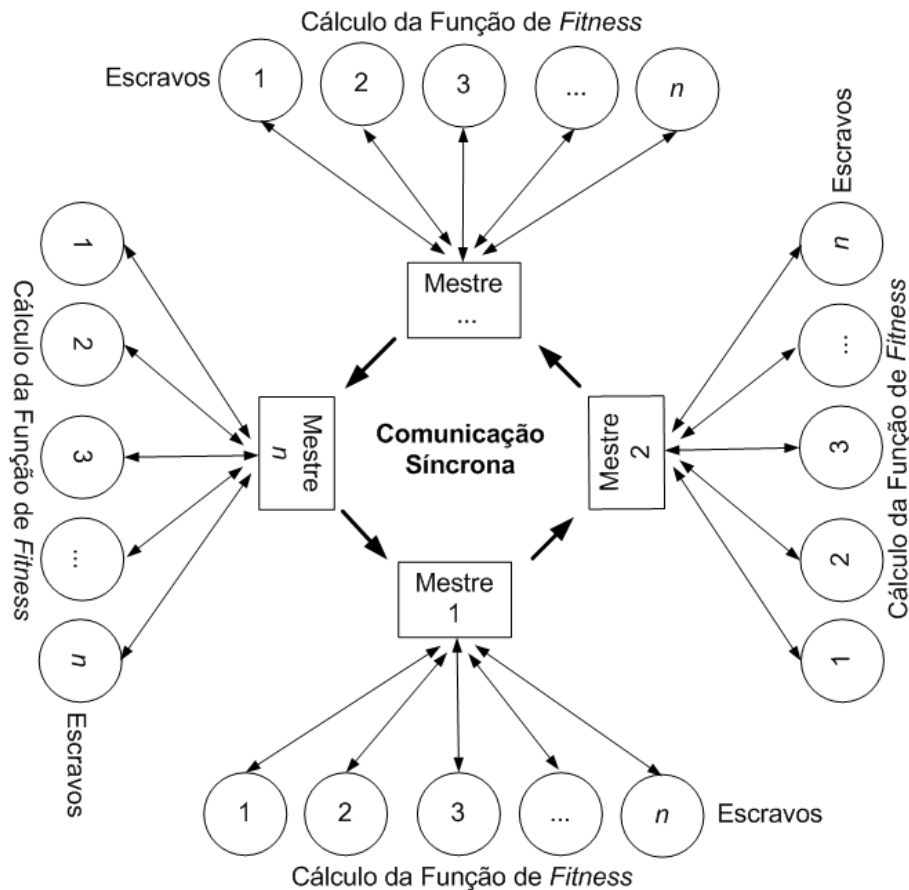


Figura 37: Arquitetura mestre-escravo.

Na abordagem adotada o processo mestre tem a incumbência de executar o algoritmo evolucionário bem como controlar todo o processo de comunicação entre ele e seus respectivos processos escravos. Todo processo escravo executa a mesma tarefa que, neste caso, é o cálculo da função de *fitness*. Caso opte-se por uma arquitetura composta por mais de uma ilha cria-se um processo colaborativo. Todas as ilhas executam exatamente as mesmas tarefas. No entanto, de tempos em tempos, os processos mestres de cada uma das ilhas comunicam-se entre si trocando informações que podem melhorar a solução de determinado problema. Este processo colaborativo é chamado de co-evolução (FREITAS, 2002). A co-evolução emerge de um processo de simbiose, onde simbiose define-se como relacionamentos entre indivíduos de diferentes populações, sendo que a aptidão de um indivíduo afeta diretamente a aptidão de outro.

6.2.2 Modelagem do PIRGEP

A modelagem do PIRGEP pode ser representada pelo fluxograma da Figura 38.

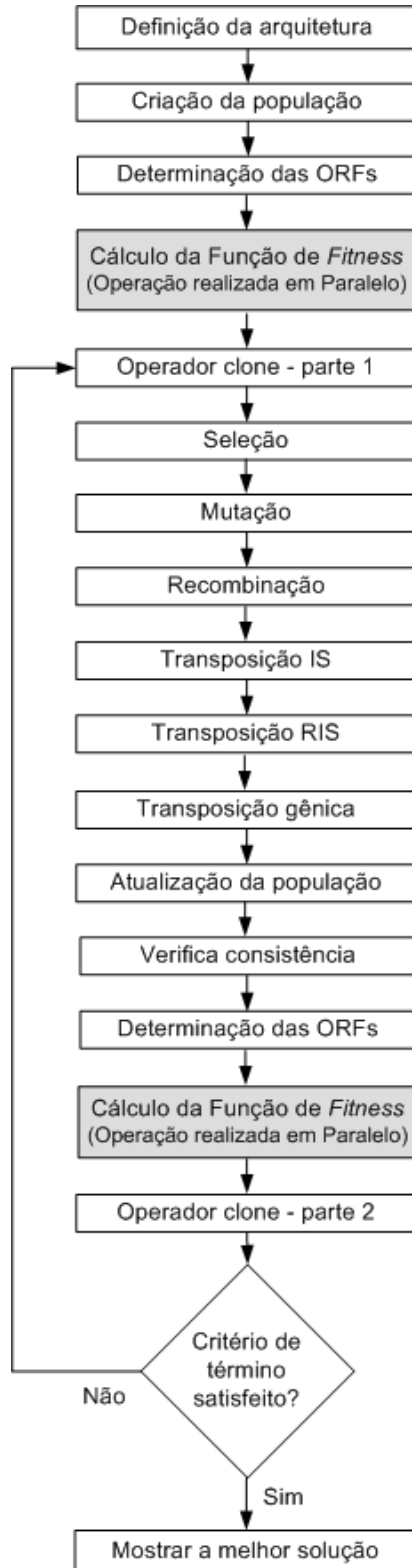


Figura 38: Fluxograma do PIRGEP.

Todos os parâmetros que alimentam o PIRGEP são configurados em um arquivo próprio de configuração de parâmetros.

Seguindo o fluxograma da Figura 38 como primeira tarefa tem-se a definição da arquitetura lógica que será utilizada para execução do algoritmo evolucionário. Basicamente, dois parâmetros definem esta estrutura. O primeiro é o número de processos mestres que serão criados e o segundo o número de processos escravos subordinados a cada um dos referidos processos mestres. Por exemplo, se uma arquitetura for constituída de um único processo mestre e 10 processos escravos, teremos um sistema paralelo formado por 11 processos. Frisa-se que a utilização de um único processo mestre elimina a possibilidade de co-evolução, pois o sistema constitui-se de uma única ilha de processamento. No entanto, para uma arquitetura constituída de 5 processos mestres e 10 processos escravos para cada processo mestre, tem-se um sistema paralelo formado por 55 processos que se subdividem em 5 ilhas de processamento onde os processos mestres interconectam-se numa topologia em formato anel conforme apresentado na Figura 37. A comunicação entre processos mestres ocorre sempre em sentido unidirecional de forma síncrona sob supervisão de um dos processos mestres. O MPI tentará destinar cada processo para uma unidade de processamento distinta. Se existirem mais processos do que unidades de processamento, algumas unidades receberão um número maior de processos. O MPI não oferece nenhum mecanismo para balanceamento automático de carga, sendo assim, é responsabilidade do desenvolvedor implementar tais mecanismos.

Neste trabalho o balanceamento de carga relaciona-se diretamente com o número de indivíduos que compõem a população de soluções do algoritmo evolucionário e o número de processos escravos relacionados a um processo mestre. Uma vez que cada indivíduo é submetido ao processo de cálculo da função de *fitness* e este cálculo é realizado pelos processos escravos, procura-se distribuir, por igual, o número de indivíduos entre os processos escravos. Quando a divisão entre o número de indivíduos e o número de processos escravos não é exata, alguns processos escravos recebem mais indivíduos do que outros para que todos sejam processados. Infelizmente, isto gera um pequeno gargalo no sistema de comunicação síncrono, pois alguns processos terminam antes do que outros e precisam aguardar até que o processo mestre solicite o resultado do processamento.

Os indivíduos são criados aleatoriamente segundo o conjunto de funções e terminais especificados no arquivo de configuração. As regras de preenchimento das estruturas cabeça e cauda, são as mesmas do GEPCLASS (Seção 2.1.3.1). Esta abordagem paralela não contempla a utilização de indivíduos com tamanhos variáveis. O algoritmo utilizado como gerador de números aleatórios é o Mersenne Twister (MATSUMOTO; NISHIMURA, 1998). O processo

de determinação das ORFs também é igual ao implementado no GEPCLASS (Seção 2.1.3.1, Figura 4).

Os procedimentos de criação da população inicial e determinação das ORFs são inteiramente executados pelo processo mestre. Concluída esta etapa o processo mestre distribui os indivíduos para os processos escravos e estes executam o cálculo da função de *fitness*. Duas funções diferentes de *fitness* estão disponíveis. Uma para problemas genéricos de classificação e outra para indução de regras de ACs. As Seções 6.2.2.1 e 6.2.2.2 apresentam os detalhes de implementação destas duas abordagens. Após a conclusão do cálculo da função de *fitness* os processos escravos aguardam por uma solicitação de seu processo mestre a fim de retornarem o resultado obtido.

Retornado o controle para o processo mestre inicia-se o processo de evolução propriamente dito. O primeiro operador genético denomina-se clone parte 1. Ele guarda o indivíduo de melhor *fitness* para devolvê-lo à população no final da aplicação dos operadores genéticos. Esta devolução é realizada pelo operador clone parte 2 que substituiu o indivíduo de menor valor de *fitness* presente na população pelo indivíduo armazenado. Este procedimento também é conhecido como elitismo. E ele pode ou não ser executado, dependendo da configuração escolhida.

Na sequência, os indivíduos são selecionados para sofrerem modificações pelos diversos operadores genéticos. Dois métodos de seleção estão disponíveis: roleta (GOLDBERG, 1989) e torneio estocástico (KOZA, 1992). Os operadores implementados são: mutação, recombinação, transposição IS, transposição RIS e transposição gênica. Detalhes a respeito destes operadores e das propriedades de fechamento, que garantem apenas a manipulação de indivíduos válidos estão descritos na Seção 2.1.3.1.

Após a aplicação dos operadores genéticos obtem-se uma nova população de indivíduos. A fim de garantir que todos os indivíduos representem soluções válidas também implementou-se um módulo de verificação de consistência. Basicamente este módulo analisa se o material genético apresentado na região codificadora do gene, ou seja, até sua ORF, gera realmente uma solução passível de ser mapeada para uma regra. Caso se detecte alguma inconsistências num indivíduo, este é descartado da população e um novo indivíduo aleatoriamente criado o substitui.

Então os indivíduos são novamente enviados para os processos escravos a fim que as funções de *fitness* sejam computadas. Por fim, testa-se o critério de término, que é o número máximo de gerações. Caso o mesmo tenha sido atingido apresenta-se a melhor solução encontrada, senão retorna-se ao processo evolutivo.

6.2.2.1 Modelo para Problemas Genéricos de Classificação

As regras de classificação são, em geral expressas na forma “SE A ENTÃO C”. O antecedente “A” pode ser formado por um conjunto de condições e “C” representa o conseqüente da regra.

As condições das regras são t -uplas da forma $A_i Op V_{ij}$, onde A_i é o i -ésimo atributo, Op é um operador relacional (normalmente =, \neq , > ou <) e V_{ij} é o j -ésimo valor que pertence ao domínio do atributo A_i . Para fazer a combinação das possíveis condições são utilizados operadores lógicos (AND, OR ou NOT).

O conseqüente da regra consiste de uma simples condição na forma $\langle M_i = V_{ij} \rangle$, onde M_i é um possível atributo meta e V_{ij} é um dos possíveis valores do domínio do atributo selecionado.

Neste modelo a codificação dos indivíduos da população segue a abordagem de Michigan (FREITAS, 2002). Segundo esta abordagem um indivíduo representa uma única regra dentro do sistema de classificação que se deseja obter.

A Figura 39 exemplifica o modelo de codificação adotado. O processo de construção e interpretação deste modelo já foi descrito na Seção 2.1.3.1.

Um indivíduo é formado por um único cromossomo, o que faz destas duas denominações, sinônimos. A Figura 39 apresenta um cromossomo formado por 2 genes, no entanto, este número é variável à escolha do usuário, bem como o tamanho da cabeça de cada gene, que neste exemplo é de tamanho 3. Os genes são interligados por meio de uma função de ligação, que pode ser AND ou OR. A solução candidata obtida a partir da interpretação da AE sempre retorna uma regra que cobre os exemplos da classe dita positiva. A classe positiva é a classe que se pretende classificar e todas as outras são ditas classes negativas.

No arquivo de configuração do sistema encontram-se definidos os atributos pertencentes ao problema e o número de classes do mesmo. Um segundo arquivo contém os casos de *fitness* que são exemplos de diferentes combinações entre valores de atributos para diferentes classes. Uma análise deste arquivo define o domínio dos atributos. O domínio de um atributo constituiu-se dos valores que o mesmo pode assumir. Se este for do tipo categórico, poderá assumir um valor dentre uma lista que contém todos os valores atribuídos a ele. Caso seja contínuo, poderá assumir um determinado valor dentro de um intervalo de valores atribuídos a ele.

Este modelo permite a construção de um classificador hierárquico (FREITAS, 2002). Para isto, deve-se definir quais classes serão classificadas nos diferentes níveis da hierarquia. Por exemplo, considere um problema de classificação formado por 3 classes: A , B e C , e um arquivo

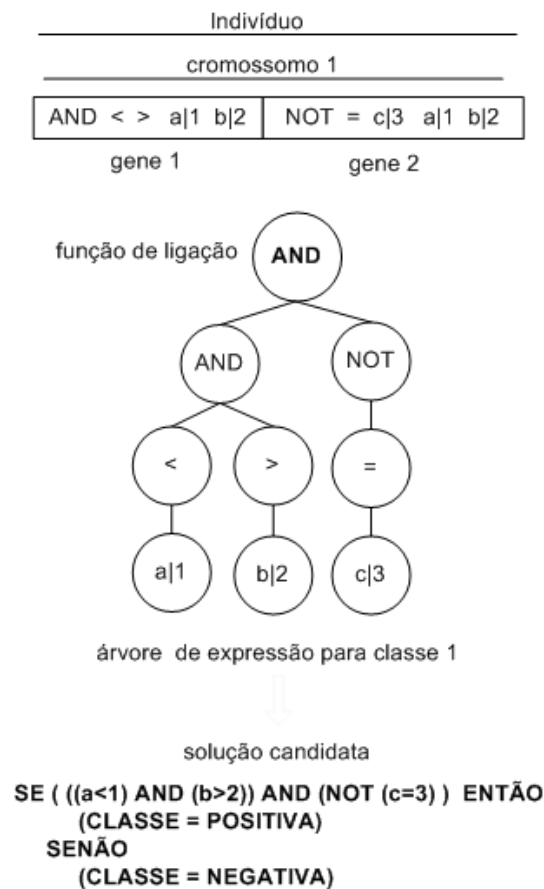


Figura 39: Codificação para indução de regras em problemas de classificação.

de casos de *fitness* contendo 2500 exemplos. Destes, 1200 da classe *A*, 800 da classe *B* e 500 da classe *C*. A definição de qual classe será prevista num determinado nível hierárquico é do usuário. No entanto, é importante manter sempre um mesmo critério, como por exemplo, o número de amostras de cada uma das classes. Sendo assim, no primeiro nível hierárquico teria-se a classe *A* como representante da classe positiva e as classes *B* e *C* como representantes da classe negativa. Para o segundo nível hierárquico ficaram apenas com as classes *B* e *C*, haja visto que a classe *A* já foi classificada. Sendo a classe *B* representante da classe positiva, e a classe *C* da classe negativa. O resultado desta hierarquia de classificação é uma regra que cobre a classe *A*, uma outra que cobre a classe *B*, e a classe *C* passou a ser a classe padrão. Ou seja, se o exemplo não pertencer à classe *A*, nem à classe *B* é classificado como sendo integrante da classe *C*.

A função de *fitness* utilizada é aquela apresentada pela Equação 1 (Seção 2.1.3.1). Esta função é amplamente utilizada pela literatura. Ela sempre retorna valores pertencentes ao intervalo $[0..1]$, onde valores próximo de 1 indicam soluções de alta qualidade. Esta função independe do problema de classificação que está sendo tratado, assim adapta-se perfeitamente

ao propósito deste trabalho que é o desenvolvimento de uma metodologia genérica para classificação. Outro aspecto interessante é que esta função considera tanto a habilidade da regra em reconhecer exemplos que realmente pertencem a sua classe quanto em rejeitar exemplos que realmente não pertencem a sua classe.

6.2.2.2 Modelo para Indução de Regras de Autômatos Celulares

O modelo para indução de regras de ACs também utiliza regras na forma “SE A ENTÃO C” como definido na Seção 6.2.2.1.

Este modelo vislumbra a generalidade, ou seja, deve ser capaz de induzir regras para diferentes problemas modelados sobre o conceito de ACs. A idéia central consiste em, induzir, dado um estado inicial e um estado final para um determinado AC, bem como sua configuração de vizinhança, uma regra de transição que represente tal comportamento. Os estados inicial e final do AC são introduzidos no sistema por arquivos tipo texto.

Neste modelo a codificação dos indivíduos da população segue a abordagem de Pittsburgh (FREITAS, 2002). Segundo esta abordagem um indivíduo representa um conjunto de regras dentro do sistema de classificação. Sendo assim, cada indivíduo passa a ter mais de um cromossomo, onde cada cromossomo codifica uma regra, como explicado na sequência.

Como já mencionado, o objetivo é encontrar uma regra que represente o comportamento desejado. Para exemplificar o procedimento, utiliza-se um AC unidimensional onde o conjunto de estados possíveis para cada célula do AC é representado por $\Sigma = 1, 2, 3$ e a vizinhança a ser considerada, tem raio igual a 1.

A Seção 2.2.1.2 apresentou as formulações que permitem o cálculo do número de combinações de vizinhança, neste caso 27, e o número de regras que se aplicam a tal configuração, que no caso, é de aproximadamente 7×10^{12} .

A partir destes dados torna-se possível construir a tabela de transição de estados (Tabela 14) aplicável a este exemplo. A coluna “regra” ainda encontra-se indefinida, pois depende da regra codificada pelo indivíduo apresentado na Figura 40 .

Neste exemplo, Σ é composto por 3 elementos. Cada elemento é definido como uma classe. Ou seja, fazendo uma analogia com a abordagem para indução de regras de classificação (Seção 6.2.2.1), depara-se com um problema de classificação hierárquica composto por 3 classes. Diferentemente dos problemas de classificação, os problemas de indução de regras de ACs não possuem casos de *fitness*. Assim, a idéia central consiste em se partir da regra codificada no indivíduo e preencher a coluna “regra” da Tabela 14 segundo a abordagem de Pittsburgh.

Tabela 14: Vizinhanças para uma regra de transição ainda desconhecida.

| transição | vizinhança | regra |
|-----------|------------|-------|
| | a b c | |
| 0 | 111 | ? |
| 1 | 112 | ? |
| 2 | 113 | ? |
| ... | ... | ... |
| 24 | 331 | ? |
| 25 | 332 | ? |
| 26 | 333 | ? |

A codificação dos indivíduos depende da definição do conjunto de terminais e seus respectivos domínios. Os terminais representam a vizinhança. Neste exemplo cada vizinhança é composta por 3 elementos, assim serão utilizados 3 terminais. Dá-se um rótulo para cada terminal, por exemplo: a, b e c (Tabela 14). Todos os terminais possuem o mesmo domínio. Este domínio é definido pelos estados que as células do AC podem assumir, neste exemplo: 1, 2 e 3.

As classes do problema também são representadas pelos estados que as células do AC podem assumir. Neste exemplo, tem-se um indivíduo que codifica 3 regras de maneira hierárquica, sendo a primeira responsável por cobrir combinações de vizinhança que mapeiam o estado 1. A segunda, executada somente se a primeira não for aplicável, é responsável por cobrir combinações de vizinhança que mapeiam o estado 2. A terceira regra, executada somente se a segunda não for aplicável, é a regra padrão, ou seja, cobre todas as combinações de vizinhança que não foram cobertas pela primeira e segunda regras. A regra padrão não é codificada no indivíduo. A ordem hierárquica de cobertura dos estados das células é inserida como dado de entrada no sistema.

A Figura 40 apresenta um indivíduo codificado. O processo de construção e interpretação deste modelo já foi descrito na Seção 2.1.3.1. Cada cromossomo que compõe o indivíduo é mapeado para sua respectiva AE. Neste exemplo, os cromossomos são formados por 2 genes, mas este parâmetro é variável, bem como o tamanho da cabeça, que está com valor 3. A solução candidata, a qual vai permitir o preenchimento da coluna “regra” na Tabela 14 apresenta-se como uma regra hierárquica, onde o primeiro nível representa o primeiro cromossomo e o segundo nível o segundo cromossomo. Por fim, adiciona-se a regra padrão que cobre a classe 3.

A Tabela 15 apresenta a regra (terceira coluna) que foi obtida a partir da aplicação da soluções candidata sobre as diferentes combinações de vizinhança, que será utilizada para simular a dinâmica do AC.

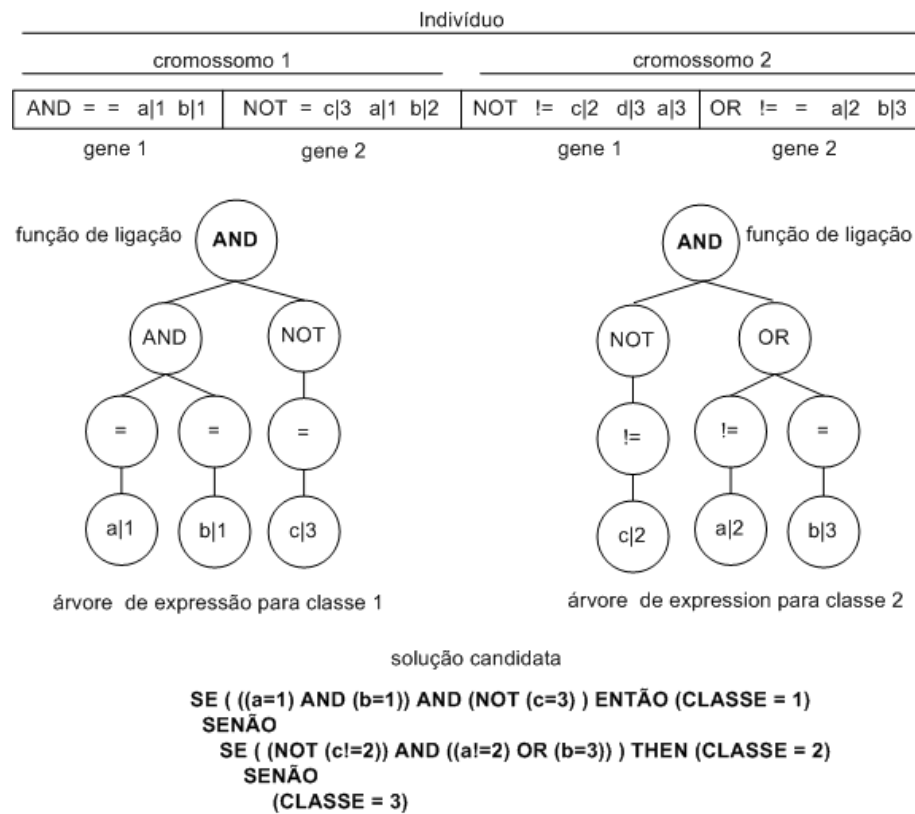


Figura 40: Exemplo de codificação para indução de regras de transição em ACs.

A função de *fitness* implementada precisa ser genérica e aplicável a qualquer dimensão de AC independentemente do alfabeto utilizado. A função proposta que é definida pela Equação 7 utiliza a mesma idéia da medida da distância de Hamming (PETERSON; WELDON, 1972) apresentada na Seção 4.2.1 (Equação 6). Ela quantifica o número de semelhanças entre dois ACs, célula a célula. Os ACs devem ter a mesma dimensão e o mesmo número de células. Então divide-se este número pelo número total de células de um dos ACs. Se o resultado da divisão for 1 tem-se uma semelhança completa, se for 0 a semelhança inexistente, e um valor entre 0 e 1 indica o grau de semelhança entre os dois ACs. No caso, os dois ACs são representados pela configuração de AC desejada e a configuração obtida pelo sistema a partir de uma dada configuração inicial, no processo de simulação da dinâmica, tal que

$$fitness = \frac{\sum_{i=1}^{d_3} (\sum_{j=1}^{d_2} (\sum_{k=1}^{d_1} (SE(ACD_{i,j,k} = ACO_{i,j,k}; 1; 0))))}{d_3 * d_2 * d_1} \quad (7)$$

onde,

- d_3 , d_2 e d_1 : tamanho da dimensão 3, 2 e 1 respectivamente;
- ACD : autômato celular desejado;

Tabela 15: Determinação da regra de transição.

| transição | vizinhança | regra |
|-----------|------------|-------|
| | $a b c$ | |
| 0 | 111 | 1 |
| 1 | 112 | 1 |
| 2 | 113 | 2 |
| ... | ... | ... |
| 24 | 331 | 3 |
| 25 | 332 | 2 |
| 26 | 333 | 3 |

- *ACO*: autômato celular obtido;
- *SE*: função que retorna 1 caso o teste seja verdadeiro e 0 caso contrário;
- i, j, k : índices que endereçam as células dos ACs *ACD* e *ACO*.

A priori o sistema não sabe quantas iterações serão necessárias para que determinada regra simule o comportamento desejado. O que o sistema faz é utilizar um limiar máximo de iterações. A cada iteração ele executa a função de *fitness* e se esta retornar valor 1 ele encerra o processo. Senão, na iteração referente ao limiar utilizado retorna um valor de semelhança pertencente ao intervalo [0..1].

A implementação atual do PIRGEP permite a manipulação de ACs unidimensionais de raio 1, 2 e 3 e ACs bidimensionais que utilizam vizinhança de Neumann e de Moore. Tais modelos de vizinhança encontram-se descritos na Seção 2.2.1.2. O sistema também está apto para trabalhar com ACs tridimensionais. No entanto, ainda não foi implementado um modelo de vizinhança para o mesmo. A adição de novos modelos é relativamente simples, visto que o sistema foi construído de maneira modular, já com intuito de facilitar a adição de novas funcionalidades.

6.3 RESULTADOS

6.3.1 Problemas de Classificação

O modelo desenvolvido para problemas de classificação foi testado sobre 10 bases de dados obtidas no repositório de bases de dados para aprendizado de máquina da *University of California at Irvine*². As bases foram aleatoriamente escolhidas, o único filtro utilizado sobre as bases

²Disponível em: <http://archive.ics.uci.edu/ml/datasets.html>

escolhidas foi a inexistência de amostras com valores de atributos indeterminados. A Tabela 16 apresenta um resumo das características de cada uma das bases analisadas.

Tabela 16: Características das bases de dados.

| Base | Nº de Classes | Nº de Atributos | Nº de Amostras | Tipo dos Atributos |
|--------------------------------|---------------|-----------------|----------------|-------------------------|
| <i>Abalone</i> | 3 | 8 | 4177 | contínuos e categóricos |
| <i>Balance Scala</i> | 3 | 4 | 625 | contínuos |
| <i>Car evaluation</i> | 4 | 6 | 1728 | categóricos |
| <i>Haberman</i> | 2 | 3 | 306 | contínuos |
| <i>Blood Transfusion</i> | 2 | 4 | 748 | contínuos |
| <i>Ljubljana Breast Cancer</i> | 2 | 9 | 277 | contínuos |
| <i>Wisconsin Breast Cancer</i> | 2 | 9 | 683 | contínuos |
| <i>Iris</i> | 3 | 4 | 150 | contínuos |
| <i>Nursery</i> | 4 | 8 | 12958 | categóricos |
| <i>Letter Recognition</i> | 16 | 16 | 20000 | contínuos e categóricos |

O número de classes de cada base é diferente. Assim, para cada base cria-se um modelo de classificação hierárquica. Para explicar como funciona a interpretação destes modelos utiliza-se o modelo de classificação hierárquica para base de dados *Abalone* apresentado na Figura 41. O modelo é representado por uma árvore binária, onde no nó raiz encontram-se definidos o nome da base e o número de amostras da mesma. Estas amostras formam os casos de *fitness*. A árvore binária é segmentada em níveis hierárquicos numerados do nó raiz para os nós folhas. Cada nível hierárquico apresenta, no nó à esquerda, a classe considerada positiva para o algoritmo de classificação e o número de amostras desta classe, e no nó à direita, as classes consideradas negativas para o algoritmo de classificação bem como o somatório do número de amostras destas classes.

Para cada nível hierárquico são realizados 3 experimentos. Estes experimentos têm o intuito de verificar o desempenho do modelo proposto.

O primeiro experimento é realizado com um algoritmo de classificação C4.5, consagrado na literatura (QUINLAN, 1993). O C4.5 foi apresentado por Quinlan (1993) e faz parte do pacote WEKA (*Waikato Environment for Knowledge Analysis*) versão 3.5.7, escrita em linguagem Java (WITTEN; FRANK, 2000). Esta ferramenta foi desenvolvida na Universidade de *Waikato*, situada na Nova Zelândia e implementa um pacote de algoritmos de mineração de dados. Por ser implementada em linguagem Java o WEKA tem como principais vantagens ser portátil (rodar nas mais variadas plataformas), aproveitar os benefícios de uma linguagem orientada a objetos (modularidade, polimorfismo, encapsulamento e reutilização de código) e ser um *software* de

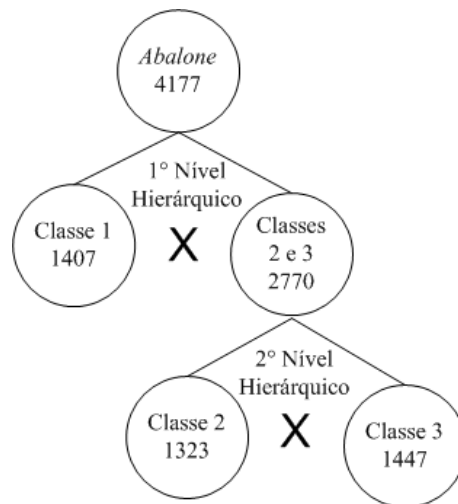


Figura 41: Classificação hierárquica para base *Abalone*.

domínio público³.

O segundo experimento contempla a utilização do modelo mestre-escravo em uma única ilha de processamento e o terceiro experimento em 5 ilhas, explorando os benefícios da co-evolução.

Apesar de modelos paralelos sugerirem comparativos de desempenho em relação ao tempo de processamento, estes não foram realizados nesta seção de experimentos. O motivo é que o algoritmo C4.5 é determinístico e rápido, sendo difícil para um algoritmo evolucionário qualquer tipo de competição neste sentido. Experimentos neste sentido serão apresentados na seção que trata da aplicação do modelo proposto para problemas de indução de regras em ACs.

O desempenho nos 3 diferentes experimentos é mensurado sobre dois aspectos. O primeiro aspecto mede a relação: sensibilidade x especificidade. Esta relação retorna valores pertencentes ao intervalo [0..1], onde quanto mais próximo de 1 melhor a qualidade da solução em termos de precisão do classificador. O segundo aspecto mede a complexidade da regra obtida através do parâmetro número de nós. Quanto menor for este parâmetro mais fácil é a compreensão da regra obtida. Segundo Freitas (2002), a compreensão do conhecimento é importante por duas razões. Primeiro, pelo fato de que o conhecimento, usualmente, será utilizado como ferramenta de suporte a decisão do usuário. E segundo, se o conhecimento descoberto não for compreensível pelo usuário, o mesmo não será capaz de validá-lo. Assim, se o número de regras descobertas e/ou número de nós que compõem cada regra for grande, a regra encontrada dificilmente poderá ser chamada de compreensível.

³Disponível em: <http://www.cs.waikato.ac.nz/ml/weka/>

Todos os experimentos foram realizados utilizando validação cruzada de tamanho 5 (HAND, 1997). Tomou-se o cuidado de sempre se preservar as mesmas amostras dentro de cada uma das partições de treinamento e teste dos algoritmos, bem como, de manter a proporção entre as amostras de cada classe.

Os algoritmos evolucionários, independente do número de ilhas utilizaram os seguintes parâmetros:

- número de indivíduos na população: 50;
- número de genes por cromossomo: 3;
- tamanho da cabeça de cada gene: 15;
- conjunto de funções: AND, OR e NOT;
- função de ligação das AEs: AND;
- método de seleção: torneio contemplando 10% dos indivíduos da população;
- mutação: 2% de probabilidade;
- recombinação: 80% de probabilidade;
- transposição IS: 70% de probabilidade;
- transposição RIS: 70% de probabilidade;
- transposição gênica: 70% de probabilidade;
- clonagem (elitismo): habilitada;
- número de gerações: 50;
- número de processos escravo por processo mestre: 10;
- migração: um indivíduo, de cada ilha, selecionado pelo método do torneio a cada 5 gerações, migra para ilha vizinha substituindo o pior indivíduo desta ilha.

É importante salientar que, até o presente momento, para problemas de classificação, devido a sua generalidade e abrangência no que diz respeito as aplicações, ainda não foi definido um conjunto de valores ótimos para cada um dos parâmetros evolucionários. Os valores dos

parâmetros definidos para estes experimentos são fruto da experiência do autor em outros trabalhos onde foi utilizada a PEG técnica evolucionária para resolução de diferentes problemas (WEINERT, 2004; WEINERT; LOPES, 2006, 2007).

Os resultados obtidos são apresentados em tabelas. Todos os valores representam as médias obtidas a partir do processo de validação cruzada. As siglas utilizadas nas tabelas são:

- C4.5: Algoritmo C4.5;
- M/E: Algoritmo mestre-escravo executado em uma única ilha de processamento;
- M/E ilhas: Algoritmo mestre-escravo executado em 5 ilhas de processamento;
- Sen: medida de sensibilidade, definida na Seção 2.1.3.1 pela Equação 3;
- Esp: medida de especificidade, definida na Seção 2.1.3.1 pela Equação 2;
- Sen x Esp: medida de desempenho, definida na Seção 2.1.3.1 pela Equação 1;
- N^o Nós: número de nós, definido como medida de complexidade;
- DP: desvio-padrão.

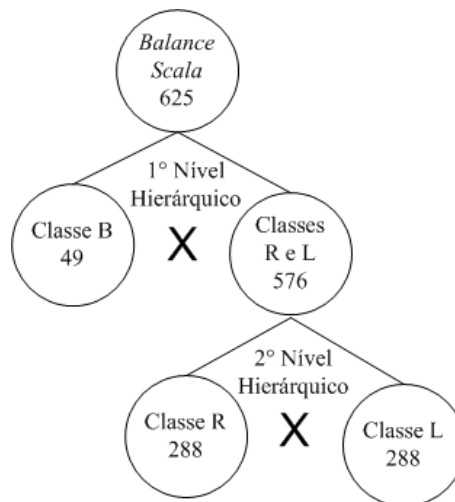
Na sequência são apresentados os resultados obtidos sobre as 10 bases analisadas.

A primeira base é a *Abalone*. Esta base possui 8 atributos, sendo 7 contínuos e 1 categórico, e está dividida em 3 classes. A classe 1 com 1407 amostras, a classe 2 com 1323 e a classe 3 com 1447. A classificação hierárquica segue as especificações definidas na Figura 41 e os resultados obtidos estão descritos na Tabela 17.

Tabela 17: Base de dados *Abalone*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|-------------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,675 ± 0,030 | 0,895 ± 0,075 | 0,603 ± 0,036 | 129,200 ± 23,763 |
| M/E | 0,800 ± 0,098 | 0,808 ± 0,099 | 0,636 ± 0,057 | 26,600 ± 5,128 |
| M/E Ilhas | 0,800 ± 0,088 | 0,813 ± 0,091 | 0,646 ± 0,057 | 25,600 ± 5,505 |
| 2º Nível Hierárquico | | | | |
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,637 ± 0,063 | 0,614 ± 0,086 | 0,387 ± 0,024 | 157,200 ± 41,877 |
| M/E | 0,568 ± 0,063 | 0,632 ± 0,075 | 0,373 ± 0,083 | 23,600 ± 8,142 |
| M/E Ilhas | 0,659 ± 0,113 | 0,583 ± 0,084 | 0,380 ± 0,053 | 34,200 ± 12,988 |

A segunda base analisada é a *Balance Scala*. Esta base possui 4 atributos, sendo todos contínuos, e está dividida em 3 classes. A classe B com 49 amostras, a classe R com 288 e a classe L com 288. A classificação hierárquica segue as especificações definidas na Figura 42 e os resultados obtidos estão descritos na Tabela 18.

Figura 42: Classificação hierárquica para base *Balance Scala*.Tabela 18: Base de dados *Balance Scala*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|------------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,000 ± 0,000 | 1,000 ± 0,000 | 0,000 ± 0,000 | 1,000 ± 0,000 |
| M/E | 0,000 ± 0,000 | 0,351 ± 0,353 | 0,000 ± 0,000 | 21,600 ± 8,019 |
| M/E Ilhas | 0,231 ± 0,275 | 0,408 ± 0,165 | 0,086 ± 0,123 | 29,600 ± 12,512 |
| 2º Nível Hierárquico | | | | |
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,640 ± 0,389 | 0,641 ± 0,321 | 0,332 ± 0,267 | 42,600 ± 9,529 |
| M/E | 0,477 ± 0,436 | 0,667 ± 0,318 | 0,227 ± 0,248 | 21,600 ± 3,362 |
| M/E Ilhas | 0,588 ± 0,212 | 0,716 ± 0,207 | 0,406 ± 0,157 | 29,600 ± 7,369 |

A terceira base analisada é a *Car Evaluation*. Esta base possui 6 atributos, sendo todos categóricos, e está dividida em 4 classes. A classe “acc” com 384 amostras, a classe “good” com 69, a classe “unacc” com 1210 e a classe “vgood” com 65. A classificação hierárquica segue as especificações definidas na Figura 43 e os resultados obtidos estão descritos na Tabela 19.

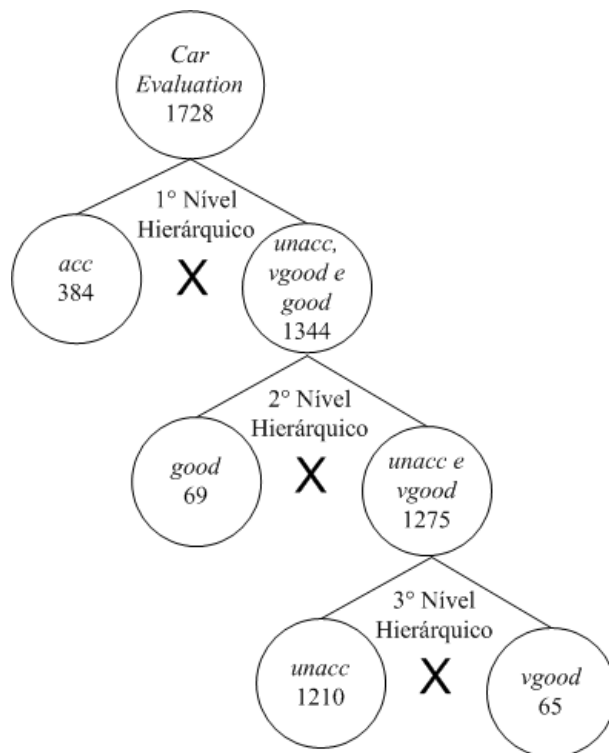


Figura 43: Classificação hierárquica para base *Car Evaluation*.

Tabela 19: Base de dados *Car Evaluation*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|-------------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,478 ± 0,314 | 0,878 ± 0,066 | 0,413 ± 0,250 | 122,400 ± 16,802 |
| M/E | 0,817 ± 0,159 | 0,690 ± 0,219 | 0,560 ± 0,204 | 26,800 ± 2,280 |
| M/E Ilhas | 0,836 ± 0,132 | 0,746 ± 0,205 | 0,617 ± 0,182 | 28,400 ± 5,857 |
| 2º Nível Hierárquico | | | | |
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,643 ± 0,378 | 0,792 ± 0,256 | 0,458 ± 0,286 | 34,800 ± 8,167 |
| M/E | 0,743 ± 0,279 | 0,650 ± 0,220 | 0,449 ± 0,144 | 24,800 ± 8,815 |
| M/E Ilhas | 1,000 ± 0,000 | 0,784 ± 0,100 | 0,784 ± 0,100 | 24,600 ± 5,459 |
| 3º Nível Hierárquico | | | | |
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,709 ± 0,343 | 0,892 ± 0,241 | 0,601 ± 0,312 | 19,800 ± 1,643 |
| M/E | 0,745 ± 0,081 | 1,000 ± 0,000 | 0,745 ± 0,081 | 33,400 ± 7,436 |
| M/E Ilhas | 0,783 ± 0,146 | 1,000 ± 0,000 | 0,783 ± 0,146 | 28,800 ± 4,382 |

A quarta base analisada é a *Haberman*. Esta base possui 3 atributos, sendo todos contínuos, e está dividida em 2 classes. A classe 1 com 225 amostras e a classe 2 com 81. A classificação hierárquica segue as especificações definidas na Figura 44 e os resultados obtidos estão descritos na Tabela 20.

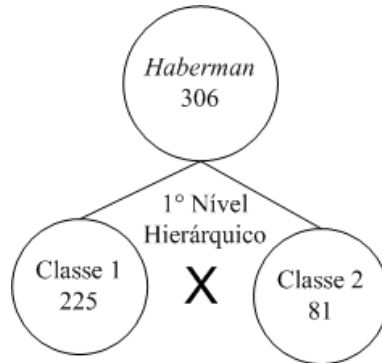


Figura 44: Classificação hierárquica para base *Haberman*.

Tabela 20: Base de dados *Haberman*.

| 1º Nível Hierárquico | | | | |
|----------------------|-------------------|-------------------|--------------------------|---------------------------|
| | Sen \pm DP | Esp \pm DP | Sen x Esp \pm DP | Nº Nós \pm DP |
| C4.5 | 0,818 \pm 0,134 | 0,413 \pm 0,268 | 0,310 \pm 0,187 | 4,600 \pm 2,191 |
| M/E | 0,689 \pm 0,122 | 0,593 \pm 0,089 | 0,405 \pm 0,067 | 22,600 \pm 2,702 |
| M/E Ilhas | 0,702 \pm 0,081 | 0,557 \pm 0,104 | 0,386 \pm 0,052 | 27,200 \pm 5,167 |

A quinta base analisada é a *Blood Transfusion*. Esta base possui 4 atributos, sendo todos contínuos, e está dividida em 2 classes. A classe 0 com 570 amostras e a classe 1 com 178. A classificação hierárquica segue as especificações definidas na Figura 45 e os resultados obtidos estão descritos na Tabela 21.

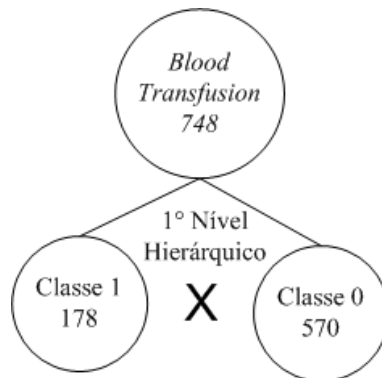
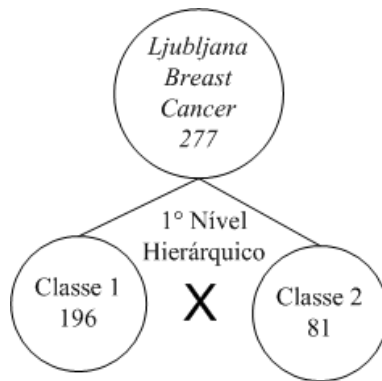


Figura 45: Classificação hierárquica para base *Blood Transfusion*.

Tabela 21: Base de dados *Blood Transfusion*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|------------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,911 ± 0,074 | 0,347 ± 0,102 | 0,316 ± 0,100 | 12,200 ± 6,419 |
| M/E | 0,581 ± 0,194 | 0,714 ± 0,183 | 0,392 ± 0,093 | 36,000 ± 12,903 |
| M/E Ilhas | 0,632 ± 0,181 | 0,726 ± 0,074 | 0,448 ± 0,092 | 25,800 ± 6,058 |

A sexta base analisada é a *Ljubljana Breast Cancer*. Esta base possui 9 atributos, sendo todos contínuos, e está dividida em 2 classes. A classe 1 com 196 amostras e a classe 2 com 81. A classificação hierárquica segue as especificações definidas na Figura 46 e os resultados obtidos estão descritos na Tabela 22.

Figura 46: Classificação hierárquica para base *Ljubljana Breast Cancer*.Tabela 22: Base de dados *Ljubljana Breast Cancer*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|------------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,862 ± 0,086 | 0,358 ± 0,062 | 0,312 ± 0,083 | 20,600 ± 7,266 |
| M/E | 0,643 ± 0,065 | 0,666 ± 0,096 | 0,430 ± 0,086 | 25,600 ± 12,361 |
| M/E Ilhas | 0,633 ± 0,065 | 0,654 ± 0,070 | 0,416 ± 0,077 | 21,000 ± 4,301 |

A sétima base analisada é a *Wisconsin Breast Cancer*. Esta base possui 9 atributos, sendo todos contínuos, e está dividida em 2 classes. A classe 1 com 444 amostras e a classe 2 com 239. A classificação hierárquica segue as especificações definidas na Figura 47 e os resultados obtidos estão descritos na Tabela 23.

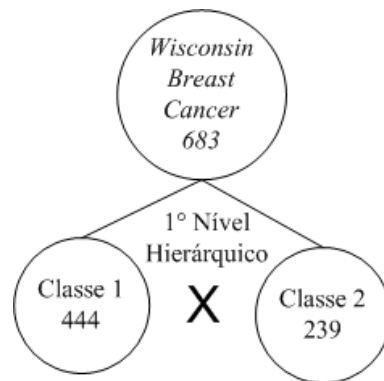


Figura 47: Classificação hierárquica para base *Wisconsin Breast Cancer*.

Tabela 23: Base de dados *Wisconsin Breast Cancer*.

| 1º Nível Hierárquico | | | | |
|----------------------|-------------------|-------------------|--------------------------|---------------------------|
| | Sen \pm DP | Esp \pm DP | Sen x Esp \pm DP | Nº Nós \pm DP |
| C4.5 | 0,960 \pm 0,019 | 0,933 \pm 0,063 | 0,895 \pm 0,060 | 17,000 \pm 2,449 |
| M/E | 0,926 \pm 0,032 | 0,941 \pm 0,052 | 0,871 \pm 0,053 | 26,200 \pm 7,259 |
| M/E Ilhas | 0,948 \pm 0,023 | 0,946 \pm 0,035 | 0,897 \pm 0,048 | 28,000 \pm 6,782 |

A oitava base analisada é a *Iris*. Esta base possui 4 atributos, sendo todos contínuos, e está dividida em 3 classes. A classe “setosa” com 50 amostras, a classe “versicolor” com 50 e a classe “virginica” com 50. A classificação hierárquica segue as especificações definidas na Figura 48 e os resultados obtidos estão descritos na Tabela 24.

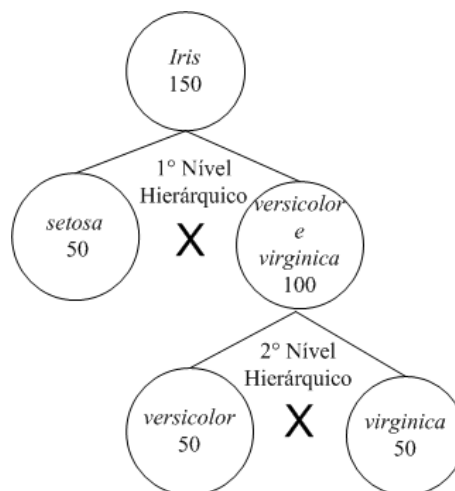


Figura 48: Classificação hierárquica para base *Iris*.

Tabela 24: Base de dados *Iris*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|-----------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,980 ± 0,045 | 1,000 ± 0,000 | 0,980 ± 0,045 | 3,000 ± 0,000 |
| M/E | 1,000 ± 0,000 | 0,960 ± 0,065 | 0,960 ± 0,065 | 29,800 ± 7,950 |
| M/E Ilhas | 1,000 ± 0,000 | 0,980 ± 0,027 | 0,980 ± 0,027 | 23,000 ± 3,317 |
| 2º Nível Hierárquico | | | | |
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,940 ± 0,089 | 0,900 ± 0,122 | 0,846 ± 0,140 | 6,200 ± 1,095 |
| M/E | 0,900 ± 0,071 | 0,860 ± 0,167 | 0,778 ± 0,188 | 26,800 ± 6,723 |
| M/E Ilhas | 0,920 ± 0,110 | 0,920 ± 0,084 | 0,844 ± 0,108 | 26,400 ± 6,656 |

A nona base analisada é a *Nursery*. Esta base possui 8 atributos, sendo todos categóricos, e está dividida em 4 classes. A classe “priority” com 4266 amostras, a classe “not_recom” com 4320, a classe “very_recom” com 328 e a classe “spec_prior” com 4044. A classificação hierárquica segue as especificações definidas na Figura 49 e os resultados obtidos estão descritos na Tabela 25.

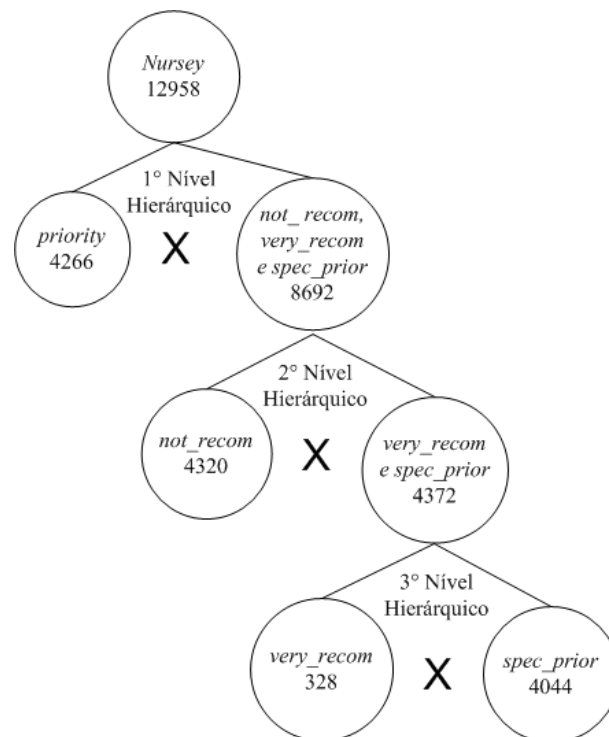
Figura 49: Classificação hierárquica para base *Nursery*.

Tabela 25: Base de dados *Nursery*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|--------------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,370 ± 0,153 | 0,774 ± 0,209 | 0,296 ± 0,175 | 374,800 ± 108,410 |
| M/E | 0,720 ± 0,420 | 0,734 ± 0,231 | 0,499 ± 0,327 | 24,800 ± 7,050 |
| M/E Ilhas | 0,878 ± 0,273 | 0,637 ± 0,339 | 0,586 ± 0,395 | 21,000 ± 1,732 |
| 2º Nível Hierárquico | | | | |
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 1,000 ± 0,000 | 1,000 ± 1,000 | 1,000 ± 0,000 | 4,000 ± 0,000 |
| M/E | 0,824 ± 0,205 | 1,000 ± 0,000 | 0,824 ± 0,205 | 29,200 ± 8,167 |
| M/E Ilhas | 1,000 ± 0,000 | 1,000 ± 0,000 | 1,000 ± 0,000 | 26,800 ± 9,066 |
| 3º Nível Hierárquico | | | | |
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,806 ± 0,434 | 0,696 ± 0,383 | 0,562 ± 0,488 | 14,400 ± 4,930 |
| M/E | 0,969 ± 0,043 | 0,879 ± 0,108 | 0,849 ± 0,068 | 29,800 ± 9,149 |
| M/E Ilhas | 0,988 ± 0,027 | 0,919 ± 0,066 | 0,908 ± 0,075 | 27,800 ± 6,760 |

A décima base analisada é a *Letter Recognition*. A base *Letter Recognition* possui 16 atributos, sendo 15 contínuos e 1 categórico. Esta base possui 16 classes. No entanto, apenas um nível hierárquico foi analisado. Neste nível foi considerado como classe positiva a classe 9, com 8047 amostras, e como classe negativa, todas as outras. Nas classes negativas totalizam-se 11953 amostras. A classificação hierárquica segue as especificações definidas na Figura 50 e os resultados obtidos estão descritos na Tabela 26.

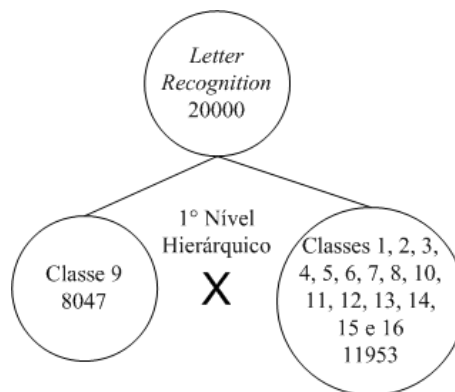
Figura 50: Classificação hierárquica para base *Letter Recognition*.

Tabela 26: Base de dados *Letter Recognition*.

| 1º Nível Hierárquico | | | | |
|----------------------|---------------|---------------|----------------------|--------------------------|
| | Sen ± DP | Esp ± DP | Sen x Esp ± DP | Nº Nós ± DP |
| C4.5 | 0,465 ± 0,097 | 0,868 ± 0,215 | 0,389 ± 0,067 | 780,952 ± 445,780 |
| M/E | 0,725 ± 0,043 | 0,528 ± 0,197 | 0,376 ± 0,127 | 31,200 ± 3,421 |
| M/E Ilhas | 0,717 ± 0,083 | 0,595 ± 0,151 | 0,424 ± 0,112 | 30,000 ± 8,216 |

6.3.2 Autômatos Celulares

Numa primeira etapa foram realizados 6 experimentos controlados. Cada experimento consiste da indução de n regras para uma mesma configuração inicial de AC. Assim, cada regra é obtida em uma rodada independente do algoritmo. A configuração final desejada para o AC é o único parametro que se altera em cada rodada, e é representada pela configuração do AC em uma determinada iteração. Os experimentos são ditos controlados porque, primeiro, sabe-se a priori qual a regra se deseja obter. Segundo, sabe-se o número de iterações necessárias para conduzir a configuração inicial do AC até a configuração final desejada durante o processo de simulação da dinâmica. No entanto, é importante lembrar, que para problemas reais estas duas informações (regras e número de iterações) são desconhecidas.

Para os experimentos seguintes utiliza-se um AC binário de 149 células, inicializado aleatoriamente com distribuição uniforme. A vizinhança é formada por um raio igual a 1. A configuração deste AC é apresentada na Figura 51 onde células com estado 0 aparecem em branco e células com estado 1 em preto.



Figura 51: Configuração inicial do AC de 149 células.

O primeiro experimento controlado avalia o desempenho do algoritmo em induzir a regra elementar 160. Esta regra apresenta um comportamento dinâmico nulo segundo a classificação de Li e Packard (1990). O sistema apresenta-se estável a partir da sexta iteração. A Figura 52 demonstra a dinâmica do sistema por 20 iterações.

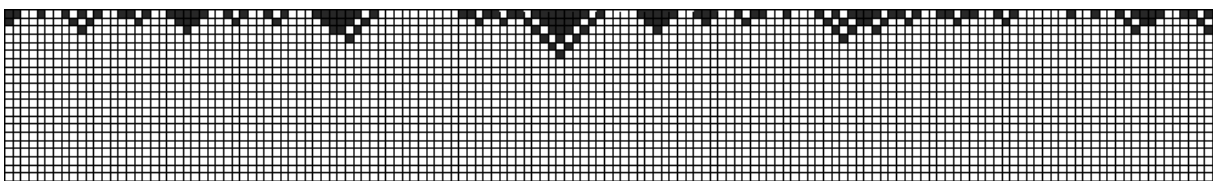


Figura 52: Simulação da dinâmica da regra 160 por 20 iterações.

Basicamente, os parâmetros evolucionários utilizados são os mesmos da Seção 6.3.1. Ape-

nas reduziu-se o número de indivíduos para 30, devido ao pequeno tamanho do espaço de busca deste problema, e não foi utilizado o conceito de co-evolução através do trabalho em múltiplas ilhas. Este experimento foi segmentado em 7 rodadas. Sendo que em cada rodada busca-se uma configuração final de AC diferente. Considerou-se como configuração final do AC a configuração alcançada em uma determinada iteração no processo de simulação da dinâmica (Figura 52). As rodadas contemplam as configurações finais obtidas nas iterações 1, 2, 3, 4, 5, 6 e 10. Em todas as rodadas o sistema foi capaz de induzir a regra 160. A Tabela 27 resume algumas informações a respeito dos resultados obtidos. Cada linha corresponde a uma rodada. Na segunda coluna tem-se informação a respeito do comportamento dinâmico do sistema até aquele momento. Esta questão será discutida na próxima seção. A terceira coluna indica a iteração utilizada como determinante do estado final do AC desejado, e a quarta coluna apresenta o número de gerações necessárias para que a solução fosse encontrada.

Tabela 27: Resultados para a regra 160.

| Rodada | Comportamento | Iteração | N ^o de Gerações |
|--------|-----------------------|----------|----------------------------|
| 1 | | 1 | 1 |
| 2 | Período Transiente | 2 | 1 |
| 3 | | 3 | 1 |
| 4 | | 4 | 1 |
| 5 | | 5 | 1 |
| 6 | | 6 | 1 |
| 7 | Nulo | 10 | 1 |

Além da regra desejada o sistema também apresentou outras regras como solução para as rodadas 6 e 7. Para a rodada 6 foram as regras elementares 0 e 2. Para a rodada 7 foram as regras elementares 0, 128, 4, 64, 32, 8 e 136. Isto já era esperado, uma vez que estas regras pertencem ao espaço de soluções do algoritmo evolucionário e apresentam comportamento nulo.

O segundo experimento controlado avalia o desempenho do algoritmo em induzir a regra elementar 118. Esta regra apresenta um comportamento dinâmico periódico segundo a classificação de Li e Packard (1990). O sistema apresenta-se estável a partir da sexta iteração apresentando um período cíclico de tamanho 3. A Figura 53 demonstra a dinâmica do sistema por 20 iterações. Os parâmetros evolucionários utilizados são os mesmos do experimento anterior. Este experimento foi segmentado em 11 rodadas. Os critérios de definição dos estados finais dos ACs para cada rodada são os mesmos da regra 118. As rodadas contemplam as configurações finais obtidas nas iterações 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 e 11. Em todas as rodadas o sistema foi capaz de induzir a regra 118. A Tabela 28 resume algumas informações a respeito dos resultados obtidos.

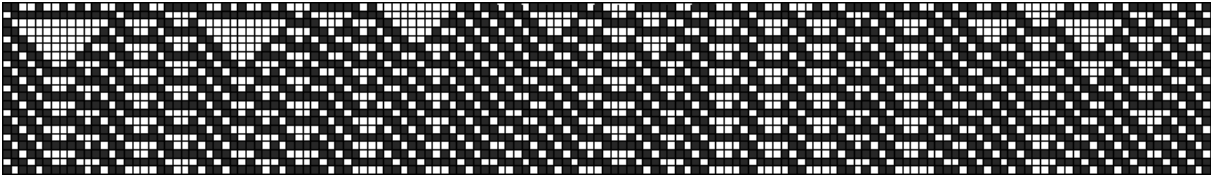


Figura 53: Simulação da dinâmica da regra 118 por 20 iterações.

Tabela 28: Resultados para regra 118.

| Rodada | Comportamento | Iteração | N ^o de Gerações |
|--------|-----------------------|----------|----------------------------|
| 1 | | 1 | 31 |
| 2 | Período Transiente | 2 | 46 |
| 3 | | 3 | 71 |
| 4 | | 4 | 125 |
| 5 | | 5 | 156 |
| 6 | | 6 | 282 |
| 7 | Cíclico | 7 | 152 |
| 8 | | 8 | 173 |
| 9 | | 9 | 86 |
| 10 | Cíclico | 10 | 14 |
| 11 | | 11 | 18 |

Neste experimento o sistema não encontrou nenhuma outra regra, além da própria regra 118, que pudesse solucionar o problema em nenhuma das diferentes rodadas.

O terceiro experimento controlado avalia o desempenho do algoritmo em induzir a regra elementar 16. Esta regra apresenta um comportamento dinâmico de ponto fixo segundo a classificação de Li e Packard (1990). O sistema apresenta-se estável a partir da primeira iteração. A Figura 54 demonstra a dinâmica do sistema por 20 iterações. Este experimento foi segmentado em 2 rodadas. As rodadas contemplam as configurações finais obtidas nas iterações 1 e 10. Em ambas as rodadas o sistema foi capaz de induzir a regra 16. A Tabela 29 resume algumas informações a respeito dos resultados obtidos.

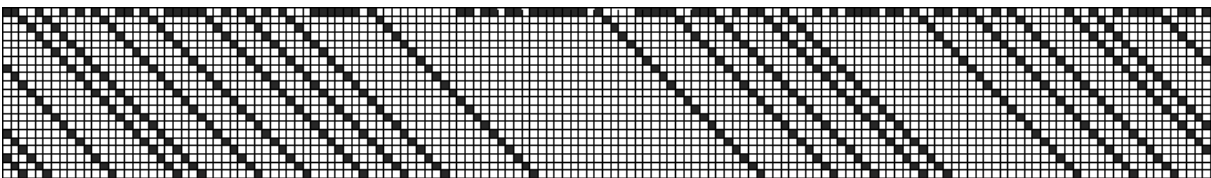


Figura 54: Simulação da dinâmica da regra 16 por 20 iterações.

Tabela 29: Resultados para regra 16.

| Rodada | Comportamento | Iteração | Nº de Gerações |
|--------|---------------|----------|----------------|
| 1 | Ponto Fixo | 1 | 3 |
| 2 | | 10 | 3 |

O quarto experimento controlado avalia o desempenho do algoritmo em induzir a regra elementar 95. Esta regra apresenta um comportamento dinâmico de ciclo duplo segundo a classificação de Li e Packard (1990). O sistema apresenta-se estável a partir da primeira iteração. A Figura 55 demonstra a dinâmica do sistema por 20 iterações. Este experimento foi segmentado em 4 rodadas. As rodadas contemplam as configurações finais obtidas nas iterações 1, 2, 3 e 10. Em todas rodadas o sistema foi capaz de induzir a regra 95. A Tabela 30 resume algumas informações a respeito dos resultados obtidos.



Figura 55: Simulação da dinâmica da regra 95 por 20 iterações.

Tabela 30: Resultados para regra 95.

| Rodada | Comportamento | Iteração | Nº de Gerações |
|--------|---------------|----------|----------------|
| 1 | Ciclo Duplo | 1 | 16 |
| 2 | | 2 | 14 |
| 3 | | 3 | 13 |
| 4 | | 10 | 5 |

O quinto experimento controlado avalia o desempenho do algoritmo em induzir a regra elementar 183. Esta regra apresenta um comportamento dinâmico caótico segundo a classificação de Li e Packard (1990). O sistema não estabiliza. A Figura 56 demonstra a dinâmica do sistema por 20 iterações. Este experimento foi segmentado em 3 rodadas. As rodadas contemplam as configurações finais obtidas nas iterações 1, 10 e 20. O algoritmo evolucionário configurado com 30 indivíduos não retornou a regra desejada em 350 gerações. Então rodou-se novamente o algoritmo com um tamanho de população igual a 100 indivíduos. Com este valor a regra 183 foi encontrada para as rodadas 1 e 2. A rodada 3 somente atingiu seu objetivo com uma população de 200 indivíduos. A Tabela 31 resume algumas informações a respeito dos resultados obtidos.

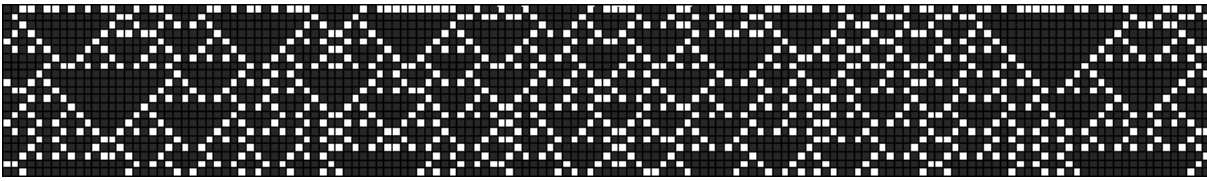


Figura 56: Simulação da dinâmica da regra 183 por 20 iterações.

Tabela 31: Resultados para regra 183.

| Rodada | Comportamento | Iteração | N ^o de Gerações |
|--------|---------------|----------|----------------------------|
| 1 | | 1 | 184 |
| 2 | Caótico | 10 | 187 |
| 3 | | 20 | 16 |

O sexto experimento controlado avalia o desempenho do algoritmo em induzir a regra elementar 124. Esta regra apresenta um comportamento dinâmico complexo segundo a classificação de Li e Packard (1990). O sistema estabiliza após milhares de iterações. A Figura 57 demonstra a dinâmica do sistema segmentada em duas regiões, da iteração 1 até a 20 (Figura 57(a)) e da iteração 3980 até a 4000 (Figura 57(b)). Este experimento foi segmentado em 2 rodadas. As rodadas contemplam as configurações finais obtidas nas iterações 1 e 4000. O algoritmo evolucionário volta a utilizar 30 indivíduos. A Tabela 32 resume algumas informações a respeito dos resultados obtidos.

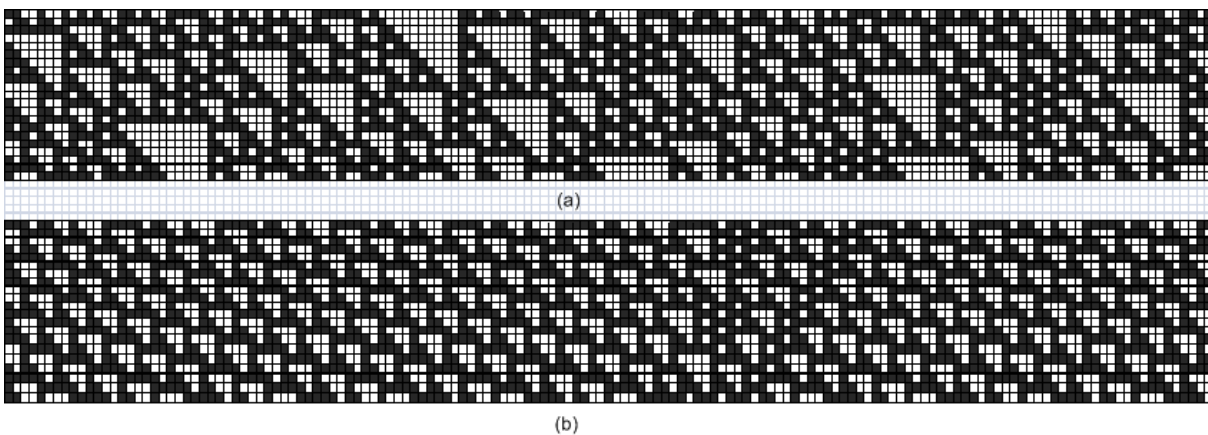


Figura 57: Simulação da dinâmica da regra 124 por 4000 iterações. (a) Apresenta as primeiras 20 iterações e (b) as últimas 20.

Concluída a etapa de experimentos controlados apresenta-se um experimento com um problema hipotético. Neste problema deseja-se descobrir uma regra que seja capaz de conduzir o AC bidimensional (5x5) apresentado pela Figura 58(a) ao AC apresentado pela Figura 58(b). A

Tabela 32: Resultados para regra 124.

| Rodada | Comportamento | Iteração | N ^o de Gerações |
|--------|--------------------|----------|----------------------------|
| 1 | Período Transiente | 1 | 81 |
| 2 | Complexo | 4000 | 27 |

única definição existente é quanto a configuração de vizinhança, que deve ser a de Neumann. Este modelo de vizinhança encontra-se descrito na Seção 2.2.1.2.

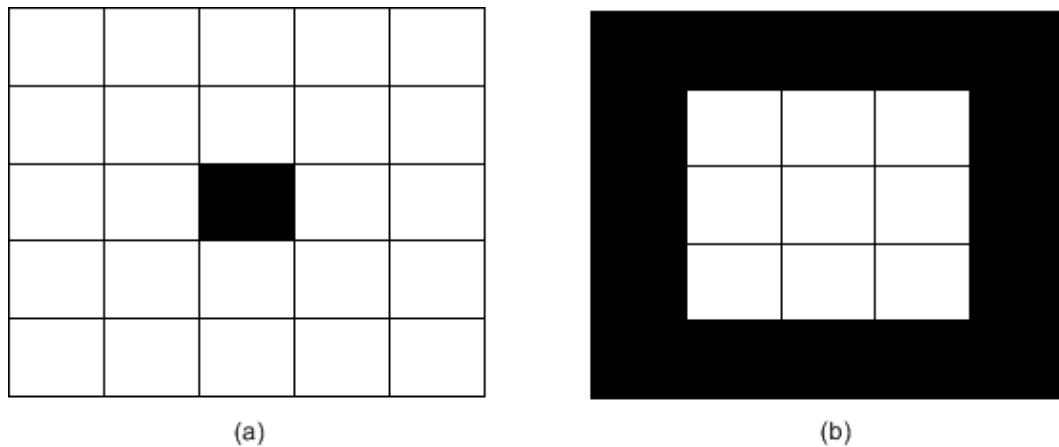


Figura 58: Configurações inicial (a) e final (b) do AC bidimensional.

Para resolução deste problema, foi utilizado o modelo insular composto por 5 ilhas. Foram utilizados os seguintes parâmetros:

- número de indivíduos na população: 1000;
- número de genes por cromossomo: 2;
- tamanho da cabeça de cada gene: 10;
- conjunto de funções: AND, OR e NOT;
- função de ligação das AEs: AND;
- método de seleção: torneio contemplando 10% dos indivíduos da população;
- mutação: 20% de probabilidade;
- recombinação: 80% de probabilidade;
- transposição IS: 70% de probabilidade;

- transposição RIS: 70% de probabilidade;
- transposição gênica: 70% de probabilidade;
- clonagem (elitismo): habilitada;
- gerações: 350;
- número de processos escravo por processo mestre: 20;
- migração: indivíduo selecionado pelo método do torneio a cada 5 gerações;
- número máximo de iterações do AC: 30.

O algoritmo encontrou a solução na geração 103, sendo a regra: 10100000101001010010000000100011. Esta regra conduz o AC da Figura 58(a) ao AC da Figura 58(b) em duas iterações.

6.3.2.1 Balanceamento de Carga na Arquitetura Paralela

A arquitetura de processamento paralelo proposta neste trabalho, dentre outras características, deve apresentar uma relação eficiente quando se compara o tempo de processamento com a quantidade de tarefas a serem executadas. Esta relação está diretamente ligada ao número de processos escravos utilizados na arquitetura e ao custo da comunicação dos mesmos com seus respectivos processos mestres.

Para analisar esta característica realizou-se um experimento baseado no problema de classificação pela regra da maioria descrito na Seção 2.2.4.1 executado em uma única ilha de processamento. Neste experimento a função de *fitness* consiste em simular a dinâmica de 10000 ACs, por 200 iterações durante 50 gerações.

O número de indivíduos variou de 100 a 600, de 100 em 100, e o número de escravos variou de 10 a 100, de 10 em 10. Assim, executaram-se 60 rodadas formadas das combinações possíveis entre o número de indivíduos da população e o número de processos escravos. É importante ressaltar que cada processo escravo é executado em uma unidade física de processamento distinta.

O gráfico de superfície apresentado pela Figura 59 mostra os resultados obtidos. Neste gráfico tem-se 3 eixos: tempo, escravos e tarefas. O tempo é medido em segundos. Os escravos representam os processos executores da função de *fitness* e as tarefas representam as regras, que são na verdade, os indivíduos da população do algoritmo evolucionário. Pelo gráfico percebe-se uma relação aproximadamente linear entre o aumento de tarefas e o aumento do número de escravos.

O balanceamento de carga consiste, basicamente, em dividir de forma uniforme o número de tarefas entre os processadores disponíveis.

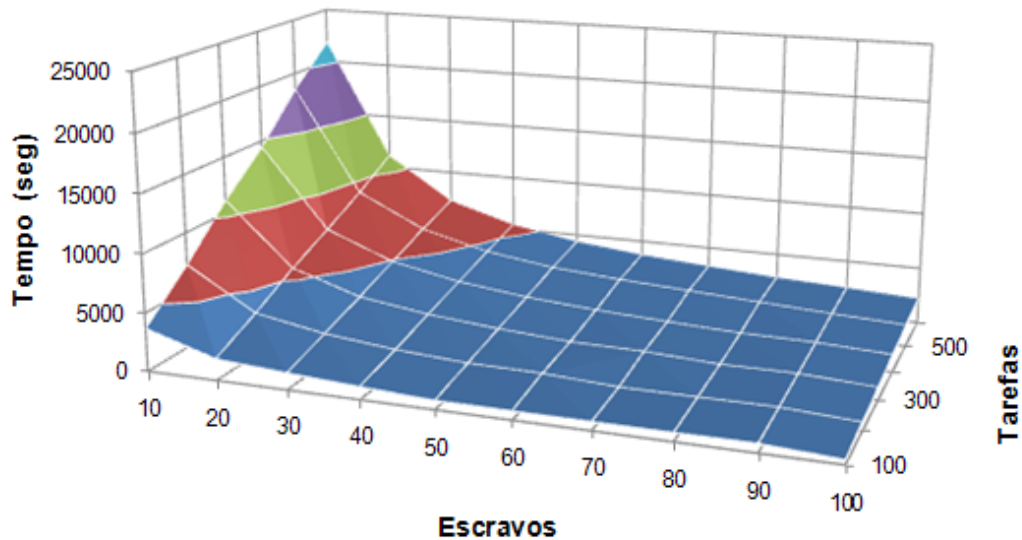


Figura 59: Relação entre o tempo de processamento, número de tarefas e número de processadores.

6.3.2.2 Otimização dos Parâmetros Evolucionários

Definir um conjunto de parâmetros considerados ótimos para todos os problemas é uma tarefa muito difícil. Então, o próximo experimento se propõe a avaliar alguns parâmetros e propor alguns valores, que embora não possam ser chamados de ótimos, podem ser considerados parâmetros de referência. Os parâmetros selecionados são: número de indivíduos, número de genes por classe e probabilidade de mutação. Estes parâmetros têm uma relação direta com a manutenção da diversidade genética dentro da população e por isto foram selecionados. Outro aspecto avaliado é utilização de uma estrutura composta por uma única ilha de processamento e uma estrutura composta por 5 ilhas de processamento.

Os parâmetros evolucionários descritos abaixo permanecem fixos para todas as rodadas.

- tamanho da cabeça de cada gene: 10;
- método de seleção: torneio contemplando 10% dos indivíduos da população;
- recombinação: 80% de probabilidade;

- transposição IS: 70% de probabilidade;
- transposição RIS: 70% de probabilidade;
- transposição gênica: 70% de probabilidade;
- clonagem (elitismo): habilitada;
- migração: indivíduo selecionado pelo método do torneio a cada 5 gerações.

A Figura 60 apresenta a simulação da dinâmica da regra GKL (Tabela 2), durante 20 iterações, com vizinhança de raio 3, sobre a configuração inicial apresentada na Figura 51. Esta regra apresenta um comportamento nulo segundo a classificação de Li e Packard (1990). Para tornar o problema de indução de regras extremamente difícil apresenta-se ao sistema de indução como AC inicial a configuração apresentada pela Figura 51 e como configuração AC final a configuração apresentada pela décima iteração na Figura 60. O período analisado ainda apresenta um comportamento transiente.

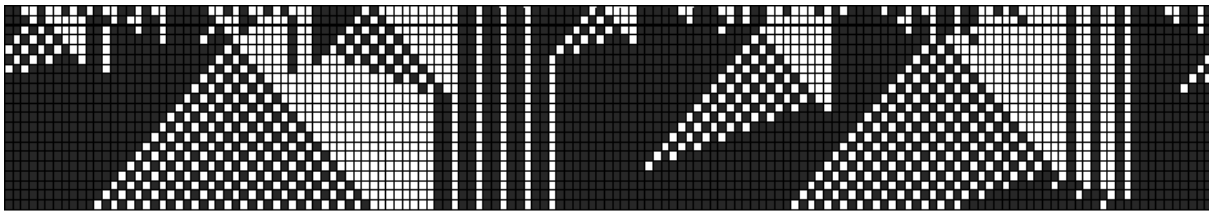


Figura 60: Simulação da dinâmica da regra GKL por 20 iterações.

Um total de 54 diferentes combinações de parâmetros foi avaliada. Cada combinação foi executada por 10 vezes, totalizando a execução de 540 rodadas. Apresentam-se como resultado os valores médios das melhores soluções encontradas, bem como o desvio-padrão entre estes valores. As Tabelas 33 e 34 apresentam os resultados obtidos para uma única ilha de processamento e para um conjunto de 5 ilhas, respectivamente. As linhas que aparecem em destaque em ambas as tabelas salientam os melhores resultados obtidos.

Tabela 33: Experimentos, com 1 ilha, para definir valores para parâmetros do algoritmo evolucionário.

| Ilhas | Indivíduos | N ^o de Genes por Classe | Mutação (Taxa em %) | <i>Fitness</i> Médio \pm DP |
|----------|------------|------------------------------------|---------------------|-------------------------------------|
| 1 | 100 | 1 | 2 | 0,7852 \pm 0,0352 |
| 1 | 100 | 1 | 5 | 0,7852 \pm 0,0308 |
| 1 | 100 | 1 | 10 | 0,7711 \pm 0,0446 |
| 1 | 100 | 2 | 2 | 0,7711 \pm 0,0132 |
| 1 | 100 | 2 | 5 | 0,7731 \pm 0,0186 |
| 1 | 100 | 2 | 10 | 0,7657 \pm 0,0208 |
| 1 | 100 | 3 | 2 | 0,7724 \pm 0,0156 |
| 1 | 100 | 3 | 5 | 0,7597 \pm 0,0129 |
| 1 | 100 | 3 | 10 | 0,7657 \pm 0,0107 |
| 1 | 150 | 1 | 2 | 0,7798 \pm 0,0330 |
| 1 | 150 | 1 | 5 | 0,7751 \pm 0,0303 |
| 1 | 150 | 1 | 10 | 0,7912\pm0,0261 |
| 1 | 150 | 2 | 2 | 0,7872 \pm 0,0296 |
| 1 | 150 | 2 | 5 | 0,7118 \pm 0,0297 |
| 1 | 150 | 2 | 10 | 0,7879 \pm 0,0251 |
| 1 | 150 | 3 | 2 | 0,7664 \pm 0,0169 |
| 1 | 150 | 3 | 5 | 0,7724 \pm 0,0193 |
| 1 | 150 | 3 | 10 | 0,7724 \pm 0,0162 |
| 1 | 200 | 1 | 2 | 0,7879 \pm 0,0290 |
| 1 | 200 | 1 | 5 | 0,7805 \pm 0,0476 |
| 1 | 200 | 1 | 10 | 0,7892 \pm 0,0427 |
| 1 | 200 | 2 | 2 | 0,7912\pm0,0213 |
| 1 | 200 | 2 | 5 | 0,7885 \pm 0,0245 |
| 1 | 200 | 2 | 10 | 0,7885 \pm 0,0255 |
| 1 | 200 | 3 | 2 | 0,7711 \pm 0,0185 |
| 1 | 200 | 3 | 5 | 0,7765 \pm 0,0277 |
| 1 | 200 | 3 | 10 | 0,7751 \pm 0,0259 |

Tabela 34: Experimentos, com 5 ilhas, para definir valores para parâmetros do algoritmo evolucionário.

| Ilhas | Indivíduos | N ^o de Genes por Classe | Mutação (Taxa em %) | <i>Fitness</i> Médio \pm DP |
|----------|------------|------------------------------------|---------------------|-------------------------------------|
| 5 | 100 | 1 | 2 | 0,8093 \pm 0,0180 |
| 5 | 100 | 1 | 5 | 0,8134 \pm 0,0142 |
| 5 | 100 | 1 | 10 | 0,8116 \pm 0,0138 |
| 5 | 100 | 2 | 2 | 0,8021 \pm 0,0195 |
| 5 | 100 | 2 | 5 | 0,7936 \pm 0,0211 |
| 5 | 100 | 2 | 10 | 0,8067 \pm 0,0175 |
| 5 | 100 | 3 | 2 | 0,7699 \pm 0,0145 |
| 5 | 100 | 3 | 5 | 0,7851 \pm 0,0197 |
| 5 | 100 | 3 | 10 | 0,7813 \pm 0,0211 |
| 5 | 150 | 1 | 2 | 0,8136 \pm 0,0120 |
| 5 | 150 | 1 | 5 | 0,7908 \pm 0,0299 |
| 5 | 150 | 1 | 10 | 0,8055 \pm 0,0194 |
| 5 | 150 | 2 | 2 | 0,7995 \pm 0,0189 |
| 5 | 150 | 2 | 5 | 0,8045 \pm 0,0207 |
| 5 | 150 | 2 | 10 | 0,7978 \pm 0,0191 |
| 5 | 150 | 3 | 2 | 0,7765 \pm 0,0146 |
| 5 | 150 | 3 | 5 | 0,7881 \pm 0,0189 |
| 5 | 150 | 3 | 10 | 0,7775 \pm 0,0149 |
| 5 | 200 | 1 | 2 | 0,8169 \pm 0,0081 |
| 5 | 200 | 1 | 5 | 0,8093 \pm 0,0195 |
| 5 | 200 | 1 | 10 | 0,8187\pm0,0033 |
| 5 | 200 | 2 | 2 | 0,8045 \pm 0,0154 |
| 5 | 200 | 2 | 5 | 0,8025 \pm 0,0167 |
| 5 | 200 | 2 | 10 | 0,7966 \pm 0,0205 |
| 5 | 200 | 3 | 2 | 0,7871 \pm 0,0146 |
| 5 | 200 | 3 | 5 | 0,7934 \pm 0,0176 |
| 5 | 200 | 3 | 10 | 0,7868 \pm 0,0181 |

O parâmetro mutação exerceu um papel importante nos experimentos, isto reafirma a importância de se manter a diversidade genética dentro da população.

6.4 DISCUSSÃO

O sistema implementado batizado de PIRGEP foi totalmente implementado em linguagem C seguindo um modelo estruturado de programação. Esta abordagem proporciona um elevado desempenho do algoritmo no que se refere a tempo de processamento. Diferentemente do sistema GEPCLASS, que foi implementado em linguagem C++ sobre a plataforma proprietária C++Builder® 6.0. Nesta plataforma o processo de codificação torna-se mais simples, no entanto, agrega-se um elevado custo computacional de processamento.

O modelo paralelo insular implementado no PIRGEP para indução de regras foi primeiramente avaliado na resolução de problemas genéricos de classificação e posteriormente em problemas de indução de regras para ACs.

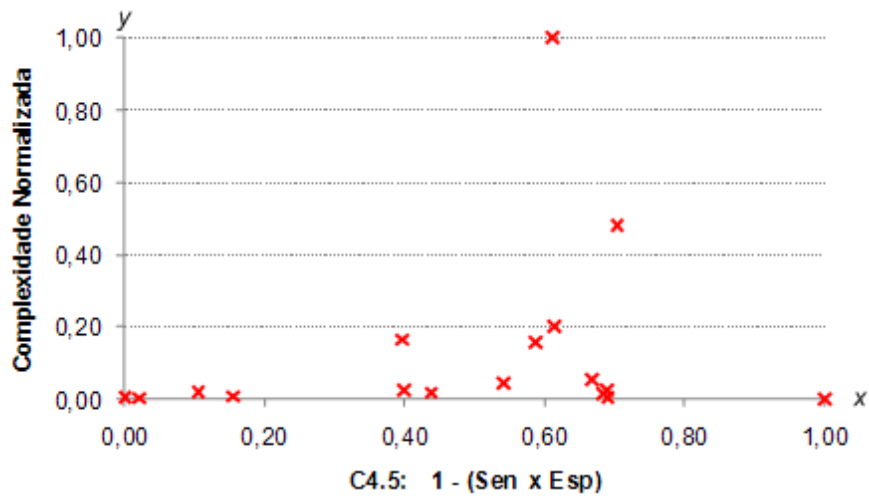
Como exemplos de problemas de classificação foram utilizadas 10 bases de dados diferentes. Cada base apresenta um número variável de atributos, classes e exemplos. Os resultados apresentados pelos modelos aqui propostos foram comparados com o algoritmo C4.5.

O processo de comparação seguiu a abordagem baseada no conceito de validação cruzada. Neste conceito, diferentes partições de dados alternam-se na composição das classes positivas e negativas durante o processo de treinamento e teste do algoritmo evolucionário. O algoritmo C4.5 utilizou as mesmas partições de dados em todos os níveis hierárquicos avaliados. Duas medidas receberam destaque neste processo de comparação: o produto entre a sensibilidade e a especificidade do algoritmo, e o número de nós que compõem a regra encontrada. A primeira mede a qualidade da solução e a segunda a complexidade. Deve-se salientar que as soluções mais interessantes são as que apresentam alto valor de qualidade, e um número baixo no quesito complexidade. A qualidade apresenta-se, sempre, como um valor normalizado entre [0..1] e a complexidade apenas possui um limite inferior que é 1.

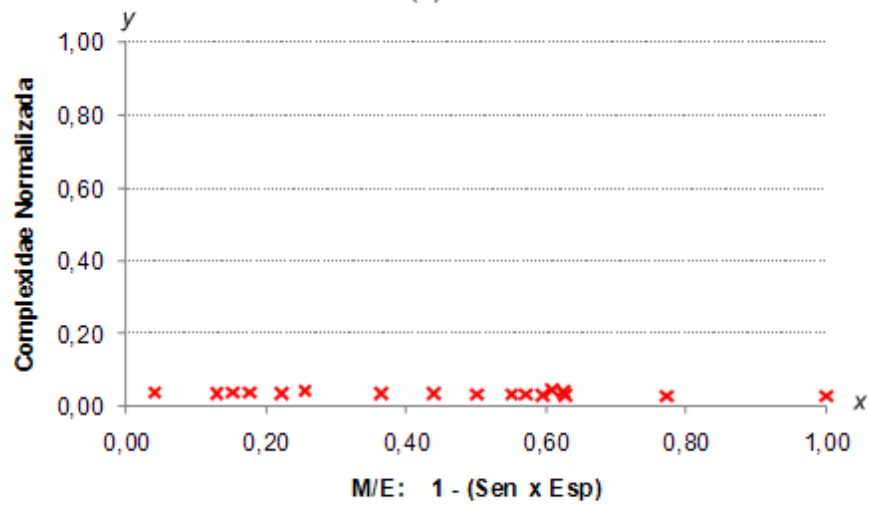
Nas 10 bases de dados foram totalizados 17 comparativos entre a tríade: C4.5, modelo mestre-escravo (M/E) de uma ilha e modelo mestre-escravo de 5 ilhas (M/E Ilhas). Cada comparativo reflete um nível hierárquico, por exemplo, na base *Abalone* (Figura 41) contabiliza-se 2 comparativos.

Os gráficos de Pareto mostrados na Figura 61 apresentam uma visão geral do desempenho dos 3 algoritmos: C4.5 (a), M/E (b) e M/E Ilhas (c), para todas as bases analisadas. No eixo x de cada gráfico encontra-se o complemento da: Sensibilidade x Especificidade, e no eixo y a complexidade normalizada no intervalo [0..1]. Assim, as melhores soluções apresentam-se mais próximas do ponto de intersecção (0,0) entre os eixos (x,y) .

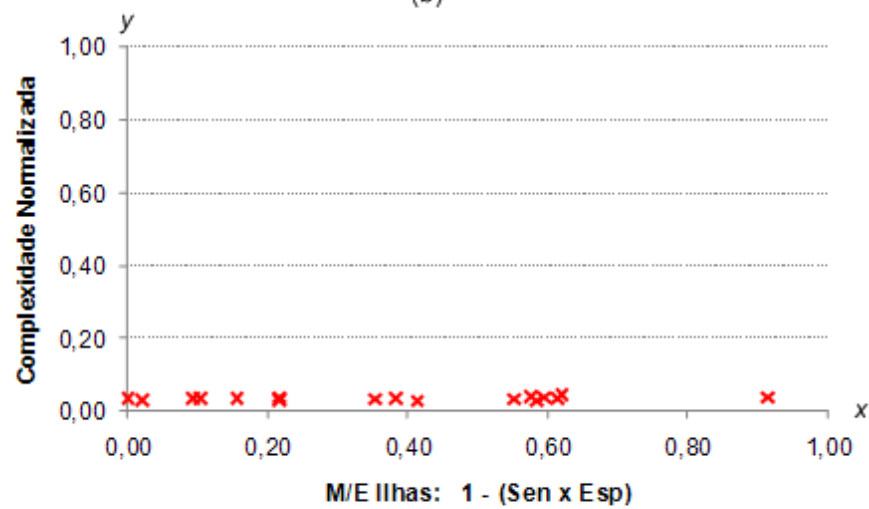
Pelos gráficos da Figura 61 é possível observar que o algoritmo C4.5 (Figura 61 (a)) apresentou o pior desempenho, tanto no quesito qualidade quanto no quesito complexidade. O algoritmo M/E (Figura 61 (b)) melhorou, consideravelmente, os resultados até então alcançados pelo C4.5, principalmente no quesito complexidade. E, o algoritmo M/E Ilhas (Figura 61 (c)) além de manter os níveis de complexidade apresentados pelo algoritmo M/E proporcionou melhores resultados no quesito qualidade.



(a)



(b)



(c)

Figura 61: Gráficos de Pareto: (a) C4.5, (b) M/E e (c) M/E Ilhas.

Na sequência discutem-se algumas particularidades dos 17 comparativos. Observando isoladamente a medida de qualidade, em 12 dos 17 comparativos o modelo mestre-escravo de 5 ilhas mostrou-se mais eficaz. O modelo mestre-escravo de 1 ilha destacou-se apenas em um comparativo e o C4.5 em outros 4. Agora, observando isoladamente a medida de complexidade, em 9 dos 17 comparativos o algoritmo C4.5 apresentou uma solução de menor complexidade, enquanto o modelo-mestre escravo com 5 ilhas contemplou 6, e o modelo-mestre escravo de 1 ilha contemplou 2.

Quando se associa a medida de qualidade à medida de complexidade tem-se o seguinte cenário: o modelo mestre-escravo com 5 ilhas apresentou a melhor associação em 5 comparativos. O modelo mestre-escravo de 1 ilha não se enquadrou neste cenário, e o C4.5 apresentou melhor associação em 3 comparativos.

Assim, ainda restam 9 comparativos que necessitam de uma análise mais criteriosa para que se possa endossar uma ou outra abordagem.

Na sequência é discutido cada um destes casos.

Na base de dados *Abalone*, 2^o nível hierárquico (Tabela 17) tem-se o algoritmo C4.5 como de melhor qualidade. No entanto, esta medida de qualidade é 0,007 maior que medida alcançada pelo modelo mestre-escravo com 5 ilhas. Além disto, a solução apresentada pelo C4.5 é, aproximadamente, 5 vezes mais complexa da que a apresentada pelo sistema mestre-escravo com 5 ilhas. Logo, a solução mais simples pode ser considerada como mais adequada para o problema.

Na base de dados *Balance Scala*, existem dois casos a serem estudados. No 1^o nível hierárquico (Tabela 18) fica evidente que nenhum método foi capaz de atingir uma solução satisfatória. Provavelmente, isto se deve ao desbalanceamento do número de amostras entre as classes positiva e negativa. A classe negativa possui 11,75 vezes mais amostras do que a classe positiva. No 2^o nível hierárquico existe uma forte tendência em eleger-se o método mestre-escravo de 5 ilhas como mais apropriado. Pois, apesar do mesmo apresentar uma solução com 9 nós a mais do que a apresentada pelo modelo mestre-escravo de 1 ilha, apresenta, aproximadamente, o dobro da qualidade.

Na base de dados *Car Evaluation*, também existem dois casos a serem estudados. No 1^o nível hierárquico (Tabela 19) tem-se uma qualidade melhor obtida pelo método mestre-escravo com 5 ilhas e uma menor complexidade obtida pelo método mestre-escravo de 1 ilha. Como a diferença é de apenas 2 nós na questão da complexidade, o método de melhor qualidade pode ser escolhido como melhor solução. No 3^o nível hierárquico desta mesma tabela a opção pelo C4.5 ou pelo mestre-escravo de 5 ilhas deve considerar algo além do que os valores obtidos para cada

uma das medidas. Observando-se apenas os valores tem-se a medida de qualidade favorecendo o método mestre-escravo de 5 ilhas e a medida de complexidade favorecendo o C4.5. Um bom critério de escolha poderia ser a utilização posterior da regra obtida. Se a mesma for manipulada por um sistema computacional, o mais interessante é ficar com a solução de melhor qualidade. No entanto, se a mesma for interpretada manualmente por algum especialista, o fator complexidade é muito mais importante.

Na base de dados *Haberman*, 1^o nível hierárquico (Tabela 44) depara-se com a mesma situação do 3^o nível hierárquico da base *Car evaluation*. No entanto, a dúvida agora contempla a escolha do método C4.5 ou do mestre-escravo de 1 ilha. Novamente vai depender da aplicabilidade futura da regra. Se for por qualidade escolhe-se o mestre-escravo de 1 ilha, se for por simplicidade o C4.5.

Na base de dados *Blood Transfusion*, 1^o nível hierárquico (Tabela 21) têm-se o modelo mestre-escravo de 5 ilhas com uma qualidade 30% superior ao C4.5. Neste caso a diferença de complexidade, na ordem de 12 nós, não justificaria a opção pelo C4.5.

Na base de dados *Wisconsin Breast Cancer*, 1^o nível hierárquico (Tabela 23) têm-se o modelo mestre-escravo de 5 ilhas com uma qualidade equivalente ao C 4.5. No entanto, a complexidade do C4.5 é menor, assim utilizá-lo como melhor solução é a melhor escolha.

Na base de dados *Nursery*, 3^o nível hierárquico (Tabela 23) têm-se o modelo mestre-escravo de 5 ilhas com uma qualidade 61% melhor do que o C4.5. Nestas condições a diferença de complexidade não é suficiente para justificar a escolha do método C4.5.

Os resultados alcançados, para os problemas de indução de regras de classificação, são bastante satisfatórios. O modelo proposto, na grande maioria das vezes, apresentou soluções mais interessantes do que as apresentadas pelo modelo C4.5, já consagrado na literatura. Além disto, o modelo paralelo insular em seu formato co-evolutivo mostrou-se capaz de melhorar muitos dos resultados alcançados pelo modelo baseado em uma única ilha de processamento.

Encerrado o processo de discussão dos resultados obtidos com os problemas de indução de regras de sistemas de classificação, inicia-se a discussão dos resultados para os problemas de indução de regras de ACs.

O modelo proposto para indução de regras de ACs induz regras para diferentes configurações de ACs. Basicamente, informa-se ao sistema a configuração inicial do AC, a configuração final do AC, o tipo de vizinhança a ser considerado no processo iterativo, bem como o número máximo de iterações aceito para levar o AC inicial ao AC final no processo de simulação da dinâmica.

Inicialmente, com intenção de verificar se o sistema estava realmente fazendo o que se propunha a fazer, foram realizados alguns experimentos controlados. Este termo foi utilizado pelo fato de que nestes experimentos as regras desejadas já eram previamente conhecidas.

Foram realizados 6 experimentos controlados com as regras elementares: 160, 118, 16, 95, 183 e 124. Estas regras apresentam, respectivamente, os comportamentos: nulo, periódico, ponto fixo, ciclo duplo, caótico e complexo. A configuração inicial do AC foi a mesma para todos os experimentos. Para cada experimento, diferentes configurações finais foram testadas. E sempre determinou-se o número exato de iterações necessárias para se conduzir a configuração inicial à configuração final desejada.

Para a regra 160 foram executadas 7 diferentes rodadas (Tabela 27), e o algoritmo encontrou todas as regras já na primeira geração com uma população de 30 indivíduos. Uma característica interessante deste experimento é que nas rodadas 6 e 7 foram encontradas outras regras que não a 160 que também solucionam o problema proposto. As rodadas 6 e 7 já se encontram fora do período transiente do sistema, ou seja, o sistema já se encontra estável. A partir da 6ª iteração já não existe mais mudanças nos estados das células que compõem o AC, então já era esperado que outras regras, também de comportamento nulo, fossem apresentadas como soluções para o problema. Por outro lado, nas rodadas que contemplam a análise do período transiente, somente a própria regra 160 foi apresentada como solução. Este comportamento pode ser explicado pela teoria de bacia de atração (WUENSCHÉ, 1994). Segundo esta teoria um estado de um AC possui um único estado sucessor, determinado pela aplicação da regra, mas pode possuir um número arbitrário de predecessores, conhecidos como pré-imagens. O espaço de regras no qual a regra 160 está inserida é composto por 256 regras, sendo que 30 delas, representadas pelos indivíduos da população (possivelmente existem regras repetidas, visto que nenhum mecanismo foi implementado para que isto não ocorresse), apresentam-se como solução para o problema na primeira geração. No início do processo iterativo, ou seja, dentro do período transiente, os diferentes estados dos ACs possuem poucos estados na condição de pré-imagem, ou seja as regras neste momento dão origem a ACs com estados bastante distintos, logo, somente tais regras são capazes de representar determinada dinâmica. Esta característica vai desaparecendo conforme o sistema vai entrando em seu estado estável.

Para a regra 118 foram executadas 11 diferentes rodadas (Tabela 28), e o algoritmo também encontrou todas as regras. No entanto foram necessários esforços computacionais diferenciados. A regra 118 apresenta um comportamento periódico com ciclo de tamanho 3. Novamente, evidencia-se a questão de que se os experimentos encontrarem-se dentro do período estável do sistema, será mais fácil para o algoritmo evolucionário encontrar a solução.

Para a regra 16 (Tabela 29) e 95 (Tabela 30) também não foi difícil encontrar a solução. Em poucas gerações as regras foram encontradas.

Para a regra 183 (Tabela 31) de comportamento caótico, foi necessário aumentar a diversidade genética para que fosse possível alcançar a solução. Isto ocorre, primeiramente, devido ao comportamento caótico ser minoria dentro das regras que compõem o espaço de busca. Segundo, que no comportamento caótico o sistema não se estabiliza.

Para a regra 124 (Tabela 32) de comportamento complexo, 30 indivíduos foram suficientes, uma vez que este comportamento estabiliza após um longo processo de simulação da dinâmica.

Os experimentos controlados apesar de muito simples do ponto de vista do algoritmo evolucionário, devido ao tamanho do espaço de busca, foram de suma importância para a compreensão do funcionamento do sistema como um todo. Foi possível observar diferentes comportamentos dinâmicos e entender que a busca de soluções pertencentes a períodos transientes pode ser muito difícil. Na verdade, esta é a realidade da grande maioria das aplicações reais para as quais este sistema se destina. Como são introduzidos apenas estados iniciais e finais dos ACs, o sistema não sabe nada a respeito da dinâmica que ele deve produzir para solucionar o problema de indução.

Na sequência realizou-se um experimento que avalia o potencial do algoritmo evolucionário. Este experimento consiste em encontrar uma regra para simulação da dinâmica apresentada pela Figura 58. Neste experimento o espaço de busca é formado por 4294967296 regras. A regra foi encontrada na geração 103 no modelo mestre-escravo de 5 ilhas. Ou seja, após avaliar, aproximadamente, 615000 regras o algoritmo encontrou a solução. Isto equivale a examinar apenas 0,014% do espaço de busca, o que mostra a eficiência do método utilizado.

Quando se utilizam arquiteturas paralelas de processamento, uma característica importante a ser considerada é o balanceamento de carga. Neste modelo ficou bem definido qual é o papel do processo mestre e qual o papel dos processos escravos. O balanço de carga consiste em distribuir as tarefas entre os processos escravos de forma a minimizar os tempos de comunicação, minimizando também, o tempo de computação da função de *fitness*. Não é possível fixar uma política de balanço de carga para todos os problemas, visto que cada um necessita de tempos de processamento distintos. Assim, esta análise deve ser especialmente realizada para cada problema que se pretende tratar. A Figura 59 apresenta um estudo de balanço de carga para uma instância do problema de classificação pela regra da maioria. Segundo o gráfico existe uma relação bastante linear entre o aumento de tarefas e o aumento do número de escravos. No entanto, este gráfico é válido apenas para este problema que, no caso, apresentou o melhor desempenho na relação de 100 tarefas para 100 escravos com o tempo de 370,93 segundos. Muitas vezes

pode-se pensar que a utilização do maior número possível de escravos é a melhor opção. No entanto, se o tempo de processamento da função de *fitness* for relativamente pequeno, um número elevado de processos escravos pode tornar-se um gargalo devido ao tempo de comunicação que será utilizado por estes processos.

O último experimento realizado tem a pretensão de definir um conjunto de parâmetros de referência para o algoritmo evolutivo. Os parâmetros avaliados têm uma relação direta com a manutenção da diversidade dentro da população. Foram realizados 54 experimentos, com um problema de simulação baseado na regra GKL dentro de seu período transiente (Figura 60), o que torna o problema de indução bastante difícil. O espaço de busca deste problema é de 2^{128} . A Tabela 33 apresenta os resultados para o modelo mestre-escravo de uma única ilha e a Tabela 34 para o modelo mestre-escravo de 5 ilhas. Apesar dos resultados não apresentarem diferenças muito significativas, evidenciou-se que os melhores resultados são fruto de um número maior de indivíduos aliado a um número menor de genes por classe. O parâmetro mutação com valor 10% pode parecer um pouco alto, se comparado com os valores utilizados na maioria dos algoritmos evolucionários. O que explica o bom desempenho deste parâmetro com valor 10% é o alfabeto binário a qual cada atributo está relacionado. Cada vez que um atributo vai mutar seu valor sempre existe 50% de chance deste permanecer com o mesmo valor, ou seja, muitas mutações não fazem nenhuma modificação no material genético. Assim, valores mais altos para mutação aumentam a chance de mutações reais acontecerem.

7 CONCLUSÕES E TRABALHOS FUTUROS

7.1 CONCLUSÕES

As principais contribuições desta tese são:

- Desenvolvimento de um modelo para indução de regras de ACs baseado em PPCD;
- Desenvolvimento de um modelo para previsão de comportamento dinâmico nulo;
- Diminuição do custo computacional referente ao processo de simulação da dinâmica dos ACs;
- Implementação de uma arquitetura híbrida baseada em *hardware* e *software* para indução de regras de ACs;
- Exploração do paralelismo intrínseco dos ACs;
- Implementação de um modelo paralelo insular para indução de regras em um ambiente dinâmico com alto poder de processamento;
- Uma implementação paralela para o algoritmo da PEG que pode ser aplicada em diferentes classes de problemas.

O problema de indução de regras para ACs mostrou-se complexo. Esta tese analisou o problema sobre quatro diferentes aspectos.

Num primeiro momento considerou-se a utilização dos PPCD no processo de indução de regras. Trinta e uma regras (Tabela 2) foram utilizadas no processo de análise dos PPCD. Os parâmetros analisados foram: *S*, *DV*, *PA*, *AA*, *Z_W*, *A*, *CM* e *Z_Li*. O principal objetivo desta análise foi verificar a capacidade individual e coletiva destes parâmetros no processo de previsão do comportamento dinâmico. Adotou-se como objeto de estudo o problema intitulado classificação pela regra da maioria. Este problema exibe uma característica importante e fundamental para a análise realizada. Esta característica refere-se ao comportamento exibido pelas regras

que são conhecidas como soluções para o problema. Todas pertencem à classe rotulada como nulas, segundo a classificação de Li e Packard (1990). Os resultados alcançados sugerem que não é possível identificar uma fronteira de valores para os PPCD para indução de regras nulas. Assim, conclui-se que regras com valores de PPCD bastante próximos dos valores de referência utilizados podem apresentar comportamentos dinâmicos distintos. Isto pode comprometer a eficiência de algoritmos de computação evolucionária para problemas de indução de regras em ACs que utilizam tais parâmetros. Da forma como os PPCD foram utilizados neste trabalho, faz-se necessária a existência de um conjunto de valores de referência que normalmente é desconhecido. Sendo assim, os resultados sugerem a necessidade de se desenvolver PPCD que possam capturar com mais expressividade os aspectos dinâmicos do sistema modelado.

No estudo sobre comportamento dinâmico foram analisados dois métodos apresentados pela literatura e foi proposto um novo método, que não considera apenas as regras de transição, como os PPCD, mas também considera as características apresentadas em num período parcial da dinâmica do sistema para previsão do comportamento dinâmico. A primeira análise apresentou a metodologia proposta por Wolfram baseada no cálculo da distância de Hamming entre duas configurações de AC idênticas exceto pela mudança aleatória de um estado em uma das configurações. Os resultados desta metodologia são bastante satisfatórios, no entanto, só permitem uma análise a respeito do comportamento local do sistema, uma vez que apenas dois ACs são utilizados na análise. A segunda análise destina-se especificamente à previsão de comportamentos dinâmicos determinada em função de valores de parâmetros de previsão de comportamento, ou seja, não utiliza a simulação da dinâmica no processo. Novamente, os resultados demonstram que a utilização destes parâmetros como previsores de comportamento não é satisfatória, visto que regras que apresentam valores de parâmetros similares, quando submetidas ao processo de simulação do sistema, demonstram comportamentos dinâmicos distintos. Por fim, introduziu-se o conceito de dinâmica parcial. O método proposto mostrou-se eficiente na tarefa de previsão de comportamento dinâmico nulo. O número de iterações necessárias para se determinar o comportamento dinâmico do sistema foi reduzido em 70% sem perda de desempenho. Os resultados até então obtidos encorajam o desenvolvimento de novos trabalhos. Sugere-se a aplicação deste conceito de simulação parcial da dinâmica, com as devidas adaptações, para outras classes de comportamento dinâmico além do comportamento nulo.

Na sequência dos trabalhos desenvolveu-se um modelo para indução de regras de ACs baseado na abordagem conhecida como *Hardware-Software Co-design*. O objetivo aqui não foi encontrar a melhor regra para o problema de classificação pela regra da maioria, mas sim, propor uma abordagem viável capaz de acelerar o cálculo da função de *fitness* dentro de um algoritmo evolucionário. Embora o uso de um servidor *web* tenha sido adequada para este projeto,

o seu desenvolvimento foi complexo devido à dificuldade em sincronizar elementos de *hardware* e *software* entre o computador e o sistema integrado. Em versões preliminares o sistema de comunicação baseou-se em interfaces seriais e paralelas, no entanto, a comunicação através de uma rede *Ethernet*, embora mais lenta, é mais estável e fácil de usar, sendo então empregada neste projeto. A comparação mostrou que o desempenho do modelo proposto foi 283 vezes mais rápido do que uma abordagem executada somente em *software*. Se considerássemos a situação hipotética em que as frequências de *clock* da FPGA e do ambiente de trabalho fossem os mesmos, teria-se um *speedup* normalizado na casa dos milhares. Este resultado indica fortemente que a metodologia proposta pode ser útil para muitos problemas computacionais tratados com técnicas de computação evolucionária na qual a função de *fitness* é computacionalmente cara. Apesar do bom resultado alcançado no que se refere a arquitetura implementada, a mesma não trouxe boas perspectivas, no sentido de ser extrapolada para o desenvolvimento de uma metodologia genérica de evolução de regras para ACs. A principal restrição do modelo refere-se à especificidade da aplicação implementada no *hardware* reconfigurável. Qualquer alteração no modelo demanda um alto custo de desenvolvimento e reconfiguração da FPGA. Outra restrição é o número de células lógicas programáveis disponíveis dentro da FPGA. Dependendo das operações que precisam ser implementadas para uma determinada aplicação, o número de células lógicas disponíveis não é suficiente para implementação dos circuitos que realizariam tais operações. Com o contínuo desenvolvimento tecnológico, novos dispositivos FPGA de maior capacidade e velocidade estarão disponíveis. De maneira análoga, sistemas de desenvolvimento mais ágeis e eficientes poderão diminuir a complexidade de novas implementações. Desta forma, a abordagem de *Hardware-Software Co-design* poderá vir a ser uma alternativa promissora num futuro próximo.

Por fim, apresenta-se um modelo paralelo insular para indução de regras. Duas vertentes foram exploradas: a indução de regras para problemas genéricos de classificação e a indução de regras para ACs. O modelo proposto disponibiliza um ambiente dinâmico com alto poder de processamento. Um sistema de codificação genérico permite que diferentes problemas sejam tratados sem qualquer modificação no modelo. Os resultados obtidos foram bastante satisfatórios. Os problemas de classificação tinham uma função especial dentro de todo o processo. Embora não sejam o foco principal deste trabalho, eles foram utilizados como referência para que fosse possível avaliar o funcionamento e as características do modelo evolucionário proposto. Como os problemas de indução de regras para ACs são significativamente mais complexos que os problemas de classificação, foi de suma importância verificar se o modelo estava funcionando corretamente antes de aplicá-lo aos problemas de indução de regras para ACs. Nestes problemas o sistema também se demonstrou eficiente. Os resultados enfatizaram a difi-

culdade em se induzir regras que contemplam sistemas dinâmicos ainda em períodos de instabilidade.

Entende-se que existem pontos a serem explorados na continuidade deste trabalho. No entanto, todos os objetivos iniciais foram satisfatoriamente alcançados. As contribuições deste trabalho foram tanto teóricas, no caso das avaliações realizadas sobre os PPCD e os diferentes métodos de previsão de comportamento dinâmico dos sistemas, quanto metodológicas, nas duas arquiteturas de processamento propostas. Nos Anexos I (WEINERT; LOPES, 2007) e II (WEINERT; LOPES, 2009) encontram-se 2 artigos publicados resultantes desta tese.

7.2 TRABALHOS FUTUROS

Este trabalho apresenta um universo de possibilidades para pesquisa científica.

No modelo proposto para indução de regras utilizando os PPCD seria importante testar outras funções de *fitness*. A função utilizada é composta pela multiplicação de 6 parâmetros. Se um for igual a zero, toda a função é penalizada. Uma abordagem multiobjetivo poderia melhorar a distinção entre os indivíduos e, conseqüentemente, favorecer a busca por soluções de melhor qualidade.

O modelo de previsão de comportamento dinâmico baseado no conceito de simulação parcial da dinâmica pode ser ampliado para previsão de outros comportamentos dinâmicos, além do nulo. Acredita-se que a mesma metodologia possa ser aplicada sobre comportamentos periódicos. Nestes comportamentos, deixariam de existir transições dominantes e passariam a existir transições oscilantes no que se refere ao número de ocorrência das mesmas.

O modelo híbrido baseado na técnica de *Hardware-Software Co-design* pode ser particularizado para diferentes problemas. Embora existam algumas restrições físicas relacionadas ao número de células lógicas disponíveis para programação, a abordagem é insuperável na questão desempenho. Funções de *fitness* com características paralelizáveis podem se beneficiar desta arquitetura.

O modelo paralelo proposto pode ser aplicado a diferentes problemas complexos de indução de regras para ACs. Dentre estes problemas pode-se destacar os relacionados a bioinformática como: modelagem de reações enzimáticas, simulação do crescimento de tumores, espalhamento de infecções, alinhamento de seqüências de DNA, dobramento de proteínas, entre outros. Bem como, também ser aplicado a uma generalidade de problemas de classificação, onde sua principal vantagem em relação a outros métodos de indução de regras apresenta-se na simplicidade e conseqüente compreensibilidade das regras obtidas.

No processo de co-evolução testou-se apenas a arquitetura insular com 5 ilhas de processamento numa topologia em anel. A literatura apresenta uma série de variações topológicas e políticas migratórias que também podem ser exploradas no modelo proposto.

O carácter genérico do sistema dificulta a determinação de um conjunto ótimo de parâmetros evolucionários. O ajuste automático através de um processo co-evolucionário poderia trazer bons resultados.

Um trabalho interessante seria a união dos modelos propostos. Esta união poderia ser concebida em dois estágios. Primeiramente, desenvolvendo-se um novo PPCD baseado no modelo de previsão de dinâmica parcial que pudesse ser incorporado a função de *fitness* implementada no PIRGEP para tarefa de indução de regras de ACs. Num segundo estágio interconectar um conjunto de FPGAs ao *cluster*. Apesar das limitações das FPGAs elas poderiam processar módulos genéricos, aplicáveis a um conjunto de problemas, e os computadores interligados no *cluster* processar módulos mais específicos, ou que necessitem de recursos não disponibilizados pelas FPGAs. Esta nova arquitetura de processamento poderia reduzir os tempos de processamento de muitos problemas.

REFERÊNCIAS

- AHAMED, E.; ELGAZZAR, A. S. On some applications of cellular automata. *Physica A*, v. 296, n. 3-4, p. 529–538, 2001.
- AHLGREN, P.; JARNEVING, B.; ROUSSEAU, R. Requirements for a cocitation similarity measure, with special reference to Pearson's correlation coefficient. *Journal of the American Society for Information Science and Technology*, v. 54, n. 6, p. 550–560, 2003.
- ALONSO-SANZ, R.; MARTIN, M. Elementary cellular automata with elementary memory rules in cells: the case of linear rules. *Journal of Cellular Automata*, v. 1, n. 1, p. 71–81, 2006.
- ANDRE, D.; BENNETT III, F. H.; KOZA, J. R. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: *Proceedings of the First Annual Conference on Genetic Programming - GECCO '96*. Cambridge, MA, USA: MIT Press, 1996. p. 3–11.
- BANZHAF, W.; NORDIN, P.; KELLER, R. E.; FRANCONI, F. D. *Genetic Programming: An Introduction*. San Francisco: Morgan Kaufmann, 1998.
- BAR-YAM, Y. *Dynamics of Complex Systems*. Reading, USA: Addison-Wesley, 1997.
- BATTY, M. Geocomputation using cellular automata. In: *Geocomputation editado por Openshaw, S. e Abrahart, R.J. (Eds.)*. New York: Taylor & Francis, 2000. p. 95–126.
- BEAUCHEMINA, C.; SAMUEL, J.; TUSZYNSKIA, J. A simple cellular automaton model for influenza A viral infections. *Journal of Theoretical Biology*, v. 232, n. 2, p. 223–234, 2005.
- BECKER, J.; HARTENSTEIN, R. Configware and morphware going mainstream. *Journal of Systems Architecture*, v. 49, n. 4-6, p. 127–142, 2003.
- BENKINIOUAR, M.; BENMOHAMED, M. Cellular automata for cryptography. In: *Proceedings of International Conference on Information and Communication Technologies: From Theory to Applications*. Damascus, Syria: Rifai, 2004. p. 423–424.
- BERLEKAMP, E.; CONWAY, J.; GUY, R. *Winning Ways for Your Mathematical Plays*. 2nd. Natick, MA: AK Peters, 2001.
- BINDER, P. M. A phase diagram for elementary cellular automata. *Complex Systems*, v. 7, n. 1, p. 241–247, 1993.
- BORTOT, J. C.; DE-OLIVEIRA, P. P. B.; OLIVEIRA, G. M. B. Multiobjective, heuristic evolutionary search in a cooperative environment leads to the best cellular automaton rule in the density classification task. In: *Proceedings of VIII Brazilian Symposium on Neural Networks*. São Luís: IEEE Press/SBC, 2004. p. CD-ROM:Paper 3565.

- CABRERA, A.; SANCHEZ-SOLANO, S.; BROX, P.; BARRIGA, A.; SENHADJI, R. Hardware/software codesign of configurable fuzzy control systems. *Applied Soft Computing*, v. 4, n. 3, p. 271 – 285, 2004.
- CÂMARA, A. S.; FERREIRA, F.; CASTRO, P. Spatial simulation modeling. In: *Spatial Analytical Perspective on GIS editado por M. M. Fisher, H. J. Scholten e D. J. Unwin*. London: Taylor & Francis, 1996. p. 201–228.
- CHEN, Q.; MYNETT, A. E. Effects of cell size and configuration in cellular automata based prey-predator modelling. *Simulation, Modelling, Practice and Theory*, v. 11, n. 7-8, p. 609–625, 2003.
- COLEY, D. A. *An Introduction to Genetic Algorithms for Scientists and Engineers*. Singapore: World Scientific Publishing Company, 1997.
- COMPTON, K.; HAUCK, S. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, v. 34, n. 2, p. 171–210, 2002.
- CORNE, D. W.; FRISCO, P. Dynamics of HIV infection studied with cellular automata and conformon-P system. *BioSystems*, n. 91, p. 531–544, 2008.
- CRANNY, T.; BOSSOMAIER, T. R. J. *The density classification problem for cellular automata: searching within structure*. Austrália, 1999.
- CULIK, K.; HURD, L. P.; YU, S. Computation theoretic aspects of cellular automata. *Physica D*, v. 45, n. 1-3, p. 357–378, 1990.
- DAS, R.; CRUTCHFIELD, J.; MITCHELL, M. A genetic algorithm discovers particle-based computation in cellular automata. *Lecture Notes Computer Science*, v. 866, p. 344–353, 1994.
- DAS, R.; CRUTCHFIELD, J.; MITCHELL, M.; HANSON, J. Evolving globally synchronized cellular automata. In: *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann, 1995. p. 336–343.
- DAVIS, L. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.
- DE-OLIVEIRA, P. P. B.; BORTOT, J. C.; OLIVEIRA, G. M. B. The best currently know class of dynamically equivalent cellular automata rules for density classification. *Neurocomputing*, v. 70, n. 1-3, p. 35–43, 2006.
- DIAO, Y.; MA, D.; WEN, Z.; YIN, J.; XIANG, J.; LI, M. Using pseudo amino acid composition to predict transmembrane regions in protein: cellular automata and Lempel-Ziv complexity. *Amino Acids*, v. 34, n. 1, p. 111–117, 2007.
- DONGARRA, J.; FOX, G.; KENNEDY, K.; TORCZON, L.; GROPP, W.; FOSTER, I.; WHITE, A. *The Sourcebook of Parallel Computing*. New York: Morgan Kaufmann, 2002.
- DU, X.; DING, L.; JIA, L. Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm. In: *Proceedings of the Fourth International Conference on Natural Computation*. Jinan, China: IEEE Computer Society, 2008. p. 433–437.
- EIBEN, A. E.; SMITH, J. E. *Introduction to Evolutionary Computing*. Berlin, Germany: Springer Verlag, 2008.

- FERREIRA, C. Gene expression programming: a new adaptative algorithm for solving problems. *Complex Systems*, v. 13, n. 2, p. 87–129, 2001.
- FERREIRA, C. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Portugal: Angra do Heroísmo, 2002a.
- FERREIRA, C. Discovery of the boolean functions to the best density-classification rules using gene expression programming. *Lecture Notes in Computer Science*, v. 2278, p. 51–60, 2002b.
- FERREIRA, C. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. 2nd. Berlin: Springer-Verlag, 2006.
- FOGEL, D. B.; OWENS, A. J.; WALSH, M. J. *Artificial Intelligence Through Simulated Evolution*. New York: John Wiley & Sons, 1996.
- FREITAS, A. A. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Berlin: Springer-Verlag, 2002.
- FU, S. C.; MILNE, G. Epidemic modeling using cellular automata. In: *Proceedings of the 1st Australian Conference on Artificial Life*. Canberra, Australia: World Scientific, 2003. p. 43–57.
- GACS, P.; KURDYUMOV, G. L.; LEVIN, L. A. One dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii*, v. 14, n. 3, p. 92–98, 1978.
- GANGADHAR, D. K. Pelican - protein-structure alignment using cellular automata models. In: *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms*. Coimbra, Portugal: Springer-Verlag, 2005. p. 308–311.
- GEORGOUDAS, I. G.; SIRAKOULIS, G. C.; ANDREADIS, I. Modelling earthquake activity features using cellular automata. *Mathematical and Computer Modelling*, v. 46, p. 124–137, 2007.
- GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, USA: Addison-Wesley, 1989.
- GONÇALVES, R. M. *Modelagem preditiva através de autômatos celulares aplicada a uma sequência de imagens adquiridas por sensoriamento remoto*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática Industrial e Engenharia Elétrica - UTFPR-PR, Curitiba-PR, 2004.
- GROPP, W.; LUSK, E.; SKJELLUM, A. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. 2nd. Cambridge, MA, USA: MIT Press, 1999.
- GROPP, W.; LUSK, E.; THAKUR, R. *Using MPI-2: Advanced Features of the Message-Passing Interface*. Cambridge, MA, USA: MIT Press, 1999.
- GUTOWITZ, H. Cryptography with dynamical systems. In: *Cellular Automata and Cooperative Phenomena editado por Goles, E. e Boccara, N. (Eds.)*. Norwell: Norwell Academic Press, 1995. p. 237–274.
- HAND, D. *Construction and Assessment of Classification Rules*. Chichester, England: John-Wiley & Sons Ltd, 1997.

- HAUCK, S.; DEHON, A. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing*. Burlington, MA, USA: Morgan Kaufman, 2008.
- HOFMAN, K.; BUCHER, P.; FALQUET, L.; BAIROCH, A. The PROSITE database, its status in 1999. *Nucleic Acids Research*, v. 27, n. 1, p. 215–219, 1999.
- HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.
- ITO, S. A.; CARRO, L. A comparison of microcontrollers targeted to FPGA-based embedded applications. In: *Proceedings of IEEE 13th Symposium on Integrated Circuits and Systems Design*. Manaus, Brazil: IEEE Computer Society, 2000. p. 397–402.
- JIANG, M.; LUO, Y.; SZETO, K.; YANG, S. Dynamics of negotiating agent in a soap froth world. In: *Proceedings of the 8th International Conference on Neural Information Processing*. Shanghai, China: Springer Verlag, 2001.
- JUILLÉ, H.; POLLACK, J. B. Coevolving the ideal trainer: application to the discovery of cellular automata rules. In: et al, J. K. e (Ed.). *Proceedings of the Third Annual Conference in Genetic Programming*. San Francisco: Morgan Kaufmann, 1998. p. 519–527.
- KANSAL, A. R.; TORQUATO, S.; IV, G. R. H.; CHIOCCA, E. A.; DEISBOECK, T. S. Cellular automaton of idealized brain tumor growth dynamics. *BioSystems*, v. 55, n. 1-3, p. 119–127, 2000.
- KHAN, A. R.; CHOUDHURY, P. P.; DIHIDAR, K.; VERMA, R. Text compression using two-dimensional cellular automata. *Computers and Mathematics with Applications*, v. 37, n. 6, p. 115–127, 1999.
- KIERA, L. B.; CHENGA, C. K.; TESTAB, B.; CARRUPTB, P. A. A cellular automata model of enzyme kinetics. *Journal of Molecular Graphics*, v. 14, n. 4, p. 227–231, 1996.
- KOZA, J. R. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. Cambridge, USA: MIT Press, 1992.
- KOZA, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, USA: MIT Press, 1994.
- KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach Featuring the Internet Package*. Boston, MA, USA: Addison Wesley, 2007.
- LANGDON, W. B.; POLI, R. *Foundations of Genetic Programming*. New York: Springer, 2002.
- LANGTON, C. G. Computation at the edge of chaos: phase transitions and emergent computation. *Physica D*, v. 42, n. 1-3, p. 12–37, 1990.
- LANGTON, C. G. *Artificial Life: an Overview*. Cambridge: The MIT Press, 1995.
- LAURIO, K.; LINAKER, F.; NARAYANANA, A. Regular biosequence pattern matching with cellular automata. *Information Sciences*, v. 146, p. 89–101, 2002.
- LI, W. *Problems in Complex Systems*. Tese (Doutorado) — PhD Thesis, Columbia University, 1989.

- LI, W. *Parameterizations of cellular automata rule space*. Santa Fe, New Mexico, USA, 1991.
- LI, W. Phenomenology of nonlocal cellular automata. *Journal of Statistical Physics*, v. 68, n. 5-6, p. 829–882, 1992.
- LI, W.; PACKARD, N. The structure of elementary cellular automata rule space. *Complex Systems*, v. 4, n. 3, p. 281–297, 1990.
- LIMA, C. R. E.; ROSÁRIO, J. M. Reconfigurable architecture proposal applied to mobile robots. In: *Proceedings of 11th International Conference on Advanced Robotics, ICAR 2003*. New York: [s.n.], 2003. p. 601–606.
- LOPES, H. S. *Analogia e aprendizado evolucionário: aplicação em diagnóstico clínico*. Tese (Doutorado) — Engenharia Elétrica, Sistemas de Informação - Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, Brasil, 1996.
- LOPES, H. S.; LIMA, C. R. E.; MURATA, N. J. A configware approach for high-speed analysis of genomic data. *Journal of Circuits, Systems and Computers*, v. 16, p. 527–540, 2007.
- LOPES, H. S.; WEINERT, W. R. EGIPSYS: a enhanced gene expression programming approach for symbolic regression problems. *International Journal of Applied Mathematics and Computer Science*, v. 14, n. 3, p. 375–384, 2004.
- LUGER, G. F. *Inteligência Artificial: Estruturas e Estratégias para a Solução de Problemas Complexos*. Porto Alegre, RS: Bookmann, 2004.
- MALLETA, D. G.; PILLISB, L. G. A cellular automata model of tumor-immune system interactions. *Journal of Theoretical Biology*, v. 239, n. 3, p. 234–350, 2006.
- MARTÍNEZ, G. J. *Teoría del Campo Promedio en Autómatas Celulares Similares a “The Game of Life”*. Dissertação (Mestrado) — Centro de Investigaciones y Estudios Avanzados del I.P.N., Distrito Federal - México, 2000.
- MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, v. 8, n. 1, p. 3–30, 1998.
- Message Passing Interface Forum. Mpich2: A message passing interface standard. *International Journal of High Performance Computing Applications*, v. 12, p. 1–299, 1998.
- MITCHELL, M. Computation in cellular automata: A select review. In: GRAMSS, T.; et al. (Ed.). *Nonstandard Computation*. VHC Verlagsgesellschaft: Weinheim, 1996. p. 95–140.
- MITCHELL, M.; CRUTCHFIELD, J. P.; HRABER, P. T. Evolving cellular automata to perform computations: mechanisms and impediments. *Physica D*, v. 75, n. 1-3, p. 361–391, 1994.
- MITCHELL, M.; HRABER, P. T.; CRUTCHFIELD, J. P. Revisiting the edge of chaos: evolving cellular automata to perform computations. *Complex Systems*, v. 7, n. 2, p. 89–130, 1993.
- MIYAZAKI, T. Reconfigurable systems: a survey. In: *Proceedings of the IEEE Design Automation Conference*. Yokohama, Japan: IEEE Computer Society Press, 1998. p. 447–452.

MIZAS, C.; SIRAKOULIS, G. C.; MARDIRES, V.; KARAFYLLIDIS, I.; GLYKOS, N.; SANDALZOPOULOS, R. Reconstruction of DNA sequence using genetic algorithms and cellular automata: towards mutation prediction? *BioSystems*, v. 92, n. 1, p. 61–68, 2008.

MORALES, F. J.; CRUTCHFIELD, J. P.; MITCHELL, M. Evolving two-dimensional cellular automata to perform density classification: a report on work in progress. *Parallel Computing*, v. 27, n. 5, p. 571–585, 2001.

NEDJAH, N.; MOURELLE, L. M. Software/hardware co-design of efficient and secure cryptographic hardware. *Journal of Universal Computer Science*, v. 11, n. 1, p. 66–82, 2005.

NEUMANN, J. V.; BURKS, A. *Theory of self-reproducing automata*. Urbana Champaign, USA, 1966.

OLIVEIRA, G. M. B. Autômatos celulares: aspectos dinâmicos e computacionais. In: *III Jornada de Mini-cursos em Inteligência Artificial (MCIA)*. Campinas, São Paulo: Sociedade Brasileira de Computação, 2003. v. 8, p. 297–345.

OLIVEIRA, G. M. B.; ASAKURA, O. K. N.; DE-OLIVEIRA, P. P. B. Dynamic behaviour forecast as a driving force in the coevolution of one-dimensional cellular automata. In: *Proceedings of the VII Brazilian Symposium on Neural Networks*. Los Alamitos, CA, USA: IEEE Computer Society, 2002. p. 98–103.

OLIVEIRA, G. M. B.; ASAKURA, O. K. N.; DE-OLIVEIRA, P. P. B. Coevolutionary search for one-dimensional cellular automata, based on parameters related to their dynamic behaviour. *Journal of Intelligent & Fuzzy Systems*, v. 13, n. 2-4, p. 99–110, 2003.

OLIVEIRA, G. M. B.; BORTOT, J. C.; DE-OLIVEIRA, P. P. B. Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task. In: *Proceedings of the 8th International Conference on Artificial Life, ALIFE VIII*. Sydney: Cambridge: MIT Press, 2002. p. 202–206.

OLIVEIRA, G. M. B.; BORTOT, J. C.; DE-OLIVEIRA, P. P. B. Heuristic search for cellular automata density classifiers with a multiobjective evolutionary algorithm. In: *Proceedings of VI Congress of Logic Applied to Technology LAPTEC 2007*. Santos, SP: CD-ROM Paper ID:213, 2007. v. 1, p. 1–12.

OLIVEIRA, G. M. B.; COELHO, A. R.; MONTEIRO, L. H. A. Cellular automata cryptographic model based on bi-directional toggle rules. *International Journal of Modern Physics C*, v. 15, n. 8, p. 1061–1068, 2004.

OLIVEIRA, G. M. B.; DE-OLIVEIRA, P. P. B.; OMAR, N. Guidelines for dynamics-based parameterization of one-dimensional cellular automata rule spaces. *Complexity*, v. 6, n. 2, p. 63–71, 2000a.

OLIVEIRA, G. M. B.; DE-OLIVEIRA, P. P. B.; OMAR, N. Evolving solutions of the density classification task in 1d cellular automata, guided by parameters that estimate their dynamic behavior. In: BEDAU, M. e. a. E. editado por (Ed.). *Proceedings of the Seventh International Conference on Artificial Life, ALIFE VII*. Portland: Cambridge: MIT Press, 2000b. p. 428–436.

- OLIVEIRA, G. M. B.; DE-OLIVEIRA, P. P. B.; OMAR, N. Improving genetic search for cellular automata, using heuristics related to their dynamical behavior. In: *Proceeding of the IEEE Congress on Evolutionary Computation, CEC2001*. Seoul: IEEE Computer Society, 2001. p. 348–355.
- OLIVEIRA, G. M. B.; OMAR, N.; DE-OLIVEIRA, P. P. B. Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. *Artificial Life*, v. 7, n. 3, p. 277–301, 2001.
- OLIVEIRA, G. M. B.; SIQUEIRA, S. R. C. Using dynamic behavior prediction to guide an evolutionary search for designing two-dimensional cellular automata. In: *Advances in Artificial Life, VIIIth European Conference on Artificial Life*. Canterbury: Springer Berlin / Heidelberg, 2005. v. 3630, p. 491–499.
- OLIVEIRA, G. M. B.; SIQUEIRA, S. R. C. Parameter characterization of two-dimensional cellular automata rule space. *Physica D*, v. 15, n. 8, p. 1–6, 2006.
- O'REILLY, U. *An Analysis of Genetic Programming*. Ottawa, Canadá, 1995.
- PAGE, I. Reconfigurable processor architectures. *Microprocessors and Microsystems*, v. 20, n. 3, p. 185–196, 1996.
- PARK, H.; GRINGS, A.; SANTOS, M. V. dos; SOARES, A. S. Parallel hybrid evolutionary computation: automatic tuning os parameters for parallel gene expression. *Applied Mathematics and Computation*, v. 201, n. 1-2, p. 108–120, 2008.
- PEDRONI, V. A. *Circuit Design with VHDL*. Cambridge: MIT Press, 2004.
- PETERSON, W. W.; WELDON, E. J. *Error-correcting codes*. Cambridge: MIT Press, 1972.
- POLI, R.; LANGDON, W. B.; MCPHEE, N. F. *A Field Guide to Genetic Programming*. United Kingdom: Lulu Enterprises, 2008.
- QUINLAN, J. R. Induction of decision trees. *Machine Learning*, v. 1, n. 1, p. 81–106, 1986.
- QUINLAN, J. R. *C4.5: Programs for Machine Learning*. San Mateo: Morgan Kaufmann, 1993.
- RICHARDS, F. C.; MEYER, T. P.; PACKARD, N. H. Extracting cellular automaton rules directly from experimental data. *Physica D*, v. 45, n. 1-3, p. 189–202, 1990.
- RIGO, L. O.; BARBOSA, V. C. Two-dimensional cellular automata and the analysis of correlated time series. *Pattern Recognition Letters*, v. 27, n. 12, p. 1353–1360, 2006.
- ROOSTA, S. H. *Parallel Processing and Parallel Algorithms: Theory and Computation*. New York: Springer-Verlag, 1999.
- ROSÁRIO, J. M.; LIMA, C. R. E.; FERASOLLI, H.; PEROGARO, R. Reconfigurable architecture proposal to application on mobile embedded systems prototypes. In: *Proceedings of 7th IFAC Symposium of Robot Control-SYROCO'03*. Wroclaw, Poland: Elsevier, 2003. p. 89–94.
- ROSIN, P. L. Training cellular automata for image processing. *Lecture Notes in Computer Science*, v. 3540, p. 195–204, 2005.

SEIFERAS, J. Observations on nondeterministic multidimensional iterative array. In: *Proceedings of the ACM Symposium on the Theory of Computing*. New York: ACM Press, 1982. p. 276–289.

SEREDYNSKI, F. Discovery with genetic algorithm scheduling strategies for cellular automata. In: *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 1998. v. 1498, p. 643–652.

SIPPER, M. Co-evolving non-uniform cellular automata to perform computations. *Physica D*, v. 92, p. 193–208, 1996.

SIVANANDAM, S. N.; DEEPA, S. N. *Introduction to Genetic Algorithms*. New York: Springer, 2007.

SNIR, M.; OTTO, S.; HUSS-LEDERMAN, S.; WALKER, D.; DONGARRA, J. *MPI: The Complete Reference*. 2nd. Cambridge: MIT Press, 1997.

SPEARS, W. M.; GORDON, D. F. Evolving finite-state machine strategies for protecting resources. *Lecture Notes in Computer Science*, v. 1932, p. 166–175, 2000.

SWIECICKA, A.; SEREDYNSKI, F. Cellular automata approach to scheduling problem. In: *Proceedings of International Conference on Parallel Computing in Electrical Engineering*. Washington, DC, USA: IEEE Computer Society, 2000. p. 29–33.

TOMASSINI, M.; PERRENOUD, M. Cryptography with cellular automata. *Applied Soft Computing*, v. 1, n. 2, p. 151–160, 2001.

VIDICA, P. M.; OLIVEIRA, G. M. B. Cellular automata-based scheduling: A new approach to improve generalization ability of evolved rules. In: *Proceedings of IX Brazilian Symposium on Neural Networks*. Ribeirão Preto, SP, Brasil: IEEE Computer Society, 2006. p. 1–6.

VINCENT, W. J. *Statistics in Kinesiology*. Champaign: Human Kinetics, 2005.

WANG, D.; LI, S.; DOU, Y. Collaborative hardware/software partition of coarse-grained reconfigurable system using evolutionary ant colony optimization. In: *ASP-DAC '08: Proceedings of the 2008 Conference on Asia and South Pacific Design Automation*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2008. p. 679–684.

WEI, Y.; YING, S.; FAN, Y.; WANG, B. The cellular automata model of investment behavior in the stock market. *Physica A*, v. 325, n. 3-4, p. 507–516, 2003.

WEINERT, W. R. *Programação de expressão genética aplicada a problemas de classificação e regressão simbólica*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática Industrial e Engenharia Elétrica - UTFPR-PR, Curitiba-PR-Brasil, 2004.

WEINERT, W. R.; BENITEZ, C.; LOPES, H.; LIMA, C. Simulation of the dynamic behavior of one-dimensional cellular automata using reconfigurable computing. *Lecture Notes in Computer Science*, v. 4419, p. 385–390, 2007.

WEINERT, W. R.; LOPES, H. Evaluating of dynamic behaviour forecasting parameters in the process of transition rule induction of unidimensional cellular automata. *BioSystems*, v. 99, n. 1, p. 6–16, 2009.

- WEINERT, W. R.; LOPES, H. S. GEPCLASS: a classification rule discovery tool using gene expression programming. *Lecture Notes in Computer Science*, v. 4093, p. 871–880, 2006.
- WEINERT, W. R.; LOPES, H. S. Análise de estratégias para melhoria do desempenho do algoritmo de programação por expressão genética na tarefa de regressão simbólica. In: *Anais do I Simpósio Brasileiro de Inteligência Computacional*. Florianópolis, SC: CD-ROM, 2007.
- WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation*. San Francisco: Morgan Kaufmann Publishers, 2000.
- WOLFRAM, S. Cellular automata. *Los Alamos Science*, v. 9, p. 2–21, 1983.
- WOLFRAM, S. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, v. 55, n. 3, p. 601–644, 1983b.
- WOLFRAM, S. Universality and complexity in cellular automata. *Physica D*, v. 10, n. 1-2, p. 1–35, 1984.
- WOLFRAM, S. Computation theory of cellular automata. *Communication in Mathematical Physics*, v. 96, p. 15–57, 1984b.
- WOLFRAM, S. Advances in cryptology. *Lecture Notes in Computer Science*, v. 218, p. 429–432, 1986.
- WOLFRAM, S. Complex systems theory: emergent syntheses in science. In: *Fouding Workshops of the Santa Fe Institute*. Santa Fe: Santa Fe Institute Tech, 1988.
- WOLFRAM, S. *Cellular Automata and Complexity*. Urbana-Champaign, USA: Westview Press, 1994.
- WOLFRAM, S. *A New Kind of Science*. Champaign: Wolfram Media, 2002.
- WUENSCHÉ, A. *Complexity in One-D Cellular Automata: Gliders, Basins of Attraction and the Z Parameter*. Santa Fe, New Mexico, USA, 1994.
- WUENSCHÉ, A. Classifying cellular automata automatically: finding gliders, filtering, and relating space-time patterns, attractor basins and the Z parameter. *Complexity*, v. 4, n. 3, p. 47–66, 1999.
- WUENSCHÉ, A.; LESSER, M. *The Global Dynamics of Cellular Automata*. Reading, USA: Addison Wesley, 1992.
- XIAO, X.; SHAO, S.; DING, Y.; HUANG, A.; CHEN, X.; CHOU, K.-C. Using cellular automata to generate image representation for biological sequences. *Amino Acids*, v. 28, n. 1, p. 29–35, 2005.
- XIAO, X.; SHAO, S.; DING, Y.; HUANG, A.; CHOU, K. C. Using cellular automata images and pseudo amino acid composition to prediction protein subcellular location. *Amino Acids*, v. 30, n. 1, p. 49–54, 2006.
- ZHANG, Y.; LUO, W.; ZHANG, Z.; LI, B.; WANG, X. A hardware/software partitioning algorithm based on artificial immune principles. *Applied Soft Computing*, v. 8, n. 1, p. 383–391, 2008.

ANEXO I

Simulation of the Dynamic Behavior of One-Dimensional Cellular Automata Using Reconfigurable Computing

Wagner R. Weinert, César Benitez, Heitor S. Lopes*, and Carlos R. Erig Lima

Bioinformatics Laboratory, Federal University of Technology Paraná (UTFPR),
Av. 7 de setembro, 3165 80230-901, Curitiba (PR), Brazil
hslopes@pesquisador.cnpq.br, erig@utfpr.edu.br

Abstract. This paper presents the implementation of an environment for the evolution of one-dimensional cellular automata using a reconfigurable logic device. This configware is aimed at evaluating the dynamic behavior of automata rules, generated by a computer system. The performance of the configware system was compared with an equivalent software-based approach running in a desktop computer. Results strongly suggest that such implementation is useful for research purposes and that the reconfigurable logic approach is fast and efficient.

1 Introduction

Cellular automata (CA) are discrete distributed systems formed by simple and identical elements. The dynamic behavior of a CA is represented by its evolution along time and this evolution depends on a transition rule [8]. Finding a transition rule capable of modelling a given behavior is a rather difficult task, since the search space promptly becomes computationally intractable. Many works in the literature proposed the use of evolutionary computation techniques for the task of finding suitable transition rule that leads a CA to display a desired behavior (see, for instance, [3,5,6]). Usually, the basic approach in these cited works is to induce rules. Rules are evaluated according to a fitness function, regarding its utility to simulate the desired behavior. Iteratively, the best performed rules are selected and modified by using genetic operators. The evolutionary process finishes after n generations when a given rule achieves the a satisfactory behavior. Frequently, the computation of the fitness function is the most time-consuming task in an evolutionary computation algorithm. Recently, we have witnessed a pronounced growth of the hardware and software technologies for embedded systems, with many technological options arising every year. In particular, applications based on CA can found in recent works [1,2,7]. In our approach to use evolutionary computation for inducing transition rules for cellular automata, the fitness function is computed by simulating the dynamic behavior of the automaton. This is accomplished evolving many automata using a single

* This work was partially supported by the Brazilian National Research Council – CNPq, under research grant no. 305720/2004-0 to H.S. Lopes.

transition rule. Such automata are randomly generated and evolved during a fixed number of time steps. The computation of the fitness function is based on the comparison of the obtained behavior and the desired one. The discovery of transition rules for CAs requests the computation of a fitness function that is computationally-intensive. To reduce this computational cost, we propose a methodology for evolving CAs using reconfigurable computing. The system was developed in VHDL (VHSIC Hardware Description Language) and implemented in a FPGA (Field Programmable Gate Array) device.

The case-study addressed in this work is the classic problem of a CA to perform a density classification task [1,3,5,6]. A typical evolutionary computation algorithm for finding a transition rule for such CA will need to try some 1500 different rules until finding a good one. For each rule, its appropriateness is evaluated by applying it to a randomly generated automaton and evolving it for a given number of time steps. Due to the stochastic nature of the algorithm and the randomness of the initial condition of the automaton, around 10,000 different automata are evolved for each rule, and results are statistically evaluated.

In this work it is not aimed to propose any evolutionary computation technique for inducing CAs, but evaluating the usefulness of reconfigurable computing as a hardware accelerator for evolving CAs, in comparison with a software-based approach.

2 Cellular Automata and the Density Classification Task

A Cellular Automata is defined by its cellular space and by its transition rule. The cellular space is represented by a lattice of N cells connected according a boundary condition in a d -dimensional space. The transition rule gives the next state for the cell, considering the configuration of its current neighborhood. At each time step, all cells in the lattice update their current state, according to the transition rule (representing the dynamic nature of the system)[8]. A formal definition of CA is given by [4]:

$$\left\{ \begin{array}{l} \Sigma : \quad \text{set of possible states for each cell;} \\ k : \quad \text{cardinality of } \Sigma; \\ i : \quad \text{index of each cell;} \\ S_i^t : \quad \text{state of a given cell } i \text{ at time } t (S_i^t \in \Sigma); \\ \eta_i^t : \quad \text{neighborhood of cell } i; \\ \Phi(\eta_i) : \text{ transition rule that leads cell } i \text{ to the next state } (S_i^{t+1}) \text{ as function of } \eta_i^t. \end{array} \right. \quad (1)$$

Two parameters are necessary to deal with the neighborhood concept: (m) that represents the size of the neighborhood, and (r) representing its radius. Usually, parameter m is given as function of r , in the form: $m = 2r + 1$. Since a CA is represented by a linear structure, the leftmost cell (i_0) and the rightmost cell (i_{n-1}) of a one-dimensional automaton do not have left and right neighbors, respectively. Therefore, a bounding condition is necessary, such that the leftmost cell is connected with the rightmost cell, and then the transition rule $\Phi(\eta_i)$ can be applied to the whole lattice. Considering that the number of cells of a

given neighborhood is $2r + 1$, the number of different neighborhood that can be generated using a given rule is $k^{(2r+1)}$. Also, the number of possible transition rules that can be generated for a CA is $k^{k^{(2r+1)}}$.

The dynamic behavior of an one-dimensional CA generated by the application of a transition rule over the automaton for n time steps is usually illustrated by a spatiotemporal diagram. In such diagram, the configuration of states in a lattice is plot as a function of time.

The number of possible rules for a given automaton is $k^{k^{(2r+1)}}$. The larger k and r , the larger the set of rules applicable to a given CA. In most cases it is not computationally feasible to evaluate the whole set of rules so as to find a given one that explains the dynamic behavior of a system.

The density classification task, also known as majority problem, is a classical problem in CA theory [8]. The objective is to find a transition rule such that, when applied for M time steps to a random initial configuration, will lead all cells either to state 0 or state 1, depending on the density of the initial configuration. The parameter ρ is defined as a threshold for the density classification task [4], in such a way that ρ_0 represents the density of cells in state 1 in the initial configuration, and ρ_c is the same density in a given configuration (for $\forall t > 0$).

The density classification task can be modelled in several ways. Here we use the approach proposed by [3]. In this model, all cells can have binary states ($k = 2$), the lattice is composed by an odd number of cells ($N = 149$), and the neighborhood radius of $r = 3$. The number of possible transition rules for this CA is 2^{128} .

3 Methodology

To implement the system, we used the Altera Quartus II development system, version 5.1, and a Cyclone EP1C6Q240C8 FPGA device. The software running in the desktop PC was developed in C++ language.

The software running in the desktop PC generates a transition rule (as part of other evolutionary strategy for inducing rules), encodes and sends it to the FPGA through a parallel interface. Inside the FPGA, several hardware blocks (see Figure 1 perform the evaluation of the transition rule, and send back both, the result of such evaluation and the processing time.

Using a serial to parallel converter, a 128-bits long transition rule is assembled (from the parallel interface block) and stored in the register block.

The command decoder block decodes the received commands and executes control actions in the other FPGA blocks: the pseudo-random number generator block, the CA evolver block, the chronometer block and to the accuracy calculator block. In the current version, five different commands can be decoded and executed:

- Internal reset.
- Restart MLS pseudo-random number generator.
- Clear current transition rule in register.

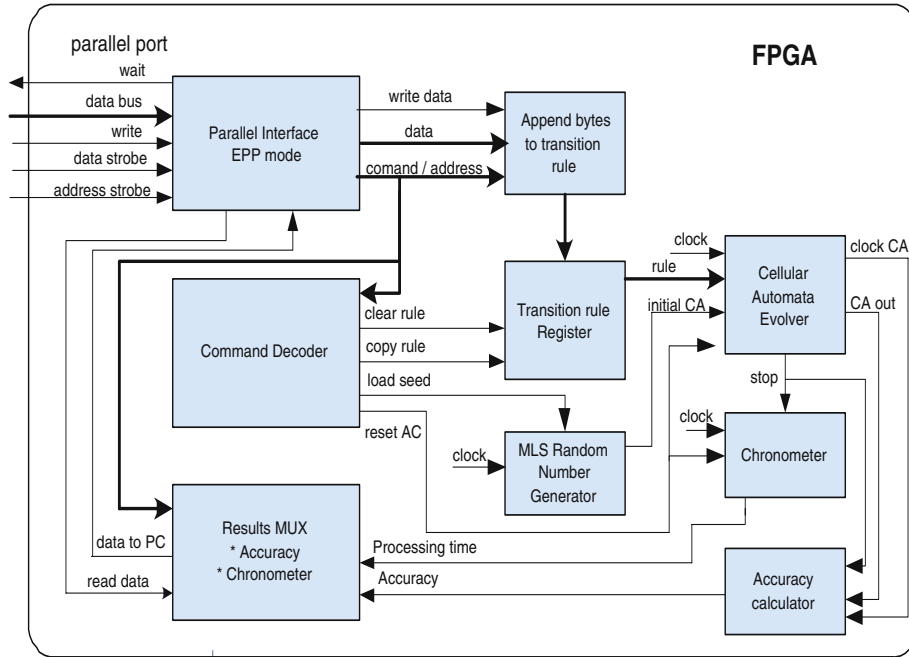


Fig. 1. Block diagram of the proposed system for evolving CAs

- Latch transition rule.
- Start CA evolution.

The CA evolver block receives a previously registered transition rule and an initial configuration for the automaton, generated by the pseudo-random number generator, and evolves the CA for a given number of iterations (in our case, 200). Inside this block there are two Finite State Machines (FSM). The first one controls the number of iterations of the CA. The second FSM controls the execution of 10,000 runs, with a random initial configuration.

The accuracy calculator block verifies if the final configuration is the expected one, considering the density of the initial configuration. If positive, the score of the rule under evaluation is incremented. At the end of all runs, this block retains the total number of hits, which, divided by the number of runs and taken as a percentage, is the accuracy rate.

To generate random initial configurations, a Maximum Length Sequence (MLS) pseudo-random number generator is used ¹. MLS is an n -stage linear shift-register that can generate binary periodical sequences of maximal period length of $L = 2^n - 1$. These sequences are referred to as maximal-length sequences (MLS), and n is said to be the degree of the sequences. In our work, we used $n = 16$, thus generating 16-bits sequences. The random seed is loaded into the

¹ Available in <http://www.ph.ed.ac.uk/~jonathan/thesis/node83.html>

shift-register by a command. Ten parallel shift-registers were implemented, and out from these 160 bits, 149 are used for the initial configuration of the CA.

The chronometer block counts and register the total elapsed time for generate, evolve and evaluate the behavior of 10,000 CAs. The multiplex block selects output data between the computed accuracy rate and the processing time, to be sent out to the desktop computer.

4 Results

The evaluation of the proposed system consists of five experiments. For each experiment, 10,000 one-dimensional CAs were randomly generated and evolved for 200 iterations. For these experiments, we used a single rule, proposed by Juillé et al. [3], named “Coevolution(1)”. To date, this rule is supposed to be the best known rule for the density classification task. Results take into account the average accuracy (that evaluates how good the rule performs) and the average processing time (that represents the computational cost). In order to compare such results with other implementation, a similar software was developed in C++ programming language, compiled without any optimization flag and run in a desktop computer under Microsoft Windows XP operational system. Exactly the same rule and parameters were used for both systems, and Table 1 shows the results obtained. The FPGA was run at 33,33 Mhz and the real clock of the desktop PC was 1800 MHz. The comparison between different hardware platforms such as a desktop PC and FPGA-based system is not completely fair, but necessary to emphasize the dramatic performance difference.

Table 1. Comparison PC versus hardware-based approach

| Device | Accuracy | Processing Time |
|-------------------------------|----------|-----------------|
| FPGA CYCLONE EP1C6Q24068 | 82.28 | 58.660ms |
| PC AMD Athlon XP 2400+, 512MB | 79.33 | 3h:9m:26.6s |

5 Conclusions and Future Work

In this work we described a reconfigurable computing system for evolving cellular automata. This system is to be used in conjunction with a software application, running in a desktop PC, to study the behavior of transition rules. The reconfigurable hardware is able to communicate with any software application, since a standard parallel interface was implemented.

For this work, the average accuracy of the system has small importance, since the main focus is the processing speed. The difference of 2.95% between the accuracy rates of the hardware-based system and the software-based system is due to different random number generators, responsible for the generation of the initial configuration of the CAs.

The processing time needed to evolve the CAs in the PC is extremely high, when compared with the time needed by the hardware approach. This is due to the fact that, in the PC, processing is sequential, and in the FPGA parallelization was largely explored. Even considering that the FPGA was running at 33,33 MHz and the PC was 54 times faster, the processing time of the later was more than 5 orders of magnitude higher than the former. This makes evident the great advantages of reconfigurable computing, strongly suggesting its adequacy for this kind of task.

Future work will focus on the implementation of some evolutionary computation technique to search transition rules for one-dimensional CA, using the reconfigurable computing module as an external accelerator. Also, other levels of parallelization will be sought, thus improving even more the efficiency of the system.

References

1. Corsonello, P., Spezzano, G., Staino, G., et al.: Efficient implementation of cellular algorithms on reconfigurable hardware. In: *Proc. of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing* (2002) 211–218.
2. Halbach, M., Hoffmann, R.: Implementing cellular automata in FPGA logic. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (2004) 258–262.
3. Juillé, H., Pollack, J.B.: Coevolving the ideal trainer: application to the discovery of cellular automata rules. In: Koza, J.R et al. (eds.), *Genetic Programming 1996: Proc. 3rd Annual Conference*, (1998) 519–527.
4. Mitchell, M.: Computation in cellular automata: a selected review. In: Gramms, T., ed., *Nonstandard Computation*. VCH Verlagsgesellschaft, Weinheim (1996).
5. Morales, F.J., Crutchfield, J.P., Mitchell, M.: Evolving two-dimensional cellular automata to perform density classification: a report on work in progress. *Parallel Computing* **27** (2001) 571–585.
6. Oliveira, G.M.B., Bortot, J.C., Oliveira, P.P.B.: Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task. In: *Proc. 8th Int. Conf. on Artificial Life*, (2002) 202–206.
7. Shackelford, B., Tanaka, M., Carter, R.J., et al.: FPGA implementation of neighborhood-of-four cellular automata random number generators. In: *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, (2002) 106–112.
8. Wolfram, S.: *Cellular Automata and Complexity*. Westview Press, Boulder (1994).

ANEXO II



Evaluation of dynamic behavior forecasting parameters in the process of transition rule induction of unidimensional cellular automata

Wagner Rodrigo Weinert^a, Heitor Silvério Lopes^{b,*}

^a Federal Institute of Education Science and Technology of Paraná (IFPR), R. Antônio Carlos Rodrigues 453, 83215-750 Paranaguá (PR), Brazil

^b Bioinformatics Laboratory/CPGEI, Federal University of Technology – Paraná (UTFPR), Av. 7 de setembro 3165, 80230-901 Curitiba (PR), Brazil

ARTICLE INFO

Article history:

Received 17 January 2009

Received in revised form 3 August 2009

Accepted 6 August 2009

Keywords:

Cellular automata

Dynamic behavior forecasting parameters

Dynamic systems

Evolutionary computation

ABSTRACT

The simulation of the dynamics of a cellular systems based on cellular automata (CA) can be computationally expensive. This is particularly true when such simulation is part of a procedure of rule induction to find suitable transition rules for the CA. Several efforts have been described in the literature to make this problem more treatable. This work presents a study about the efficiency of dynamic behavior forecasting parameters (DBFPs) used for the induction of transition rules of CA for a specific problem: the classification by the majority rule. A total of 8 DBFPs were analyzed for the 31 best-performing rules found in the literature. Some of these DBFPs were highly correlated each other, meaning they yield the same information. Also, most rules presented values of the DBFPs very close each other. An evolutionary algorithm, based on gene expression programming, was developed for finding transition rules according a given preestablished behavior. The simulation of the dynamic behavior of the CA is not used to evaluate candidate transition rules. Instead, the average values for the DBFPs were used as reference. Experiments were done using the DBFPs separately and together. In both cases, the best induced transition rules were not acceptable solutions for the desired behavior of the CA. We conclude that, although the DBFPs represent interesting aspects of the dynamic behavior of CAs, the transition rule induction process still requires the simulation of the dynamics and cannot rely only on the DBFPs.

© 2009 Elsevier Ireland Ltd. All rights reserved.

1. Introduction

Cellular automata (CAs) are discrete dynamic systems formed by simple and identical components. The dynamics of a CA is determined by its state transition rule. The application of such rule in the current configuration of the CA leads to another configuration and, thus successively changing along time. The overall dynamic behavior presents certain characteristics that can classify it. In the past, Wolfram (1984) proposed a standard classification for the dynamic behavior of CAs, qualitatively divided into four classes. Later, Li and Parckard (1990) improved Wolfram's classification to six classes.

The computational simulation of a CA system is relatively simple, provided the transition rule is known. However, finding a transition rule for a given dynamic behavior is a very difficult task, for which there is no efficient method to date. This work is focused in this gap, presenting and evaluating an approach for the induction of transition rules for CAs.

The classical problem known as classification by the majority rule (Juillé et al., 1998) (see Section 2.3) is frequently studied in the

CAs theory. Basically, this problem consists in finding a transition rule that, when applied to all the initial cells of a d -dimensional lattice, will lead them either to state 0 or state 1, after n time-steps. This feature assures that all possible solutions (rules) to this problem will exhibit a behavior defined as “null”, according to Li and Parckard (1990) classification. The expected final state of the lattice depends on the density of the CA in its original configuration.

It is important to point out that every transition rule that is said a solution for the problem presents a “null” behavior. However, the opposite does not hold, since a many rules that present a “null” behavior are not solution for the classification by the majority problem.

Several authors have proposed the use of computational intelligence techniques (more specifically, evolutionary computation), for determining appropriate transition rules for the classification by the majority rule problem (Richards et al., 1990; David et al., 1996; Juillé et al., 1998; Morales et al., 2001; Ferreira, 2002; Oliveira et al., 2002b). The general approach used by those works is the evolution of a population of individuals (in this case, rules) for y generations. At each generation a set of genetic operators actuate in the population generating hopefully better individuals. The quality of the individuals is systematically evaluated by a fitness function. The evolution process is ended when a satisfactory solution is found or a maximum number of generations is reached. In general, the

* Corresponding author.

E-mail addresses: wweinert@gmail.com (W.R. Weinert), hslopes@utfpr.edu.br, hslopes@pesquisador.cnpq.br (H.S. Lopes).

problem of rule induction for CAs uses as fitness function a measure of the performance obtained by the analysis of the dynamics of the system. Every possible candidate solution (transition rule) is submitted to a simulation procedure. The rule is applied to a random initial configuration of the CA and then iterated n times, thus achieving its final configuration. Both initial and final states of the CA are compared. If both are the same, thus the rule is a solution for the problem. Otherwise, a similarity measure quantifies the quality of the rule. Using this approach, obtaining the fitness of a rule has a high computational cost, since a very large number of initial random configurations have to be simulated. For some applications, this fact makes unfeasible the evolutionary approach previously described, since it is necessary to evaluate a large number of candidate solutions. More recently, a dedicated reconfigurable hardware was proposed to compute the fitness of transition rules for the simulation of CAs (Weinert et al., 2007). Although this seems to be a viable alternative, regarding computational cost, this methodology is restricted to a single problem (classification by the majority rule), and it is quite difficult to adapt to other rule-induction problems in CAs.

Several measures that can be used to forecast the behavior of a CA can be found in the recent literature. These include for instance: sensitivity (Binder, 1993), neighborhood domain, activity propagation and absolute activity (Oliveira et al., 2001), Wuensche's Z parameter (Wuensche, 1999), activity (Langton, 1990), mean field parameters (Li, 1991) and Li's Z parameter (Li, 1991). These parameters, known as dynamic behavior forecasting parameters (DBFPs), are computed for a given transition rule without the necessity of simulating the dynamics of the CA for many iterations. The values obtained by the DBFPs are then used to classify the dynamic behavior of the CA. However, Oliveira et al. (2002a) demonstrated that parameters sensitivity, absolute activity, neighborhood domain and activity propagation can be useful for a genetic algorithm to find rules for the majority classification rule problem. The evaluation of rules is obtained as a function of two parameters with different weights. The first parameter considers the DBFPs values and causes the search to be filtered towards rules with "null" behavior. The second parameter is computed by simulating the dynamics of the CA, and verifies if the rule satisfactory solves the problem.

The objective of this work is to develop an evolutionary computation method for the induction of transition rules of CAs capable of presenting a "null" behavior, for the classification by the majority rule problem. The central idea is to find a set of DBFPs and combine them in such a way to be useful for the evolutionary process of rule induction.

This work is organized as follows. The next subsection presents some selected references about applications of CAs, with special emphasis on the simulation of biological systems. Section 2 presents a short introduction about the concepts pertaining CAs, including the features of the dynamic behavior, the DBFPs and the classification by the majority problem. A correlation analysis of the DBFPs is presented in Section 3, aiming at gathering useful information for the rule induction procedure using a gene expression programming algorithm described in Section 4. Results are shown and commented in Section 5. Finally, Section 6 concludes the work and points out future directions of research.

1.1. Related Work

Many different problems can be modeled by CAs and, thus, can be applied to many areas such as cryptography (Wolfram, 1986; Tomassini and Perrenoud, 2001; Benkiniouar and Benmohamed, 2004), text compression (Khan et al., 1999), scheduling (Swiecicka and Seredynski, 2000), task synchronization (Das et al., 1995), image processing (Rosin, 2005), data classification (David et al.,

1996), stock market behavior (Wei et al., 2003) and earthquake activity (Georgoudas et al., 2007).

There has been a growing interest in using CAs for modeling and simulation of biological systems. For instance, Kansal et al. (2000) presented a CA-based model capable of simulating the growth of brain tumors. Authors used this model to simulate the Gompertzian tumor growth. The predicted composition and growth rates were in agreement with a test case from the medical literature. The dynamical simulation allowed the identification of clonal competition in the system, an important property of tumor growth.

Genetic algorithms and CAs were used by Mizas et al. (2008) for reconstructing the evolution of DNA sequences. This work is particularly interesting due to its complexity and its contribution to the biological sciences. They modeled a DNA chain using a unidimensional CA, which alphabet represents the four bases of DNA (A, C, G and T). The genetic algorithm was used to find a transition rule for the CA, capable of leading the chain to the desired dynamic behavior. By starting with two DNA sequences previously identified in different evolutionary moments, the rule found by the algorithm was capable of explaining the evolutionary changes of the sequences. This system can be particularly interesting for the identification of mutations that take place in the DNA throughout evolutionary periods.

Kiera et al. (1996) developed a CA model of an enzyme reaction with water substrate. The ease of simulation and the illustrative value of the model showed that CA models can be useful in the study of dynamic phenomena such as enzyme kinetics. This study opened up opportunities for the study of a general enzyme model and the effects on its behavior, including the interaction of ingredients with the solvent.

Spatial characteristics such as localized populations of dead cells might adversely affect the spread of an infection. Beauchemina et al. (2005) used a simple bidimensional CA model of a viral infection to investigate, through dynamic simulation, the influence of spatial heterogeneities on viral spread. The CA model was validated against clinical immunological data for uncomplicated influenza A infections.

Malleta and Pillisb (2006) presented a hybrid cellular automata-partial differential equation model to describe, through dynamic simulation, the interactions between a growing tumor next to a nutrient source and the immune system of the host organism.

Gangadhar (2005) presented an innovative method for protein sequence alignment using a bidimensional CA. Starting from an initial unaligned state, the CA evolves for a number of iterations according to a dynamic behavior defined by a set of transition rules. The emerged behavior led to the alignment of the sequences.

A novel way to visualize biological sequences using CAs was devised by Xiao et al. (2005). In this work, a sequence of amino acids is transformed into a digital encoding and a transition rule systematically applied over it creates a spatio-temporal diagram. Using such dynamic approach, it is possible to observe characteristics that were originally hidden in the amino acids chain.

Laurio et al. (2007) showed how to create CAs for recognizing regular patterns in protein sequences in such a way to identify families and functional sites. Starting from a regular pattern from Prosite (Hofman et al., 1999), a rule is determined using regular grammars which, in turn, is converted to a transition rule characterizing the dynamic behavior of a CA.

There are other works in the recent literature that employed CAs to model the dynamics of biological systems, for instance: epidemic propagation (Fu and Milne, 2003), simulation of cell infection caused by the HIV virus (Corne and Frisco, 2008) and prey-predator modeling (Chen and Mynett, 2003).

In the previous mentioned works, the models generated by CAs allowed a better comprehension of the dynamical behavior of specific biological systems. Usually, the computational cost of the

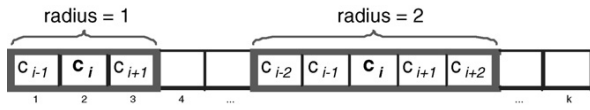


Fig. 1. Definition of an unidimensional CA.

simulation of the dynamic behavior of CA is high (see Section 2.1). Therefore, the study of methods that give some insight about the behavior of CAs, the focus of this work, may have large practical applicability in complex biological systems.

2. Cellular Automata

A CA can be represented by a d -dimensional matrix structure. All the cells of this structure are identical and they keep a neighborhood relationship determined by a predefined radius (r). The size (m) of this neighborhood is defined as a function of r , according to Eq. (1):

$$m = 2r + 1 \quad (1)$$

Boundary conditions are set so as to allow cells situated at the extremities of the matrix to be connected each other. All cells of the d -dimensional matrix are updated at the same time, in parallel, by a state transition rule. The transition rule is applied to each cell and determines the next state of the cell, considering its current state and the state of the neighboring cells. Successive iterations of the transition rule with the CA will lead to an observable dynamic behavior. Mitchell et al. (1996) defined the following formal notation for CAs:

- Σ : set of possible states of a cell;
- k : number of elements of the set Σ ;
- i : index of a specific cell;
- S_i^t : state of a cell in a given time t , such that $S_i^t \in \Sigma$;
- η_i^t : neighborhood of cell i , that is, state S_i^t of cell i together with the states of cells to which cell i is connected;
- $\Phi(\eta_i)$: transition rule de that gives the next state S_i^{t+1} for each cell i , as function of η_i^t .

An unidimensional CA is defined as a matrix data structure C of size $k(C_{1 \times k})$. For each cell c_i , its neighborhood is defined as the cells standing at its left $c_i(c_{i-1}, c_{i-2}, \dots, c_{i-k})$ and those at its right $c_i(c_{i+1}, c_{i+2}, \dots, c_{i+k})$. For the $r = 1$, the neighborhood of cell c_i is formed by its left and right neighbors, that is, (c_{i-1}) and (c_{i+1}) , respectively (see Fig. 1). For $r = 2$, the neighborhood comprehends $(c_{i-2}$ and $c_{i-1})$ to the left, and $(c_{i+1}$ and $c_{i+2})$ de c_i to the right. Similarly, higher-order neighborhoods is straightforwardly defined.

The state of a CA is the configuration of the lattice in a given instant of time. The initial state in t_0 will take to successive states in subsequent time steps t_1, t_2, \dots, t_n , by successive iteration of the transition rule of the automaton. A state of the CA has a single successor state, as a consequence of the application of the transition rule. However, it can have an arbitrary number of predecessor states, known as pre-images.

Fig. 2 exemplifies the evolution of a unidimensional CA formed by 10 cells ($C_{1 \times 10}$), during four time steps. Initially, at time zero, cells are randomly initialized by elements belonging to the binary set ($\Sigma = \{0, 1\}$): $c_1 = 1; c_2 = 0; c_3 = 0; c_4 = 1; c_5 = 0; c_6 = 0; c_7 = 1; c_8 = 1; c_9 = 1; c_{10} = 0$. In this example we consider $r = 1$, consequently, $m = 3$, according to Eq. (1). Since the graphical representation of this CA is linear, its first (c_1) and last (c_{10}) cells do not have neighbors to the left and to the right, respectively. Therefore, according to the boundary condition, cell c_1 becomes the neighbor of cell c_{10} . This procedure assures that the transition

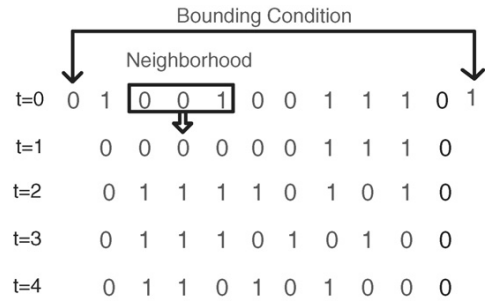


Fig. 2. Evolution of a unidimensional CA.

rule $\Phi(\eta_i)$ can be applied over all cells of the CA, without continuity failure.

An important issue of CAs is the number of transitions that composes the rule Φ is obtained by k^{2r+1} , and the number of rules represented by those transitions is $k^{k^{2r+1}}$. The CA presented in Fig. 2 can be evolved for 256 ($k^{k^{2r+1}} = 2^{2^{2 \times 1 + 1}} = 256$) different rules. For the take of example, it is shown rule 169. This number represents a rule according to the concept of elementary automata proposed by Wolfram (1983). Therefore, the binary number 10101001, extracted from the concatenation (from right to left, that is, from transition 111 \rightarrow 1 to transition 000 \rightarrow 1) of the eight outputs specified by rule Φ ($\{000 \rightarrow 1; 001 \rightarrow 0; 010 \rightarrow 0; 011 \rightarrow 1; 100 \rightarrow 0; 101 \rightarrow 1; 110 \rightarrow 0; 111 \rightarrow 1\}$) is equivalent to the number 169 in decimal.

2.1. Dynamic Behavior of a CA

The dynamic behavior of a CA can be illustrated by a spatio-temporal diagram, in which the configuration of the states in a d -dimensional lattice is plotted as function of time. Usually, cells in state 0 are represented in white, and those in state 1 are represented in black, as shown in Fig. 3.

According to Wuensche and Lesser (1992), this diagram represents the local behavior of the system, since its global behavior is independent on the initial state. The global behavior can be obtained by the analysis of a (large) set of overlapped spatio-temporal diagrams, each one constructed starting from different initial state.

Wolfram (1984) proposed a standard terminology for classifying the dynamic behavior of CAs. By studying spatio-temporal diagrams, he has demonstrated that CAs can exhibit extremely complex behaviors, and he grouped them into four qualitative classes:

- Class 1: The evolution of the CA takes to homogeneous states in which, for instance, all cells have state 1 (Fig. 3a).
- Class 2: The evolution of the CA takes to a set of simple, periodic and stable structures (Fig. 3b).
- Class 3: The evolution of the CA takes to chaotic patterns (Fig. 3c).
- Class 4: The evolution of the CA takes to complex structures and, sometimes, with long periods (Fig. 3d).

Dynamic behaviors of class 1 and class 2 can be visually identifiable. However, the distinction between behaviors of class 3 and 4 is quite difficult, since there is no clear boundary between these classes.

Li and Parckard (1990) extended the classification previously proposed by Wolfram into six classes, using a more specific definition, as follows:

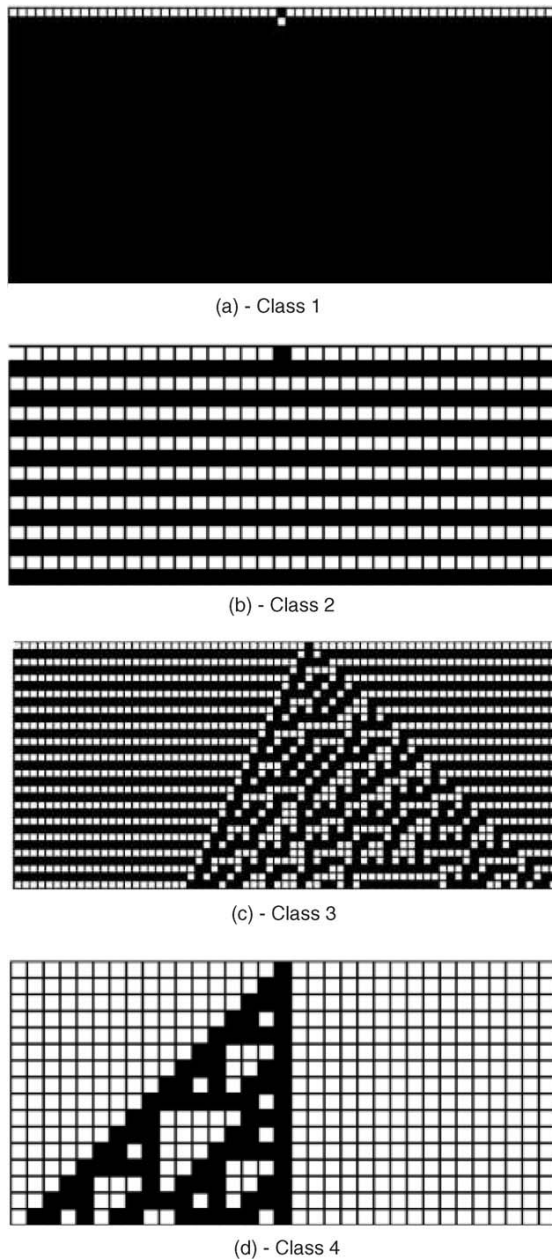


Fig. 3. Classification of the dynamic behaviors according to Wolfram (2002) (pp. 55–56). Time $t = 0$ is represented in the top of each figure and grows downwards.

- Null rules: After t iterations, all cells of the CA converge either to state 0 or state 1.
- Fixed-point rules: After t iterations, the application of a given rule maintains constant the configuration of the CA.
- Double-cycle rules: After t iterations, every two applications of the rule the CA returns to the same configuration.
- Periodic rules: After t iterations, each n repetitive applications of the rule, the CA returns to a given configuration, with $n \geq 3$.
- Chaotic rules: can produce non-periodic dynamic behaviors.
- Complex rules: can produce periodic dynamic behaviors, although they can have extremely long periods.

2.2. Parameters for Forecasting the Dynamic Behavior of CAs

The dynamic behavior forecasting parameters (DBFPs) are measures that give a quantitative evaluation of the effect of a transition rule in a CA, throughout iterations. The basic idea behind such parameters is to infer particular aspects of the dynamic behavior of a CA without the need to simulate them.

Next, a brief description of the main DBFPs found in recent literature is given, including: sensitivity (Binder, 1993), neighborhood domain, activity propagation and absolute activity (Oliveira et al., 2001), Wuensche's Z parameter (Wuensche, 1999), activity (Langton, 1990), mean field parameters (Li, 1991) and Li's Z parameter (Li, 1991). The mathematical formalization of such parameters can be found in the refereed citations.

Sensitivity measures the percent of similar transitions that are mapped to different states (Binder, 1993). A transition can be composed by n cells sequentially associated with a single output (s) in the form: $n_1, \dots, n_n \rightarrow s$. Similar transitions to $n_1, \dots, n_n \rightarrow s$ are all transitions that present a single neighbor cell different from the original transition. The analysis of similar transitions consists in counting the transitions that change s for a change in a single cell n , relative to the original transition.

The neighborhood domain (Oliveira et al., 2001) is a parameter used for quantifying the changes, due to the transition rule, in the central cell, taking into account the predominant state of the overall neighborhood.

The activity parameter was defined by Langton (1990), and refers to a statistical measure about the transitions of a rule. For binary CAs, the computation of this parameter consists in the sum of the output values of each transition, divided by the number of transitions that composes the rule.

The absolute activity (Oliveira et al., 2001) is based on the parameter activity, previously mentioned. It counts the number of transitions that takes to a state different from the current state of the central cell or of its neighbor cells.

The objective of the parameter Wuensche's Z parameter (Wuensche, 1999) is to quantify the propagation of perturbations in a given rule (Wuensche and Lesser, 1992; Wuensche, 1994). Such perturbation is computed as function of the pre-images in the state space. This parameter indicates whether the amount of pre-images is high (low Z) or low (high Z), for an arbitrary lattice configuration.

The concept of activity propagation is based on the two previous measures, and it is defined as the possibility of transition of a given cell, following the dominant state of the neighboring cells, and the possibility of this transition be sensitive to a minimal state change of its neighborhood.

Mean field parameters (Li, 1991) are formed by p components, such that $p = n + 1$, and n is the number of cells of the neighborhood. Each component is labeled as cm_p and stores the number of configurations (neighborhoods) that present p occurrences of state 1 in the n cells, and map this configuration to output with value 1.

The Z parameter defined by Li (1991) uses the same concept proposed by Binder (1993) for the parameter sensitivity. Although these parameters have been emerged in different works, both quantify the same information (Table 2 demonstrates this fact).

2.3. The Problem of Classification by the Majority Rule

The classical problem in CA literature known as classification by the majority rule can be modeled in different ways. In this work, we adopt a particular approach presented by Juillé et al. (1998), in which cells have binary states ($k = 2$), the unidimensional matrix structure of the CA is composed by a constant and predefined number of cells ($N = 149$), and the neighborhood radius is 3 ($r = 3$). Consequently, for this approach, the rule space of the CA is 2^{128} .

Table 1
Most efficient transition rules found in the literature, for the classification by the majority rule problem.

| Rule | Hexadecimal | Reference | Efficiency |
|------|-----------------------------------|------------------------------|------------|
| GKL | 005F005F005F005FFF5F005FFF5F | Gacs et al. (1978) | 81.60% |
| MHC | 0504058705000F77037755837BFFB77F | Mitchell et al. (1993) | 76.90% |
| DAV | 002F035F001FCF1F002FFC5F001FFF1F | Davis (1991) | 81.80% |
| DAS | 070007FF0F000FF0F0007FF0F310FFF | Das et al. (1995) | 82.20% |
| ABK | 050055050055005555FF55FF55FF55FF | David et al. (1996) | 82.30% |
| CRA | 00550055005500571F55FF57FF55FF57 | Cranny and Bossomaier (1999) | 82.50% |
| JP1 | 011430D7110F395705B4FF17F13DF957 | Juillé et al. (1998) | 85.10% |
| JP2 | 1451305C0050CE5F1711F55F0F53CF5F | | 86.00% |
| BOO1 | 145500CC0F14021F1715FFC0F17FF1F | Bortot et al. (2004) | 86.16% |
| BOO2 | 070017070C0057DF07BFD707CCFF559F | | 85.97% |
| BOO3 | 070017070C0057D707BFD707CDDFF55D7 | | 85.39% |
| BOO4 | 002F131F010FF91F00ECFF1F013DF91F | | 85.38% |
| BOO5 | 1071307C0000286F17313F7FF33F2B7F | | 85.35% |
| R1 | 015E0012005500571F5FFFF0F55CF5F | Oliveira et al. (2007) | 82.70% |
| R2 | 10111000531512531F15FF5FDF5DDF5F | | 82.60% |
| R3 | 00010355015511571F150F77FF5FF57 | | 81.50% |
| R4 | 070447470700460705774757F777FF77 | | 82.70% |
| R5 | 015400550050045F055FFFDF5557FF5F | | 81.90% |
| R6 | 0445004C37770E3F044500CDF7773FFF | | 81.50% |
| R7 | 15005000350077071553775FF5F77F7F | | 81.70% |
| R8 | 000104171DDF555704DF441FDDDFD557 | | 81.60% |
| R9 | 01000030011311370DFFBFFBDDFF11FF | | 81.50% |
| R10 | 0001090703030B031F1F6F37FF776F77 | | 79.20% |
| R11 | 0100050D1D9D155F05FD555FDDFF557 | | 78.90% |
| R12 | 000103021111011317F5FFFDDFF11FF | | 78.70% |
| R13 | 0001017D2113C35F4B15DF75275B9FD7 | | 78.80% |
| R14 | 015500400054563F1057BF0FB7FFB7F | | 77.80% |
| R15 | 0071023C00224D170379B53747BFFF7F | | 77.30% |
| R16 | 10041383005313DD3357CFED875F1FDF | | 78.70% |
| R17 | 040305502F06457D05013757D5F7FF7F | | 78.20% |
| R18 | 050470000006516D053FF5FF977FE77F | | 77.30% |

The problem states that all cells must reach either state 0 or state 1 after M time steps, depending upon the density of the initial configuration of the CA. According to Mitchell et al. (1994), parameter c denotes a threshold for the classification. That is, if the number of 1's in the initial CA divided by the number of total cells of the same automaton if larger or equal to c , then all cells shall converge to state 1 after M time steps, otherwise, to state 0. In this work, we used $c = 0.5$, in the same way as in Juillé et al. (1998).

3. Correlation Analysis of the Dynamic Behavior Forecasting Parameters

This section aims at identifying possible statistical relationships between the DBFPs previously presented in Section 2.2. Considering that such parameters are computed as function of the transition rules of the CA, this analysis will use rules for the classification by the majority rule problem, according to the approach described in Section 2.3. This particular problem was chosen due to the availability of a large number of publications that present several different computational methods for finding interesting solutions for it. Therefore, it is possible to have a comparison of our proposed approach with another works. Another interesting feature of the cited problem is regarding the transition rules that are currently considered as good solutions for it. These rules have a behavior classified as “null”, according to Li and Parckard (1990).

The rule space for the classification by the majority rule problem treated here is 2^{128} rules. This number of possibilities is too high to allow the systematic computation of the DBFPs for all rules. Therefore, after a deep search in recent literature, 31 rules were found, and all of them were submitted to the forthcoming analysis.

Table 1 presents these rules in hexadecimal format, together with the corresponding reference and their efficiency measure. Efficiency, in this context, was computed by evolving 10^4 random CAs for 200 iterations, for each rule. After, the final configuration is

examined, and two particular situations are of interest:

- If the density of the initial configuration of the CA is lower than 0.5 and, after 200 iterations, all cells converge to 0.
- If the density of the initial configuration of the CA is higher or equal to 0.5 and, after 200 iterations, all cells converge to 1.

For these two situations, the score of the rule is increased, since its application to the CA really leads to the desired result. Any other situation is considered failed and the score of the rule is not increased. The efficiency of the rule is obtained by the division of the score divided by the number of trials (in our case, 10^4), therefore representing the percent of success of the rule.

Table 2 presents the computed values of the DBFPs for the reference rules presented in Table 1. The acronyms in Table 2 are: sensitivity (S), neighborhood dominance (DV), activity propagation (PA), absolute activity (AA), Wuensche's Z ($Z.W$), activity (A), mean field parameters (CM) and Li's Z ($Z.Li$). This table also shows the average values (that is, $Sm, DVm, PAm, AAm, Z.Wm, Am, CMm, Z.Lim$), the standard deviation, the maximum and the minimum of these parameters, computed over the 10^4 CAs. Finally, this table also shows the Euclidean distance (DE) between values of the DBFP set and the corresponding average values. DE is defined by Eq. (2) and is aimed as a measure of similarity between the values of the individual parameters of each rule (P_i) and the average values (Pm_i). In this procedure, 8 dimensions are considered, each one corresponding to a parameter. The smaller the value of DE , the more similar will be the rule parameters regarding the average values:

$$DE = \sqrt{\sum_{i=1}^8 (P_i - Pm_i)^2} \quad (2)$$

Table 2
Computation of the dynamic behavior forecasting parameters—DBFPs.

| Rule | S | DV | PA | AA | Z.W | A | CM | Z.Li | DE |
|--------------------|------|-----------|------------|------------|------------|-------------|-----------|------------|--------------|
| GKL | 0.23 | 0.91 | 0.07 | 0.10 | 0.25 | 0.50 | 0.50 | 0.23 | 0.27 |
| MHC | 0.37 | 0.91 | 0.08 | 0.18 | 0.54 | 0.49 | 0.49 | 0.37 | 0.11 |
| DAV | 0.30 | 0.88 | 0.09 | 0.16 | 0.24 | 0.50 | 0.50 | 0.30 | 0.23 |
| DAS | 0.25 | 0.87 | 0.10 | 0.22 | 0.38 | 0.50 | 0.50 | 0.25 | 0.13 |
| ABK | 0.23 | 0.88 | 0.09 | 0.20 | 0.50 | 0.50 | 0.50 | 0.23 | 0.13 |
| CRA | 0.25 | 0.88 | 0.09 | 0.21 | 0.47 | 0.50 | 0.50 | 0.25 | 0.10 |
| JP1 | 0.40 | 0.85 | 0.11 | 0.25 | 0.48 | 0.51 | 0.51 | 0.40 | 0.13 |
| JP2 | 0.33 | 0.84 | 0.11 | 0.26 | 0.48 | 0.50 | 0.50 | 0.33 | 0.08 |
| BOO1 | 0.33 | 0.84 | 0.11 | 0.26 | 0.48 | 0.50 | 0.50 | 0.33 | 0.08 |
| BOO2 | 0.34 | 0.84 | 0.10 | 0.26 | 0.49 | 0.51 | 0.51 | 0.34 | 0.08 |
| BOO3 | 0.33 | 0.85 | 0.10 | 0.26 | 0.47 | 0.51 | 0.51 | 0.33 | 0.07 |
| BOO4 | 0.34 | 0.85 | 0.12 | 0.22 | 0.34 | 0.50 | 0.50 | 0.34 | 0.14 |
| BOO5 | 0.35 | 0.85 | 0.11 | 0.22 | 0.41 | 0.49 | 0.49 | 0.35 | 0.08 |
| R1 | 0.29 | 0.87 | 0.11 | 0.19 | 0.50 | 0.50 | 0.50 | 0.29 | 0.06 |
| R2 | 0.30 | 0.87 | 0.10 | 0.26 | 0.49 | 0.50 | 0.50 | 0.30 | 0.07 |
| R3 | 0.30 | 0.92 | 0.06 | 0.17 | 0.46 | 0.50 | 0.50 | 0.30 | 0.07 |
| R4 | 0.30 | 0.88 | 0.09 | 0.24 | 0.39 | 0.51 | 0.51 | 0.30 | 0.09 |
| R5 | 0.28 | 0.91 | 0.08 | 0.15 | 0.45 | 0.49 | 0.49 | 0.28 | 0.08 |
| R6 | 0.30 | 0.88 | 0.09 | 0.18 | 0.52 | 0.49 | 0.49 | 0.30 | 0.06 |
| R7 | 0.30 | 0.88 | 0.10 | 0.26 | 0.46 | 0.50 | 0.50 | 0.30 | 0.07 |
| R8 | 0.30 | 0.88 | 0.10 | 0.17 | 0.44 | 0.50 | 0.50 | 0.30 | 0.05 |
| R9 | 0.30 | 0.89 | 0.08 | 0.16 | 0.56 | 0.49 | 0.49 | 0.30 | 0.11 |
| R10 | 0.30 | 0.90 | 0.08 | 0.21 | 0.50 | 0.48 | 0.48 | 0.30 | 0.05 |
| R11 | 0.30 | 0.90 | 0.07 | 0.17 | 0.50 | 0.52 | 0.52 | 0.30 | 0.07 |
| R12 | 0.29 | 0.89 | 0.08 | 0.18 | 0.61 | 0.48 | 0.48 | 0.29 | 0.15 |
| R13 | 0.40 | 0.89 | 0.07 | 0.19 | 0.48 | 0.49 | 0.49 | 0.40 | 0.12 |
| R14 | 0.36 | 0.89 | 0.08 | 0.17 | 0.55 | 0.50 | 0.50 | 0.36 | 0.11 |
| R15 | 0.40 | 0.91 | 0.10 | 0.17 | 0.51 | 0.48 | 0.48 | 0.40 | 0.13 |
| R16 | 0.38 | 0.88 | 0.06 | 0.18 | 0.48 | 0.50 | 0.50 | 0.38 | 0.10 |
| R17 | 0.36 | 0.90 | 0.06 | 0.20 | 0.47 | 0.49 | 0.49 | 0.36 | 0.07 |
| R18 | 0.36 | 0.87 | 0.08 | 0.20 | 0.55 | 0.49 | 0.49 | 0.36 | 0.10 |
| | | <i>Sm</i> | <i>DVm</i> | <i>PAm</i> | <i>AAm</i> | <i>Z.Wm</i> | <i>Am</i> | <i>CMm</i> | <i>Z.Lim</i> |
| Average | | 0.32 | 0.88 | 0.09 | 0.20 | 0.47 | 0.50 | 0.50 | 0.32 |
| Standard-deviation | | 0.05 | 0.02 | 0.02 | 0.04 | 0.08 | 0.01 | 0.01 | 0.05 |
| Maximum | | 0.40 | 0.92 | 0.12 | 0.26 | 0.61 | 0.52 | 0.52 | 0.40 |
| Minimum | | 0.23 | 0.84 | 0.06 | 0.10 | 0.24 | 0.48 | 0.48 | 0.23 |

The analysis of Table 2 shows a straight relationship between parameters *S* and *Z.Li* and parameters *A* and *CM*. The values of *S* remain identical to the values of *Z.Li*. In the same way, values of *A* are identical to *CM*, for all rules under analysis. This relationship between parameters suggest the exclusion of some parameters from further analysis. We choose to exclude *Z.Li* and *CM* because both present a high complexity level for computation, when compared with *S* and *A* respectively.

Observing the average, standard deviation, maximum and minimum values for each parameter, as well as *DE* in Table 2, it can be concluded that most rules present values very close each other, except for parameter *Z.W*, which has high variation.

For a deeper analysis of correlations between the values of DBFPs previously presented (excluding *Z.Li* and *CM*), Table 3 shows a correlation matrix. The computation of this matrix is based on the Pearson's correlation coefficient (*r*) (Ahlgren et al., 2003).

According to the correlation matrix shown, there is no high correlation between parameters *S*, *DV*, *PA*, *AA*, *Z.W* and *A*. It is worth to mention that Vincent (2005) defined high correlation for absolute

Table 3
Correlation matrix of the DBFPs.

| | <i>S</i> | <i>DV</i> | <i>PA</i> | <i>AA</i> | <i>Z.W</i> | <i>A</i> |
|------------|----------|-----------|-----------|-----------|------------|----------|
| <i>S</i> | 1.00 | −0.16 | 0.02 | 0.21 | 0.30 | −0.14 |
| <i>DV</i> | | 1.00 | −0.75 | −0.76 | 0.06 | −0.39 |
| <i>PA</i> | | | 1.00 | 0.60 | −0.14 | 0.18 |
| <i>AA</i> | | | | 1.00 | 0.15 | 0.32 |
| <i>Z.W</i> | | | | | 1.00 | −0.30 |
| <i>A</i> | | | | | | 1.00 |

values higher than 0.9 in the matrix. This suggests the hypothesis that all the mentioned parameters are equally important for describing the dynamic forecasting behavior of a CA.

4. Transition Rule Induction Using DBFPs

In this section two basic issues are addressed: (a) what is the real usefulness of the DBFPs for identifying specific dynamic behaviors? and (b) Is it possible to emerge different behaviors for rules with the same DBFPs values?

The objective of these questions is to evaluate how the DBFPs can be used in the process of rule induction. That is, if it is possible to use previous knowledge about the desired dynamic behavior to devise transition rules, without the need of exhaustive simulation of the CA.

To answer the questions, an evolutionary rule induction system was developed to find transition rules that present “null” behaviors. The proposed system is described in details in the next subsection.

4.1. Rule Induction with Evolutionary Computation

In mathematics, the term “induction” is understood as a reasoning by which a given property of one element is extended to all elements of a set. In this context, it is possible to generalize conclusions obtained from the analysis of a set of premises. According to Quinlan (1986), the rule induction problem consists in finding classification rules capable of determining the class of objects by analyzing the values of their attributes. The set of attributes that describe an object usually measure or identify a relevant feature of it. Many algorithms were proposed for rule induction, including evolutionary computation (EC) methods.

EC is a growing field of research in computer science, with many successful applications to real-world problems. Most of paradigms in EC borrow inspiration from natural process of evolution. Amongst those paradigms, the genetic algorithm (GA) is, possibly, the most widely known (Holland, 1975; Goldberg, 1989), since it has been successfully applied to many optimization problems. A direct descendent of GAs is genetic programming (GP), devised by Koza (1992, 1994). The idea behind GP is induce computers to find solutions, in the form of programs, to problems for which they were not previously programmed to. That is, based on a set of pairs *inputs*, *output*, induce a program capable of incorporating a model of the problem and generating outputs for other, previously unseen inputs. To each program a fitness value is associated, based on its ability to provide good solutions to the problem in hand. Due to its distinguishing characteristics, genetic programming have been used for complex rule-induction problems in several domains (see, for instance, Bojarczuk et al., 2004), thus suggesting its applicability in finding transition rules for CAs.

An extension of GP, known as gene-expression programming (GEP) was proposed by Ferreira (2001), uses concepts from both GP and GA. The fundamental difference between these three approaches is how individuals are represented. In a GA, individuals are usually represented as a linear fixed-size string of bits (although other representations can also be found). In GP, individuals are non-linear structures of different sizes and shapes, in the form of a tree. In GEP, individuals are first encoded as a linear string of fixed-size (genome or chromosome), which is later expressed as non-linear structures of different sizes and shapes (expression trees).

GEPCLASS (Weinert and Lopes, 2006) is an implementation of GEP specifically designed for finding rules for classification problems based on supervised learning. In classification problems it is aimed to find a rule or a set of them capable of model a given domain of known samples, and then classify unseen sets of data of the same nature.

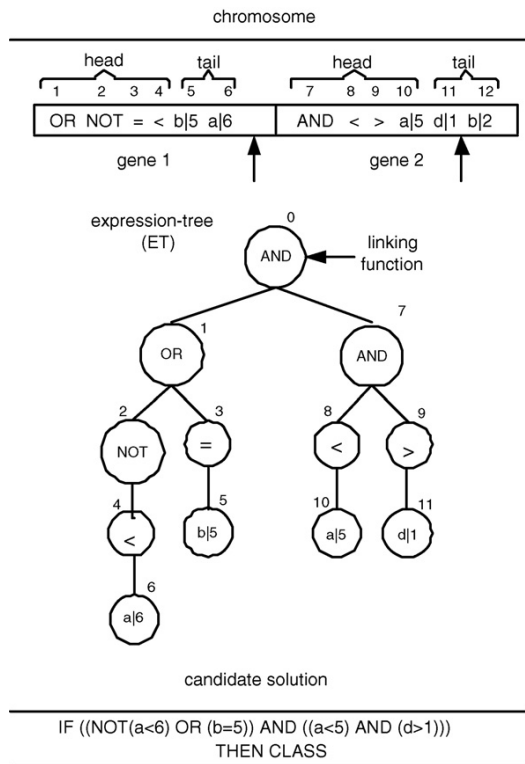


Fig. 4. Example of the structure of a chromosome, the expression tree and its corresponding rule in GEPCLASS.

GEPCLASS implements a structure where a population of individuals is evolved for a number of generations. At each generation several genetic operators is applied to selected individuals of the population, generating diversity and, hopefully, improving the quality of the overall population, regarding a given problem. Following Ferreira (2001), an individual is composed of a single chromosome that, in turn, is composed of n genes. Each gene is divided into two parts: head and tail. The head of the gene can have elements belonging only to the set of functions (operations), such as: *AND*, *OR*, *NOT*, *=*, *≠*, *>* and *<*. The tail, in turn, can have elements either from the set of functions or from the set of terminals. The set of terminals includes the attributes that describe the problem and particular values (such as constants).

GEP uses the biological concept of open read frame ORF as encoding principle. That is, the sequence of elements at the genotypical level has the potential to express phenotypically, but not necessarily all them will be used. Therefore, the encoding region of a gene (ORF) can “activate” or “deactivate” portions of the genetic material during evolution.

The mapping between the genotype to the phenotype is carried out as follows. The whole chromosome is transcribed into a variable-size expression tree. The genetic material used in this process is that one defined by the ORF in each gene. The transcription rules follow the standard of the Karva language (Ferreira, 2001), where each gene is transcribed to a separated sub-tree. Next, all sub-trees are joined together by a linking function (usually, *AND* or *OR*), thus composing the expression tree. This tree represents a candidate solution for the problem in hand and, when evaluated by a fitness function, the quality of this solution will be objectively estimated. The transcription process in GEP is illustrated in Fig. 4. For gene 1 of this figure the ORF ends at position 6, and for gene 2, it ends at position 11. The ORF of a gene is determined by the func-

tions and their respective arities (number of arguments) encoded in the head of the gene. The elements of a gene are read from left-most position towards right. For instance, in gene 1 of the figure there is a logical operator (*OR*), which arity is 2, meaning that two arguments are necessary for evaluating this function. Therefore, the next two operators after *OR* (that is, *NOT* at position 2 and *=* at position 3) belong their arguments. Recursively, one can determine the arguments of *NOT* and *=*, and so on. This procedure is repeated until there are no more functions without arguments. The size of the tail of the gene is determined according to the size of the head (Weinert and Lopes, 2006). Therefore, it is warranted that there will not be functions with missing attributes encoded in the head of the gene. It is important to noted that a precedence relationship must be defined for logical operators, relational operators and terminals.

The genetic operators implemented in GEPCLASS are: mutation, recombination and transposition. All of them comply with the closure property (Koza, 1992), thus finding valid rules. That is, the hierarchical structure of the expression trees is always maintained after the application of the operators (Weinert and Lopes, 2006). The fitness function implemented in GEPCLASS was the same used for other rule induction problems, and is shown in Eq. (3):

$$\text{fitness} = \frac{tp}{tp + fn} * \frac{tn}{tn + fp} \quad (3)$$

where

- *tp* (true positive): the rule predicts that the instance belongs to a class, and it indeed belongs;
- *fp* (false positive): the rule predicts that the instance belongs to a class, but in fact it does not;
- *tn* (true negative): the rule predicts that the instance does not belong to a class, and indeed it does not;
- *fn* (false negative): the rule predicts that the instance does not belong to a class, but in fact it does.

Further details of the genetic operators of GEPCLASS and their closure property, as well as the fitness function implemented for classification problems can be found in Weinert and Lopes (2006).

4.2. Implementation

GEPCLASS was modified to be applied to the classification by the majority rule problem. The instance of this problem used in our work was the same suggested by Juillé et al. (1998): a unidimensional CA with 149 cells and radius 3. The running parameters of GEPCLASS were:

- Number of individuals: 50.
- Number of generations: 50.
- Linking function: *AND*.
- Function set: *AND*, *OR*, *NOT*, *=* and *≠*.
- Terminal set: *a*, *b*, *c*, *d*, *e*, *f*, *g*.
- Number of genes per chromosome: 3.
- Size of head: 6–15.
- Selection method: stochastic tournament.
- Genetic operators: the same defined in Ferreira (2001), with the same probabilities.

The terminal set (*a*, *b*, *c*, *d*, *e*, *f*, *g*) are binary and represent all the possible combinations of neighborhood of the problem, thus mapping all transitions of a given rule. The number of combinations is 128, obtained as shown in Section 2. The definition of these 128 binary combinations, each one with 7 bits (one bit for each terminal), can be easily generated converting from binary to decimal base. That is, the conversion of the decimal numbers present in the interval 0...127 to their corresponding binary numbers, define

Table 4
State transition table.

| # | Transitions | | | | | | | Rule |
|-----|-------------|---|---|---|---|---|---|------|
| | a | b | c | d | e | f | g | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 127 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 128 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

the 128 possible neighborhood combinations, according to Table 4. For instance, a possible rule encoded in an individual of GEPCLASS would be:

*if ((a = 0) AND (b = 1)) then
Rule:=1; else Rule:=0;*

In fact, there is no restriction regarding the number of terminals that can appear in the rule. In this simple example only the terminals *a* and *b* (see Table 4) were considered. Any terminal that do not appear in a rule can effectively assume any value, since they do not influence the final outcome of the rule.

The concatenation of all 128 output (the last column of Table 4), that is, 00...10 forms the rule that is said a candidate solution to the problem in hand (see Section 2.3). Next, the candidate solution is submitted to the fitness computation procedure.

The fitness function is computed using the following DBFPs *S*, *DV*, *PA*, *AA*, *Z.W* and *A*. Since there is no known optimal set of values for these parameters, the average values were adopted as reference, as shown in Table 2. Eq. (4) presents the fitness function used in this work.

$$fitness = Sn * DVn * PAn * AAn * Z.Wn * An \quad (4)$$

Acronyms in this equation are those defined in Section 3 and the subscript *n* indicates that values are normalized in the range [0...1] according to Table 5 and the corresponding average values. The fitness function has the property of returning 1 only when all parameters are 1. The basic idea in this normalization is to find linear equations that return a high fitness for rules that have the values of the DBFPs around to the average values, and low fitness when the DBFPs are far from average. Fig. 5 presents plots illustrating how the linear adjustment of the DBFPs are done.

5. Results and Discussion

In the analysis of the DBFPs in Section 3 it was observed that parameters *S* and *Z.Lj*, and *A* and *CM* are numerically equivalent, for the same rule. This finding allowed the exclusion of parameters

Table 5
Linear adjustment of the DBFPs.

| Normalized parameter | Pseudocode |
|----------------------|---|
| <i>Sn</i> | <i>If (S <= Sm) Then Sn:=3.125 * S; Else Sn:= - 1.470 * S + 1.470;</i> |
| <i>DVn</i> | <i>If (DV <= DVm) Then DVn:=1.136 * DV; Else DVn:= - 8.333 * DV + 8.333;</i> |
| <i>PAn</i> | <i>If (PA <= PAm) Then PAn:=11.112 * PA; Else PAn:= - 1.098 * PA + 1.098;</i> |
| <i>AAn</i> | <i>If (AA <= AAm) Then AAn:=5 * AA; Else AAn:= - 1.250 * AA + 1.250;</i> |
| <i>Z.Wn</i> | <i>If (Z.W <= Z.Wm) Then Z.Wn:=2.127 * Z.W; Else Z.Wn:= - 1.886 * Z.W + 1.886;</i> |
| <i>An</i> | <i>If (A <= Am) Then An:=2 * A; Else An:= - 2 * A + 2</i> |

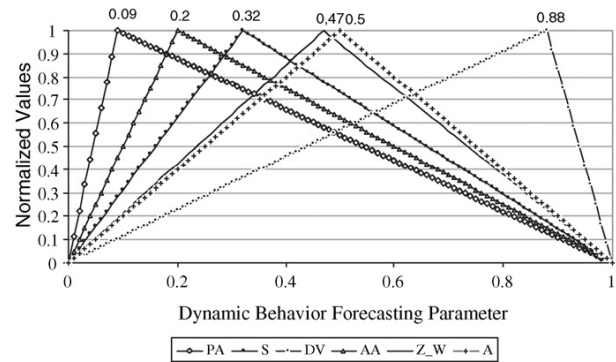


Fig. 5. Linear adjustment of the DBFPs.

Table 6
Transition rules found by GEPCLASS considering parameters individually.

| Rule | Hexadecimal | Efficiency |
|---------|----------------------------------|------------|
| Rule_S | 0505A5A50F0F0F00505A5A50F0F0 | 0.00% |
| Rule_DV | 00110033555555550515053755555555 | 50.30% |
| Rule_PA | 000000007770FFF00000000770FFF | 50.33% |
| Rule_AA | 0CF0CF0CF0CF0CF3FFF3FFF3FFF3FFF | 11.08% |
| Rule_ZW | FF33FF73FF33FF50FF33FF73FF33FF50 | 0.00% |
| Rule_A | AABB0031BBBB3333AABA0030BBB3333 | 0.00% |

Z.Lj and *CM* from the remaining analysis.

Once that all the 31 rules analyzed exhibit “null” behavior and there are no reference values for the DBFPs computed with these rules, the average values were adopted as reference. It is fair to use average values since the Euclidean distance between the parameters of the 31 rules and the average values of the parameters is small for most cases.

The analysis of correlation between parameters revealed that parameters *DV*, *PA*, *AA*, *Z.W* and *A* really provide distinct information in the process of forecasting the dynamic behavior or CAs.

A GEP-based system (GEPCLASS) was developed for transition rule induction, thus finding rules with “optimized” DBFPs. The fitness function of GEPCLASS was changed according to the specific forecasting parameter to be optimized.

Initially, GEPCLASS was run with the parameters mentioned in Section 4.2. We obtained 6 rules for the classification by the majority rule using a unidimensional CA with 149 cells and radius 3. These rules are presented in Table 6 in hexadecimal format with the respective values of efficiency. Efficiency was computed by means of the simulation of 10⁴ CAs during 200 iterations, starting with random configurations.

Each one of the rules presented in Table 6 is regarding the optimization of a given DBFP. For instance, for Rule_S GEPCLASS found a rule whose *S* value is as near as possible to the value of *Sm*. Since that other parameters were not considered in this experiment, the fitness function shown in Eq. (4) was reduced to: *fitness = Sn*. Similar procedure was adopted for all the other DBFPs.

Table 7
Values for the parameters of rules presented in Table 6.

| Rule | S | DV | PA | AA | Z.W | A | DE |
|----------|-------------|-------------|-------------|-------------|-------------|-------------|------|
| Rule_S | 0.32 | 0.62 | 0.19 | 0.40 | 0.69 | 0.44 | 0.41 |
| Rule_DV | 0.22 | 0.88 | 0.08 | 0.18 | 0.69 | 0.39 | 0.27 |
| Rule_PA | 0.20 | 0.87 | 0.09 | 0.18 | 0.66 | 0.33 | 0.28 |
| Rule_AA | 0.18 | 0.79 | 0.16 | 0.20 | 0.44 | 0.75 | 0.31 |
| Rule_Z.W | 0.19 | 0.38 | 0.33 | 0.67 | 0.47 | 0.73 | 0.77 |
| Rule_A | 0.24 | 0.52 | 0.27 | 0.45 | 0.55 | 0.50 | 0.49 |

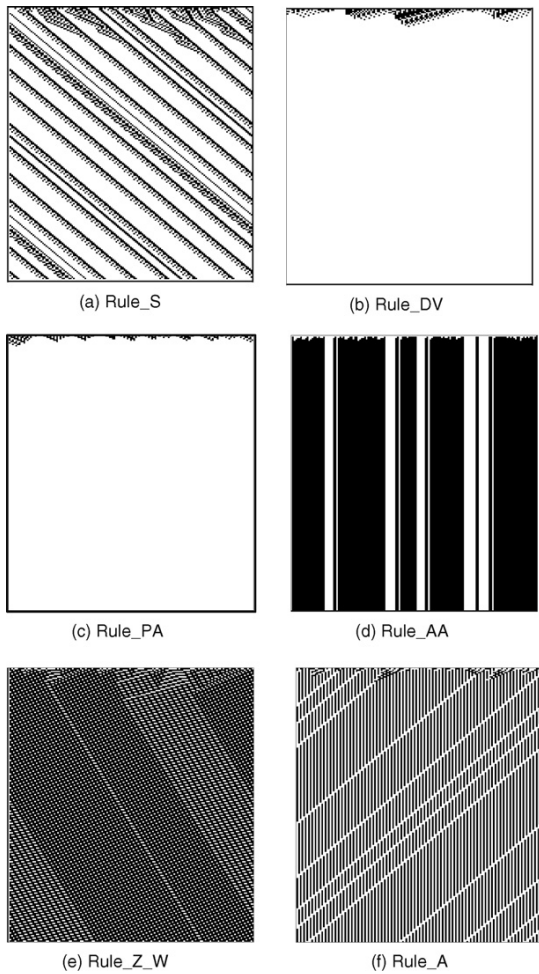


Fig. 6. Simulation of the dynamic behavior of rules shown in Table 6.

Table 7 shows the values of the DBFPs regarding to the rules previously shown in Table 6.

Fig. 6 presents an example of the simulation of the dynamic behavior for each one of the rules previously shown in Table 6.

Although GEPCLASS did not have difficulty in finding the 6 rules, the efficiency of them, as well as their dynamic behavior, indicate that they are not solutions for the classification by the majority rule problem. Also, not all rules present a “null” behavior. Therefore, these results indicate that the use of DBFPs individually, as the only source of information for the rule induction procedure, is inefficient.

The efficiency rate for the rules found were far below the expected values. For instance, rules Rule.DV and Rule.PA had efficiency rate around 50%. However, observing the simulations of these rules in Fig. 6 and the value of parameter “activity” in Table 7, it becomes clear that such rules can take any initial configuration of the CA to a state where all cells are 0, after 200 iterations. This do not solve the problem, although a “null” behavior can be identified. Rule.S, Rule.AA, Rule.Z.W and Rule.A besides presenting very low values for efficiency, they lead the CA to periodic behavior, as shown in Fig. 6.

Another experiment was done using GEPCLASS to find rules for the same problem. However, differently from those rules previously shown in Table 6, now all parameters were jointly considered

Table 8

Transition rules found by GEPCLASS considering all parameters together.

| Rule | Hexadecimal | Efficiency |
|---------|----------------------------------|------------|
| Rule_1 | 111110101111F3101F1F1F1F1FFF1F | 2.83% |
| Rule_2 | 000F000F555F555FCCFCFFDDDFDDDF | 0.00% |
| Rule_3 | 37370113377F015F37370113377F015F | 49.96% |
| Rule_4 | 0AAA0FFF011105550AAA0FFF0BBB0FFF | 0.00% |
| Rule_5 | 00000F0FFFF0F1F00110F1FFFF0F1F | 1.83% |
| Rule_6 | 333300000000F0F3F3F3F3F3F3F3F3F | 49.33% |
| Rule_7 | 0A5F0A5F0A5FFFF005500550A55B77 | 0.00% |
| Rule_8 | 0033F0330033F0330033FFFF0033FFFF | 49.73% |
| Rule_9 | 0A0F0A0F000F000F3F3F3F3F3F3F3F3F | 1.62% |
| Rule_10 | 331133113311331111FF11FF11FF11FF | 49.33% |

by using the fitness function of Eq. (4). The top 10 rules found by GEPCLASS are shown in Table 8 and the corresponding DBFPs computed for these rules are shown in Table 9. Examples of the dynamic behavior of CAs for these rules 2, 3, 4, 6, 8 and 10 are shown in Fig. 7.

The difficulty for GEPCLASS to find acceptable solutions was significantly higher than in the previous experiment, since more degrees of freedom were added to the problem. Only Rule.8 and Rule.10 were capable of maintaining values within the [maximum, minimum] range established as reference (see Table 2). Although these rules presented a “null” behavior in the dynamic simulation, they achieved an efficiency rate around 50%. Observing the simulation of these rules (Fig. 7d–f) it is clear that these rules can take any initial configuration of a CA to a state in which all cells are 0, after 200 iterations.

For all the rules found in this experiment, the parameter “sensitivity” was quite different from the reference value. However, observing Table 2 we note that some rules, such as GKL, DAS, ABK and CRA have values of the DBFPs very close to those of the 10 rules found by GEPCLASS (Table 8). Therefore, the use of an “optimized” set of DBFPs in the rule induction procedure do not take important benefits to the quality of solutions.

6. Conclusions and Future Work

A total of 31 rules were used in the analysis of eight DBFPs. The main objective of this analysis was to verify the capacity of each parameter individually and the group of parameters, for forecasting the dynamic behavior of CAs. A classical problem in CA literature was selected for this study: the classification by the majority rule. This problem is interesting because the solutions for it are known as “null” behavior rules, according to Li and Parckard (1990). That is, independently of the initial configuration of the system, it will converge to a state where all cells of the CA will take the same value (either 0 or 1) after a transient period of successive iterations.

According to the results obtained, it can be concluded that the DBFPs studied in this work, when considered individually or together, do not provide enough information for classifying efficiently the dynamic behavior of a CA. Although this conclusion is

Table 9

Values of the DBFPs for rules presented in Table 8.

| Rule | S | DV | PA | AA | Z.W | A | DE |
|---------|------|------|------|------|------|------|------|
| Rule_1 | 0.26 | 0.86 | 0.09 | 0.26 | 0.46 | 0.47 | 0.09 |
| Rule_2 | 0.24 | 0.82 | 0.13 | 0.22 | 0.49 | 0.58 | 0.14 |
| Rule_3 | 0.26 | 0.82 | 0.13 | 0.22 | 0.46 | 0.52 | 0.10 |
| Rule_4 | 0.27 | 0.80 | 0.13 | 0.22 | 0.49 | 0.52 | 0.11 |
| Rule_5 | 0.24 | 0.84 | 0.11 | 0.24 | 0.46 | 0.54 | 0.11 |
| Rule_6 | 0.25 | 0.85 | 0.10 | 0.25 | 0.50 | 0.50 | 0.10 |
| Rule_7 | 0.27 | 0.81 | 0.13 | 0.21 | 0.47 | 0.51 | 0.10 |
| Rule_8 | 0.23 | 0.85 | 0.10 | 0.20 | 0.45 | 0.50 | 0.10 |
| Rule_9 | 0.22 | 0.86 | 0.09 | 0.20 | 0.47 | 0.53 | 0.11 |
| Rule_10 | 0.25 | 0.85 | 0.10 | 0.20 | 0.50 | 0.50 | 0.08 |

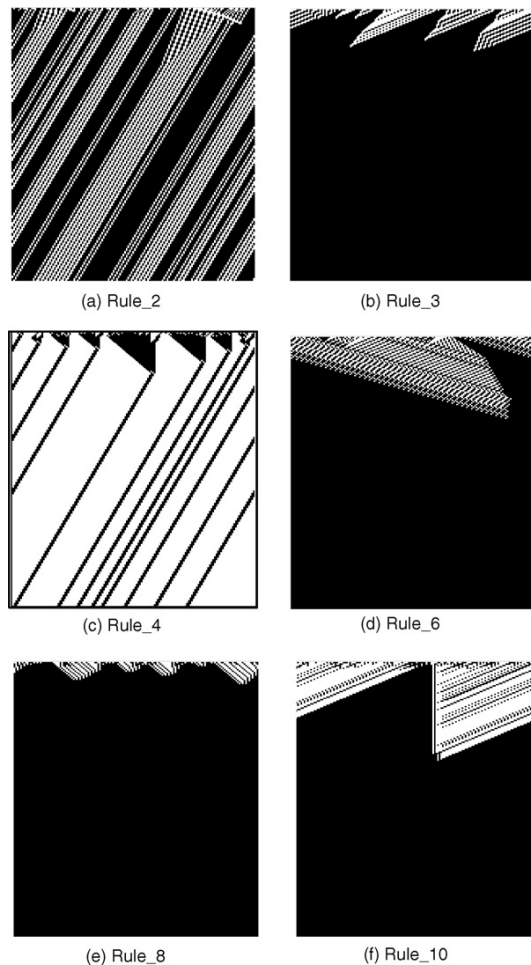


Fig. 7. Simulation of the dynamic behavior of rules 2, 3, 4, 6, 8 and 10, show in Table 8.

specific for the problem of classification by the majority rule, it is fair to expect that it can be extended to other classes of problems dealt by CAs.

It was observed that different transition rules having the values of the DBFPs very close each other can display distinct dynamic behavior in the simulation. These results found suggest that it is not possible to identify a clear boundary of values for the DBFPs studied for identifying transition rules with “null” behavior.

Consequently, this fact may have a relevant impact in approaches that use DBFPs in the process of rule-induction for CAs, such as the evolutionary computation approaches previously mentioned. The way the DBFPs were used in this work requests a set of reference values for them. Possibly, this reference set does not exist for the problem treated here, and shall be addressed in future work. Results obtained in this work suggest the need of further exploration in DBFPs. More expressive parameters, regarding the full dynamic behavior of CAs, are still needed.

A possible line of investigation is devising parameters not completely dependent on the transition rules, but that take into account features captured by sampling a simulation of the CA. Since simulations are computationally expensive, further development will have to address this issue, either using reconfigurable hardware for accelerating simulations (such as in Weinert et al., 2007) or using parallel computing.

Acknowledgements

This work was partially supported by the Brazilian National Research Council under grant no. 309262/2007-0 to H.S. Lopes.

References

- Ahlgren, P., Jarneving, B., Rousseau, R., 2003. Requirements for a cocitation similarity measure, with special reference to Pearson's correlation coefficient. *Journal of the American Society for Information Science and Technology* 54 (6), 550–560.
- Beauchemina, C., Samuelb, J., Tuszynskia, J., 2005. A simple cellular automaton model for influenza a viral infections. *Journal of Theoretical Biology* 232, 223–234.
- Benkniouar, M., Benmohamed, M., 2004. Cellular automata for cryptography. In: *Proceedings of the International Conference on Information and Communication Technologies: From Theory to Applications*, Damascus, Syria, pp. 423–424.
- Binder, P., 1993. A phase diagram for elementary cellular automata. *Complex Systems* 7, 241–247.
- Bojarczuk, J., Lopes, H., Freitas, A., 2004. A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. *Artificial Intelligence in Medicine* 30, 27–48.
- Bortot, J., de Oliveira, P., Oliveira, G., 2004. Multiobjective, heuristic evolutionary search in a cooperative environment leads to the best cellular automaton rule in the density classification task. In: *Proceedings of the VIIIth Brazilian Symposium on Neural Networks*. IEEE Press/SBC, São Luís (CD-ROM: Paper 3565).
- Chen, Q., Mynett, A., 2003. Effects of cell size and configuration in cellular automata based prey–predator modelling. *Simulation Modelling Practice and Theory* 11, 609–625.
- Corne, D., Frisco, P., 2008. Dynamics of HIV infection studied with cellular automata and conformon-p system. *BioSystems* (91), 531–544.
- Cranny, T., Bossomaier, T., 1999. The density classification problem for cellular automata: searching within structure. Tech. rep., Charles Sturt University.
- Das, R., Crutchfield, J., Mitchell, M., Hanson, J., 1995. Evolving globally synchronized cellular automata. In: *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 336–343.
- David, A., Forrest, B., Koza, H., 1996. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: *Proceedings of the First Annual Conference the Genetic Programming 1996*. MIT Press, Cambridge, MA, pp. 3–11.
- Davis, L., 1991. *Handbook of Genetic Algorithms*, 1st edition. Van Nostrand Reinhold.
- Ferreira, C., 2001. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems* 13, 87–129.
- Ferreira, C., 2002. Discovery of the Boolean functions to the best density-classification rules using gene expression programming. *Lecture Notes in Computer Science* 2278, 51–60.
- Fu, S., Milne, G., 2003. Epidemic modeling using cellular automata. In: *Proceedings of the 1st Australian Conference on Artificial Life*, pp. 43–57.
- Gacs, P., Kurdyumov, G., Levin, L., 1978. One dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii* 12, 92–98.
- Gangadhar, D., 2005. Pelican - protein-structure alignment using cellular automata models. In: *Proceedings of the International Conference in Adaptive and Natural Computing Algorithms*, Coimbra, Portugal, pp. 308–311.
- Georgoudas, I., Sirakoulis, G., Andreadis, I., 2007. Modelling earthquake activity features using cellular automata. *Mathematical and Computer Modelling* 46, 124–137.
- Goldberg, D., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st edition. Addison-Wesley Professional.
- Hofman, K., Bucher, P., Falquet, L., Bairoch, A., 1999. The prosite database, its status in 1999. *Nucleic Acids Research* 27, 215–219.
- Holland, J., 1975. *Adaptation in Natural and Artificial Systems*, 1st edition. The University of Michigan Press.
- Juillé, H., Pollack, J., 1998. Coevolving the ideal trainer: application to the discovery of cellular automata rules. In: Koza, J. (Ed.), *Proceedings of the Third Annual Conference in Genetic Programming*. Morgan Kaufmann, San Francisco, pp. 519–527.
- Kansal, A.R., Torquato, S., Harsh, G.R., IV, Chiocia, E.A., Deisboek, T.S., 2000. Cellular automaton of idealized brain tumor growth dynamics. *BioSystems* 55, 119–127.
- Khan, A., Choudhury, P., Dihadar, K., Verma, R., 1999. Text compression using two-dimensional cellular automata. *Computers and Mathematics with Applications* 37 (6), 115–127.
- Kiera, L., Chenga, C., Testab, B., Carruptb, P., 1996. A cellular automata model of enzyme kinetics. *Journal of Molecular Graphics* 14, 227–231.
- Koza, J., 1992. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, 1st edition. MIT Press.
- Koza, J., 1994. *Genetic Programming. II. Automatic Discovery of Reusable Programs*, 1st edition. MIT Press.
- Langton, C., 1990. Computation at the edge of chaos: phase transitions and emergent computation. *Physica D* 42, 12–37.
- Laurio, K., Linaker, F., Narayanan, A., 2007. Regular biosequence pattern matching with cellular automata. *Information Sciences* 146, 89–101.
- Li, W., 1991. Parameterizations of cellular automata rule space. Tech. rep., Santa Fe Institute Tech.
- Li, W., Parckard, N., 1990. The structure of elementary cellular automata rule space. *Complex Systems* 4 (3), 281–297.
- Malleta, D., Pillisb, L.D., 2006. A cellular automata model of tumor-immune system interactions. *Journal of Theoretical Biology* 239, 234–350.

- Mitchell, M., 1996. Computation in cellular automata: a select review. In: Gramss, T. (Ed.), *Nonstandard Computation*. VHC Verlagsgesellschaft, Weinheim, pp. 95–140.
- Mitchell, M., Crutchfield, J., Hraber, P., 1994. Evolving cellular automata to perform computations: mechanisms and impediments. *Physica D* 75, 361–391.
- Mitchell, M., Hraber, P., Crutchfield, J., 1993. Revisiting the edge of chaos: evolving cellular automata to perform computations. *Complex Systems* 7, 89–130.
- Mizas, C., Sirakoulis, G.C., Mardires, V., Karafyllidis, I., Glykos, N., Sandaltzopoulos, R., 2008. Reconstruction of DNA sequence using genetic algorithms and cellular automata: towards mutation prediction? *BioSystems* 92, 61–68.
- Morales, F., Crutchfield, J., Mitchell, M., 2001. Evolving two-dimensional cellular automata to perform density classification: a report on work in progress. *Parallel Computing* 27, 571–585.
- Oliveira, G., Asakura, O., de Oliveira, P., 2002a. Dynamic behaviour forecast as a driving force in the coevolution of one-dimensional cellular automata. In: *Proceedings of the VIIth Brazilian Symposium on Neural Networks*, vol. 1. IEEE Computer Society, Los Alamitos, CA, USA, pp. 98–103.
- Oliveira, G., Bortot, J., de Oliveira, P., 2002b. Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task. In: *Proceedings of the 8th International Conference on Artificial Life*. MIT Press, Cambridge, MA, USA, pp. 202–206.
- Oliveira, G., Bortot, J., de Oliveira, P., 2007. Heuristic search for cellular automata density classifiers with a multiobjective evolutionary algorithm. In: *Proceedings of the VIth Congress of Logic Applied to Technology LAPTEC 2007*, vol. 1, Santos, SP, pp. 1–12.
- Oliveira, G., Omar, N., de Oliveira, P., 2001. Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. *Artificial Life* 7, 277–301.
- Quinlan, J., 1986. Induction of decision tree. *Machine Learning* 1, 81–106.
- Richards, F., Meyer, T., Packard, N., 1990. Extracting cellular automaton rules directly from experimental data. *Physica D* 45, 189–202.
- Rosin, P., 2005. Training cellular automata for image processing. *Lecture Notes in Computer Science* 3540, 195–204.
- Swiecicka, A., Seredynski, F., 2000. Cellular automata approach to scheduling problem. In: *Proceedings of the International Conference on Parallel Computing in Electrical Engineering*. IEEE Computer Society, Washington, DC, USA, pp. 29–33.
- Tomassini, M., Perrenoud, M., 2001. Cryptography with cellular automata. *Applied Soft Computing* 1, 151–160.
- Vincent, W., 2005. *Statistics in Kinesiology*. Human Kinetics, Champaign.
- Wei, Y., Ying, S., Fan, Y., Wang, B., 2003. The cellular automata model of investment behavior in the stock market. *Physica A* 325, 507–516.
- Weinert, W., Benitez, C., Lopes, H., Lima, C., 2007. Simulation of the dynamic behavior of one-dimensional cellular automata using reconfigurable computing. *Lecture Notes in Computer Science* 4419, 385–390.
- Weinert, W., Lopes, H., 2006. GEPCLASS: a classification rule discovery tool using gene expression programming. *Lecture Notes in Computer Science* 4093, 871–880.
- Wolfram, S., 1983. Cellular automata. *Los Alamos Science* 9, 2–21.
- Wolfram, S., 1984. Universality and complexity in cellular automata. *Physica D* 10, 1–35.
- Wolfram, S., 1986. *Advances in cryptology*. *Lecture Notes in Computer Science* 218, 429–432.
- Wolfram, S., 2002. *A New Kind of Science*, 1st edition. Champaign, Wolfram Media.
- Wuensche, A., 1994. Complexity in one-D cellular automata: gliders, basins of attraction and the Z parameter. *Tech. Re 94-04-025*, Santa Fe Institute.
- Wuensche, A., 1999. Classifying cellular automata automatically: finding gliders, filtering, and relating space–time patterns, attractor basins and the Z parameter. *Complexity* 3, 47–66.
- Wuensche, A., Lesser, M., 1992. *The Global Dynamics of Cellular Automata*. Addison-Wesley, Reading, USA.
- Xiao, X., Shao, S., Ding, Y., Huang, A., Chen, X., Chou, K.-C., 2005. Using cellular automata to generate image representation for biological sequences. *Amino Acids* 28, 29–35.