



unesp

UNIVERSIDADE ESTADUAL PAULISTA

Programa de Pós-Graduação em Engenharia Elétrica

**ALGORITMO GENÉTICO EMPREGADO
NA ALOCAÇÃO DE ESTADOS EM
MÁQUINAS MARKOVIANAS**

Carlos Eduardo da Silva Santos

Ilha Solteira - SP

1210001675



unesp





UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO MESQUITA FILHO"



FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ALGORITMO GENÉTICO EMPREGADO NA ALOCAÇÃO DE ESTADOS EM MÁQUINAS MARKOVIANAS

Dissertação apresentada à Faculdade de Engenharia de Ilha Solteira da Universidade Estadual Paulista-UNESP, como parte dos requisitos necessários para obtenção do título de Mestre em Engenharia Elétrica.

Carlos Eduardo da Silva Santos

Candidato

Orientador: **Eng. Eletr. Alexandre César Rodrigues da Silva – FEIS / UNESP**

Ilha Solteira, Fevereiro de 2005.

BCoIS - FEIS • UNESP



FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação/Serviço Técnico de Biblioteca e Documentação da UNESP-Ilha Solteira

S237a Santos, Carlos Eduardo da Silva.
Algoritmo genético empregado na alocação de estados em máquinas markovianas /
Carlos Eduardo da Silva Santos. – Ilha Solteira : [s.n.], 2005
102 p. :il. (algumas color.)

Dissertação (mestrado) – Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2005

Orientador: Alexandre César Rodrigues da Silva
Bibliografia: p. 99-102

1. Eletrônica digital. 2. Algoritmos genéticos. 3. Otimização combinatória.
4. Máquinas-ferramentas. 5. Redes (Matemática). 6. Alocação de estados.

70707004

CERTIFICADO DE APROVAÇÃO

TÍTULO: ALGORITMO GENÉTICO EMPREGADO NA ALOCAÇÃO DE ESTADOS EM MÁQUINAS MARKOVIANAS

AUTOR: CARLOS EDUARDO DA SILVA SANTOS

ORIENTADOR: Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovado como parte das exigências para obtenção do Título de MESTRE em ENGENHARIA ELÉTRICA pela Comissão Examinadora:

Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

Dr. ORLANDO DEL BIANCO FILHO

Dr. LUIS ANTONIO F DE OLIVEIRA

Data da realização: 11 de fevereiro de 2005

Presidente da Comissão Examinadora
Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

UNESP - "CAMPUS DE ILHA SOLTEIRA"	
SERVIÇO TÉCNICO DE BIBLIOTECA E DOCUMENTAÇÃO	
DATA DE CHEGADA	DATA DE TOMBO
02.02.06	28.04.06
REGISTRADO POR	TOMBO
Ailza	Te. 1675
AQUISIÇÃO	CLASSIFICAÇÃO
Ilha Anta R\$10.00	S237a

1210001675



Ca sup 456883
sup 59690

BCpIS - FEIS - UNESP

Resumo

Devido à necessidade de se projetar sistemas digitais cada vez mais complexos, desenvolvidos a curto prazo, tornou-se essencial o desenvolvimento de ferramentas de síntese digital. A complexidade do circuito combinacional que controla as transições dos elementos de memória e as saídas está intimamente relacionada à fase de alocação de estados. A importância da alocação de estados fica evidente quando a área, a dissipação de energia e a testabilidade são os critérios de otimização. O problema de alocação de estado consiste em obter uma representação binária para cada um dos estados da máquina. Trata-se de um problema NP-Completo, portanto, extremamente complexo, pois devem-se testar todos os arranjos possíveis para obter a alocação ótima. Nota-se, entretanto, que nos CAD destinados a esta fase do projeto de máquinas de estados finitos, é dada pouca atenção à fase da alocação de estados. Um dos motivos é a grande quantidade de elementos de memória disponibilizados nos componentes reconfiguráveis como, por exemplo, as FPGA e os CPLD. Neste trabalho implementaram-se três algoritmos empregados na alocação de estados para máquinas de estados finitos. O programa denominado KADE utiliza-se do algoritmo de Kohavi, modificado para considerar somente as funções de próximo estado. É empregado na alocação de máquinas de estados descritas pelo modelo de Mealy. O programa denominado Nichols utiliza-se da teoria dos conjuntos, onde os elementos são formados pela combinação das variáveis de entrada e das variáveis de estado. É empregado na alocação de máquinas de estados, descritas pelo modelo de Moore. Foi desenvolvido e implementado um algoritmo genético que realiza a alocação de estados; este algoritmo possui duas fases. Na primeira fase tem-se a geração de um grafo baseado na proposta de Amaral, e na segunda fase o Algoritmo Genético propriamente dito. Este programa realiza a alocação de estados em máquinas completamente ou incompletamente especificadas descritas pelo modelo de Moore ou de Mealy. Geraram-se dessa forma três programas didáticos que serão empregados no ensino da disciplina Projetos de Sistemas Digitais. Estudos de casos mostram que o algoritmo Genético proposto neste trabalho apresentou-se tão eficiente, quanto o proposto por Amaral e o clássico JEDI.

Palavras-chave: Alocação de Estados, Algoritmo Genético, Máquinas de Estados Finitos, Otimização, Ferramentas de Síntese.

Abstract

Due the necessity to make digital systems each time more complex, developed in short time, the development of tools of digital synthesis became essential. The complexity of the combinatorial circuit that controls the transitions of the memory elements and the outputs is closely related with the phase of state assignment. The importance of the state assignment is evident when the area, the energy dissipation and the testability are the optimization criteria. The problem of state assignment consists of getting a binary representation for each one of the machine states. One is about a NP-Complete problem, therefore, extremely complex because it must be tested all the possible arrangements to get the optima assignment. It was noticed, however, that in the CAD destined to this project phase of finite states machines, gives little attention to the states assignment phase. One of the reasons is the great quantity of memory elements available in the reconfigured components as, for example, the FPGA and the CPLD. In this work was implemented three algorithms used in the states assignment for finite states machine. The program called KADE uses the Kohavi algorithm, modified to only consider the functions of next state. It is used in the allocation of described states machines through the model of Mealy. The program called Nichols uses the theory of sets, where the elements are formed by the combination of the input variable and state variable. It is used in the allocation of described states machines through the model of Moore. It was developed and implemented a genetic algorithm that realizes the states assignment, this algorithm have two phases. In the first phase generates a graph based on the proposal of Amaral and in the second phase it is the Genetic Algorithm properly. This program realizes the states assignment in machines completely or incompletely specified that were described through the model of Moore or Mealy. It was generated by this three didactic programs that will be used for teaching the subject Projects of Digital Systems. Studies of cases show that the considered Genetic algorithm in this work revealed so efficient, how much considered through Amaral and classic JEDI.

Keywords: State Assignment, Genetic Algorithm, Finite State Machine, Optimization, Synthesis Tools.

Lista de Figuras

Figura 1.1: Modelo genérico de uma máquina de estados.	12
Figura 1.2: Diagrama de estados de uma máquina de estados descrito pelo modelo de Mealy.	13
Figura 1.3: Artigos Publicados sobre alocação de estado na última década.	16
Figura 2.1: Fluxograma do Programa KADE.....	35
Figura 3.1: Diagrama de estados da máquina M_5	39
Figura 3.2: Representação dos <i>output states</i> no mapa de Karnaugh para a máquina M_5	39
Figura 3.3: Máquina M_6 representada no Mapa de Karnaugh.....	45
Figura 3.4: Cobertura mínima do <i>output set</i> (A, C).....	46
Figura 3.5: Fluxograma do programa Nichols.....	48
Figura 3.6: Arquivo de Entrada do Programa Nichols.	50
Figura 3.7: Tela inicial do programa Nichols.....	50
Figura 3.8: Relatório exibido pelo programa Nichols, contendo a alocação para a máquina M_7	51
Figura 3.9: Arquivo de saída com alocação para a máquina M_7	51
Figura 4.1: Grafo para a máquina M_8	56
Figura 5.1: Fluxograma do programa AGPS.....	71
Figura 5.2: Arquivo de entrada para a máquina M_{10}	72
Figura 5.3: Relatório apresentado por AGPS para a máquina M_{10}	73
Figura 5.4: Arquivo de saída gerado pelo AGPS, máquina M_{10}	73
Figura 6.1: Fragmento do arquivo relatório apresentando a alocação de estados de uma determinada FSM.	77
Figura 6.2: Arquivo de entrada para o KADE (Caso 1.6.1).	78
Figura 6.3: Relatório apresentado pelo KADE.....	79

Figura 6.4: Arquivo de saída gerado pelo KADE (Caso 1.6.1).....	80
Figura 6.5 : Arquivo de saída do programa Tabela.	81
Figura 6.6: Fragmento do arquivo de saída do programa TAB2VHDL para o Caso 1.6.1.....	82
Figura 6.7: Gráfico comparativo entre os métodos, considerando o número de literais (SOP).	83
Figura 6.8: Gráfico comparativo entre os métodos, considerando o número de células lógicas.	84
Figura 6.9: Arquivo de entrada do programa Nichols, representação da máquina do caso 3.61.	85
Figura 6.10: Tela de apresentação do programa Nichols.	86
Figura 6.11: Tela que apresenta a alocação obtida pelo programa Nichols.	86
Figura 6.12: Arquivo de saída do programa Nichols.	87
Figura 6.13: Gráfico comparativo entre os métodos apresentados.....	87
Figura 6.14: Arquivo de entrada da máquina M_{10}	89
Figura 6.15: Relatório apresentado pelo programa AGPS.	90
Figura 6.16: Arquivo de saída da máquina M_{10} , gerado pelo AGPS.	90
Figura 6.17: Arquivo de saída do programa Tabela para máquina M_{10}	91
Figura 6.18: Fragmento do código VHDL gerado pelo TAB2VHDL.	92
Figura 6.19: Fragmento do arquivo relatório do gerado pelo programa MAX + PLUS II.	92
Figura 6.20: Gráfico comparativo entre o programa JEDI e o AGPS.....	93
Figura 6.21: Gráfico comparativo entre JEDI e AGPS, sendo o custo a quantidade de células lógicas.....	94
Figura 6.22: Gráfico comparativo entre o programa AGPS e o proposto por Amaral.....	95

Tabela 6.2: Tabela de custos de cada caso estudado.	83
Tabela 6.3: Tabela de custos de cada método estudado.	83
Tabela 6.4: Tabela de Próximo Estado da máquina do caso 3.6.1.	85
Tabela 6.5: Resultados do Algoritmo do Amaral e AGPS.	95



Lista de Abreviaturas

AG	Algoritmo Genético
AGPS	Algoritmo Genético com Propriedade Substituição
Bit	Binary digit
BOOM	Boolean Minimizer
CAD	Computer-Aided Design
FPGA	Field Programmable Gates Array
FSM	Finite State Machine
GE	Gates Equivalentes
GPS	Global Positioning System
LUT	Lookup table
KADE	Kohavi Algorithm Decomposition
MAX + PLUS II	Multiple Array Matrix Programmable Logic User System
NP-Completo	Não Polinomial Completo
PS	Propriedade Substituição
RTL	Register Transfer Logic
SA	Simulated Annealing
SAP	State Assignment Problem
SC	Saídas Consistentes
SOP	Sum of Products
VHDL	VHSIC High Description Language
VHSIC	Very High Speed Integrate Circuit



Sumário

CAPÍTULO I - INTRODUÇÃO	11
1.1. Máquinas de Estados Finitos ou Markovianas	11
1.2. Estado da Arte	15
1.3. Métodos Clássicos de Alocação	17
1.4. Outros Métodos de Alocação	19
CAPÍTULO II - O MÉTODO DE KOHAVI.....	26
2.1. Propriedades para Decomposição de uma Máquina de Estados.....	26
2.2. Condições para Alocação de Estados pelo Método de Kohavi	30
2.3. O Algoritmo de Kohavi	31
2.4. O Programa KADE.....	34
CAPÍTULO III – O MÉTODO DE NICHOLS	38
3.1. Ouput States.....	38
3.2. Compatibilidade e Output Sets	40
3.3. O algoritmo de Nichols para a Alocação de Estados	42
3.4. O Programa Nichols	47
CAPÍTULO IV – ALGORITMO DE AMARAL	52
4.1. Propriedades que Determinam os Pesos do Grafo.....	52
4.2. Equação que Define os Pesos	54
4.3. Algoritmo Genético proposto por Amaral.....	57
CAPÍTULO V – O PROGRAMA AGPS	60
5.1. Uma Deficiência no Grafo de Amaral.....	60
5.2. Propriedade de Substituição e o Grafo de Amaral	62

5.3. Algoritmo Genético Desenvolvido.....	65
5.4. O Programa AGPS Implementado	69
CAPÍTULO VI – ESTUDOS DE CASOS	75
6.1. O Programa Tabela.....	75
6.2. Alocação Realizada pelo MAX + PLUS II	76
6.3. Avaliação do programa KADE	77
6.4. Avaliação do programa Nichols	84
6.5. O Programa JEDI	88
6.6. Avaliação do programa AGPS	89
CAPÍTULO VII – CONCLUSÃO	96
REFERÊNCIAS BIBLIOGRÁFICAS	99



CAPÍTULO I - INTRODUÇÃO

O capítulo I apresenta a definição de máquinas de estados finitos ou markovianas e a definição de alocação de estados. Demonstra-se a importância da alocação de estados e sua influência nos circuitos combinacionais e como o problema de alocação de estados vem sendo tratado na última década.

1.1. Máquinas de Estados Finitos ou Markovianas

Máquinas de Estados Finitos ou Markovianas são circuitos sequenciais que mudam de comportamento, dependendo da informação que lhes é fornecida (entradas) e da informação armazenada em sua memória (estados) e produzem resultados (saídas) que podem ser de dois tipos: os que dependem somente dos estados e os que dependem dos estados e das entradas.

Uma máquina de estados M pode ser definida como $M = (I, O, S, \delta, \lambda)$, onde I , O e S são conjuntos finitos e não vazios de entradas, saídas e estados, respectivamente, $\delta: I \times S \rightarrow S$ é a função de próximo estado, $\lambda: I \times S \rightarrow O$ é a função de saída para máquinas de Mealy e $\lambda: S \rightarrow O$ é a função de saída para máquinas de Moore.

Os modelos de máquinas como apresentado na Figura 1.1, são encontradas em diversas áreas das engenharias como sistemas de comunicação digital, sistemas digitais de controle, microprocessadores, entre outros.



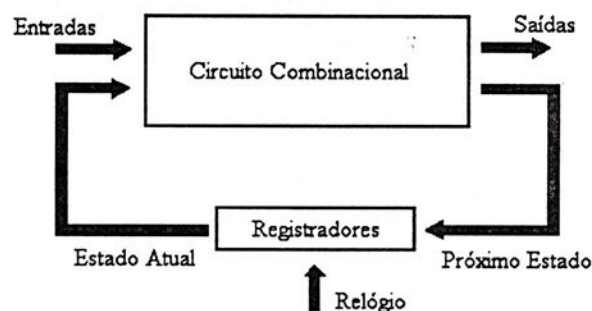


Figura 1.1: Modelo genérico de uma máquina de estados.

Na otimização do circuito combinacional de máquinas de estados finitos um aspecto muito importante do projeto é a alocação de estados, pois a complexidade do circuito combinacional que controla as transições de estados e as saídas está intimamente relacionado com a alocação de estados. Com circuitos combinacionais otimizados têm-se projetos mais eficientes nos quesitos área, velocidade, dissipação de energia e testabilidade.

A alocação de estados consiste em obter a representação binária para os estados, que se encontram na forma literal na fase inicial do projeto, descrito pelo diagrama de estados.

O exemplo 1.1 ilustra a importância da alocação de estados para a otimização do circuito combinacional de uma máquina de estados finitos.

Exemplo 1.1

O diagrama de estados da Figura 1.2 descreve o comportamento de uma máquina de estados, descrita pelo modelo de Mealy.

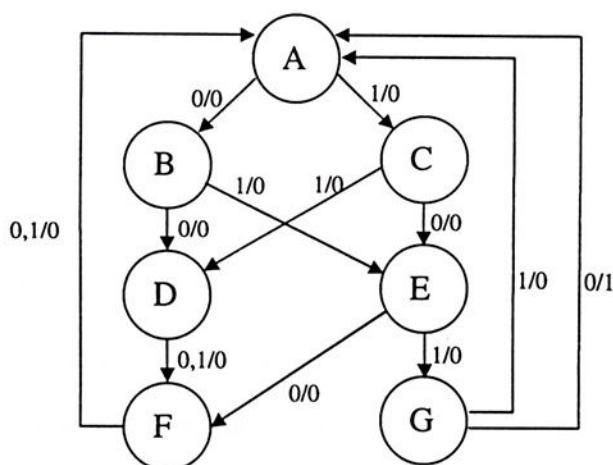


Figura 1.2: Diagrama de estados de uma máquina de estados descrito pelo modelo de Mealy.

Determinando valores binários para os estados na sua forma literal têm-se: A = 000, B = 001, C = 010, D = 011, E = 100, F = 101 e G = 111. Para esta alocação obtêm-se as seguintes funções de próximo estado e saída, quando utilizado os elementos de memória do tipo D (flip-flop D).

$$Z = x'y_1y_2y_3;$$

$$Y_1 = x'y_1'y_3 + x'y_1'y_2 + y_1y_2'y_3';$$

$$Y_2 = x'y_1'y_2'y_3 + xy_2'y_3' + x'y_1'y_3';$$

$$Y_3 = x'y_2'y_3' + xy_1'y_2' + x'y_1'y_3.$$

sendo Y_1 , Y_2 e Y_3 funções de próximo estado e Z a função de saída. As variáveis y_1 , y_2 e y_3 são variáveis de estado, e x é a variável de entrada.

Estas funções possuem um custo igual a 41 literais (SOP)¹. Este circuito pode ser modificado, alterando apenas os valores binários dos estados na sua forma literal. Considere a nova alocação, descrita a seguir: A = 000, B = 010, C = 011, D = 110, E = 111, F = 100 e G = 101.

Para essa alocação obtêm-se as seguintes funções de próximo estado e saída.

$$Z = x'y_1y_2'y_3;$$

¹ Literais (SOP) é a soma de variáveis e das parcelas que compõe uma função booleana. Este critério de custo, de um circuito combinacional, é adotado neste trabalho.

$$Y_1 = x'y_1y_2 + xy_1'y_2' + x'y_1y_2y_3;$$

$$Y_2 = y_3';$$

$$Y_3 = y_2.$$

Com a modificação na alocação de estados o novo circuito combinacional tem um custo igual a 17 literais (SOP). Pode-se verificar que houve uma simplificação considerável, sem nenhuma alteração no comportamento da máquina.

Para encontrar a melhor solução (alocação ótima) para este problema deve-se testar todos os arranjos possíveis, mas isso se torna extremamente complexo quando o número de estados aumenta, pois este problema consiste em um problema NP-Completo (Não Polinomial Completo), ou seja, não possui um polinômio que determina seu comportamento.

Para resolver problemas de otimização do tipo NP-Completo utilizam-se algoritmos especializados em buscas heurísticas como o Algoritmo Genético (AG) [1], [2], *Simulated Annealing* (SA) [3], entre outros. Algoritmos heurísticos não garantem que a solução obtida seja a solução ótima global. Podem-se também utilizar métodos híbridos que combinam métodos exatos com heurísticos, onde os heurísticos buscam uma boa solução e passam estas aos exatos ou vice-versa, para o último convergir.

Neste trabalho de dissertação desenvolveu-se e implementou um algoritmo genético, especializado na alocação de estados que a realiza de forma mais rápida e eficiente que os métodos clássicos [4].

Implementaram-se também dois algoritmos empregados na alocação de estados para máquinas de estados finitos, que deram origem aos programas KADE e Nichols. O programa denominado KADE utiliza-se do algoritmo de Kohavi, modificado para considerar somente as funções de próximo estado. É empregado na alocação de máquinas de estados descritas pelo modelo de Mealy ou Moore. O programa denominado Nichols utiliza-se da teoria dos conjuntos, onde os elementos são formados pela combinação das variáveis de entrada e das

variáveis de estado. É empregado na alocação de máquinas de estados descritas pelo modelo de Moore.

A contribuição dessa dissertação foi desenvolvimento e a implementação de um algoritmo genético denominado AGPS que realiza a alocação de estados. Este algoritmo possui duas fases. Na primeira fase tem-se a geração de um grafo baseado na proposta de Amaral [1] e na segunda fase o Algoritmo Genético desenvolvido. O programa realiza a alocação de estados em máquinas completamente ou incompletamente especificadas, descritas pelo modelo de Moore ou de Mealy.

Os três programas implementados neste trabalho poderão ser utilizados nas disciplinas Circuitos Digitais I, Circuitos Digitais II e Projetos de Sistemas Digitais.

1.2. Estado da Arte

Devido à necessidade de se desenvolver projetos digitais a curto prazo, que realizem tarefas muito complexas, surgiram as ferramentas de síntese digitais CAD (*Computer-Aided Design*) para auxiliar o projetista nesta tarefa. Com máquinas que possuem cada vez mais estados, a contribuição da alocação de estados, para minimizar o circuito combinacional, aumenta rapidamente e maior é a necessidade de se ter uma alocação ótima ou pelo menos de boa qualidade.

Uma ferramenta de descrição que evidenciou a importância de algoritmos eficientes na alocação de estados em ambientes de síntese lógica foi a linguagem VHDL [5], que surgiu em 1987 e teve algumas modificações em 1993. Isto pode ser verificado pelo aumento significativo de publicações [6] a respeito no mesmo período como mostra a Figura 1.3. Pelo fato da VHDL descrever sistemas em altos níveis de abstração, coube às ferramentas de síntese a tarefa de obter a melhor alocação de estados para o sistema, tendo sempre em vista as características da tecnologia alvo (componente a ser gravado no caso de dispositivos programáveis) [7].

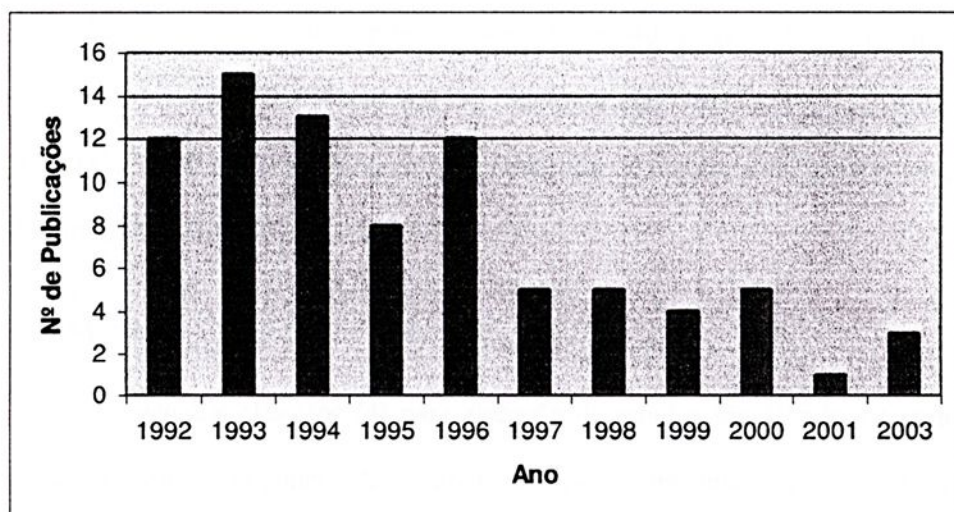


Figura 1.3: Artigos Publicados sobre alocação de estado na última década.

Em 1996 verifica-se outro pico na Figura 1.3, sendo que estas publicações estão, na sua maioria, relacionadas com dissipação de energia das máquinas seqüenciais e buscam uma alocação que diminua a atividade de troca de informação nos elementos de memória, fazendo com que o consumo de energia seja minimizado.

Estudos que buscam a minimização da dissipação de energia, provavelmente foram motivados, devido ao grande avanço da telefonia celular, computadores portáteis, GPS (*Global Positioning System*) entre outros equipamentos eletrônicos que necessitam de uma grande autonomia de energia.

Apesar da alocação de estados ser um ponto crucial na minimização dos circuitos combinacionais de uma máquina de estados, o desenvolvimento de ferramentas de síntese digital que executam esta fase do projeto de síntese não tem recebido a devida atenção nos últimos anos, como mostra a Figura 1.3.

Um fator que contribuiu para a qualidade das soluções do problema de alocação de estados, foi o avanço computacional, com processadores de alto desempenho, pode-se verificar uma grande quantidade de soluções em pequenos intervalos de tempo quando comparado com computadores da década de 80.

Nos últimos dez anos este problema foi abordado por vários algoritmos de busca heurística como *Simulated Annealing* [3], [8], Algoritmo Genético [1], [2], [9] Decomposição [10], [11] e Grafos [12] e por alguns programas que se tornaram clássicos como NOVA [13], Mustang [14] e JEDI [8].

1.3. Métodos Clássicos de Alocação

Os métodos clássicos [4] para alocação de estados são algoritmos muito simples e rápidos de serem implementados, embora não produzam uma boa alocação para máquinas relativamente pequenas.

Apresentam-se a seguir as características de alguns métodos clássicos de alocação usualmente empregadas em ambientes de síntese digitais.

1.3.1. Simplest

No *Simplest* a alocação de estados é feita de forma aleatória, usando o número mínimo de variáveis necessário para representar os estados.

1.3.2. Decomposed

Na alocação de estados pelo *Decomposed* o bit mais significativo indica se o estado é inicial ou não, os bits intermediários indicam a condição para que a saída seja 1 no estado atual e o bit menos significativo representa o valor do estado anterior.

1.3.3. One-Hot

Esta alocação utiliza mais que o número mínimo de variáveis de estados, isto é, usa um bit por estado. Além de ser simples, uma vantagem da alocação *one-hot* é que produz pequenas equações de excitação, já que apenas um elemento de memória tem valor 1 em cada estado. Uma desvantagem óbvia da alocação *one-hot*, especialmente para máquinas com muitos estados, é que desta forma é necessário muito mais que o número mínimo de elementos de memória.

1.3.4. Almost One-Hot

Neste tipo de alocação usa-se a alocação “no-hot” no estado inicial, com todos os bits iguais a zero e nos demais estados a alocação é *one-hot*, tendo duas características: é fácil inicializar a máquina e depois que a máquina está em andamento ela não volta ao estado inicial.

Como exemplo das alocações descritas nos itens 1.3.1, 1.3.2, 1.3.3 e 1.3.4, considere-se a tabela de próximo estado para a máquina de estados M_1 , apresentada na Tabela 1.1, sendo Z a variável de saída.

Tabela 1.1: Tabela de Próximo Estado da Máquina M_1 .

Estados	Entradas				Z
	00	01	11	10	
A	B	B	C	C	0
B	D	D	C	C	0
C	B	B	E	E	0
D	D	D	E	B	1
E	B	D	E	E	1

Na Tabela 1.2 é apresentada às alocações para a máquina M_1 , para cada um dos métodos definidos nos itens 1.3.1, 1.3.2, 1.3.3 e 1.3.4. As Q_i variáveis representam as variáveis de estado.

Tabela 1.2: Tabela de alocação de estados para a máquina M_1 .

Estados	Simplest	Decomposed	One-Hot	Almost One-Hot
	Q1-Q3	Q1-Q3	Q1-Q5	Q1-Q4
A	000	000	00001	0000
B	001	100	00010	0001
C	010	101	00100	0010
D	011	110	01000	0100
E	100	111	10000	1000

1.4. Outros Métodos de Alocação

Existem também outros algoritmos que realizam a alocação de estados de forma mais eficiente que os métodos clássicos e que são bastante explorados pela literatura.

1.4.1. Algoritmo Genético

Na teoria da evolução natural proposta por Darwin, os indivíduos de uma determinada espécie se tornam cada vez mais adaptados a certas adversidades do meio a cada geração, pois somente os mais adaptados às novas condições transmitem suas características aos seus descendentes.

O Algoritmo Genético [15] busca equiparar-se ao máximo com a evolução natural, dessa forma têm-se três fatores fundamentais: cruzamentos entre indivíduos (crossover), mutação e quantidade de indivíduos da população.

Os vetores-solução representam os cromossomos e a cada iteração há uma fusão de soluções; (crossover). Os resultados desta fusão dão origem à nova geração de soluções, nesta fase as soluções sofrem alterações aleatórias denominadas mutação. A taxa de mutação, os pontos de fusão, e o número de descendentes, dependem do problema abordado.

O algoritmo genético usa uma população de indivíduos, que nos problemas combinatórios significam um conjunto de configurações, para resolver um problema de otimização complexo. Portanto, o algoritmo genético deve atender os seguintes aspectos:

1. Representar adequadamente uma configuração do problema. A representação mais comum é a codificação binária, pois são facilmente simulados os operadores genéticos de recombinação e mutação;
2. Deve encontrar uma forma adequada para avaliar a função objetivo ou seu equivalente (*fitness value*). Deste modo, as melhores configurações são aquelas que apresentam funções objetivo de melhor qualidade;

3. Deve existir uma estratégia de seleção das configurações com direito a participar na formação das configurações da nova população;
4. Deve existir um mecanismo que permita implementar o operador genético de recombinação;
5. Deve existir um mecanismo que permita implementar o operador genético de mutação. Mutação geralmente é considerada um operador genético secundário nos algoritmos genéticos, mas pesquisas recentes [16] mostram que este operador é muito mais importante do que se imaginava inicialmente;
6. Deve-se especificar o tamanho da população, isto é, o número de configurações em cada geração.

Uma vez especificados todos os aspectos anteriormente mencionados para um problema específico, diz-se, então, que foi implementado um algoritmo genético básico.

Existem dois tipos básicos de seleção: a proporcional e a torneio.

Na seleção proporcional cada configuração tem o direito de gerar um número de descendentes que é proporcional ao valor de sua função de adaptação. O esquema de seleção proporcional tem duas partes fundamentais e com funções muito específicas:

- Determinação do número de descendentes que é proporcional ao valor da função objetivo;
- Integralização desses valores não inteiros, usando um método randômico.

Na seleção torneio os descendentes são escolhidos realizando n jogos, sendo n o tamanho da população. Em cada jogo são escolhidos aleatoriamente k configurações e a configuração ganhadora do jogo é aquela que tem função de adaptação de melhor qualidade.

Na literatura especializada existem ainda outros métodos de seleção. Por exemplo, o método genitor [17] que faz uma substituição parcial da população. Neste conceito, aparece o conceito de elitismo que tenta preservar as configurações de melhor qualidade.

1.4.2. Simulated Annealing

Annealing é uma das técnicas usadas pelos físicos na construção de cristais perfeitos. Nessa técnica um material é aquecido até uma temperatura elevada e depois esfriado lentamente, mantendo durante o processo o chamado quase equilíbrio termodinâmico. O processo pára quando o material atinge seu estado de energia mínima na qual se transforma num cristal perfeito. Assim, o algoritmo de simulated annealing tenta simular um processo equivalente para encontrar a configuração ótima de um problema complexo.

No processo de “annealing” [3], [18] o material sólido é aquecido a temperaturas elevadas, quando assume uma estrutura atômica altamente desordenada. Posteriormente, o material é resfriado lentamente, mantendo o quase-equilíbrio térmico, até atingir o estado de mínima energia quando o material assume a estrutura de um cristal perfeito.

Este processo de “annealing”, desenvolvido pelos físicos para a construção de cristais perfeitos, serviu como base para desenvolvimento de um algoritmo com capacidade de resolver problemas do campo da pesquisa operacional.

A grande vantagem deste método combinatorial é que ele tem a capacidade de resolver problemas complexos de grande porte e, por isso, nos últimos anos, este processo foi utilizado para resolver o problema de alocação de estados, devido ao aumento significativo da complexidade das máquinas durante as últimas décadas [8].

1.4.3. NOVA

O programa NOVA [13] surgiu em 1990, e desde então é utilizado como referência para avaliação de outros programas. A alocação de estados utilizada pelo NOVA, como a maioria dos programas, separa em grupos os estados que sob uma determinada entrada produz



o mesmo próximo estado e saída (restrição de entrada). Estados que são próximos estados de um predecessor comum são denominados adjacentes em um k-cubo (restrição de saída).

Quando este programa é executado, pode-se optar por algumas estratégias de alocação:

Greedy: Realiza a alocação de estados satisfazendo o maior número possível de restrições de entrada. Quando a opção *greedy* é usada, o NOVA analisa estritamente as soluções que melhoram a alocação, mesmo as que já foram examinadas;

Hybrid: Faz a alocação de estados baseado nas restrições de entrada. Porém o NOVA não analisa somente as soluções que melhoram o resultado, isso permite que o NOVA saia de ótimos locais no início do processo;

I/O Hybrid: Similar ao híbrido, exceto pelo fato de tentar satisfazer restrições de entrada e saída. Desta forma obtém-se resultados melhores que o híbrido;

Exact: Obtém a melhor codificação que satisfaz as restrições de entrada. Entretanto as restrições de saída não são consideradas;

Input Annealing: Similar ao híbrido, a diferença está no fato de utilizar a teoria do *Simulated Annealing*, um método mais sofisticado para obter uma alocação de estados de boa qualidade;

One-Hot: Emprega a codificação *one-hot*. Este, raramente produz o mínimo de equações, mas pode reduzir drasticamente a complexidade das equações;

Random: Usa um método aleatório para gerar a codificação. O NOVA irá gerar um número específico de alocações aleatórias e produzirá um relatório com a melhor alocação.

Como exemplo de alocação de estados realizada pelo NOVA, considere a Tabela 1.3 onde é apresentada a tabela de próximo estado de um semáforo [19].

Tabela 1.3: Tabela de Próximo Estado do Semáforo.



Entradas			Estado Atual	Próximo Estado	Saídas	
E ₂	E ₁	E ₀	Q ₁ Q ₀	P ₁ P ₀	H ₁ H ₀	F ₁ F ₀
0	X	X	HG	HG	00	10
X	0	X	HG	HG	00	10
1	1	X	HY	HY	01	10
X	X	0	FG	HY	01	10
X	X	1	FG	FG	10	10
1	0	X	FG	FG	10	00
0	X	X	FG	FY	10	00
X	1	X	FG	FY	10	00
X	X	0	FY	FY	10	01
X	X	1	FY	HG	10	01

Os estado simbólicos HG, HY, FG, FY representam os estados auto-pista verde/rodovia vermelho, auto-pista amarelo/rodovia vermelho, auto-pista vermelho/rodovia verde e auto-pista vermelho/rodovia amarelo, respectivamente. Nas saídas H₁H₂ são as variáveis do semáforo voltado para auto-pista e F₁F₂ são as variáveis do semáforo voltado para a rodovia secundária e E₂, E₁ e E₀ são variáveis de entrada.

As alocações obtidas para o semáforo, com cada uma das possíveis opções oferecida pelo NOVA é apresentada na Tabela 1.4.

Tabela 1.4: Alocação de estado do semáforo obtidas pelo NOVA.

	HG	HY	FG	FY
Greedy	00	11	01	10
Hybrid	00	11	10	01
Exact	11	10	01	00
IO Hybrid	00	01	11	10
I annealing	01	10	11	10
Random	11	00	01	10

1.4.4. Mustang

O Mustang realiza a alocação de estados através de um algoritmo heurístico que maximiza o número de cubos comuns na codificação do circuito e então minimiza o

número de literais resultando em um circuito combinacional de multi-níveis otimizado. O Mustang possui cinco métodos alternativos de alocação:

Random: Realiza uma codificação aleatória;

Sequential: Aloca os estados na ordem sequencial do código binário;

One-Hot: Faz a alocação *one-hot*;

Fan-in: Trabalha com a entrada e o *fan-in* de cada estado. Próximos estados produzidos pelas mesmas entradas de um conjunto similar de estados presentes, são codificados com binários adjacentes. A alocação é escolhida de forma a maximizar o número de subexpressões comuns nas funções de próximo estado;

Fan-Out: Trabalha com a saída e o *fan-out* de cada estado, sem considerar as entradas. Estados atuais com mesma saída que produz conjuntos de próximos estados similares recebem codificação binária adjacente. Então, maximiza-se o número de subexpressões comuns nas funções de próximo estado e nas funções de saída. Este método tem melhores resultados para máquinas sequenciais com muitas saídas e poucas entradas.

Como exemplo, tem-se o semáforo apresentado na Tabela 1.3. A alocação obtida para cada uma das possíveis opções do Mustang encontra-se na Tabela 1.5.

Tabela 1.5: Alocação de estado para o semáforo obtido pelo Mustang.

	HG	HY	FG	FY
Random	01	10	11	00
Sequential	01	10	11	00
Fan-in	00	01	10	11
Fan-out	10	11	00	01

1.4.5. Métodos de Decomposição

Muitas vezes trabalha-se com máquinas de grande porte e fazer alocação dessas máquinas tem um custo computacional elevado, descartando uma quantidade enorme de soluções que compromete o processo de otimização. Fundamentado neste caso, utiliza-se algoritmos de decomposição [11], que divide máquinas complexas em máquinas pequenas e mais simples, facilitando assim a alocação de estados.

O que diferencia os métodos de alocação de estados por decomposição é a forma como ela é feita e levando em consideração a interdependência das máquinas resultantes.

Por exemplo, pode-se dividir o grafo [12] de estado original em muitas partes, com cada parte sendo implementada como uma máquina separada com um estado de espera, então, exatamente uma máquina está ativa em qualquer instante enquanto as outras esperam no estado *reset*. Neste caso, a decomposição é vista como a ligação de duas partições disjuntas. Um outro método de decompor uma máquina sequencial é baseado na fatoração do grafo de estado original que divide a máquina em duas máquinas menores e, este caso pode ser visto como encontrar duas partições ortogonais do conjunto de estados da máquina original [20].

Nos capítulos II e III, apresenta-se os algoritmos para alocação de estados propostos por Kohavi [20] e Nichols [21] respectivamente. Embora estes algoritmos sejam da década de 60, a maioria dos algoritmos propostos atualmente utilizam-se de suas propriedades.

Devido a esta importância, estes algoritmos foram estudados, dando origem aos programas KADE [22] e Nichols[23].



CAPÍTULO II - O MÉTODO DE KOHAVI

Neste capítulo apresenta-se o algoritmo de Kohavi para a alocação de estados. O algoritmo decompõe os estados de uma FSM em subconjuntos que possuem a Propriedade de Substituição e a de Saídas Consistentes.

A Propriedade de Substituição (P.S.) e a de Saídas Consistentes (S.C.) apresentadas por Kohavi [20], embora sejam da década de 60, ainda são utilizadas por vários algoritmos contemporâneos [24], [25], [26], que se baseiam nessas propriedades para realizar suas alocações.

Buscando compreender estas propriedades fundamentais à alocação de estados, implementou-se em linguagem de programação C o algoritmo de Kohavi, do qual se originou o programa KADE (*Kohavi Algorithm Decomposition*).

2.1. Propriedades para Decomposição de uma Máquina de Estados

O Método de Kohavi é aplicado a máquinas de estados descritas pelo modelo de Mealy e completamente especificadas, sendo as funções de saídas destas máquinas dependentes das variáveis de estado atual e das variáveis de entrada.

A alocação de estados de uma máquina M , realizada pelo método de Kohavi [20] é baseado na decomposição do conjunto de estados S , ou seja, na partição da máquina, sendo o conceito de partição dada pela seguinte lei de formação:

$$\pi = \{M_1, M_2, \dots, M_r\} \text{ com } \bigcup_{p=1}^r M_p = S \text{ e } \bigcap_{p=1}^r M_p = \emptyset,$$

sendo, M_p subconjuntos de S e r a quantidade de subconjuntos de S na partição π . Os subconjuntos M_p de π são denominados de blocos.



De acordo com a teoria apresentada, as equações que controlam uma máquina de estados finitos têm uma dependência reduzida das variáveis de estado quando a Propriedade de Substituição e a de Saídas Consistentes são consideradas. São essas duas propriedades que particionam o conjunto S , sendo que a Propriedade de Substituição dada pela definição I, e a de Saídas Consistentes pela definição II.

Definição I

Uma partição π de um conjunto de estados de uma máquina seqüencial M é dita ter a Propriedade de Substituição ou Fechamento em relação a M , se para quaisquer dois estados S_i e S_j pertencentes ao mesmo bloco de π e para qualquer entrada I , tem-se os próximos estados $\delta(I_a, S_i)$ e $\delta(I_a, S_j)$ contidos em um bloco em comum de π (δ representa a função de próximo estado conforme definido na seção 1.1) [20].

$$\pi = \{S_i, S_j \in M_p \mid \delta(I_a, S_i) \text{ e } \delta(I_a, S_j) \in M_p\}, \forall p = 1, 2, \dots, r,$$

sendo, M_p blocos de S , δ a função de próximo estado, I_a uma condição de entrada e r a quantidade de blocos.

Definição II

Uma partição τ de uma máquina seqüencial M é dita ter a propriedade de saídas consistentes, se para todos os blocos de τ , os estados contidos em cada bloco possuem saídas iguais [20].

$$\tau = \{S_i, S_j \in M_p \mid \lambda(I_a, S_i) = \lambda(I_a, S_j)\}, \forall p = 1, 2, \dots, r,$$

sendo, M_p blocos de S , λ a função de saída, I_a uma condição de entrada e r a quantidade de blocos.

Teorema 2.1:

Uma máquina seqüencial M com n estados tem $s = \lceil \log_2 n \rceil$ variáveis de estado que podem ser divididas em duas partes, tal que as primeiras k variáveis $1 \leq k < s$, e as outras $s - k$ variáveis podem ser calculadas independentemente, se e somente se existem duas partições

não triviais π_1 e π_2 com P.S. (Propriedade Substituição) ou S.C. (Saídas Consistentes) que satisfazem as seguintes condições:

- 1) Qualquer bloco de π_1 tem no máximo um elemento em comum com qualquer um dos blocos de π_2 ($\pi_1 \cdot \pi_2 = 0$);
- 2) $[\log_2 \#(\pi_1)] + [\log_2 \#(\pi_2)] = s$ (o símbolo # significa a cardinalidade do conjunto).

A operação representada pelo símbolo $(\cdot)$, determina a interseção entre cada bloco de π_1 e todos os blocos de π_2 , resultando em um bloco da partição solução. O processo se repete para todos blocos de π_1 . Essa operação é definida como:

$$\pi_1 \cdot \pi_2 = \{a_i \cap b_j\},$$

sendo, a_i e b_j blocos das partições π_1 e π_2 respectivamente.

Denomina-se partição trivial a partição que possui apenas blocos unitários ou a partição que possui apenas um bloco igual ao conjunto S de estados.

Para calcular as partições de uma máquina seqüencial M , com conjunto de estados S e entradas I , tenta-se $s_n(s_n+1)/2$ pares distintos de estados para determinar se não existe somente a partição trivial [20]. Sendo s_n o número de estados da máquina de estados.

Para determinar todas as partições com P.S. de M , escolhe-se dois estados quaisquer S_i e S_j e verifica se os seus próximos estados para cada entrada. Identifica-se o par $\delta(I_k, S_i)$ e $\delta(I_k, S_j)$. Se estes conjuntos são disjuntos, então têm-se dois pares identificados, caso contrário, deve-se aplicar a propriedade transitiva e repetir o processo com o par $\delta(I_k, S_i)$ e $\delta(I_k, S_j)$. O processo se repete até que todos os estados estejam identificados.

O exemplo 2.1 ilustra como determinar as partições de uma máquina de estados pelo processo descrito anteriormente.

Exemplo 2.1:

Considere a máquina seqüencial M_2 descrita pelo modelo de Mealy apresentada na Tabela 2.1:

Tabela 2.1: Tabela de Próximo Estado da Máquina M_2 .

Estados	Entrada		Saídas	
	0	1	0	1
A	D	C	1	0
B	F	C	0	1
C	E	B	0	0
D	B	E	1	0
E	A	D	0	1
F	C	D	1	0

Inicialmente identifica-se um par de estados aleatório, por exemplo, o par (A, B). A entrada em nível baixo identifica os próximos estados (D, F). Como a entrada com sinal em nível alto identifica somente o estado C, esta não é considerada, pois o sinal em nível baixo identifica um número maior de estados. Assim, os pares de estados (A, B) e (D, F) são disjuntos, então é adicionado o bloco (D, F) à partição.

Repetindo o processo para (D, F), obtêm-se os pares (B, C) e (E, D), desta vez, pela Lei Transitiva obtêm-se os estados (A, B, C) e (D, E, F), e desta forma, todos os estados estão identificados formando a partição $\pi_1 = \{(A, B, C); (D, E, F)\}$ com a Propriedade de Substituição.

Pode-se também escolher o par de estados (A, F), sendo que a entrada com sinal em nível baixo tem-se os estados (D, C) que, por sua vez, independentemente da entrada tem-se os estados (E, B) e repetindo o processo, identifica-se todos os estados. Então gera-se a partição $\pi_2 = \{(A, F); (B, E); (C, D)\}$ com P.S.

Saídas Parcialmente Independentes

Quando o número de variáveis de estado para distinguir os blocos das partições π e τ é maior que o número mínimo necessário, ou seja, $\log_2 \#(\pi) + \log_2 \#(\tau) > s$, não se produz uma alocação desejável para uma máquina seqüencial com s estados.

Para contornar este problema, introduz-se o conceito de independência parcial das saídas, na qual deve-se formar partições τ com S.C., verificando-se cada entrada independentemente formando τ_i partições, fazendo a permutação das n partições na operação $\tau_{ij} = \tau_i \cdot \tau_j$. A permutação que tiver o menor número de blocos será utilizada.

Dependência Parcial das Variáveis de Estado

Existem máquinas que não possuem uma partição P.S. não-trivial quando se considera todas as entradas da máquina. Neste caso, utiliza-se o conceito de dependência parcial das variáveis de estado, que desconsidera n entradas da máquina para obter a partição com P.S. sendo, $n = 1, \dots, r-1$, em que r é a quantidade de condições de entradas da máquina.

Partições obtidas a partir deste conceito geram uma alocação que determina equações de controle que dependem parcialmente de outra variável além da que foi designada para distinguir entre os blocos.

2.2. Condições para Alocação de Estados pelo Método de Kohavi

Para minimizar o circuito combinacional de uma máquina seqüencial não se utiliza uma partição π qualquer com P.S. ou τ com S.C.. Para obter a alocação completa da máquina deve-se observar as condições apresentadas nos Caso I, Caso II e Caso III, discutidos a seguir.

Os problemas de minimização dos circuitos combinacionais que controlam os elementos de memória são contraditórios aos problemas de minimizar os circuitos de saída de uma máquina de estados, o que dificulta a minimização da máquina como um todo.

Não deve haver uma partição com as propriedades P.S. e S.C. simultaneamente, portanto, a máquina deve estar com seus estados reduzidos.

Caso I: $\pi \cdot \tau = 0$ e $\log_2 \#(\pi) + \log_2 \#(\tau) = s$

Trata-se do caso mais simples, sendo que as primeiras k variáveis são independentes das $s - k$ variáveis. A saída do circuito depende somente das variáveis $s - k$ e das entradas.

Caso II: $\pi \cdot \tau = 0$ e $\log_2 \#(\pi) + \log_2 \#(\tau) > s$

Neste caso pode-se alocar os estados somente em função de π ou τ . Fazer a alocação usando os dois conjuntos, implica em se ter um número de variáveis de estado igual a $\log_2 \#(\pi) + \log_2 \#(\tau)$, que é maior que a quantidade mínima de variáveis necessárias, podendo, assim, tornar o circuito mais complexo. Utiliza-se, então, a propriedade de Saídas Parcialmente dependentes.

Caso III: $\pi \cdot \tau \neq 0$ e $\log_2 \#(\pi) + \log_2 \#(\tau) < s$

Neste caso, deve-se encontrar uma partição τ_1 tal que $\pi \cdot \tau \cdot \tau_1 = 0$ e $\log_2 \#(\pi) + \log_2 \#(\tau) + \log_2 \#(\tau_1) = s$. Sendo a obtenção de τ_1 possível, tem-se saídas dependentes somente das variáveis designadas aos blocos de τ e, as variáveis designadas aos blocos de π serão independentes de τ e τ_1 , enquanto as variáveis designadas a τ_1 dependerão de todas as outras.

2.3. O Algoritmo de Kohavi

O algoritmo de Kohavi consiste nos seguintes passos:

Passo 1: Encontre as partições π com P.S. e τ com S.C.;

Passo 2: Verifique se $\pi \cdot \tau = 0$ e $\log_2 \#(\pi) + \log_2 \#(\tau) = s$. Se verdade vá ao passo 3. Se $\pi \cdot \tau = 0$ e $\log_2 \#(\pi) + \log_2 \#(\tau) > s$ veja o caso II e vá ao passo 3, ou se $\pi \cdot \tau \neq 0$ e $\log_2 \#(\pi) + \log_2 \#(\tau) < s$ proceda como no caso III e vá ao passo 3;

Passo 3: Escolha as variáveis para distinguir entre os blocos de π (i.e. y_k), e as variáveis para distinguir entre os blocos de τ (i.e. y_j).

Os estados no mesmo bloco de π e τ terão o mesmo valor y_k e y_j respectivamente.

O exemplo 2.2 mostra com detalhes a aplicação do Algoritmo de Kohavi para a alocação de estados.

Exemplo 2.2

Considere a máquina seqüencial M_3 , descrita pelo modelo de Mealy apresentada na Tabela 2.2:

Tabela 2.2: Tabela de Próximo Estado de M_3 .

Estados	Entrada		Saídas	
	0	1	0	1
A	B	D	1	0
B	A	C	0	1
C	D	A	0	1
D	C	B	1	0

Para realizar a alocação de acordo com o algoritmo proposto por Kohavi seguem-se os passos descritos no item 2.3.

Passo 1:

Para a máquina M_3 é possível encontrar uma partição π com P.S. considerando todas as entradas da máquina.

Uma possível partição com P.S. para M_3 é $\pi = \{(A, B); (C, D)\}$.

A partição τ com S.C. é obtida colocando-se na mesma partição os estados que possuem saídas iguais para a mesma entrada.

Assim sendo, tem-se $\tau = \{(A, D); (B, C)\}$.

Passo 2: Verificar se: $\pi \cdot \tau = 0$

Como $\{(A, B); (C, D)\} \cdot \{(A, D); (B, C)\} = \{(A); (B); (C); (D)\}$, obtém-se uma partição apenas com blocos unitários e, por definição, conclui-se que $\pi \cdot \tau = 0$.

Da equação:

$$\log_2 \#(\pi) + \log_2 \#(\tau) = s$$

Tem-se,

$$s = 2, \#(\pi) = 2, \#(\tau) = 2$$

$$\log_2 2 + \log_2 2 = 2$$

A quantidade de variáveis necessárias para distinguir entre as partições de π e τ é igual à quantidade mínima de variáveis necessárias para representar o total de estados. Então se procede como no Caso I.

Passo 3: Determinar variáveis de estados para distinguir entre os blocos.

$\pi = \{(A, B); (C, D)\}$, para o bloco (A, B) designa-se “0” e para (B, C) designa-se “1”.

Esta escolha é aleatória;

$\tau = \{(A, D); (B, C)\}$, para o bloco (A, D) designa-se “0” e para (B, C) designa-se “1”,

esta escolha também é aleatória.

Realizando a alocação pelo método de Kohavi, tem-se a alocação apresentada na Tabela 2.3:

Tabela 2.3: Alocação dos Estados Máquina M_3 .

Estados	Kohavi	
	Y_1	Y_2
A	0	0
B	0	1
C	1	1
D	1	0

A partir da alocação apresentada na Tabela 2.3, obtém-se f_1 e f_2 como funções de próximo estado, e f_3 como função de saída. Os elementos de memória escolhidos são do tipo D, sendo Y_1 e Y_2 as variáveis de estado e x é a variável de entrada.

$$f_1(x, y_1, y_2) = x'y_1 + xy_1';$$

$$f_2(x, y_1, y_2) = x'y_2' + y_1'y_2' + xy_1y_2;$$

$$f_3(x, y_1, y_2) = x'y_2' + xy_2.$$

Verifica-se que as funções f_1 e f_3 dependem apenas da variável de entrada e de uma das variáveis de estado, desta forma tem-se um circuito simplificado.

2.4. O Programa KADE

Baseado no algoritmo proposto por Kohavi implementou-se o programa denominado KADE, uma variação deste algoritmo. Resultados comprovam que o programa implementado realiza a alocação de estados de forma mais eficiente do que os algoritmos utilizado pelo programa MAX+PLUS II da Altera [27].

No algoritmo do programa KADE, não são consideradas as saídas da máquina de estado, pois otimizar as funções de controle e as funções de saída torna o problema ainda mais complexo.

Na implementação deste programa utilizou-se estruturas de dados, em linguagem de programação C. As estruturas foram divididas em duas partes: uma que representa vários tipos de informações e outra que contém um campo com o endereço de outra estrutura.

Uma seqüência destas estruturas é denominada de listas encadeadas [28], [29]. Estas listas encadeadas possibilitam o armazenamento das informações de forma dinâmica por não possuir um tamanho inicial pré-determinado e por não ser necessário que haja memória contínua fisicamente. Além disso, quando não são mais necessárias podem ser apagadas, liberando memória para outras tarefas.

A estrutura descritora contém as informações sobre as listas como: último nó, primeiro nó e número de nós. Com essas informações armazenadas o programa torna-se mais eficiente, pois não é preciso percorrer a lista todas as vezes que estas informações forem necessárias.

O programa KADE é constituído de listas encadeadas onde os dados são armazenados e manipulados.



A Figura 2.1 apresenta o fluxograma do programa KADE, que ilustra o seu comportamento.

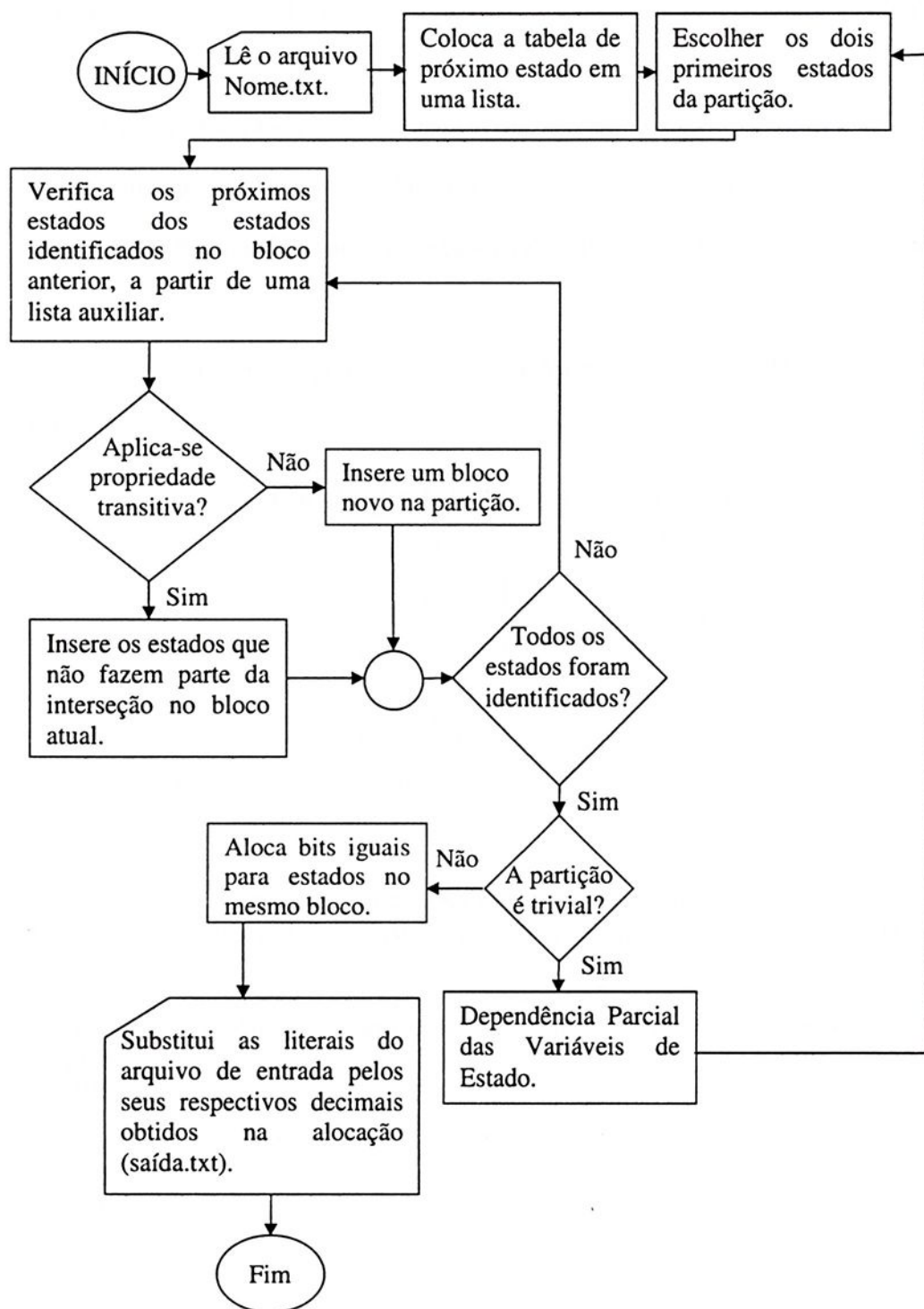


Figura 2.1: Fluxograma do Programa KADE.

Pode-se notar do fluxograma que, uma vez encontradas as partições, utiliza-se a quantidade mínima de variáveis de estado para diferenciar entre os blocos da partição. A determinação da variável para distinguir entre os estados no mesmo bloco é feita de forma aleatória.

Com a alocação realizada é gerado um arquivo de saída semelhante ao arquivo de entrada, onde os estados na forma literal são substituídos por decimais correspondentes à sua representação binária. O emprego do programa KADE é apresentado através do exemplo 2.3.

Exemplo 2.3:

Considere uma máquina de estados M_4 descrita pelo modelo Mealy, apresentada na Tabela 2.4.

Tabela 2.4: Tabela de próximo estado da máquina M_4 .

Estados	Entrada		Saída	
	0	1	0	1
A	B	D	1	0
B	A	C	0	1
C	D	A	0	1
D	C	B	1	0

A entrada do programa é um arquivo com extensão txt (nome.txt) que tem a seguinte disposição:

2 → representa o número de elementos de memória.
D → representa o tipo dos elementos de memória.
D
1 1 → representa o número de entrada e saída respectivamente.
A B 0 1 → Estado atual, próximo estado, entrada e saída, respectivamente.
A D 1 0
B A 0 0
B C 1 1
C D 0 0
C A 1 1
D C 0 1
D B 1 0
-100 → indica o final do arquivo.

O programa realiza a alocação e gera um arquivo com o seguinte formato:



2
D
D
1 1
0 1 0 1
0 2 1 0
1 0 0 0
1 3 1 1
3 2 0 0
3 0 1 1
2 3 0 1
2 1 1 0
-100

Nota-se que os estados na forma literal, descritos no arquivo de entrada, foram substituídos por decimais que representam a alocação realizada pelo Programa KADE, gerando um arquivo de saída com extensão tab (nome.tab), que é entrada do Programa Tabela [30].

O Programa Tabela gera a tabela de transição de uma máquina seqüencial a partir da tabela de próximo estado e minimiza as funções de transições internas, correspondentes aos elementos de memória utilizados, e as funções de saída do circuito, utilizando-se do algoritmo de *Quine-McCluskey* [31].

No capítulo III, apresenta-se o método de Nichols para alocação de estados, que utiliza a teoria dos conjuntos para decompor a máquina de estados. A implementação computacional deste método originou o programa Nichols.



CAPÍTULO III – O MÉTODO DE NICHOLS

Neste capítulo será abordado o algoritmo proposto por Nichols, que trata o problema de alocação de estado como um problema de minimização de funções booleanas. Para que isto seja possível algumas propriedades devem ser consideradas.

O método apresentado por Nichols [21] aplica-se somente a máquinas de Moore completamente especificadas.

Este método foi implementado para melhor compreender as suas propriedades, pois trata de uma maneira singular o problema da alocação de estados.

Para máquinas relativamente pequenas o desempenho do algoritmo proposto por Nichols é satisfatório quando comparado com os métodos utilizados pelo MAX + PLUS II da Altera.

O algoritmo baseia-se em duas propriedades fundamentais: *Output States* e Compatibilidade.

3.1. Output States

O algoritmo proposto por Nichols considera como entrada do circuito combinacional, as variáveis de estado da máquina e as variáveis de entrada do sistema. As entradas que produzem as mesmas saídas pertencem ao mesmo conjunto, que é chamado de *output states*.

Estes conjuntos podem ser representados em um Mapa de Karnaugh [32], onde cada *output state* posiciona-se na célula correspondente a entrada do circuito combinacional.

Para esclarecer estes conceitos considere o exemplo a seguir.



Exemplo 3.1

Uma máquina sequencial M_5 descrita pelo modelo de Moore é representada pelo diagrama de estados da Figura 3.1.

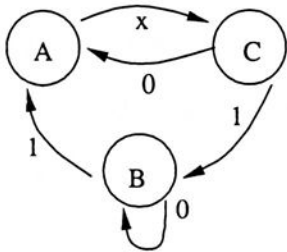


Figura 3.1: Diagrama de estados da máquina M_5 .

O agrupamento das entradas em *output states* é apresentado na Tabela 3.1, sendo E_0 a variável de entrada e, V_1 e V_0 representam os estados atuais sendo que A, B e C valem respectivamente 00, 01 e 10, que corresponde a alocação *simplest*.

Tabela 3.1: Tabela com os agrupamentos em *output states* da máquina M_5 .

Entrada			Estados
E_0	V_1	V_0	$V_1^+V_0^+$
0	0	0	C
0	0	1	B
0	1	0	A
1	0	0	C
1	0	1	A
1	1	0	B

A partir da Tabela 3.1 é possível obter as funções booleanas dos *output sets*, cujos conceitos são utilizados no item 3.2.

A Figura 3.2 apresenta a representação dos *output states* no Mapa de Karnaugh, que será utilizado para obter os custos do *output sets*.

		E_0			
V_0	A ⁵	7	3	B ¹	
	C ⁴	B ⁶	A ²	C ⁰	
		V_1			

Figura 3.2: Representação dos *output states* no mapa de Karnaugh para a máquina M_5 .

3.2. Compatibilidade e Output Sets

Um conjunto Q de estados é denominado *output set*, se cada estado do conjunto e de seu complemento Q' pode ser colocado nos vértices de $q-1$ subcubo (é dito subcubo de um k -cubo, um cubo de dimensão menor que o k -cubo²), sendo q o número mínimo de variáveis de estado necessário para alocar os estados. Pode-se definir um *output set* também em função do número de elementos do conjunto, sendo A um *output set* se, e somente se, a cardinalidade de Q e Q' é menor ou igual a 2^{q-1} [21].

Desse modo, as cardinalidades possíveis dos *output sets*, para uma máquina com s estados são:

$$s - 2^{q-1} \leq \#(Q) \leq 2^{q-1} \quad (3.1)$$

sendo que o símbolo $\#$ representa a cardinalidade de Q . O exemplo 3.2 ilustra como encontrar os *output sets*.

Exemplo 3.2

Encontrar a cardinalidade dos *output sets* da máquina M_5 apresentada na Figura 3.1.

Pela expressão 3.1 tem-se:

- $s = 3$, representa o número de estados da máquina;
- $q = 2$, quantidade necessária de variáveis de estado para alocar 3 estados;

Dessa forma tem-se a expressão:

$$3 - 2^{2-1} \leq \#(A) \leq 2^{2-1}$$

$$1 \leq \#(A) \leq 2$$

Então para uma máquina de estados que possui 3 estados obtêm-se 6 *output sets* combinando esses 3 estados em grupos de 1 e 2 elementos.

² K-cubo é um cubo de dimensão k . Cada vértice desse cubo possui um valor binário disposto de modo que cada vértice adjacente tenha um valor com distância de a *Hamming* igual a 1.

A partir do mapa de Karnaugh apresentado na Figura 3.2, pode-se minimizar as funções booleanas relacionadas aos *output sets* e obter seus custos.

Outra propriedade importante para o algoritmo de Nichols é a compatibilidade dos *output sets*, e é essa propriedade que garante a qualidade da alocação, evitando que estados diferentes tenham o mesmo valor binário, ou mesmo, que no resultado final alguns estados permaneçam sem nenhuma representação binária.

Dois ou mais *output sets* são ditos compatíveis se as condições 1, 2 e 3 descritas a seguir forem satisfeitas.

1. Nenhum par de conjuntos é complementar;
2. Todos os conjuntos e seus complementares podem ser colocados em q-cubo simultaneamente com cada *output set* ocupando $(q - 1)$ cubo e seu complementar ocupando o complementar do $(q - 1)$ cubo;
3. Dois ou mais conjuntos são compatíveis se a cardinalidade da interseção de todas as combinações destes conjuntos com seus complementares é menor ou igual a 2^{q-m} , sendo 2^m o número de combinações e q o número de variáveis de estado.

Tendo como exemplo os conjuntos A e B , então a condição 3 é satisfeita se, e somente se, $\#\{(A \cap B), (A \cap B'), (A' \cap B), (A' \cap B')\} \leq 2^{q-m}$.

O exemplo 3.3 ilustra casos de *output sets* que atendem somente algumas condições de compatibilidade.

Exemplo 3.3

Considere os *output sets* de uma máquina seqüencial com cinco estados (A, B, C, D, E).

1. Para $Q_1 = (A, B, C)$, $Q_2 = (D, E)$ e $Q_3 = (E)$, tem-se que os conjuntos cabem simultaneamente em um $(q-1)$ cubo, mas Q_1 e Q_2 são complementares e, portanto, não são compatíveis;
2. Para $Q_1 = (A, C)$, $Q_2 = (A, B, C)$ e $Q_3 = (E)$, a cardinalidade da intersecção desses conjuntos é maior que 2^{q-m} , ou seja, $\#(Q_1 \cap Q_2 \cap Q_3') = \#(A, C) = 2$, é maior que $2^{3-3} = 1$ e, portanto, não são compatíveis;
3. Para $A_1 = (A, C)$, $A_2 = (B, D)$ e $A_3 = (A, B, E)$, conclui-se que atende todas as condições de compatibilidade. Não há nenhum par de conjuntos complementares, todos cabem em um q -cubo simultaneamente e a cardinalidade da intersecção de todas as combinações entre os *output sets* e seus complementares são menores ou iguais a 2^{q-m} e, portanto, são compatíveis.

3.3. O algoritmo de Nichols para a Alocação de Estados

O algoritmo proposto por Nichols busca minimizar o custo das funções de saída de uma máquina de estados. Os passos do algoritmo para encontrar a alocação são os seguintes:

Passo 1: Identificar quais entradas do circuito combinacional geram as mesmas saídas e colocá-las no mesmo grupo gerando assim os *output states*, que são os estados da máquina;

Passo 2: Definir quantos elementos deve ter os *output sets* pela expressão:

$$s - 2^{q-1} \leq \#(A) \leq 2^{q-1}$$

sendo, s a quantidade de estados que a máquina possui e q o número mínimo de variáveis de estado necessárias para alocar s estados. Esta expressão garante que q conjuntos cabem simultaneamente em um q -cubo;

Passo 3: Pela equação:

$$N = \sum_{k=s-2^{q-1}}^{2^{q-1}} \binom{s}{k}$$

sendo, N quantidade de possíveis *output states*, s o número de estados da máquina e q o número mínimo de variáveis de estados necessário para representar s estados.

Pode-se determinar o número de *output states* possíveis formados com a quantidade de elementos calculados no passo 1 por:

$$\binom{s}{k} = 0, \text{ se } k < 0 \text{ ou } k > s;$$

Passo 4: Calcular os custos de todos os conjuntos, colocando os *output sets* no Mapa de Karnaugh e realizando a cobertura. Os valores dos custos são obtidos somando a quantidade de literais (SOP). Depois de calculado os custos de cada *output sets*, deve-se classificá-los em ordem crescente de acordo com os custos, para facilitar a execução do passo 5;

Passo 5: Dentre os *output sets* escolhe-se q conjuntos que são compatíveis e que possuem o menor custo possível;

Passo 6: Para realizar a alocação efetue a intersecção de todas as combinações entre os q *output sets* compatíveis e seus complementares, onde o complementar representa o “0” (nível baixo) e o próprio *output set* o “1” (nível alto). O estado resultante da intersecção recebe este binário na alocação.

O exemplo 3.4 mostra como encontrar a alocação de estados de acordo com o algoritmo de Nichols.

Exemplo 3.4

Uma máquina sequencial M_6 descrita pelo modelo de Moore é designada pela tabela de próximo estado apresentada na Tabela 3.2: Note que a saída do circuito é a própria alocação de estados.

Tabela 3.2: Tabela de Próximos Estados da máquina M_6 .

Estados	Entrada			
	00	01	10	11
A	A	B	D	C
B	B	C	A	D
C	C	D	B	A
D	D	A	C	B

Passo 1:

Combina-se os valores binários das entradas com uma alocação inicial dos estados, formando assim a Tabela Verdade do circuito combinacional da máquina.

A alocação inicial utilizada neste exemplo foi o *simplest* onde A, B, C e D valem respectivamente 00, 01, 10 e 11.

Tabela 3.3: Tabela verdade do circuito da máquina M_6 .

Entradas				Saída
E_1	E_0	Y_1	Y_0	$Y_1^* Y_0^*$
0	0	0	0	A
0	0	0	1	B
0	0	1	0	C
0	0	1	1	D
0	1	0	0	B
0	1	0	1	C
0	1	1	0	D
0	1	1	1	A
1	0	0	0	D
1	0	0	1	A
1	0	1	0	B
1	0	1	1	C
1	1	0	0	C
1	1	0	1	D
1	1	1	0	A
1	1	1	1	B

Deste modo, têm-se os seguintes *output states* para serem alocados: A, B, C e D.

Passo 2:

Determinar quantos elementos deve haver em cada *output set* para que possa ser colocado simultaneamente com seu complementar em um q-cubo, pela expressão.

$$s - 2^{q-1} \leq \#(A) \leq 2^{q-1}$$

Tem-se $s = 4$ e $q = 2$, então:

$$4 - 2^{2-1} \leq \#(A) \leq 2^{2-1}$$

$$2 \leq \#(A) \leq 2$$

Pode-se deste modo formar *output sets* com apenas dois elementos.

Passo 3:

Formar todos os *output sets* com o número de elementos calculados no passo 2. A quantidade de *output sets* é dado pela equação:

$$N = \sum_{k=s-2^{q-1}}^{2^{q-1}} \binom{s}{k} \Rightarrow N = \sum_{k=4-2^{2-1}}^{2^{2-1}} \binom{4}{k} \Rightarrow N = 6$$

Os seis grupos possíveis são: (A, B), (A, C), (A, D), (B, C), (B, D) e (C, D).

Passo 4:

Coloca-se os *output states* no Mapa de Karnaugh de acordo com a Tabela 3.3, e fazendo a cobertura mínima dos possíveis *output sets* encontra-se os custos de cada um.

A Figura 3.3 apresenta a disposição dos *output states* no Mapa de Karnaugh.

	E ₁				
Y ₁	B ¹⁰	A ¹⁴	D ⁶	C ²	Y ₀
	C ¹¹	B ¹⁵	A ⁷	D ³	
	A ⁹	D ¹³	C ⁵	B ¹	
	D ⁸	C ²	B ⁴	A ⁰	
	E ₀				

Figura 3.3: Máquina M₆ representada no Mapa de Karnaugh.

Considerando A e C como sendo os mintermos no Mapa Karnaugh da Figura 3.4 e fazendo a cobertura mínima do *output set* (A, C) tem-se:

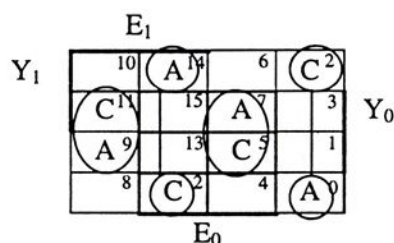


Figura 3.4: Cobertura mínima do *output set* (A, C).

A partir da cobertura mínima tem-se a função booleana do *output set* (A, C) e desta o seu custo em literais (SOP).

Repetindo o processo para os demais *output sets* tem-se os seguintes custos: $(B, D) = 9$, $(A, C) = 16$, $(A, B) = 20$, $(B, C) = 20$, $(A, D) = 24$ e $(C, D) = 36$.

Passo 5:

Escolhe-se q *output sets* que são compatíveis. Os conjuntos compatíveis que possuem menor soma dos custos são os conjuntos (A, C) e (A, B).

$$Q_0 \rightarrow (A, C) = 16$$

$$Q_1 \rightarrow (A, B) = 20$$

O custo total do circuito é 36 literais (SOP). Este custo não corresponde ao custo do circuito final da máquina de estado.

Passo 6:

Fazendo-se a interseção de todas as combinações possíveis entre Q_0 , Q_1 e seus complementares tem-se:

$$Q_0 = (A, C) \quad Q_0' = (B, D)$$

$$Q_1 = (A, B) \quad Q_1' = (C, D)$$

$$Q_1' \cap Q_0' = D \rightarrow 00$$

$$Q_1 \cap Q_0' = C \rightarrow 10$$

$$Q_1' \cap Q_0 = B \rightarrow 01$$

$$Q_1 \cap Q_0 = A \rightarrow 11$$

A Tabela 3.4 apresenta a alocação obtida pelo Algoritmo de Nichols.

Tabela 3.4: Alocação de estados para a máquina M_6 .

Estados	Nichols	
	Y_1	Y_2
A	1	1
B	0	1
C	1	0
D	0	0

A partir da alocação apresentada na Tabela 3.4, para a máquina M_6 , tem-se um circuito combinacional com o custo de 32 literais (SOP).

3.4. O Programa Nichols

Com o objetivo de criar um programa que realize a alocação de estados de forma mais eficiente do que os algoritmos utilizados pelo programa MAX + PLUS II da Altera, testar máquinas mais complexas e tornar a disseminação desta teoria mais atraente, implementou-se o algoritmo proposto por Nichols em linguagem de programação C, dando origem ao programa Nichols.

O programa desenvolvido utiliza o programa BOOM (*Boolean Minimizer*) [33] para minimizar as funções booleanas geradas a partir dos *output sets*.

O arquivo de entrada do programa Nichols possui a mesma sintaxe do arquivo de entrada do programa KADE e a mesma estrutura de dados como descrito no Capítulo II.

Para melhor entendimento do programa Nichols considere o fluxograma apresentado na.Figura 3.5

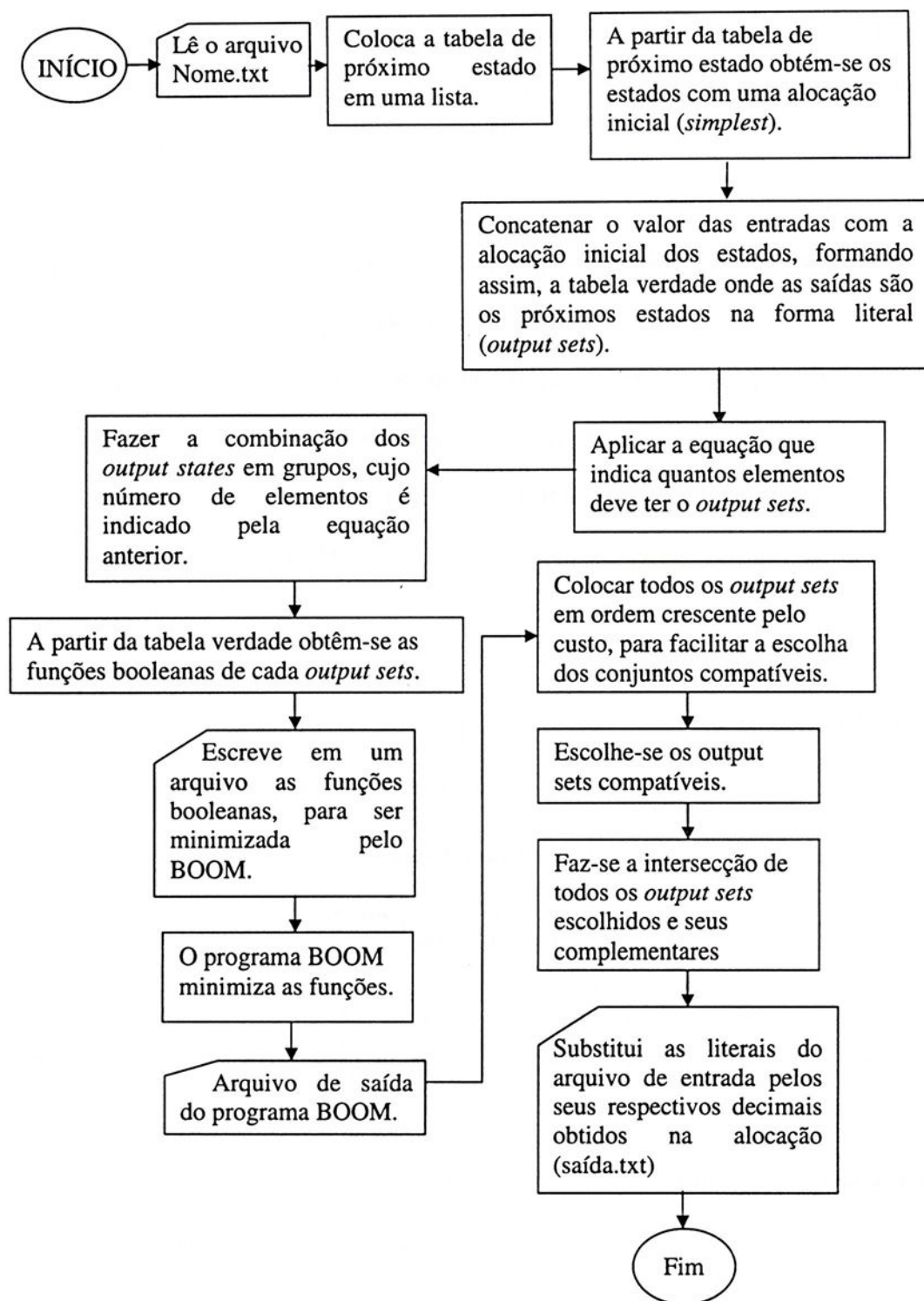


Figura 3.5: Fluxograma do programa Nichols.

Para que o programa Nichols funcione corretamente é necessário que os arquivos Gera_sets, BOOM, Compat e Nichols estejam no mesmo diretório (pasta) onde o programa será executado.

O programa Gera_sets, determina os *output states* e os *output sets*, gerando as funções booleanas. Estas funções são minimizadas pelo programa BOOM. A partir do arquivo de saída do programa BOOM é obtido o custo de cada função em número de literais (SOP), então o programa Compat coloca os output sets em ordem crescente por custo e escolhe os conjuntos compatíveis com os menores custos possíveis.

O arquivo Nichols tem extensão *bat*, a partir do qual os programas são executados e os arquivos temporários excluídos.

No exemplo 3.5 é apresentado o funcionamento do programa Nichols.

Exemplo 3.5

Considere a máquina de estados M_7 descrita pelo modelo de Moore, sendo sua tabela de próximo estado apresentada na Tabela 3.5.

Tabela 3.5: Tabela de próximo estado da máquina M_7 .

Estados	Entrada				Saídas
	00	01	10	11	
A	A	B	C	D	00
B	B	C	A	D	01
C	C	D	B	A	10
D	D	A	C	B	11

A Figura 3.6 apresenta o arquivo de entrada para o Programa Nichols, que representa a tabela de próximo estado da máquina de estados M_7 .

```

caso342 - Bloco ...
Arquivo  Editar  Formatar  Exibir
Ajuda
P
D
D
2
A A 0 0
A B 1 1
A C 3 2
A D 2 3
B B 0 1
B C 1 2
B A 2 0
B D 3 3
C C 0 2
C D 1 3
C B 2 1
C A 3 0
D D 0 3
D A 1 0
D C 2 2
D B 3 1
-100

```

Figura 3.6: Arquivo de Entrada do Programa Nichols.

Quando o programa é executado é exibido no terminal uma *interface*, onde o usuário deve digitar o nome do arquivo de entrada e o nome do arquivo de saída. A Figura 3.7 é um exemplo da tela inicial do programa Nichols.

```

Selecionar C:\WINDOWS\system32\...
*****
*****
*****
** Bem Vindo ao Programa **
**          NICHOLS          **
*****
*****
*****
digite o nome do arquivo de ENTRADA
maq?
digite o nome do arquivo de SAIDA
maq?_s_

```

Figura 3.7: Tela inicial do programa Nichols.

A alocação realizada pelo Programa Nichols é exibida ao término da execução do programa. Na Figura 3.8 encontra-se a alocação obtida pelo Programa Nichols.

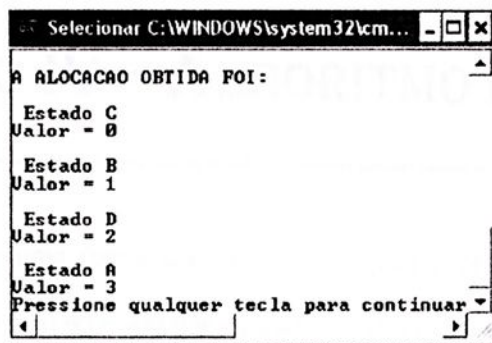


Figura 3.8: Relatório exibido pelo programa Nichols, contendo a alocação para a máquina M_7 .

O Programa Nichols gera o arquivo de saída apresentado na Figura 3.9, substituindo as literais pelos decimais de acordo com o relatório exibido na Figura 3.8.

```

2
0 0
0 0
2 2
3 3 0 0
3 2 1 1
3 1 3 2
3 0 2 3
2 2 0 1
2 1 1 2
2 3 2 0
2 0 3 3
1 1 0 2
1 0 1 3
1 2 2 1
1 3 3 0
0 0 0 3
0 3 1 0
0 1 2 2
0 2 3 1
-100

```

Figura 3.9: Arquivo de saída com alocação para a máquina M_7 .

Os valores binários possuem a quantidade mínima de variáveis de estados necessárias para representar os estados da máquina. No arquivo de saída do programa Nichols os valores binários para a alocação estão na representação decimal.

O arquivo de saída do programa Nichols tem extensão tab (nome.tab), que é o arquivo de entrada do programa Tabela que gera as equações de controle dos elementos de memória e as equações de saída.

Apresenta-se no Capítulo IV, o algoritmo proposto por Amaral [1], este transforma o problema de alocação de estados em um grafo completamente conectado, com pesos, porém sem orientação e utiliza-se de um algoritmo genético para encontrar uma solução de boa qualidade.

CAPÍTULO IV – ALGORITMO DE AMARAL

Na busca de uma formulação matemática para o problema de alocação de estados encontrou-se o algoritmo proposto por Amaral [1]. O algoritmo de Amaral divide-se em duas fases.

A primeira fase consiste na geração de um grafo completamente conectado, com arcos sem orientação, e com pesos nos arcos. A segunda fase consiste em um algoritmo genético que utiliza no seu critério de custo os pesos do grafo obtido na primeira fase.

4.1. Propriedades que Determinam os Pesos do Grafo

Os pesos dos arcos representam o quanto se deseja que dois estados tenham uma alocação com distância de *hamming*³ mínima. Portanto o valor dos pesos deve estar diretamente relacionado com as características e as propriedades da máquina.

As propriedades e características a serem consideradas são as seguintes: Sucessores, Predecessores, Saídas Consistentes e se a máquina é descrita pelo modelo de Mealy ou Moore.

- **Sucessores:** O conjunto dos sucessores de um estado S_k são seus próximos estados $\delta(I_j, S_k)$, sendo k uma constante e $j = 1, 2, \dots, n$, a quantidade de possíveis condições de entrada, sendo δ a função de próximo estado. A notação de conjunto dos sucessores de um estado S_k é $Suc(S_k)$.
$$Suc(S_k) = \{ \delta(I_j, S_k) \in S \mid \forall I_j \in I, S_k \in S \};$$
- **Predecessores:** O conjunto dos predecessores de um estado S_k é formado pelos estados S_j que sob a mesma condição de entrada I_a transitam para o estado S_k . A

³ Distância de *Hamming* é o número de *bits* diferentes entre duas representações binárias.



notação de conjuntos dos predecessores de um determinado estado sob um determinada condição de entrada é $Pred(S_k, I_a)$.

$$Pred(S_k, I_a) = \{S_j \in S \mid \forall S_j \in S, I_a \in I, \delta(I_a, S_j) = S_k\};$$

sendo, os índices a e k constantes, $j = 1, 2, \dots, n$ cardinalidade de S e δ a função de próximo estado.

- **Saídas parcialmente dependentes:** São estados que possuem as mesmas saídas para máquinas descritas pelo modelo de Moore, e para máquinas descritas pelo modelo de Mealy, são estados que sob uma determinada entrada possuem a mesma saída.

$$O_{Mealy} = \{S_j \in S \mid \forall S_j \in S, I_a \in I, O_k \in O, \lambda(I_a, S_j) = O_k\};$$

sendo, $\lambda: I \times S \rightarrow O$ a função de saída para máquinas de Mealy, a e k índices constantes e $j = 1, 2, \dots, n$ a cardinalidade de S .

$$O_{Moore} = \{S_j \in S \mid \forall S_j \in S, O_k \in O, \lambda(S_j) = O_k\};$$

sendo, $\lambda: S \rightarrow O$ a função de saída para máquinas de Moore, k um índice constante e $j = 1, 2, \dots, n$ a cardinalidade de S .

O exemplo 4.1. ilustra a aplicação das propriedades sucessores, predecessores e saídas consistentes.

Exemplo 4.1

Considere um detector de três zeros implementado através da máquina de estados M_8 , descrita pelo modelo de Moore, sendo sua tabela de próximo estado apresentado na Tabela 4.1.

Tabela 4.1: Tabela de próximo estado da máquina M_8 .

Estados	Entrada		Saída
	0	1	
A	B	A	0
B	C	A	0
C	D	A	0
D	B	A	1

A máquina de estados M_8 , possui os seguintes conjuntos com a propriedade sucessores, $Suc(A) = \{B, A\}$, $Suc(B) = \{C, A\}$, $Suc(C) = \{D, A\}$, $Suc(D) = \{B, A\}$ e com a propriedade de predecessores obtêm-se os conjuntos, $Pred(A, 1) = \{A, B, C, D\}$, $Pred(B, 0) = \{A, D\}$, $Pred(D, 0) = \{C\}$ e a partição com a propriedade saídas consistentes é $O_{Moore} = \{(D), (A, B, C)\}$.

4.2. Equação que Define os Pesos

Os pesos são empregados para determinar qual o desejo de se ter dois estados próximos na alocação. Estes pesos são determinados de acordo com as características da máquina que são representadas pelas propriedades de sucessores, predecessores e pelo seu modelo que pode ser de Moore ou Mealy.

Para formalizar a determinação dos pesos entre dois estados i e j , tem-se a equação abaixo:

$$P_{ij} = P_{ji} = R_1 \sum_{l=0}^{s-1} \alpha_{il} \alpha_{lj} \delta_{ij} + R_2 \sum_{a=0}^{c-1} \sum_{l=0}^{s-1} \beta_{lia} \beta_{lja} \delta_{ij} + R_3 M \sum_{b=0}^{c-1} \gamma_{ijb} \delta_{ij} + R_3 (1-M) \sum_{a=0}^{c-1} \sum_{b=0}^{v-1} \phi_{ijab} \delta_{ij} + R_4 (\alpha_{ij} + \alpha_{ji}) \delta_{ij}$$

sendo, c o número de condições de entrada, v é o número de variáveis de saídas, s é o número de estados, i e j são os estados nas extremidades do arco em questão. O comportamento das funções α , β , γ , δ e M são apresentados a seguir:

$$\alpha_{lm} = \begin{cases} 1 & \text{se } S_m \in Suc(S_l); \\ 0 & \text{caso contrário.} \end{cases}$$

$$\beta_{lma} = \begin{cases} 1 & \text{se } S_m \in Ped(S_l, I_a); \\ 0 & \text{caso contrário.} \end{cases}$$

$$\gamma_{ijb} = \begin{cases} 1 & \text{se } Z_b(S_i) = Z_b(S_j); \\ 0 & \text{caso contrário.} \end{cases}$$

$$\delta_{ij} = \begin{cases} 1 & \text{se } i \neq j; \\ 0 & \text{caso contrário.} \end{cases}$$

$$\phi_{ijab} = \begin{cases} 1 & \text{se } Z_b(S_i, I_a) = Z_b(S_j, I_a); \\ 0 & \text{caso contrário.} \end{cases}$$

$$M = \begin{cases} 1 & \text{para máquinas de Moore;} \\ 0 & \text{para máquinas de Mealy.} \end{cases}$$

A primeira parcela da equação refere-se aos pares de estados i e j , que têm um estado do conjunto de sucessores em comum. A segunda parcela se refere aos pares de estados que tem um predecessor em comum diante de uma determinada condição de entrada. A terceira e a quarta parcela referem-se aos pares de estados pertencentes à mesma partição com a propriedade de Saídas Consistentes para uma determinada variável de saída. O último termo indica se existe transição entre os estados i e j , este termo é usado como desempate entre pesos.

Os valores de R_1 , R_2 , R_3 e R_4 são constantes e valem respectivamente 3, 4, 2 e 1. Esses valores foram determinados por Amaral de acordo com a relevância de cada propriedade.

O Exemplo 4.2 ilustra como obter os pesos dos arcos do grafo.

Exemplo 4.2

Considere a máquina de estados M_8 descrita pelo modelo de Moore, o peso do arco AB será definido por:

$$P_{ij} = P_{ji} = R_1 \sum_{l=0}^{c-1} \alpha_{li} \alpha_{lj} \delta_{ij} + R_2 \sum_{a=0}^{c-1} \sum_{l=0}^{c-1} \beta_{lia} \beta_{lja} \delta_{ij} + R_3 M \sum_{b=0}^{c-1} \gamma_{ijb} \delta_{ij} + R_3 (1-M) \sum_{a=0}^{c-1} \sum_{b=0}^{c-1} \phi_{ijab} \delta_{ij} + R_4 (\alpha_{ij} + \alpha_{ji}) \delta_{ij}$$

Desenvolvendo os somatórios tem-se,

$$P_{AB} = P_{BA} = R_1 (\alpha_{AA} \alpha_{AB} \delta_{AB} + \alpha_{BA} \alpha_{BB} \delta_{AB} + \alpha_{CA} \alpha_{CB} \delta_{AB} + \alpha_{DA} \alpha_{DB} \delta_{AB}) + R_2 (\beta_{AA0} \beta_{AB0} \delta_{AB} + \beta_{BA0} \beta_{BB0} \delta_{AB} + \beta_{CA0} \beta_{CB0} \delta_{AB} + \beta_{DA0} \beta_{DB0} \delta_{AB} + \beta_{AA1} \beta_{AB1} \delta_{AB} + \beta_{BA1} \beta_{BB1} \delta_{AB} + \beta_{CA1} \beta_{CB1} \delta_{AB} + \beta_{DA1} \beta_{DB1} \delta_{AB}) + R_3 (\gamma_{AB0} \delta_{AB} + \gamma_{AB1} \delta_{AB}) + R_4 (\alpha_{AB} + \alpha_{BA}) \delta_{AB}$$

e calculando o valor de cada função tem-se,

$$P_{AB} = P_{BA} = 3(1 \cdot 1 \cdot 1 + 1 \cdot 0 \cdot 1 + 0 \cdot 1 \cdot 1 + 1 \cdot 1 \cdot 1) + 4(0 \cdot 0 \cdot 1 + 1 \cdot 0 \cdot 1 + 0 \cdot 0 \cdot 1 + 0 \cdot 0 \cdot 1 + 1 \cdot 1 \cdot 1 + 0 \cdot 0 \cdot 1 + 0 \cdot 0 \cdot 1 + 0 \cdot 0 \cdot 1) + 2(1 \cdot 1 + 0 \cdot 1) + 1(1 + 1)1$$

e deste modo o peso do arco $P_{AB} = P_{BA}$ é igual a 14.

Determinando os pesos de todos os arcos da máquina M_8 , tem-se uma matriz simétrica com a diagonal principal nula, como mostra a Tabela 4.2.

Tabela 4.2: Matriz de pesos dos arcos da máquina M_8 .

	A	B	C	D
A	0	14	10	12
B	14	0	7	5
C	10	7	0	5
D	12	5	5	0

Como cada estado representa um nó do grafo, sendo este completamente conectado, com arcos sem orientação e considerando a matriz de pesos apresentada na Tabela 4.2, tem-se o grafo da Figura 4.1.

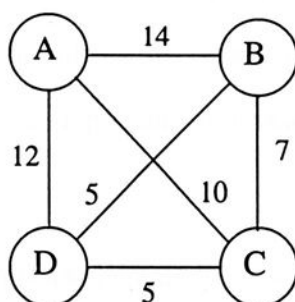


Figura 4.1: Grafo para a máquina M_8 .

Desta forma, o problema de alocação de estados, resume-se em encontrar uma alocação que minimize a função:

$$\text{Min } f = \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} \alpha_{ij} P_{ij},$$

sendo, s a quantidade de estados, α_{ij} e P_{ij} , a distância de *hamming* e o peso entre os estados i e j respectivamente.

Desta forma o problema da alocação de estados pode ser tratado facilmente através de um algoritmo genético, pois o custo de cada alocação pode ser calculado rapidamente através de f .

4.3. Algoritmo Genético proposto por Amaral

Como o problema da alocação de estados consiste em um problema NP-Completo, este é tratado de forma heurística. Sendo assim foi escolhido o algoritmo genético.

Conforme apresentado no item 1.4.1, do capítulo I, um algoritmo genético consiste nas seguintes fases: codificação, geração da população inicial, *crossover*, mutação, seleção e critério de parada.

Nesta seção são descritas as características básicas do algoritmo genético proposto por Amaral para cada uma dessas fases.

4.3.1. Codificação

A codificação no algoritmo de Amaral é feita através de uma matriz binária com s linhas e k colunas, sendo s a quantidade de estados da FSM e k a quantidade mínima de variáveis de estado necessárias para representar os estados da FSM.

4.3.2. Seleção

O operador seleção é utilizado para decidir quais os indivíduos que permanecerão na próxima geração e quais indivíduos vão gerar descendentes, ou seja, quem participa do *crossover*.

No algoritmo genético proposto por Amaral foi utilizado um tipo de seleção proporcional chamado roleta.

O método de seleção roleta tem uma seletividade pobre quando utilizada de forma tradicional. Para melhorar a seletividade deste método de seleção foi proposta a seguinte equação para determinar o desejo de um indivíduo ser selecionado.

$$R(I_k) = F(I_k) - (q + 1) F(\min) + q F(\max),$$

sendo, $R(I_k)$ valor utilizado na roleta para o indivíduo I_k , $F(I_k)$ o custo do indivíduo, $F(\min)$ o custo do indivíduo menos adaptado da população, $F(\max)$ o custo do indivíduo mais adaptado da população. A constante q é arbitrária, e é usada para determinar a seletividade da roleta.

A relação entre a possibilidade de escolher o melhor indivíduo $P(m)$ e a probabilidade de escolher o pior indivíduo $P(p)$ é dada por:

$$P(p) = \frac{q}{q+1} P(m),$$

No trabalho de Amaral o valor adotado para q é 0,01.

4.3.3. Crossover

O operador *crossover* é utilizado para fazer o cruzamento das informações entre dois indivíduos.

Para aplicar o *crossover* é escolhido inicialmente, um ponto aleatório. Neste ponto os indivíduos são divididos e suas partes são permutadas, gerando dois novos indivíduos.

Este tipo de cruzamento de informação pode gerar infactibilidade, ou seja, dois ou mais estados podem ter a mesma representação binária. Neste caso a informação do indivíduo menos adaptado é alterada.

A Tabela 4.3 ilustra a realização de um *crossover*. As duas primeiras colunas representam o pai #1 e a terceira coluna o pai #2.

Tabela 4.3: Aplicação do operador *crossover*.

	PAI #1	PAI #2	TRANSIÇÃO	FILHO
S ₀	000	001	001	001
S ₁	100	011	101	101
S ₂	010	010	010	010
S ₃	011	110	010	011
S ₄	001	100	000	000
S ₅	110	101	111	111

Pode-se verificar que na coluna de transição os estados S_2 e S_3 possuem o mesmo valor binário. Para eliminar esta infactibilidade foi alterado o valor que representa o pai #2, pois se considerou este menos adaptado.

4.3.4. Mutação

O operador mutação projetado para o SAP (*State Assignment Problem*) aplica uma sequência de operações de troca de informações, possibilitando dessa maneira que um valor binário se torne qualquer outro valor do espaço de busca.

A forma como Amaral realiza a mutação é apresentado no exemplo 4.2.

Exemplo 4.2

Considere-se a tabela contendo uma possível solução onde será aplicado o operador mutação.

Na Tabela 4.4 duas operações de troca são realizadas durante a mutação, uma entre S_1 e S_2 e a outra entre S_3 e S_5 .

Tabela 4.4: Exemplo de mutação, algoritmo de Amaral.

Estados	Antes da Mutação	Depois da Mutação
S_0	001	001
S_1	110	010
S_2	010	110
S_3	011	111
S_4	000	000
S_5	111	011

A população inicial é gerada aleatoriamente e o tamanho desta população depende do teste a ser realizado, pois não é determinada nenhuma formulação para esta.

O critério de parada é determinado pelo número de gerações que também depende do teste a ser realizado.

No capítulo V será apresentado o algoritmo para alocação de estados em FSM desenvolvido neste trabalho. Este possui duas fases, sendo a primeira baseada no grafo proposto por Amaral acrescida da propriedade substituição e a segunda em um algoritmo genético que possui outra metodologia em sua codificação e seus operadores.

O algoritmo implementado deu origem ao programa denominado de AGPS (Algoritmo Genético com Propriedade Substituição).

CAPÍTULO V – O PROGRAMA AGPS

Neste capítulo são propostas algumas alterações no grafo apresentado por Amaral. Também foi desenvolvido e implementado um algoritmo genético para realizar a alocação de estados.

As alterações realizadas no grafo e cada uma das fases do algoritmo genético proposto são descritas com detalhes neste capítulo.

5.1. Uma Deficiência no Grafo de Amaral

Uma máquina de estados pode ter seu circuito combinacional otimizado através da alocação de estados.

Neste processo de otimização busca-se minimizar a área e o consumo de energia através de uma distância *hamming* mínima entre dois estados, pois desta forma uma quantidade mínima de elementos de memória necessitam trocar a sua informação durante uma transição, consumindo menos energia e por outro lado uma quantidade menor de literais (SOP) são necessárias para controlar a troca de informação nos elementos de memória. Desta forma, minimiza-se a área do circuito combinacional da máquina de estados finitos.

Do exposto, surge uma questão: Quais estados devem ter prioridade nesta distância de *hamming* mínima?

Para solucionar este problema, Amaral propôs um grafo com pesos, no qual está determinado o desejo de se ter dois estados com a distância de *hamming* mínima.

Para determinar os pesos desse grafo são consideradas algumas propriedades que as máquinas de estados possuem. São elas: sucessores, predecessores e saídas consistentes. Estas propriedades e a geração do grafo foram discutidas no Capítulo IV.



Desta forma, o SAP consiste em minimizar a seguinte função:

$$\text{Min } f = \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} \alpha_{ij} P_{ij},$$

sendo, s a quantidade de estados, α_{ij} e P_{ij} , a distância de *hamming* e o peso entre os estados i e j respectivamente.

Entretanto, para algumas máquinas de estados, o valor da função f mínimo não significa que a quantidade de literais (SOP) também seja o mínimo, pois existem valores para f não ótimos que resultam em circuitos com a quantidade de literais (SOP) inferiores. Para sanar esta pequena deficiência do algoritmo de Amaral, introduz-se a propriedade substituição na equação que determinam os pesos dos arcos do grafo.

No exemplo 5.1 tem-se uma máquina de estados que apresenta esta característica.

Exemplo 5.1

Considere-se a máquina de estados M_9 , descrita pelo modelo de Mealy, apresentada pela Tabela 5.1.

Tabela 5.1: Tabela de Próximo Estado da máquina M_9 .

Estados	Entrada		Saída	
	0	1	0	1
A	D	B	0	0
B	A	C	0	1
C	B	A	0	0
D	A	E	0	0
E	D	A	0	1

A matriz de pesos dos arcos do grafo da máquina M_9 , é apresentada na Tabela 5.2

Tabela 5.2: Matriz de pesos da máquina M_9 .

	A	B	C	D	E
A	0	9	6	9	10
B	9	0	4	11	2
C	6	4	0	2	8
D	9	11	2	0	4
E	10	2	8	4	0

Realizando a alocação de modo que minimize f tem-se, A, B, C, D valendo respectivamente 000, 010, 101, 011 e 001. Com esta alocação tem-se f com um custo igual a 92 e o número de literais (SOP) do circuito obtido a partir desta alocação é 19. Fez-se o complementar desta alocação o que mantém f com um custo igual a 92, porém para esta última alocação tem-se um circuito que possui 16 literais (SOP).

Pode-se observar que os pesos não representam corretamente o comportamento do SAP para diferentes tipos de alocação, pois a variação na quantidade de literais (SOP) deve corresponder a uma variação na soma dos pesos. Deste modo, torna-se impossível prever qual a alocação a ser determinada para este tipo de máquina.

Nesta dissertação propõe-se acrescentar aos valores dos pesos do grafo a propriedade de substituição, para melhor representar as características das máquinas de estados. Aumenta-se a diferença entre os pesos, pode-se determinar uma alocação de estados de boa qualidade de forma menos complexa.

5.2. Propriedade de Substituição e o Grafo de Amaral

Algumas máquinas de estados podem ser decompostas em sub-máquinas que podem ser conectadas em série ou em paralelo.

No caso de serem decompostas em série, tem-se que as sub-máquinas predecessoras não dependem de suas sucessoras. E quando decompostas em paralelo, há uma independência entre as sub-máquinas.

A propriedade de Substituição é dada pela definição I do Capítulo II.

A existência de uma partição τ qualquer e uma partição π com a Propriedade de Substituição sobre conjunto de estados S de uma máquina M , tal que $\pi \cdot \tau = \pi(0)$, garante que M pode ser decomposta em duas sub-máquinas conectadas em série.



A primeira sub-máquina da conexão consiste em r (sendo r o menor inteiro maior ou igual a $\log_2 \#(\pi)$) elementos de memória e seus circuitos de controle, correspondente às variáveis designadas para distinguir entre os blocos de π . Assim essas variáveis são independentes das variáveis para distinguir entre os estados nos blocos.

Uma máquina de estados M que possui duas partições π_1 e π_2 não triviais, com a Propriedade de Substituição de tal forma que $\pi_1 \cdot \pi_2 = \pi(0)$, sendo $\pi(0)$ a partição trivial, implica que pode ser decomposta em duas sub-máquinas operando em paralelo independente uma da outra.

Uma submáquina consiste em r_1 (sendo r_1 o menor inteiro maior ou igual a $\log_2 \#(\pi_1)$) elementos de memória, correspondendo às variáveis de estado designada para distinguir entre os blocos de π_1 . A segunda sub-máquina consiste em r_2 (sendo r_2 o menor inteiro maior ou igual a $\log_2 \#(\pi_2)$) elementos de memória, correspondendo às variáveis de estado designada para distinguir entre os blocos de π_2 .

Devido à contribuição na otimização dos circuitos de controle dos elementos de memória através da alocação de estados, a propriedade de substituição foi introduzida aos pesos do grafo proposto por Amaral.

Para agregar o conceito de propriedade de substituição ao valor dos pesos do grafo de Amaral foi adicionado a seguinte parcela a equação que determina os pesos:

$$R_5 \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} \omega_{ij} \alpha_{ij} ;$$

sendo s a quantidade de estados, i e j são os estados nas extremidades do arco em questão e o valor de R_5 é igual a 4. O comportamento da função ω é apresentado a seguir.

$$\omega_{ij} = \begin{cases} 1 & \text{se } S_i \in \pi_l \text{ e } S_j \in \pi_l, \quad l=1, 2, \dots, n; \\ 0 & \text{caso contrário.} \end{cases}$$

sendo, π_l um bloco da partição π e n a quantidade de blocos.

No exemplo 5.2 tem-se a variação dos pesos quando considera-se a propriedade de substituição.

Exemplo 5.2

Considere a máquina de estados M_{10} descrita pelo modelo de Moore, sendo seu comportamento dado pela Tabela 5.3.

Tabela 5.3: Tabela de próximo estado da máquina de estados M_{10} .

Estado Atual	Entradas		Saída
	0	1	
A	D	C	0
B	F	C	0
C	E	B	0
D	B	E	1
E	A	D	0
F	C	D	0

Na Tabela 5.4 tem-se a matriz indicando o peso dos arcos do grafo obtido sem a propriedade de substituição.

Tabela 5.4: Pesos do grafo para a máquina M_{10} sem a propriedade substituição.

	A	B	C	D	E	F
A	0	6	3	4	3	2
B	6	0	4	1	8	3
C	3	4	0	6	3	6
D	4	1	6	0	2	1
E	3	8	3	2	0	6
F	2	3	6	1	6	0

Pode-se observar que existem vários arcos com valores iguais. Arcos que possuem a mesma prioridade tornam a tomada decisão, entre qual deve ter a distância de *hamming* igual a 1, aleatória prejudicando o resultado final da alocação.

Na Tabela 5.5 tem-se a matriz com pesos dos arcos considerando a propriedade de substituição.

Tabela 5.5: Pesos do grafo para a máquina M_{10} com a propriedade substituição.

	A	B	C	D	E	F
A	0	6	3	4	3	6
B	6	0	4	1	12	3
C	3	4	0	10	3	6
D	4	1	10	0	2	1
E	3	12	3	2	0	6
F	6	3	6	1	6	0

Embora após a aplicação da propriedade substituição, ainda exista arcos com valores iguais, a quantidade de estados com os mesmos valores é menor, tornando a alocação menos aleatória.

5.3. Algoritmo Genético Desenvolvido

O emprego de algoritmos genético tem mostrado um eficiente método para encontrar soluções de boa qualidade para problemas do tipo NP-Completo.

Em geral, um algoritmo genético básico consiste nos seguintes passos: A codificação é a forma como a solução é representada; a geração de uma população inicial aleatória; cálculo do custo de cada indivíduo (solução); seleção dos indivíduos que irão participar da produção da próxima geração e o operador mutação que realiza uma modificação aleatória nos indivíduos para assegurar que o algoritmo não fique preso em um mínimo local.

Nesta seção, apresenta-se cada um desses passos e como eles foram implementados neste trabalho. A alocação de estados é realizada através da minimização de f .

5.3.1. Codificação

Para o AG (Algoritmo Genético) desenvolvido tem-se um vetor de tamanho n , sendo n o número de estados. Cada posição desse vetor representa um estado e o valor designado para um determinado estado está armazenado na posição do vetor correspondente a este na forma decimal.

O exemplo 5.3 ilustra a codificação para a máquina M_{10} .

Exemplo 5.3

Considere a máquinas de estados M_{10} , assim sua codificação será:

A	B	C	D	E	F
0	3	2	6	4	1

Embora a codificação binária seja a mais comum para o AG, a codificação foi realizada através da representação em números decimais, pois deste modo o tratamento da infactibilidade do SAP torna-se mais simples.

5.3.2. Geração da população inicial

A população inicial é gerada de forma aleatória, pois a qualidade das soluções finais deve ser independente da população inicial.

O número de indivíduos é dado pela seguinte equação:

$$I = 2 \cdot n \cdot r,$$

sendo, I o número de indivíduos da população, n a quantidade de estados e r a quantidade mínima de variáveis de estados necessárias para alocar n estados e r o menor inteiro maior ou igual a $\log_2 n$.

Embora a quantidade de indivíduos na população inicial, em alguns casos, gere resultados melhores para valores menores que I e em outros maiores, esta relação na média mostrou-se bastante satisfatória.

O problema de alocação de estados possui uma restrição onde, dois ou mais estados distintos não podem ter a mesma representação binária.

Alguns algoritmos genéticos, podem permitir soluções infactíveis, mas no caso do SAP, estados com a mesma representação binária possuem distância de *hamming* igual a zero, fazendo o resultado convergir rapidamente para uma solução infactível.

Como a geração da população inicial é aleatória, surgem indivíduos infactíveis. Então foi desenvolvido um artifício para que todos os indivíduos se tornem factíveis.

Este artifício consiste em um mapa que possui todos os valores possíveis que r variáveis de estados podem assumir. À medida que os valores são determinados para os estados de um indivíduo, estes valores são eliminados do mapa, evitando que este valor volte a ser escolhido para outro estado no mesmo indivíduo. O exemplo 5.4 ilustra a aplicação deste artifício.

Os custos dos indivíduos são determinados pela função f .

Exemplo 5.4

Considere-se a máquina M_{10} . Inicialmente, o mapa está completo e nenhum valor foi alocado para o estado de um determinado indivíduo.

Mapa

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

Indivíduo

A	B	C	D	E	F

Uma posição do mapa é escolhida aleatoriamente e então o valor dessa posição é alocado para o estado A. Esta célula do mapa é excluída para que seu valor não seja alocado para outro estado.

Mapa

0	1	2	3	5	6	7
---	---	---	---	---	---	---

Indivíduo

A	B	C	D	E	F
4					

Este processo se repete até que todos os estados tenham recebido um valor. Para cada indivíduo é gerado um novo mapa.

5.3.3. Seleção

A estratégia de seleção utilizada neste algoritmo foi o torneio. Nesta estratégia os descendentes são escolhidos realizando $n/2$ jogos, sendo n o tamanho da população, pois neste algoritmo a população não é integralmente substituída.

Em cada jogo são escolhidos aleatoriamente k indivíduos e a configuração ganhadora do jogo é aquela que tem função de adaptação de melhor qualidade. O valor de k escolhido foi 10% da população.

A maior vantagem da estratégia de seleção torneio é a eliminação de dois problemas existentes na seleção proporcional tradicional. A seleção torneio depende apenas dos valores relativos dos custos dos indivíduos e este processo encontra um número de descendentes inteiro e, portanto, dispensa o uso de um gerador de números aleatórios, tornando a seleção por torneio muito rápida e eficiente, embora muito simples.

5.3.4. Crossover

No algoritmo genético, o *crossover* consiste em trocar parcelas de duas configurações para formar duas novas configurações. Os tipos de *crossover* se diferem na quantidade de pontos de troca de informação, que podem ser de um ponto, de dois pontos, multiponto e uniforme.

O *crossover* implementado neste algoritmo foi o de um ponto e todos os indivíduos selecionados são submetidos a este operador.

Após o *crossover* podem surgir indivíduos infactíveis, então o artifício utilizado na geração da população inicial é utilizado para torná-los factíveis.

Este processo de tornar as soluções factíveis é considerado como mutação, pois é introduzida uma informação aleatória no indivíduo.

O exemplo 5.5 ilustra como é utilizado o artifício para tornar as soluções factíveis e a aplicação do operador mutação.

Exemplo 5.5

Considere-se a máquina de estados M_{10} e P_1 e P_2 duas possíveis alocações para esta máquina.

			↓			
	A	B	C	D	E	F
P_1	1	5	7	3	2	0

			↓			
	A	B	C	D	E	F
P_2	2	4	1	3	7	6

	A	B	C	D	E	F
F_1	1	5	7	3	7	6

	A	B	C	D	E	F
F_2	2	4	1	3	2	0

As setas indicam o ponto de *crossover*.

Pode-se observar que em F_2 os estados A e E possuem a mesma representação binária, tornando a solução infactível.

Utiliza-se então, o mapa para torná-lo factível e aplica-se neste indivíduo o operador mutação.

Mapa

5	6	7
---	---	---

	A	B	C	D	E	F
F_2	2	4	1	3	5	0

A representação decimal do estado E foi trocada de 2 para 5, ou seja, foi inserido o primeiro elemento do mapa depois de excluir os que já existiam no indivíduo, de modo a inserir uma informação aleatória (mutação) sem gerar outra infactibilidade.

O indivíduo mais adaptado de cada geração, ou seja, o que possui menor custo é armazenado. A solução é o indivíduo mais adaptado após n iterações.

5.4. O Programa AGPS Implementado

O algoritmo genético descrito na seção 5.3, foi implementado em linguagem de programação C e a estrutura de dados utilizada foram as listas encadeadas.

Com listas encadeadas pode-se utilizar alocação dinâmica de memória, o que possibilita a alocação de estados de máquinas complexas.

Utilizando listas encadeadas, o programa torna-se ágil no seu desempenho, pois não utiliza a troca de informações, mas o seu endereçamento para indicar posição da informação na memória.

A Figura 5.1 apresenta o fluxograma do programa AGPS (Algoritmo Genético com Propriedade Substituição), que ilustra o seu comportamento.

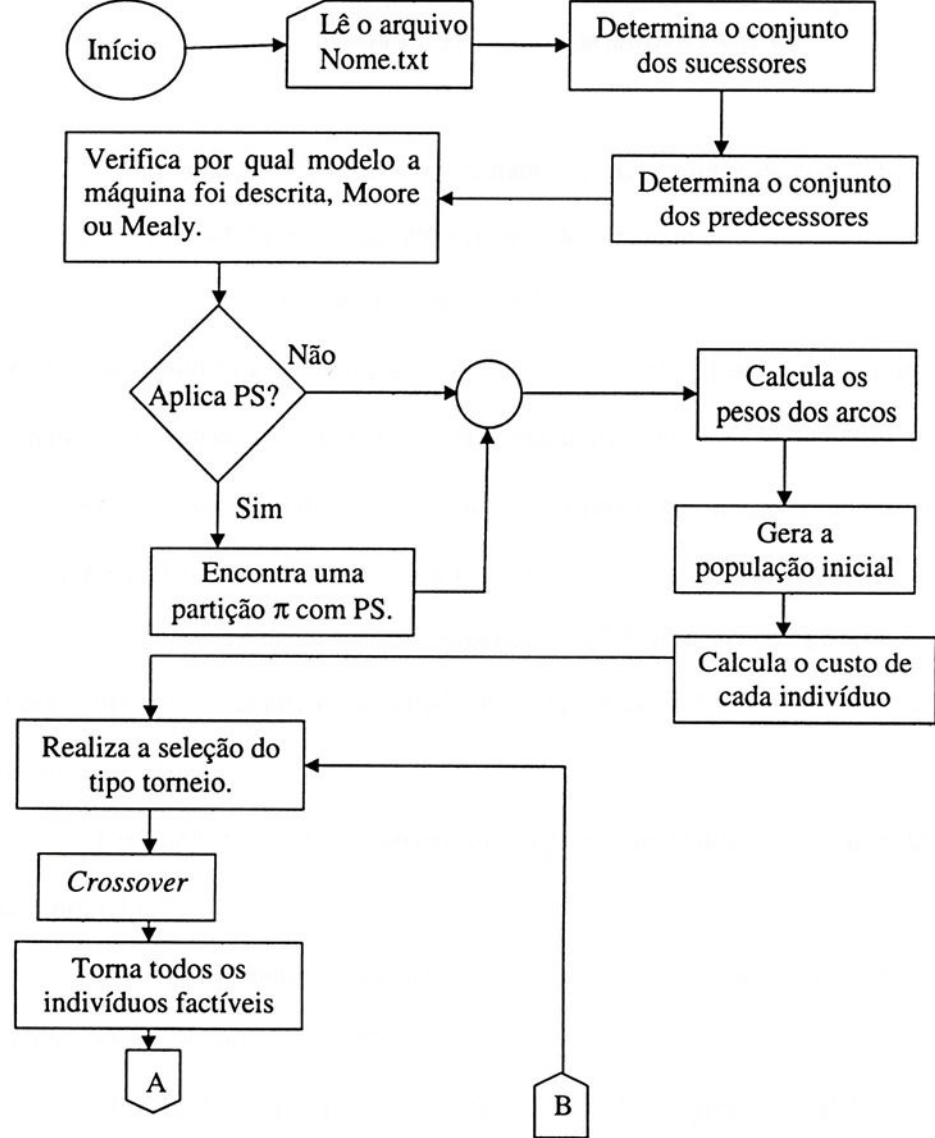


Figura 5.1: Fluxograma do programa AGPS (continua).

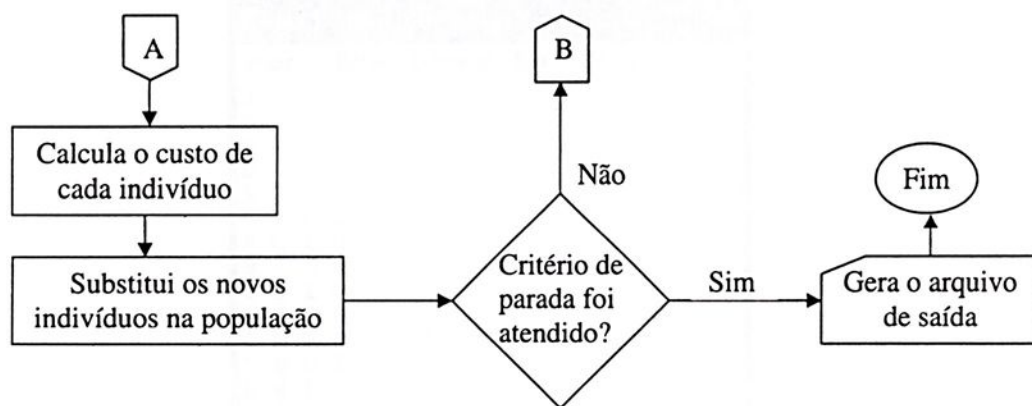


Figura 5.1: Fluxograma do programa AGPS.

O programa AGPS deve ser executado a partir da linha de comando (modo texto) do sistema operacional. Tem-se o seguinte padrão de comando.

AGPS i [opção] arq_entrada arq_saída

sendo, *i* a quantidade de iterações. A opção pode ser *A* para gerar o grafo proposto por Amaral ou *P* para considerar a propriedade substituição na geração do grafo.

Os arquivos executável e de entrada do programa devem estar no mesmo diretório (pasta) na qual é feita a chamada ao programa.

O arquivo de entrada que o programa AGPS utiliza deve possuir extensão *txt* e a mesma sintaxe dos arquivos de entrada dos programas KADE e Nichols. O arquivo de saída possui extensão *tab*.

O exemplo 5.6 ilustra como executar o programa AGPS e os detalhes dessa operação.

Exemplo 5.6

Considere a máquina de estados M_{10} . Este exemplo ilustra como realizar a alocação de estados através do programa AGPS.

A Figura 5.2 representa o arquivo de entrada para o programa AGPS.



Figura 5.2: Arquivo de entrada para a máquina M_{10} .

O programa é executado digitando a seguinte linha de comando:

AGPS 10 P m11 m1lp

sendo, 10 a quantidade de iterações e P a opção que considera a propriedade substituição na geração do grafo, *m11* é o nome do arquivo de entrada e *m1lp* o nome do arquivo de saída.

O programa exibe um relatório no terminal indicando o modelo pelo qual a máquina foi descrita, a partição com propriedade substituição encontrada e a alocação realizada pelo programa.

A Figura 5.3 apresenta este relatório.

```

Selecionar C:\WINDOWS\system32\cmd.exe
D:\projeto\textos\disserata\teste_disserata>genet
VERIFICANDO SE EXISTE A PROPRIEDADE PECHAMENTO
Maquina de Moore
Particao: 4
Numero de Blocos: 3
Bloco
Marca 0
Numero de estados: 2
Estados: B
Estados: E
Bloco
Marca 1
Numero de estados: 2
Estados: D
Estados: C
Bloco
Marca 1
Numero de estados: 2
Estados: A
Estados: F

Indice<51>
Custo: 100
A -> 0
B -> 1
C -> 5
D -> 4
E -> 3
F -> 2
D:\projeto\textos\disserata\teste_disserata>

```

Figura 5.3: Relatório apresentado por AGPS para a máquina M_{10} .

O arquivo de saída é gerado a partir da substituição das literais do arquivo de entrada pelos decimais correspondentes a alocação encontrada pelo programa AGPS.

A Figura 5.4 ilustra o arquivo de saída obtido.

```

M11P - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
B
D
D
D
1 1
0 4 0 0
0 5 1 0
1 2 0 0
1 5 1 0
5 3 0 0
5 1 1 0
4 1 0 1
4 3 1 1
3 0 0 0
3 4 1 0
2 5 0 0
2 4 1 0
-100

```

Figura 5.4: Arquivo de saída gerado pelo AGPS, máquina M_{10} .

O arquivo de saída do programa AGPS é o arquivo de entrada do programa Tabela [30] que gera as equações booleanas a partir da alocação obtida.

Devido às características do operador seleção, o programa seleciona para a próxima geração vários indivíduos com a mesma configuração.

Uma população com baixa variedade genética tende a se extinguir, ou seja, a solução permanece em um mínimo local. Buscando solucionar este problema, indivíduos com a mesma configuração da nova geração são excluídos. Entretanto, a variedade genética após certo número de gerações tende a baixar.

No capítulo VI, os programas KADE e Nichols foram submetidos a uma série de testes a fim de validar sua eficiência diante de outros algoritmos utilizados por ambientes de síntese comerciais.

O programa AGPS, por se tratar de um algoritmo heurístico, tem a capacidade de alocar estados para máquinas complexas, sendo assim avaliado e comparado com algoritmos especializados da literatura através de *benchmarks*.



CAPÍTULO VI – ESTUDOS DE CASOS

Neste capítulo, as alocações realizadas pelos programas Nichols e KADE são avaliadas e comparadas com as alocações realizadas pelo ambiente de síntese MAX + PLUS II da Altera. O objetivo das comparações é o de validar os programas e comprovar a eficiência dos métodos.

O programa AGPS é avaliado e comparado com o JEDI [8] e ao algoritmo proposto por Amaral através de *benchmarks* [34].

Os custos dos circuitos combinacionais são obtidos a partir do arquivo de saída do programa Tabela [30], que gera os circuitos de controle dos elementos de memória e as funções de saída. Os custos são dados em número de literais (SOP), pois é a unidade que melhor representa a complexidade do circuito.

6.1. O Programa Tabela

O programa Tabela foi desenvolvido para sintetizar FSM e foi implementado em linguagem de programação Pascal.

A partir da tabela de próximo estado gerado nos arquivos de saída dos programas KADE e Nichols, o programa Tabela obtém as funções de próximo estado e as funções de saída. Pode-se escolher entre dois tipos de elementos de memória, o JK e o D. As funções combinacionais são obtidas na sua forma mínima, cujo algoritmo de minimização utilizado pelo Tabela é o *Quine-McCluskey*.

O programa Tabela foi utilizado neste trabalho para obter os custos das funções de controle dos elementos de memória e das funções de saída de cada alocação de estado apresentada neste trabalho, uma vez que o programa Tabela não realiza a alocação.



Com o emprego do Tabela uniformizou-se o processo de síntese das máquinas de estados estudadas. Dessa forma, pode-se afirmar com segurança que as diferenças nos custos finais de cada implementação foram decorrentes somente em função da alocação de estado efetuada. Dessa forma o circuito que implementa a função $f(A, B, C) = A \cdot B' \cdot C + A' \cdot B' \cdot C$ tem um custo igual a 8, pois contém duas portas *ANDs* de três entradas (custo 6) e uma porta *ORs* de duas entradas (custo 2).

6.2. Alocação Realizada pelo MAX + PLUS II

O MAX + PLUS II é um ambiente de síntese desenvolvido pela Altera que implementa circuitos digitais em dispositivos programáveis. A linguagem de descrição utilizada pelo ambiente é a VHDL, a AHDL e o Verilog HDL.

O MAX + PLUS II é um ambiente muito versátil, pois possibilita ao projetista descrever o *hardware* tanto em modo texto quanto em modo gráfico e a criar funções que podem ser utilizadas em projetos maiores.

O MAX + PLUS II compila a descrição da máquina e gera alguns arquivos relatórios apresentando as definições utilizadas para descrever o circuito.

No arquivo *nomedoprojeto.rpt* encontram-se registradas as funções de controle dos elementos de memória e as funções saída, a alocação de estados e a quantidade de unidades lógicas utilizadas pelo projeto, entre outras características da descrição.

A partir do arquivo relatório tem-se a alocação utilizada pelo MAX + PLUS II, que depende do dispositivo para o qual está configurado. Para os CPLD's (*Complex Programmable Logic Device*) da família FLEX 10K20 tem-se a alocação *Almost One-Hot*, devido à grande quantidade de elementos de memória presente neste componente.

Na Figura 6.1 é mostrado o fragmento do arquivo relatório que contém a alocação.

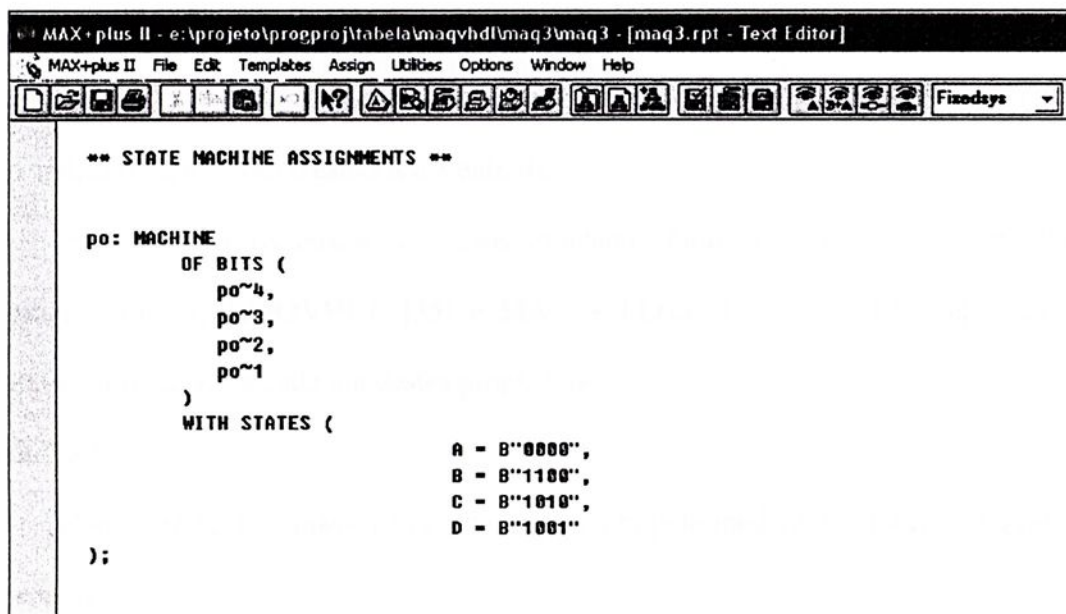


Figura 6.1: Fragmento do arquivo relatório apresentando a alocação de estados de uma determinada FSM.

Na compilação foi configurado um componente da família FLEX 10K20. Pode-se observar que a alocação realizada pelo ambiente MAX + PLUS II da Altera foi a *almost one-hot*. Quando configurado um componente da família MAX7000S, a alocação utilizada pelo MAX + PLUS II da Altera é a *simplest*.

6.3. Avaliação do programa KADE

Para comparar os custos dos circuitos combinacionais gerados pelo programa Tabela, quando empregado à alocação realizada pelo programa KADE e a alocação realizada pelo ambiente de síntese MAX + PLUS II da Altera, foram realizados alguns testes.

Escolheu-se o programa Tabela para gerar as funções de controle e saída das máquinas de estados, pois este utiliza o método de *Quine-McCluskey* para minimizar os circuitos combinacionais, enquanto que o método de minimização de funções booleanas utilizado pelo MAX + PLUS II da Altera não é apresentado em seus arquivos relatórios.

Os casos estudados foram rotulados de acordo com o número do caso, número de estados da máquina apresentada no caso e pelo número de entradas. Dessa forma, a notação 1.6.1 indica o caso 1 com 6 estados e 1 entrada.

Para verificar os custos, os casos estudados foram sintetizados com auxílio dos programas Tabela, TAB2VHDL [35] e MAX + PLUS II. O Caso 1.6.1 apresenta com detalhes a utilização de cada um destes programas.

Caso 1.6.1

Considere-se uma máquina de estados descrita pelo modelo de Moore, apresentada na Tabela 6.1.

Tabela 6.1: Tabela de Próximo Estado da máquina do Caso 1.6.1.

Estados	Entrada		Saída
	0	1	
A	D	C	0
B	F	C	0
C	E	B	0
D	B	E	1
E	A	D	0
F	C	D	0

A Figura 6.2 apresenta o arquivo de entrada para o Programa KADE, que representa a tabela de próximo estado da máquina de estados do Caso 1.6.1.

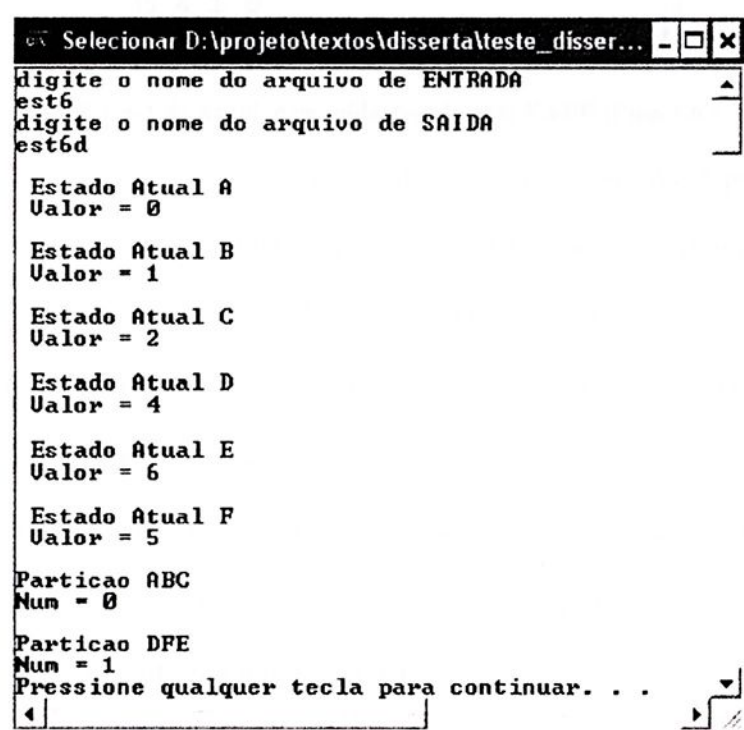


Figura 6.2: Arquivo de entrada para o KADE (Caso 1.6.1).

Quando o programa KADE é executado, apresenta no terminal um relatório informando as partições encontradas e a alocação obtida.

A mensagem “Partição Trivial” indica que não foi encontrada nenhuma partição considerando todas as entradas da máquina de estados. Então é utilizado o conceito de dependência parcial das variáveis de estado.

O relatório exibido no terminal pelo KADE, para a máquina de estados do Caso 1.6.1, é exposto na Figura 6.3.



```
Selecionar D:\projeto\textos\ disserta\teste_disser...
digite o nome do arquivo de ENTRADA
est6
digite o nome do arquivo de SAIDA
est6d

Estado Atual A
Valor = 0

Estado Atual B
Valor = 1

Estado Atual C
Valor = 2

Estado Atual D
Valor = 4

Estado Atual E
Valor = 6

Estado Atual F
Valor = 5

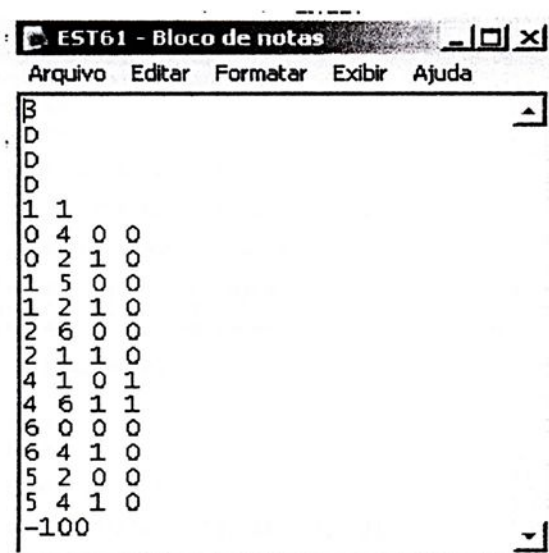
Particao ABC
Num = 0

Particao DFE
Num = 1

Pressione qualquer tecla para continuar. . .
```

Figura 6.3: Relatório apresentado pelo KADE.

O Programa KADE gera o arquivo de saída da Figura 6.4, substituindo as literais do arquivo de entrada pelos seus respectivos decimais de acordo com o relatório da Figura 6.3.



```

B
D
D
D
1 1
0 4 0 0
0 2 1 0
1 5 0 0
1 2 1 0
2 6 0 0
2 1 1 0
4 1 0 1
4 6 1 1
6 0 0 0
6 4 1 0
5 2 0 0
5 4 1 0
-100

```

Figura 6.4: Arquivo de saída gerado pelo KADE (Caso 1.6.1).

Para avaliar os custos de diferentes alocações foi utilizado o programa Tabela. O arquivo da Figura 6.4 é um exemplo de arquivo de entrada para o programa Tabela.

Através das informações contidas no arquivo de entrada o Programa Tabela gera as funções booleanas que controlam os elementos de memória e as funções de saída.

Na Figura 6.5, encontra-se o fragmento do arquivo de saída do programa Tabela, contendo uma função que controla um dos elementos de memória e a função de saída da máquina do Caso 1.6.1. A última linha do arquivo informa o custo total do circuito combinacional desta máquina em literais (SOP).

```

est6 - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
FUNCAO D0
*****
MINTERMOS : 1; 10; 4;
DON'T CARE STATES : 15; 11; 7; 3;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 4      REDUNDANCIA: 0 -> 0100
ESSENCIAL: 10     REDUNDANCIA: 1 -> 101X
ESSENCIAL: 1      REDUNDANCIA: 2 -> 00X1
CUSTO FINAL DE D0 = 13

FUNCAO Z0
*****
MINTERMOS : 4; 12;
DON'T CARE STATES : 15; 11; 7; 3;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 4      REDUNDANCIA: 8 -> X100
CUSTO FINAL DE Z0 = 3

CUSTO TOTAL DAS 4 FUNCOES = 38

```

Figura 6.5 : Arquivo de saída do programa Tabela.

Uma vez gerado as funções booleanas, o programa TAB2VHDL [35] gera o código em VHDL a partir do arquivo de saída do programa Tabela.

O código gerado pelo TAB2VHDL descreve o *hardware* no modo comportamental (Sequencial e RTL). Na Figura 6.6, encontra-se um fragmento do código em VHDL do Caso 1.6.1.


```

est6.vhd  Text Editor
END IF;
Q1 <= UE1;
END PROCESS;

-- Inserindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
        UE2 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
        UE2 <= D2;
    END IF;
    Q2 <= UE2;
END PROCESS;

-- Processos que implementam as funcoes combinacionais de controle
-- dos Elementos de memoria e de Saidas.

D2 <= ( (X0) AND (UE2)) OR (NOT(X0) AND NOT(UE2));
D1 <= ( (X0) AND NOT(UE2) AND NOT(UE1)) OR (NOT(X0) AND (UE2) AND (UE0))
      OR ((X0) AND NOT(UE1) AND NOT(UE0)) OR (NOT(X0) AND NOT(UE2) AND (UE1));
D0 <= ( NOT(X0) AND (UE2) AND NOT(UE1) AND NOT(UE0)) OR ((X0) AND NOT(UE2)
      AND (UE1)) OR (NOT(X0) AND NOT(UE2) AND (UE0));
Z0 <= ( (UE2) AND NOT(UE1) AND NOT(UE0));

```

Line 72 Col 7 INS

Figura 6.6: Fragmento do arquivo de saída do programa TAB2VHDL para o Caso 1.6.1.

O código em VHDL gerado pelo TAB2VHDL pode ser compilado pelo MAX + PLUS II e desta forma, é possível avaliar quantidade de células lógicas utilizadas pelo circuito da máquina em questão no CPLD.

As alocações realizadas pelo Programa KADE foram comparadas com as alocações realizadas por outros dois métodos, o *Simplest* [36], que o ambiente de síntese MAX + PLUS II da Altera utiliza quando os componentes da família MAX7000S são selecionados e a alocação *Almost One-Hot* [36] quando selecionados os componentes da família FLEX 10K20.

Na Figura 6.2, encontra-se a quantidade de literais (SOP) utilizada pelos circuitos combinacionais. O número 1.6.1 representa respectivamente o número do caso, a quantidade de estados e a quantidade de entradas.

Tabela 6.2: Tabela de custos de cada caso estudado.

Caso	KADE	A-One hot	Simplest
1.6.1	38	69	40
2.4.2	40	62	40
3.4.1	14	27	16
4.5.1	24	54	24
5.8.1	61	-	56
6.6.2	38	48	30

Na Figura 6.7, os custos da Tabela 6.2 são apresentados em forma de histrograma para uma melhor visualização.

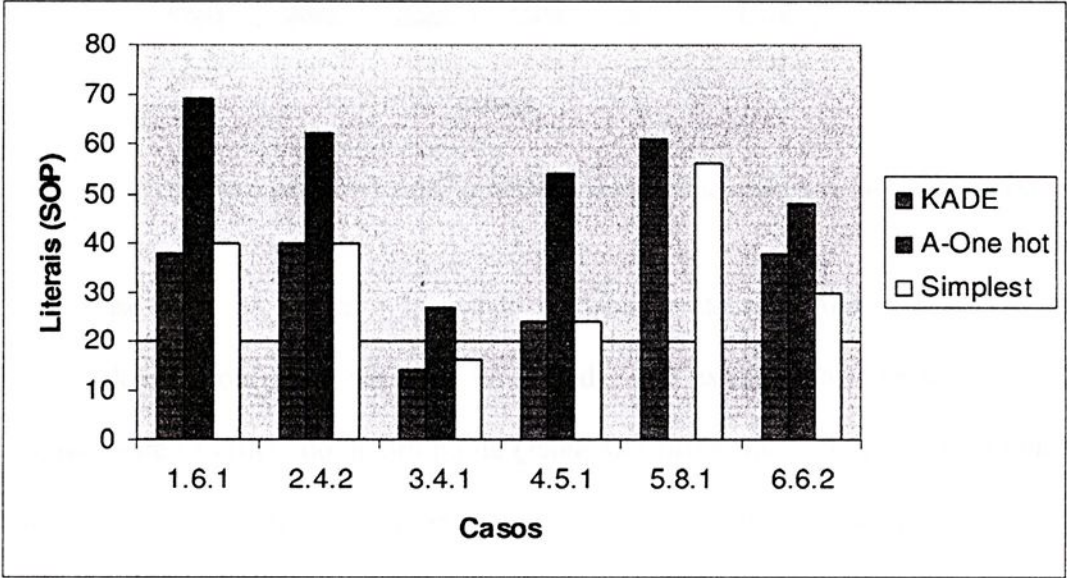


Figura 6.7: Gráfico comparativo entre os métodos, considerando o número de literais (SOP).

Na Tabela 6.3, encontra-se a área utilizada pelas máquinas, quando considerado como critério de custo as células lógicas. O componente configurado foi o EPM7032SLC44-5.

Tabela 6.3: Tabela de custos de cada método estudado.

Casos	Kohavi	A-One Hot	Simplest
1.6.1	4	7	4
2.4.2	3	5	3
3.4.1	3	5	3
4.5.1	3	5	3
5.8.1	4	-	4
6.6.2	4	6	3

Na Figura 6.8 os custos da Tabela 6.3 são apresentados em forma de histograma para uma melhor visualização.

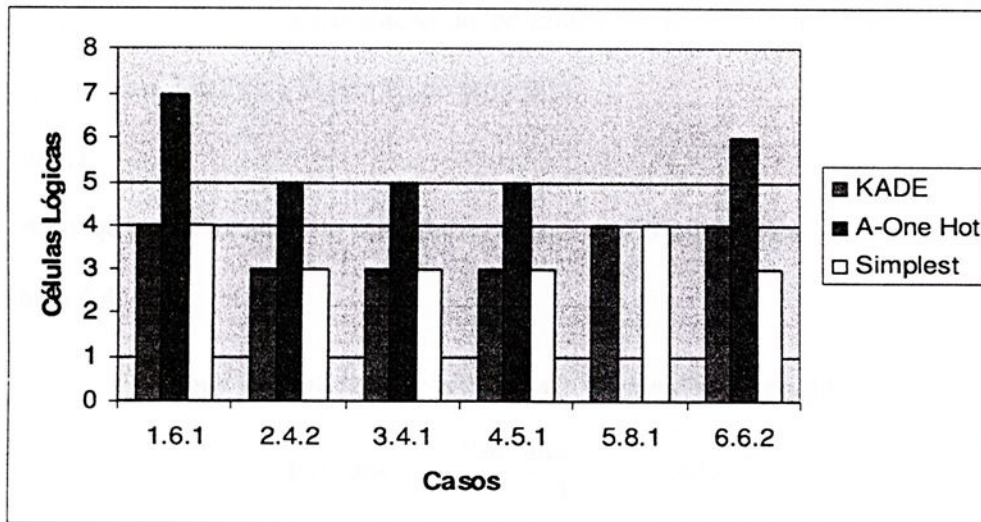


Figura 6.8: Gráfico comparativo entre os métodos, considerando o número de células lógicas.

O Caso 5.8.1 da Figura 6.7, quando realizada a alocação *Almost One-Hot*, possui funções booleanas com nove variáveis de entrada, que extrapola a capacidade programa Tabela, pois este se utiliza do algoritmo de *Quine-McCluskey* que gera todos os implicantes primos. Gerar todos os implicantes primos é um problema de alto custo computacional.

O programa KADE tem um desempenho inferior nos casos 5.8.1 e 6.6.2, pois apenas as variáveis para distinguir entre as partições não são aleatórias.

Diante resultados apresentados, verifica-se que o programa KADE produz uma minimização significativa nos circuitos combinacionais independente da tecnologia alvo, uma vez que o MAX + PLUS II da Altera utiliza a alocação *Simplest* para componentes da família MAX7000S e a alocação *Almost One-Hot* para componentes da família FLEX 10K20.

6.4. Avaliação do programa Nichols

Realizou-se a alocação em algumas máquinas de estados utilizando o programa Nichols, e os métodos utilizados pela ferramenta de síntese comercial MAX + PLUS II da

Altera, o *Simplest* e o *Almost One-Hot*, com o objetivo de comparar os circuitos combinacionais resultantes.

Para melhor ilustrar a utilização do programa Nichols, considere-se o exemplo 6.1, que apresenta com detalhes a execução do programa.

Exemplo 6.1

A máquina de estados do caso 3.6.1, descrita pelo modelo de Moore, é representada pela Tabela 6.4.

Tabela 6.4: Tabela de Próximo Estado da máquina do caso 3.6.1.

Estados	Entrada		Saída
	0	1	
A	D	C	0
B	F	C	0
C	E	B	0
D	B	E	1
E	A	D	0
F	C	D	0

Então, tem-se como arquivo de entrada, representado pela Figura 6.9, para o programa Nichols.



Figura 6.9: Arquivo de entrada do programa Nichols, representação da máquina do caso 3.61.


```

est6_said - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
B
D
D
D
1 1
1 0 0 1
1 2 1 0
7 4 0 0
7 2 1 0
2 6 0 0
2 7 1 0
0 7 0 0
0 6 1 0
6 1 0 0
6 0 1 1
4 2 0 0
4 0 1 1
-100

```

Figura 6.12: Arquivo de saída do programa Nichols.

Como pode ser observado, o programa Nichols pode ser utilizado de forma didática, pois tem-se acesso a cada etapa do processo de alocação, e junto com o programa Tabela possibilita ao projetista avaliar o circuito combinacional obtido para diversas alocações.

A Figura 6.13 apresenta, em forma de histograma, os custos dos circuitos combinacionais obtidos provenientes de cada alocação. Pode-se verificar que o programa Nichols teve um desempenho muito superior ao método *Almost One-Hot* utilizado pelo MAX + PLUS II quando configurado os componentes da família FLEX 10 K20. O programa Nichols teve um desempenho semelhante ao *Simplest*, método utilizado pelo MAX + PLUS II quando configurado componentes da família MAX7000S.

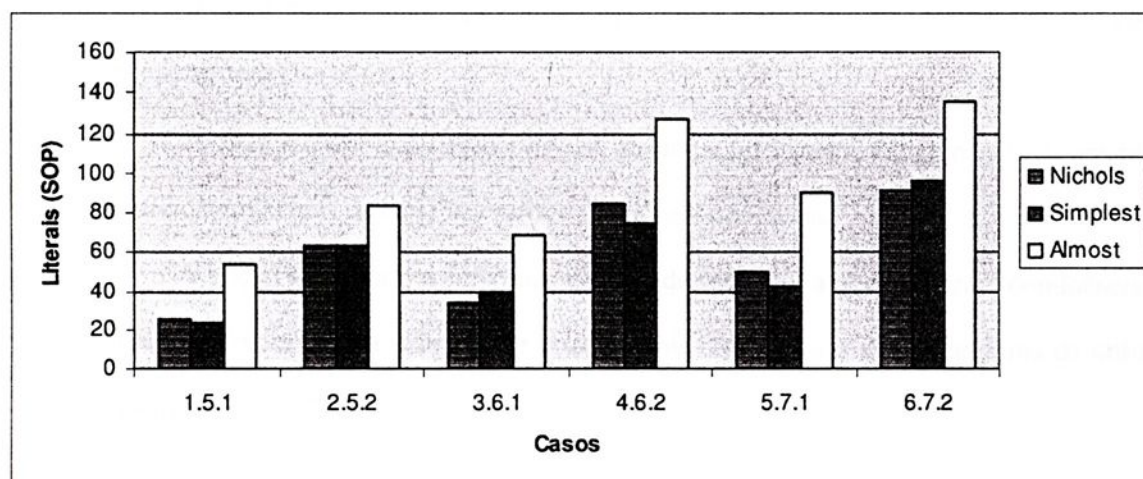


Figura 6.13: Gráfico comparativo entre os métodos apresentados.

A qualidade final da alocação realizada pelo programa Nichols está diretamente relacionada à qualidade da minimização realizada pelo programa BOOM referentes aos *output sets*.

O alto custo computacional deste método está na geração dos *output sets*, pois a quantidade de combinações cresce rapidamente.

6.5. O Programa JEDI

JEDI [8] é um programa para alocação de estados que visa à implementação multi-nível. O programa consiste em duas fases, a fase de determinação dos pesos de um grafo e a de alocação de estados.

Na primeira fase o JEDI determina pesos para todos os possíveis pares de estados, gerando um grafo semelhante ao proposto neste trabalho, porém as características que determinam a afinidade entre os estados são: entradas dominantes, saídas dominantes, pares e variação.

Na segunda fase é aplicado o método heurístico denominado *Simulated Annealing*, que realiza a alocação de estados.

O *Simulated Annealing* encontra frequentemente as soluções ótimas de problemas de pequeno e médio porte e soluções quase-ótimas de problemas de grande porte.

O JEDI permite realização da alocação de estados utilizando vários métodos através das opções que este oferece.

Pode-se realizar a alocação *One-hot*, aleatória, priorizando a dominância de entrada ou de saída, dominância de entrada e saída e diretamente do mapa.

A opção utilizada nos próximos estudos de casos foi a que prioriza a dominância das entradas e das saídas da máquina de estados, pois esta opção é a mais próxima da utilizada pelo algoritmo AGPS.

6.6. Avaliação do programa AGPS

Vários *benchmarks* tiveram sua alocação realizada pelo programa AGPS com o intuito de validar e comparar os seus resultados com programas clássicos como o JEDI.

Para ilustrar como foi realizada a comparação e a avaliação considere-se o exemplo 6.2.

Exemplo 6.2

Considere também a máquina de estados M_{10} , descrita pelo modelo de Moore apresentada na Tabela 5.3. A Figura 6.14 representa o arquivo de entrada da máquina M_{10} para o programa AGPS.

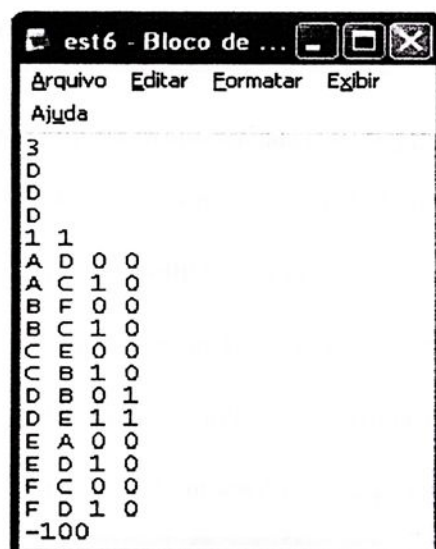


Figura 6.14: Arquivo de entrada da máquina M_{10} .

Executando o programa AGPS é apresentado no terminal o relatório sobre a alocação de estados realizada por este programa. Este relatório é apresentado na Figura 6.15.

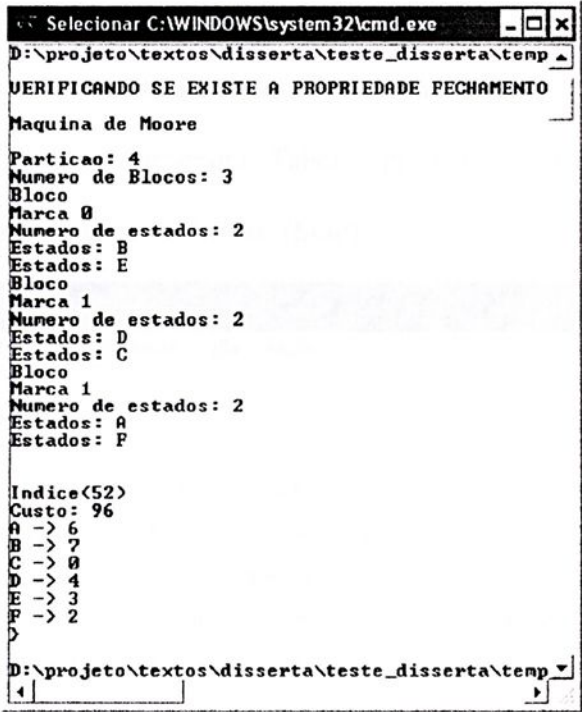


Figura 6.15: Relatório apresentado pelo programa AGPS.

No relatório, é exibida a partição com a Propriedade de Substituição considerada nos pesos do grafo, quando esta opção é escolhida, o modelo pelo qual a máquina foi descrita e a alocação realizada pelo programa após o número de iterações indicado pelo usuário.

No arquivo de saída, as literais do arquivo de entrada são substituídas pelos decimais correspondentes à alocação. A Figura 6.16 apresenta do arquivo de saída para a máquina M_{10} .

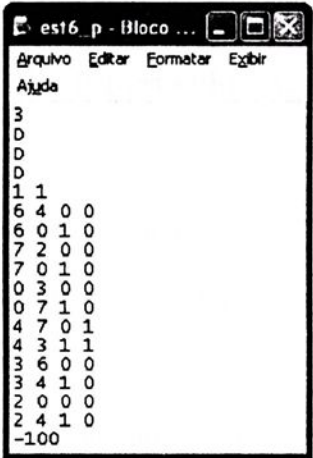
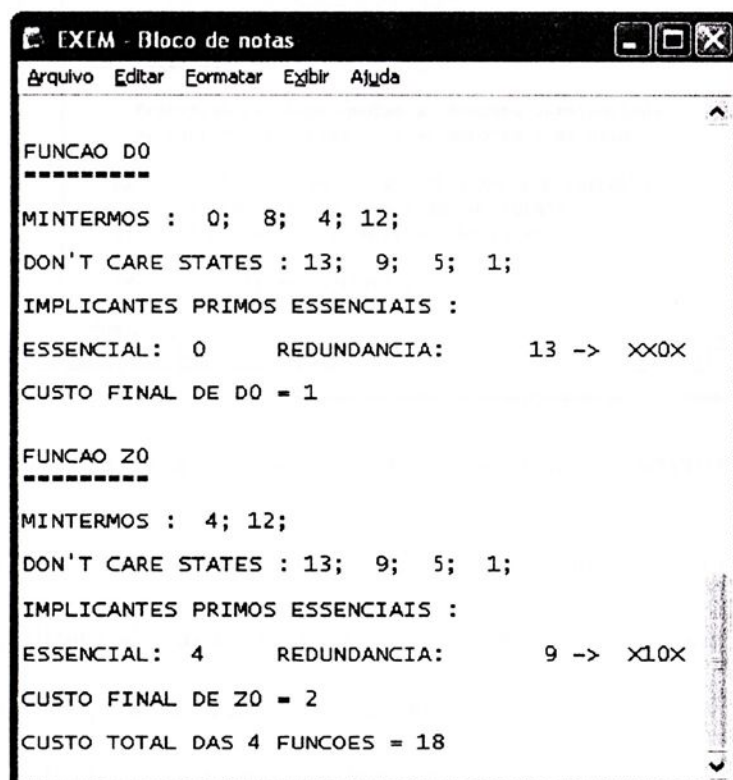


Figura 6.16: Arquivo de saída da máquina M_{10} , gerado pelo AGPS.

Para avaliar a alocação de estados realizada, deve-se obter o circuito combinacional correspondente da máquina para esta alocação. O programa utilizado foi o Tabela.

No arquivo de saída do programa Tabela apresentado na Figura 6.17, têm-se as funções booleanas e os seus custos em literais (SOP).



```

EXEM - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda

FUNCAO D0
*****
MINTERMOS : 0; 8; 4; 12;
DON'T CARE STATES : 13; 9; 5; 1;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 0 REDUNDANCIA: 13 -> XX0X
CUSTO FINAL DE D0 = 1

FUNCAO Z0
*****
MINTERMOS : 4; 12;
DON'T CARE STATES : 13; 9; 5; 1;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 4 REDUNDANCIA: 9 -> X10X
CUSTO FINAL DE Z0 = 2
CUSTO TOTAL DAS 4 FUNCOES = 18
  
```

Figura 6.17: Arquivo de saída do programa Tabela para máquina M_{10} .

Entretanto, para avaliar a implementação física em uma FPGA da máquina de estados deve-se obter a quantidade de células lógicas utilizadas na implementação da máquina de estados.

Para gerar os códigos dos casos estudados de forma automática, foi utilizado o programa TAB2VHDL que gera o código em VHDL da máquina a partir do arquivo de saída do programa Tabela.

A Figura 6.18 apresenta o fragmento do código VHDL gerado pelo TAB2VHDL.

```

MAX+plus II - Untitled1 - [exem.vhd - Text Editor]
MAX+plus II File Edit Templates Assign Utilities Options Window Help

-- Inserindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
        UE2 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
        UE2 <= D2;
    END IF;
    Q2 <= UE2;
END PROCESS;

-- Processos que implementam as funcoes combinacionais
-- de controle dos Elementos de memoria e de Saidas.

D2 <= ( NOT(UE2) AND (UE0)) OR ((X0) AND NOT(UE2))
      OR (NOT(X0) AND (UE2) AND NOT(UE0));
D1 <= ( NOT(UE1)) OR (NOT(X0) AND (UE0));
D0 <= ( NOT(UE1));
Z0 <= ( (UE2) AND NOT(UE1));

END RTL;
Line 92 Col 9 INS

```

Figura 6.18: Fragmento do código VHDL gerado pelo TAB2VHDL.

O código VHDL é compilado pelo MAX+PLUS II que gera um arquivo relatório, onde se pode visualizar todas as configurações do componente e da compilação, inclusive a quantidade de células lógicas utilizadas pela máquina.

Na Figura 6.19, tem-se o fragmento deste arquivo relatório apresentando a quantidade de células lógicas utilizadas.

MAX+plus II - d:\projeto\txtos\disserat\teste_disserat\templexem - [exem.rpt - Text Editor]

MAX+plus II File Edit Templates Assign Utilities Options Window Help

Fixedsys

**** DEVICE SUMMARY ****

Chip/ POF	Device	Input Pins	Output Pins	Bidir Pins	Shareable LCs	Expanders	% Utilized
exem	EPM7032SLC44-5	3	4	0	4	0	12 %
User Pins:		3	4	0			

Line 1 Col 1 INS

Figura 6.19: Fragmento do arquivo relatório do gerado pelo programa MAX + PLUS II.

Para todos os casos foi utilizada a mesma metodologia do exemplo 6.2, para o programa JEDI e as duas opções possíveis oferecidas pelo programa AGPS.

A partir do arquivo de saída do programa Tabela, obtêm-se os custos em literais (SOP), e a partir do arquivo relatório apresentado pelo programa MAX + PLUS II, tem-se a quantidade de células lógicas utilizadas por cada caso quando configurado o componente EPM7032SLC44-5.

O gráfico da Figura 6.20 ilustra as diferenças entre os custos, em literais (SOP), de cada caso estudado.

A legenda AGPS_PS indica que foi considerada a Propriedade de Substituição pelo programa AGPS. AGPS_A indica que foram consideradas apenas as propriedades propostas por Amaral.

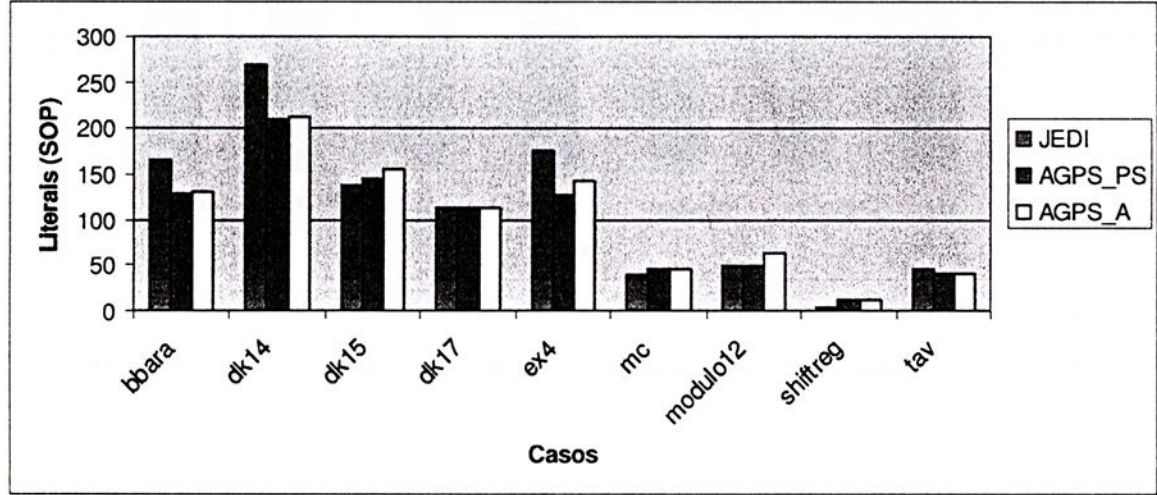


Figura 6.20: Gráfico comparativo entre o programa JEDI e o AGPS.

O SAP, por se tratar de um problema NP-Completo possui um comportamento imprevisível. Em máquinas que possuem uma quantidade de entradas, saídas e estados semelhantes, um programa pode obter resultados melhores que o outro para uma determinada máquina devido à natureza e complexidade de suas transições.

Este fato pode ser observado nos casos dk14 e dk15 apresentados na Figura 6.20. No primeiro caso o programa AGPS teve uma grande vantagem sobre o programa JEDI, enquanto que no dk15 o programa JEDI obteve resultado um pouco melhor.

O programa AGPS teve resultados satisfatórios diante do programa JEDI quando selecionado qualquer uma das opções oferecidas pelo programa AGPS.

A Figura 6.21, apresenta-se o gráfico comparativo entre os programas JEDI e AGPS considerando a quantidade de células lógicas utilizadas por cada caso quando configurado o componente EPM7032SLC44-5.

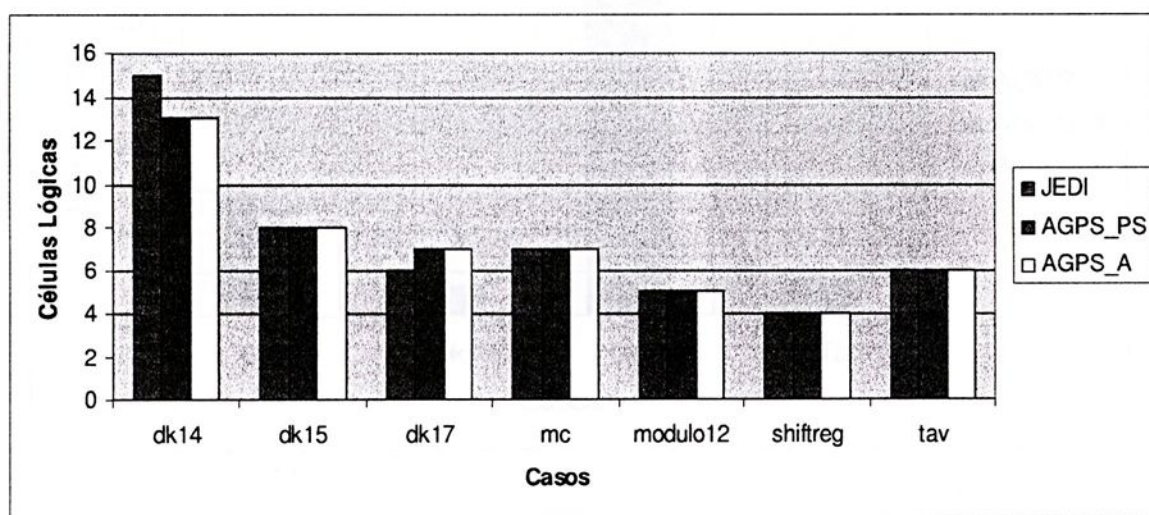


Figura 6.21: Gráfico comparativo entre JEDI e AGPS, sendo o custo a quantidade de células lógicas.

Pode-se observar que os programas tiveram desempenho semelhante, pois este critério de custo não é muito sensível a pequenas variações. Uma célula lógica mesmo que não utilizada por completo é somada ao custo.

A Tabela 6.5 apresenta os resultados obtidos pelo Algoritmo de Amaral e o AGPS. Apesar de algumas máquinas não possuírem a Propriedade de Substituição os resultados mostram-se satisfatórios.

Tabela 6.5: Resultados do Algoritmo do Amaral e AGPS.

Máquinas	AGPS_P	AGPS_A	Amaral
Bbara	149	127	132
Dk14	36	34	40
Donfile	384	417	376
Tav	40	40	40

Na Figura 6.22 é apresentado o gráfico comparativo entre o Algoritmo Genético proposto por Amaral e o AGPS.

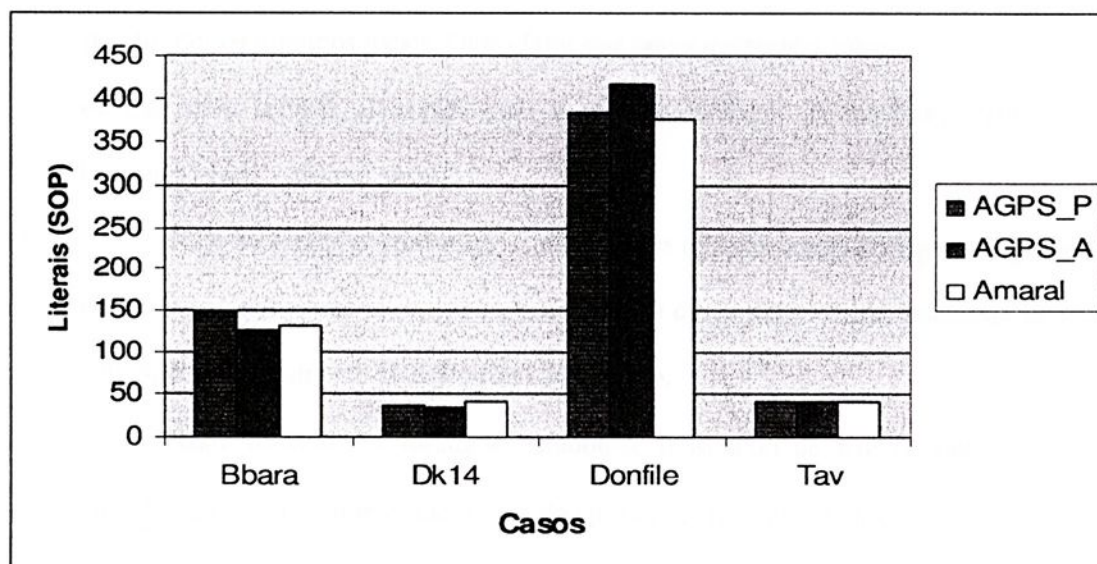


Figura 6.22: Gráfico comparativo entre o programa AGPS e o proposto por Amaral.

Pelo gráfico da Figura 6.22 pode-se observar que no algoritmo do AGPS têm-se resultados iguais ou um pouco melhores do que o proposto por Amaral na maioria dos casos avaliados. Cabe salientar que a qualidade do AGPS não está somente ligada à Propriedade Substituição, mas também ao AG implementado.

No capítulo VII, apresenta-se as conclusões obtidas a partir dos estudos realizados sobre alocação de estados e os trabalhos futuros.

CAPÍTULO VII – CONCLUSÃO

Com o surgimento de novas ferramentas de síntese de alto desempenho produzem-se máquinas de estados finitos muito complexas, o que faz a alocação de estados se tornar cada vez mais importante em um projeto de sistemas digitais.

O SAP trata-se de um problema muito complexo para obter o seu ótimo global, pois possui muitos mínimos locais. Outro fator que deixa evidente a complexidade do SAP é o fato de que nem sempre alocações com a mesma distância de *hamming* obtêm-se funções booleanas com o mesmo custo.

Pôde-se observar com estes estudos que o complementar de algumas alocações pode gerar circuitos maiores ou menores, o que indica que a distância de *hamming*, às vezes, não corresponde diretamente ao custo final do circuito.

Neste trabalho de mestrado estudou-se o Método de Kohavi que se mostrou um método de partição muito eficiente, sendo aplicado a máquinas de Mealy.

A limitação do método proposto por Kohavi é ter que encontrar uma partição com a Propriedade de Substituição. Existem máquinas que não possuem partição com essa propriedade. O custo computacional para encontrar uma partição com esta propriedade é relevante, devido às características do algoritmo.

Mesmo com essas limitações é justificável a utilização da Propriedade de Substituição, pois esta possibilita encontrar uma alocação quase ótima.

O algoritmo de Kohavi foi implementado originando o programa KADE que realiza a alocação de estados, visando apenas os circuitos de controle dos elementos de memória. Em trabalhos futuros, a propriedade de Saídas Consistentes será adicionada ao programa que então tratará também as equações de saída.



Devido ao fato do programa KADE não utilizar a propriedade Saídas de Consistentes, pode-se utilizá-lo para realizar alocação em máquinas descritas pelo modelo de Moore ou Mealy.

O programa KADE, não leva em consideração o número de estados que elas possuem.

Atualmente, vários algoritmos apresentados têm como referência o algoritmo de Kohavi, o que mostra sua importante contribuição para a alocação de estados.

O método de alocação de estados proposto por Nichols é aplicável somente em máquinas de Moore, e seu custo operacional cresce rapidamente com o aumento dos estados, porém tem um custo menor do que testar todas as alocações possíveis de uma determinada máquina.

Os testes apresentados mostram que tanto o programa KADE quanto o programa Nichols tiveram um desempenho satisfatório quando comparados com os algoritmos utilizados por ferramentas de síntese comerciais, como o MAX + PLUS II da Altera.

O algoritmo que transforma as máquinas de estados em um grafo com pesos, completamente conectado e sem direção apresentado por Amaral, mostra-se bastante eficiente e com algumas modificações, o grafo poderá ser tratado como fluxo em rede, o que possibilitará a formulação matemática do problema de alocação de estados como um problema de programação linear inteira 0 e 1.

Com o desenvolvimento do programa AGPS comprovou-se que o Algoritmo Genético é eficiente na alocação de estados; entretanto, pode-se melhorar muito este mecanismo de alocação.

Considerando a Propriedade de Substituição na geração dos pesos do grafo proposto por Amaral, obteveram-se circuitos combinacionais menores na maioria dos casos. Entretanto, algumas máquinas não possuem partição com a Propriedade de Substituição.

Os programas KADE, Nichols e AGPS podem ser utilizados como ferramentas didáticas para ilustrar a importância da alocação de estados e tornar mais atraente suas teorias. Com estes programas os alunos podem testar vários algoritmos para alocação e tirar suas próprias conclusões.

Apesar dos componentes reconfiguráveis como as FPGA e os CPLD possuírem uma quantidade imensa de unidades lógicas (LUT), que são compostas de um pequeno circuito combinacional e um elemento de memória, muitas vezes é necessário utilizar o circuito combinacional de unidades adjacentes, o que deixa evidente a necessidade de minimizar os circuitos combinacionais e, portanto, a importância da alocação de estados.

Em trabalhos futuros, os programas AGPS, Reduplex [37], Tabela, Mínimo [30] e TAB2VHDL serão integrados gerando um CAD mais completo, onde é fornecida a tabela de próximo estado e é obtido o código em VHDL já com os estados reduzidos, uma alocação de boa qualidade e com as funções que controlam os elementos de memória e funções de saída minimizados.



REFERÊNCIAS BIBLIOGRÁFICAS

- [1] José Nelson Amaral, Kagan Tuner, Joydeep Ghosh. "Designing Genetic Algorithms for the State Assignment Problem", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, Nº 4, April 1995.
- [2] Julian F. Miller, Peter Thomson, "Combinational and Sequential Logic Optimisation using Genetic Algorithms", *Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp.34-38 Nº 414, 12-14 September 1995.
- [3] Gagan Hasteer, Prithviraj Banerjee., "Simulated Annealing Based Parallel State Assignment of Finite States Machines", *10th International Conference on VLSI Design*, pp. 69-75, January 1997.
- [4] Wakerly, J.F., "Digital Design Principles and Practices", *State Assignment*, Chap. 5 p. 387-390. ed. Prentice Hall. 1990.
- [5] Perry, Douglas L., "VHDL (VHSIC High Description Language)", ed. McGraw-Hill, 2ª ed, 1993.
- [6] www.ieee.org/pubs/authors.html. Acessado em: 28/04/2003.
- [7] L. Jozwiak, A. Slusarczyk, A. Chojnacki, "Fast and compact sequential circuits for the FPGA-based reconfigurable systems", *Journal of Systems Architecture*, nº 49, pp. 227-246, 2003.
- [8] Bill Lin, A. Richard Newton, "Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages", *Proc. VLSI 89 Conference*, Munich, West Germany, August 1989.
- [9] Mariusz Chyzy, Witold Kosinski, "Evolutionary algorithm for state assignment of finite state machines", *Artificial Intelligence Methods*, Gliwice, Poland, pp. 51-52, november 5-7, 2003.
- [10] José C. Monteiro, Arlindo L. Oliveira, "Finite State Machine Decomposition for Low Power", *Proceedings 35th Design Automation Conference*, San Francisco, pp.758-763, 1998.
- [11] Srinivas Devadas, A. R. Newton, "Decomposition and factorization of sequential finite state machines", *IEEE Trans. on CAD*, vol. 8, Nº 11, pp. 1206-1217, November 1989.

- [12] Peter Vanbekbergen, Bin Lin, Hugo De Man, G. Goossens "A Generalized State Assignment Theory for Transformations on Signal Transition Graphs", *In Proc. Int. Conf. Computed-Aided Design*, pp.112-117, 1992.
- [13] Tiziano Villa, Alberto Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementation" , *IEEE Transactions on Computer Aided Design*, vol. 9, Nº 9, September 1990.
- [14] Srinvas Devadas, Hi-Keung Ma, A. Richard Newton, A. Sangiovanni-Vincentelli, "MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations" *IEEE Transactions on Computer Aided Design*, vol. 7, nº 12, December 1988.
- [15] Grefenstette J.J., Gopal R., Rosmaita B., Van Gucht D., "Genetic Algorithm for the TSP", *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, pp. 160-168.
- [16] Xia, Y., and Almaini, A.E.A, "Genetic Algorithm based state assignment for power and area optimization", *IEE Proc., Comput. Digit. Tech.*, 2002, 149, (4), pp. 128-133.
- [17] Whitley D., "The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of reproductive Trials is Best", *Proceedings of the 3rd International Conference on Genetic Algorithms*, D. Schaffer, ed., pp. 116-121, Morgan Kaufmann, 1989.
- [18] I. H. Osman: "Heuristic for the Generalized Assignment Problem – Simulated Annealing and Tabu Search Approaches", *OR Spectrum*, 17(4), 211-225, 1995.
- [19] State Assignment, <http://www2.ele.ufes.br/~ailson/digital/chapter09.doc3.html>.
Acessado em: 14/03/2003.
- [20] Kohavi, Zvi, "Reduction of Output Dependency in Sequential Machines", *IEEE Transactions on Electronic Computers*, pp. 932-934, December 1965.
- [21] Nichols, A.J., "State Assignments in Combinational Networks", *IEEE Transactions on Eletronic Computers*, EC – 14, pp. 343-349, June 1965.
- [22] Santos, Carlos Eduardo S., Alexandre César R. S., Mariângela Carvalho B., "KADE: Decomposição de Máquinas de Estados Finitos para Otimização do Circuito Combinacional", *Congresso Temático de Dinâmica, Controle e Aplicações*, pp. 1168-1174, Ilha Solteira – SP, maio 2004.

- [23] Santos, Carlos Eduardo S., Alexandre César R. S., Mariângela Carvalho B., "Alocação de Estados Através do Algoritmo de Nichols", *World Congress on Engineering and Technology Education*, pp.792-796, Santos – SP, march 2004.
- [24] Ashar, Pranav. Srinivas Devadas, A. Richard Newton, "Optimum and Heuristic Algorithms for an Approach to Finite State Machine Decomposition", *IEEE Transactions on Computer-Aided Design*, vol. 10, nº 3, march 1991.
- [25] Geiger, Martin. Thomas Muller-Wipperfurth "FSM Decomposition Revisited: Algebraic Structure Theory Applied to MCNC Benchmark FSMs", *28th ACM/IEEE Design Automation Conference*, nº 11.4, pp. 182-185, 1991.
- [26] Józwiak, L. A. Slusarczyk, A. Chojnacki, "Fast and compact sequential circuits for the FPGA-based reconfigurable systems", *Elsevier, Journal of Systems Architecture*, nº 49, pp. 227-246, 2003.
- [27] MAX + PLUS II Getting Started, version 8.1, September 1997.
- [28] Kruse, Robert L., "Lists", *Data Structures and Program Design*, ed. 3ª, Prentice Hall International Editions, 1994, pp. 195-211.
- [29] Horowitz, Ellis, Sahni, Sartaj, "Listas Encadeadas", *Fundamentos de Estruturas de Dados*, ed. 3ª, editora Campus 1997, pp.104-140.
- [30] Silva, Alexandre César Rodrigues da, *Contribuição à Minimização e Simulação de Circuitos Lógicos*. Campinas, 1989. 140 p. Dissertação (Mestrado em Engenharia Eletrônica) – Universidade Estadual de Campinas.
- [31] McCluskey, Edward J. "Logic Design Principles with Emphasis on Testable Semicustom Circuits" ed. Prentice Hall. 1986.
- [32] M. Karnaugh. "The Map Method for Synthesis of Combinational Logic Circuits". *Transactions of AIEE, Part I, Communications and Electronics*, 72, pp. 593-99. 1953.
- [33] Hlavieka, Jan, Petr Fiser, "Boom- A Heuristic Boolean Minimize", *IEEE 2001*, pp. 439-442, Department of Computer Science and Engineering, Czech Technical University, 2001.

- [34] R. Lisanke, "ed. FSM Benchmark Suite". *Microelectronics Center of North Carolina*, Research Triangle Park, NC, 1987.
- [35] Trancredo, Leandro de Oliveira, "TAB2VHDL: Um ambiente de síntese lógica para máquinas de estados finitos". Ilha Solteira, 2002. 122 p. Dissertação (mestrado) – Univerdade Estadual Paulista – Faculdade de Engenharia de Ilha Solteira.
- [36] Ohira M.I., Zanella, F. P., Silva A. C. R., "Comparative Study Between Synthesis Tools for Digital System", *International Conference on Engineering and Computer Education*, Santos – SP, Brazil, March 2003.
- [37] Silva, Alexandre César Rodrigues da, "Contribuição à Síntese de Circuitos Digitais Utilizando Programação Linear Inteira 0 e 1". Campinas, 1993. 127 p. Dissertação (Doutor em Engenharia Eletrônica) – Universidade Estadual de Campinas.





**,UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"**

Câmpus de Ilha Solteira

Programa de Pós-Graduação em Engenharia Elétrica

Av. Brasil Centro, 56

15385-000 Ilha Solteira - SP

www.dee.feis.unesp.br

