UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CURITIBA

GERÊNCIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL - CPGEI

RAFAEL RODRIGUES DA SILVA

ESTUDO E APLICAÇÃO DE UM ALGORITMO GENÉTICO COMPACTO USANDO ELITISMO E MUTAÇÃO

DISSERTAÇÃO DE MESTRADO

CURITIBA
ABRIL - 2008.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

---

DISSERTAÇÃO
apresentada à Universidade Tecnológica Federal do Paraná
para obtenção do grau de

MESTRE EM CIÊNCIAS

por

RAFAEL RODRIGUES DA SILVA

ESTUDO E APLICAÇÃO DE UM ALGORITMO GENÉTICO COMPACTO USANDO ELITISMO E MUTAÇÃO

Banca Examinadora:

Presidente e Orientador:

PROF. DR. CARLOS RAIMUNDO ERIG LIMA	**UTFPR**
PROF. DR. HEITOR SILVÉRIO LOPES	**UTFPR**

Examinadores:

PROF. DR. FÁBIO KURT SCHNEIDER	**UTFPR**
PROF. DR. LEANDRO DOS SANTOS COELHO	**PUC-PR**

Curitiba, abril de 2008.

RAFAEL RODRIGUES DA SILVA

ESTUDO E APLICAÇÃO DE UM ALGORITMO GENÉTICO COMPACTO USANDO ELITISMO E MUTAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do grau de "Mestre em Ciências" – Área de Concentração: Informática Industrial.

Orientações:

Prof. Dr. Carlos Raimundo Erig Lima (Orientador)

Prof. Dr. Heitor Silvério Lopes (Co-Orientador)

Curitiba

2008

Ficha catalográfica elaborada pela Biblioteca da UTFPR – Campus Curitiba

S586e Silva, Rafael Rodrigues da
Estudo e aplicação de um algoritmo genético compacto usando elitismo e mutação / Rafael Rodrigues da Silva. Curitiba. UTFPR, 2008
XV, 129 p. : il. ; 30 cm

Orientador: Prof. Dr. Carlos Raimundo Erig Lima
Co-Orientador: Prof. Dr. Heitor Silvério Lopes
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2008

1. Circuitos eletrônicos. 2. Algoritmos genético. 3. Hardware. I. Lima, Carlos Raimundo Erig, orient. II. Lopes, Heitor Silvério, co-orient. III. Universidade Tecnológica Federal do Paraná. Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial. IIIV. Título.

CDD: 511.8

À minha família e amigos por tudo
que representam em minha vida.

AGRADECIMENTOS

Agradeço ao professor Carlos Raimundo Erig Lima e ao professor Heitor Silvério Lopes pela orientação prestada no desenvolvimento deste trabalho. Aos colegas de laboratório que sempre estiveram dispostos a ajudar. Aos professores da banca examinadora, o professor Fábio Schneider e o professor Leandro Coelho, pela revisão e sugestões que enriqueceram ainda mais este trabalho.

Agradeço a empresa Companhia Siderúrgica Nacional, em especial ao Márcio Lins, assessor da Diretoria Executiva, e ao Carlos Frederico Rangel, gerente da CSN-PR, por permitirem meus estudos no horário de trabalho.

Com muito carinho agradeço minha família e minha namorada pela compreensão nos meus momentos de ausência e pelo suporte nos momentos mais difíceis.

Agradeço a todos aqueles que me ajudaram, de uma forma ou outra, a realizar uma travessia como esta. São pessoas especiais que não preciso indicar seus nomes, pois elas sabem quem são.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CE	- Computação Evolucionária
IC	- Inteligência Computacional
PE	- Programação Evolucionária
AG	- Algoritmo Genético
EE	- Estratégias Evolutivas
AGS	- Algoritmo Genético Simples
BC	- Bloco Construtivo
AED	- Algoritmo de Estimação de Distribuição
AGC	- Algoritmo Genético Compacto
PBIL	- *Population Based Incremental Learning*
UMDA	- *Univariate Marginal Distribution Algorihtm*
BSC	- *Bit-Based Simulated Crossover*
LVQ	- *Learning Vector Quantization*
ECGA	- *Extended Compact Genetic Algorithm*
MPM	- *Marginal Product Model*
MDL	- *Minimum Description Length*
LK	- *Lin-Kerninghan*
MBBCGA	- *Mutated by Bit Compact Genetic Algorithm*
FPGA	- *Field Programmable Gate Array*
peCGA	- *Persistent Elitism Compact Genetic Algorithm*
neCGA	- *Nonpersistent Elitism Compact Genetic Algorithm*
eCGA	- *Compact Genetic Algorithm with Elitism*
mCGA	- *Compact Genetic Algorithm with elitism and Mutation*
rmCGA	- *Compact Genetic Algorithm with elitism, Mutation and Resampling*
CoCGA	- *Cooperative Approach to Compact Genetic Algorithm*
emCGA	- *Elitism with Mutation Compact Genetic Algotithm*
AC	- Autômatos Celulares
CC	- *Confidence Counter*
ASIC	- *Application Specific Integrated Circuits*
HDL	- *Hardware Description Language*

ISP - *Instruction Set Processor*

ABEL - *Advanced Boolean Equation Language*

PLD - *Programmable Logic Devices*

VHDL - *Very high speed integrated circuit Hardware Description Language*

MEF - Máquina de Estados Finita

HGA - *Hardware Genetic Algorithm*

SPGA - *Splash Parallel Genetic Algorithm*

SSGA - *Steady-State Genetic Algorithm*

PGA - *Pipelined Genetic Algorithm*

RNG - *Random Number Generator*

LFSR - *Linear Feedback Shift Registers*

MLS - *Maximum Length Sequence*

MATLAB - *Matrix Laboratory*

DLL - *Dynamic Link Library*

MPM - *Mutation Probabily Memory*

BG - *Bit Generator*

PRM - *Probability Memory*

BM - *Bit Memory*

EFM - *Elite Fitness Memory*

CMP - *Comparator*

FEV - Fitness EValuation

MO - *Memory Output*

CV - *Convergence Verify*

PSO - *Particle Swarm Optimization*

RGB - *Red, Green and Blue*

ABE - Algoritmo de Busca Exaustiva

Fit. - *Fitness*

Aval. - Número de Avaliações de *Fitness*

RESUMO

O AG (Algoritmo Genético) é uma ferramenta eficiente para problemas de otimização, mas, em muitas aplicações, o custo computacional do AG pode ser alto demais ou pode demandar muitos recursos de *hardware*. O AGC (Algoritmo Genético Compacto) imita um AGS (Algoritmo Genético Simples) reduzindo a demanda por recursos através da utilização um vetor de probabilidades no lugar da população. Aliado a isto, o AGC é um algoritmo recente, introduzido em 1999, e não completamente explorado. Desta forma, este trabalho propôs um novo operador de mutação para o AGC com elitismo, denominado emCGA (*elitism with mutation Compact Genetic Algorithm*). Observou-se o desempenho deste novo algoritmo usando diferentes valores de probabilidade de mutação em diferentes problemas de *benchmark*. O emCGA pode alcançar melhor qualidade de solução com um menor custo computacional e utilizando um tamanho menor de população, se comparado com outros algoritmos semelhantes encontrados na literatura, isto é: neCGA (*Nonpersistent Elitism Compact Genetic Algorithm*) e mCGA (*Compact Genetic Algorithm with elitism and Mutation*). Ainda, o emCGA foi aplicado ao problema de detecção de objetos em imagens, onde pode-se observar que este algoritmo pode alcançar bons resultados em problemas reais, considerando o compromisso entre qualidade de solução e custo computacional. Aliando-se ao ganho de desempenho e à reduzida demanda por *hardware* na implementação em lógica reconfigurável, recomenda-se o uso do emCGA em problemas reais que demandem um algoritmo compacto e eficiente.

ABSTRACT

The Genetic Algorithm is an efficient search technique used in computing for optimization and search problems. However, in many applications, where the small size and power efficiency are critical design considerations, its computational cost cannot be adequate. The Compact Genetic Algorithm was introduced in 1999 and can mimic a Simple Genetic Algorithm saving memory, and thus, power and space by representing the population with a probability vector rather than as sets of bit strings. Additionally, it is a recent algorithm and has a lot of room to be explored. This work proposes a new mutation operator for Compact Genetic Algorithm using elitism, named emCGA (elitism with mutation Compact Genetic Algorithm). The performance of this new algorithm is tested with different mutation probabilities in different benchmark problems. The emCGA could overcome the performance, considering the tradeoff between quality of solution and computation cost, of other Compact Genetic Algorithms described in the literature, i.e. neCGA (Non-persistent Elitism Compact Genetic Algorithm) and mCGA (Compact Genetic Algorithm with Elitism and Mutation). The emCGA is applied for the object detection in images, and its good performance suggests that emCGA could be applied for real problems. The emCGA is also implemented in reconfigurable hardware. This implementation demonstrates a significant speedup comparing to a conventional software implementation. Comparing to the original hardware Compact Genetic Algorithm, this new hardware implementation does not demand much more resources. Hence, the emCGA can be recommended for applications in real problems where the small size and power efficiency are critical design considerations.

CAPÍTULO 1

INTRODUÇÃO

1.1 MOTIVAÇÕES

O AG (Algoritmo Genético) é uma ferramenta eficiente em problemas de otimização em diversas áreas da Engenharia. Contudo, em muitas aplicações, o custo computacional do AG pode ser alto demais, exigindo um tempo de execução proibitivo ou ainda demandando muitos recursos de *hardware* para sua implementação. Uma possível alternativa é buscar um algoritmo genético com menor complexidade computacional. Este algoritmo simplificado pode ser executado em sistemas menos poderosos e ainda, caso o problema demande, ser implementado em arquiteturas dedicadas paralelas com a utilização de dispositivos lógicos reconfiguráveis por *hardware*.

Uma possível limitação na implementação de um AG é a utilização da memória para armazenar a população. Isto é particularmente verdade em uma implementação em *hardware*. No AG a busca é feita sobre um conjunto de soluções (a população) e não somente em uma única solução. Outra limitação importante é que a maioria das técnicas computacionais utilizadas nos AGs exige muito recursos se aplicadas técnicas de paralelismo. Estas técnicas foram inicialmente desenvolvidas utilizando processadores seqüenciais, que possuem mais recursos. Desta forma, restringindo sua aplicação em arquiteturas paralelas, no que se diz respeito à demanda por recursos, particularmente em *hardware* reconfigurável.

Neste contexto, o AGC (Algoritmo Genético Compacto) imita os operadores genéticos do AGS (Algoritmo Genético Simples), demandando entre outras coisas, menos memória para armazenar a população. Isto se deve ao fato do AGC, em vez de manipular a população inteira, manipular um vetor de probabilidades. Além disto, o AGC utiliza técnicas computacionais que atualizam o vetor de probabilidades na direção dos melhores indivíduos imitando o comportamento do AGS. Devido à simplicidade destas técnicas e a baixa utilização de memória, alguns trabalhos propuseram a sua implementação em *software* e em *hardware*, mostrando bons resultados com significativa redução dos recursos necessários para sua construção.

Aliado a isto, o AGC é um algoritmo recente, com poucos trabalhos publicados e não completamente explorado. Isto permite apontar o AGC como um interessante objeto de estudo, particularmente na análise das suas vantagens e limitações, escolha adequada de parâmetros de operação e problemas adequados à sua aplicação.

1.2 OBJETIVOS

O objetivo geral deste trabalho é estudar e propor um algoritmo genético compacto (AGC), implementar o mesmo em *software* e *hardware* e aplicar a problemas de *benchmark* visando determinar o seu desempenho em termos de melhor qualidade de solução e custo computacional.

São objetivos específicos deste trabalho:

- Levantar referências bibliográficas que indiquem o estado da arte do AGC. A partir deste estudo, propor um algoritmo modificado e analisar seu desempenho em comparação com outros algoritmos semelhantes.
- Implementar o algoritmo proposto em *software*.
- Implementar o algoritmo proposto em *hardware*, particularmente usando lógica reconfigurável por *hardware*.
- Executar experimentos com o algoritmo implementado visando verificar desempenho e demanda por recursos.
- Aplicar o algoritmo proposto na solução do problema de detecção de objetos em imagens, por tratar-se de um problema complexo e com aplicações importantes no mundo real.

1.3 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada em cinco capítulos. No Capítulo 2 faz-se uma revisão da literatura sobre o AG e o AGC, bem como sobre Computação Reconfigurável. O Capítulo 3 descreve em detalhes o desenvolvimento da metodologia proposta. No Capítulo 4 relatam-se os experimentos e resultados obtidos. E, finalmente, o Capítulo 5 apresenta a discussão dos resultados, as conclusões do trabalho e as propostas de trabalhos futuros.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

2.1 COMPUTAÇÃO EVOLUCIONÁRIA

A CE (Computação Evolucionária) é um ramo da IC (Inteligência Computacional) que utiliza técnicas inspiradas na evolução natural. As primeiras técnicas de CE surgiram na década de 60 e 70 introduzidas por Lawrence J. Fogel com PE (Programação Evolucionária), por John Henry Holland com AG (Algoritmo Genético) (HOLLAND, 1975) e por Ingo Rechenberg e Hans-Paul Schwefel com EE (Estratégias Evolutivas). A partir da década de 70 houve maior desenvolvimento nestas técnicas, mas somente no começo da década de 90 surgiu o conceito de que todas eram diferentes abordagens da mesma tecnologia- no caso, a CE.

2.1.1 Algoritmo Genético

O AG (Algoritmo Genético) é um algoritmo de busca categorizado como heurística global. O AG é uma classe particular da Computação Evolucionária, e inspirado no mecanismo de seleção natural e na genética, isto é: hereditariedade, seleção natural, mutação e recombinação (*crossover*).

Em síntese, o AG é uma simulação computacional de um modelo de sistema natural, onde os indivíduos de uma população evoluem no decorrer das gerações através de um processo repetitivo de seleção e reprodução. A população é um conjunto de indivíduos, também chamados de criaturas ou fenótipos. Os indivíduos possuem um ou mais cromossomos, que constituem o genótipo ou genoma, e são uma representação abstrata de uma solução. Tradicionalmente, os cromossomos são representados por um vetor de bits, mas outras codificações também são possíveis. Numa representação binária, cada bit do cromossomo é denominado alelo, onde um conjunto de alelos representando uma variável do problema é denominado de gene. Estas denominações são ilustradas na figura 1. A evolução dos indivíduos se inicia com uma população gerada aleatoriamente. No decorrer das gerações,

os indivíduos pais são avaliados e alguns deles são selecionados para se reproduzirem e gerarem uma população de descendentes. Na seleção fica evidenciado o princípio Darwiniano da sobrevivência do mais bem adaptado, sendo privilegiados os indivíduos com melhor aptidão. A aptidão dos indivíduos é medida por uma função de *fitness*. Os indivíduos selecionados se reproduzirão, gerando uma população de descendentes através de operadores genéticos, como por exemplo: *crossover* e mutação. Este processo de seleção e reprodução se repetirá até satisfazer um determinado critério de parada, ou seja: alcançar um determinado número máximo de gerações ou um determinado valor de *fitness*.

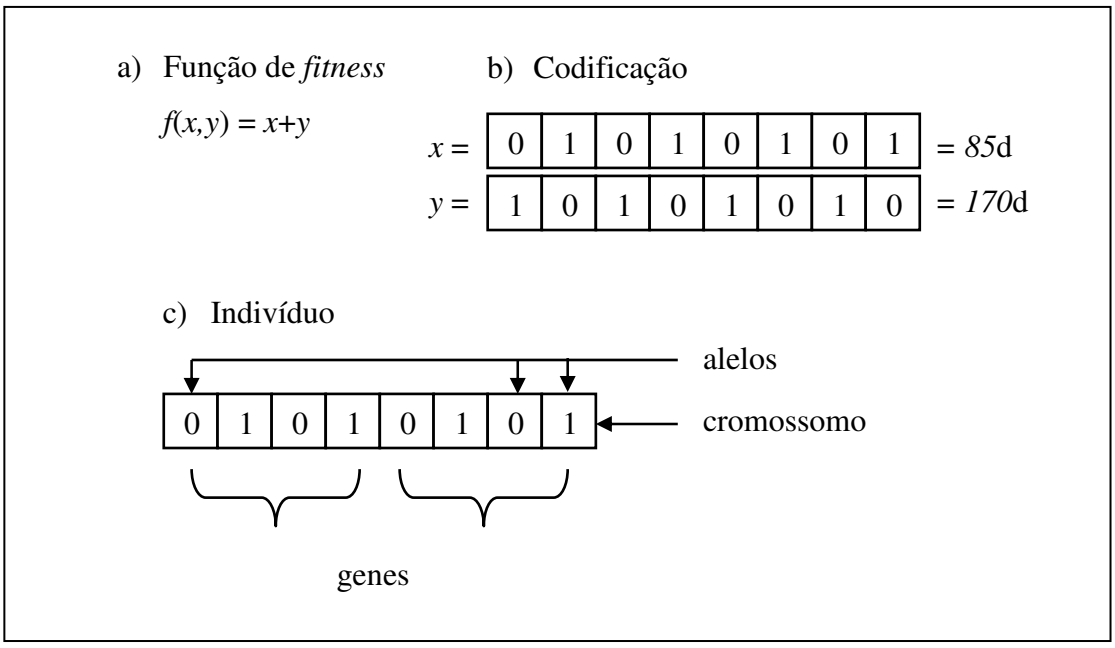


Figura 1: Ilustração das denominações comuns para o AG. Na figura *a* é exemplificado uma função de *fitness* com duas variáveis. A codificação das variáveis desta função é ilustrada na figura *b*. A figura *c* ilustra as denominações dos componentes de um indivíduo.

Os Algoritmos Genéticos diferem dos algoritmos tradicionais de otimização em basicamente quatro aspectos (GOLDBERG, 1989):

- Baseiam-se em uma codificação dos parâmetros da otimização, e não nos parâmetros em si;
- A busca é realizada sobre uma população de soluções, e não sobre uma solução única;
- Não necessitam de nenhum conhecimento derivado do problema, apenas de uma forma de avaliação do resultado;
- Usam transições probabilísticas e não regras determinísticas.

A primeira monografia com o assunto AG foi *Adaptation in Natural and Artificial Systems* apresentado por John Holland (1975). O autor iniciou seu trabalho em AG no início da década de 60 e teve dois objetivos: aumentar o entendimento do processo de adaptação natural e desenvolver um sistema artificial que tem propriedades similares ao sistema natural (GOLDBERG, 1989). Os resultados obtidos neste trabalho impulsionaram diversas pesquisas que aprimoraram e ainda aprimoram o AG. Desta forma, se encontram aplicações para o AG em diversas áreas, por exemplo: ciência da computação, engenharia, economia, química, manufatura, matemática e física, entre outras áreas.

2.1.1.1 Algoritmo Genético Simples

O AGS (Algoritmo Genético Simples) define os mecanismos básicos de um AG e foi definido por David E. Goldberg (1989). Este algoritmo é essencialmente simples e não envolve nada muito mais complexo do que copiar e mover vetores de bits. Mas nesta simplicidade se esconde um algoritmo com grande poder de busca, o que torna este algoritmo muito atrativo.

Antes de se iniciar o AGS é necessário ter definidas uma forma de representação genética da solução e uma função de *fitness*. A forma de representação do AGS é um vetor de bits de tamanho fixo, e a função de *fitness* é uma função que recebe uma das possíveis soluções e avalia a sua qualidade. Estas definições são dependentes do problema e são críticas para o desempenho do AGS.

O tamanho da população é pré-determinado e normalmente não muda no decorrer das gerações. Logo, é um parâmetro que deve ser definido antes de se iniciar o algoritmo. A sua escolha afeta o desempenho do AGS e, dependendo da natureza do problema, pode ser desde centenas até milhares de indivíduos.

Inicialmente um grupo de indivíduos é gerado para formar uma população inicial. Tradicionalmente estes indivíduos são gerados aleatoriamente, cobrindo uniformemente o espaço de busca. No entanto, quando é possível prever regiões do espaço de busca onde é mais provável encontrar a solução ótima, podem ser inseridos alguns indivíduos que contêm soluções nestas regiões.

A cada geração, os melhores indivíduos avaliados pela função de *fitness* são selecionados para participar do processo de reprodução. Este processo de seleção é inspirado

no principio da seleção natural Darwiniana, onde os indivíduos com melhores valores de *fitness* têm mais probabilidade de serem selecionados. Os métodos de seleção mais populares são roleta e torneio estocástico. Na seleção roleta os indivíduos são selecionados aleatoriamente com probabilidade proporcional ao seu valor de *fitness*. No método de seleção por torneio, um pequeno grupo de indivíduos escolhidos aleatoriamente na população compete entre si. O indivíduo que tiver o melhor *fitness* é o vencedor e será selecionado para participar do processo de reprodução.

No processo de reprodução, os indivíduos selecionados terão seus cromossomos modificados através de operadores genéticos- como *crossover* e mutação. Dois indivíduos pais dão origem a dois indivíduos descendentes, que compartilharão de muitas das características dos pais, simulando, assim, o processo natural de reprodução sexuada. Este processo se repetirá até que a nova população seja gerada. Existem diversas técnicas de *crossover*, e as mais comuns são: *crossover* de um ponto, *crossover* de dois pontos e *crossover* uniforme. No *crossover* de um ponto uma posição nos cromossomos pais é selecionada aleatoriamente, através da qual os descendentes são gerados com genes trocados a partir desta posição. No *crossover* de dois pontos duas posições são selecionadas aleatoriamente e os descendentes são gerados trocando os genes entre estas posições. O *crossover* uniforme gera os descendentes trocando os genes dos pais com uma probabilidade determinada, tipicamente 0,5. Estas técnicas de *crossover* são ilustradas na figura 2.

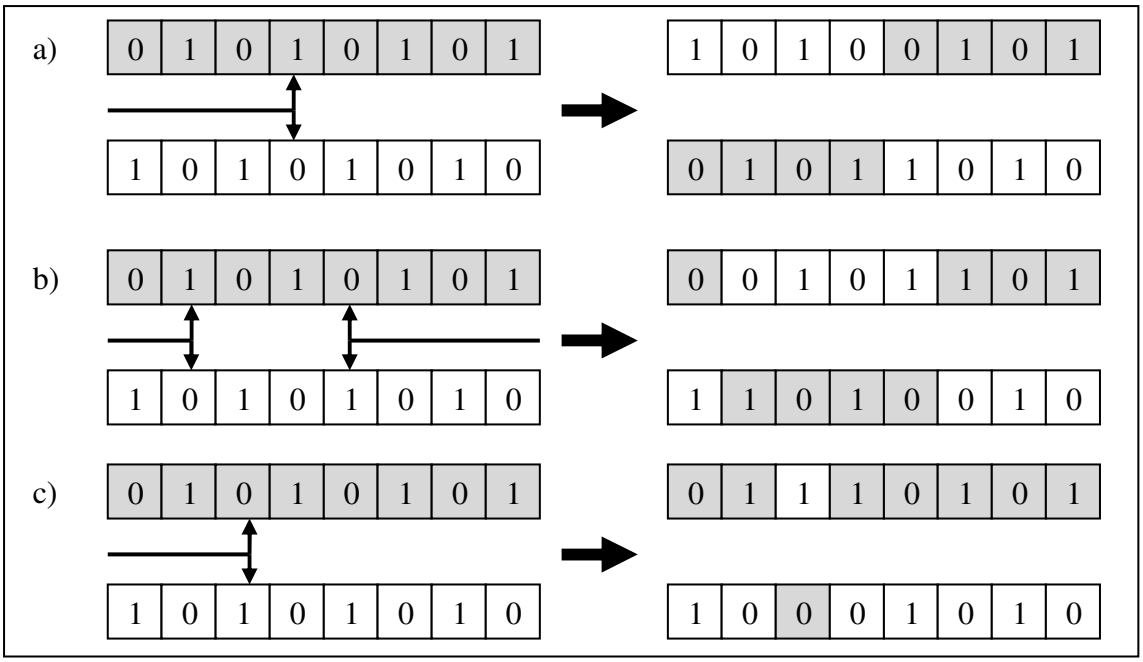


Figura 2: Exemplo de operadores de *crossover*. O *crossover* de um ponto é exemplificado na figura *a*. Na figura *b* é exemplificado o *crossover* de dois pontos. Um exemplo de *crossover* uniforme é ilustrado na figura *c*.

O AGS repete o processo de seleção e reprodução até atingir um critério de parada. Alguns exemplos de critérios de paradas são: uma solução alcançar um determinado valor de *fitness* ou alcançar um determinado número de gerações. Outros critérios- ou até a combinação de mais de um critério- também podem ser utilizados.

2.1.1.2 Hipótese dos Blocos Construtivos (*Building Blocks Hypothesis*)

Apesar da simplicidade de implementação de um AG, seu comportamento é de difícil compreensão. Em particular, é de difícil compreensão o porquê destes algoritmos normalmente gerarem soluções com um bom *fitness*. Alguns conceitos foram definidos para buscar esta compreensão, como *schema* (plural, *schemata*) e BC (Bloco Construtivo).

Um *schema* é uma representação capaz de descrever diversos cromossomos (HOLLAND, 1968); (HOLLAND, 1975). Considere uma representação binária, na qual o cromossomo é um vetor de bits, isto é, os alelos são cada bit deste vetor. Desta forma, cada alelo é definido sobre um alfabeto binário {0,1}. Um conjunto destes alelos formando uma

variável do problema é definido como sendo um gene. Se o alfabeto dos alelos for estendido para {0,1,*}, onde * significa “tanto faz”, e são criados vetores com este novo alfabeto, então estes vetores são denominados como *schema*. Por exemplo, se uma determinada população possui indivíduos com cromossomo de comprimento de 7 bits, então um *schema* 011*1** representará os cromossomos {0110100, 0110101, 0110110, 0110111, 0111100, 0111101, 0111110 e 0111111}.

No entanto, nem todos *schemata* são iguais, podendo ser diferenciado pela ordem e pelo comprimento definitório (*defining length*). A ordem quantifica o quão específico é um *schema*, sendo definida como o número de genes representados por um valor fixo, isto é {0,1}. Se compararmos o *schema* 011*1** e o 0******, o primeiro, que possui ordem 4, é mais específico do que o segundo, que possui ordem 1. Por sua vez, o comprimento definitório quantifica a extensão da definição do *schema*, definido como a distância entre o primeiro e o último valor fixo. Desta forma, o *schema* 1****1*, que possui comprimento definitório 5, se estende mais do que 1*1****, que possui comprimento definitório 2.

Os *schemata* que possuem comprimento definitório curto, baixa ordem e valores de *fitness* altos foram denominados por David E. Goldberg (1989) como Blocos Construtivos (BCs). A definição dos BCs foi inspirada pelo Teorema do *Schema* de Holland, apresentado por John Holland (1975). Este teorema mostra que na seleção, os *schemata* que possuem comprimento definitório curto, baixa ordem e valores de *fitness* acima da média aumentam o seu número exponencialmente durante execuções consecutivas de um AG.

A Hipótese dos Blocos Construtivos procura explicar o comportamento de um AG (GOLDBERG, 1989). Esta hipótese diz que um AG busca um desempenho quase-ótimo por meio de justaposição de BCs. No entanto, críticas a esta hipótese são levantadas, como em (WRIGHT, VOSE e ROWE, 2003), onde os autores argumentam que falta embasamento teórico para a hipótese em questão e que, em alguns casos, a mesma é incoerente. Esta incoerência pode ser exemplificada em um trabalho anterior, onde foi realizado um estudo do desempenho do *crossover* uniforme (SYSWERDA, 1989). O autor mostra que a técnica de *crossover* uniforme é extremamente ruidosa em *schemata* com comprimento definitório curto, diferentemente do que acontece com as técnicas de *crossover* de um ponto e *crossover* de dois pontos. Estas últimas possuem menos ruído, pois possuem maior probabilidade de conservar os *schemata* com comprimento definitório curto e de combinar seus bits fixos nos descendentes.

Este debate demonstra que a definição de como o AG alcança soluções com bons valores de *fitness* ainda está longe de ser completamente estabelecida. Com este objetivo, várias pesquisas foram e continuam a ser feitas, e são apresentadas novas propostas de como entender o AG ou seus operadores.

2.1.2 Algoritmo de Estimação de Distribuição

O AED (Algoritmo de Estimação de Distribuição) é um novo campo na área da Computação Evolucionária, onde os clássicos operadores *crossover* e mutação são substituídos por amostragens baseadas em uma distribuição probabilística, estimada de um grupo de indivíduos selecionados na população (LARRAÑAGA E LOZANO, 2002). A população inicial de um AED é gerada aleatoriamente como em um AG. A cada geração, um grupo de indivíduos é selecionado, o qual é usado para estimar uma distribuição probabilística. A partir desta distribuição são gerados indivíduos descendentes que irão substituir a população atual, ou parte dela. Este processo de seleção e geração é repetido até que um critério definido seja alcançado.

O primeiro AED foi apresentado em (MÜHLENBEIN e PAAß, 1996), e desde então diversos AEDs foram apresentados, bem como suas aplicações. Os AEDs podem ser divididos pelos tipos de modelo de distribuição probabilística empregado, isto é, discreto e contínuo, e do tipo de dependência das variáveis deste modelo, ou seja, sem dependência, dependência bivariável e múltipla dependência.

Estes modelos probabilísticos são baseados em modelos probabilísticos gráficos como as redes Bayesianas e as redes Gaussianas. As redes Bayesianas aplicam-se em modelos probabilísticos discretos e as redes Gaussianas em modelos contínuos. Nestas redes são grafos acíclicos direcionados (*directed acyclic graph*) que representam um grupo de variáveis e suas interdependências. As variáveis são representadas pelos nodos do grafo acíclico direcionado e as suas dependências condicionais são codificadas pelos arcos. Estes modelos podem ter desde nenhuma dependência, ou até mesmo múltiplas dependências. Os modelos sem dependência entre os nodos são conhecidos como AED sem dependência. Na figura 3a é ilustrado o grafo acíclico direcionado de um AED sem dependência, onde os círculos representam os nodos. Nos AEDs com dependência entre pares de variáveis, onde é suficiente considerar modelos probabilísticos de segunda ordem, são denominados de AEDs com

dependência bivariável. A figura 3b apresenta o grafo acíclico direcionado de um AED com dependência bivariável, onde, além das variáveis, também é representada os arcos entre os nodos através de uma seta. No modelo mais complexo, onde distribuições de probabilidade que requerem modelos estatísticos de ordem mais alta que segunda, são denominados como AED com múltipla dependência e um exemplo é ilustrado na figura 3c.

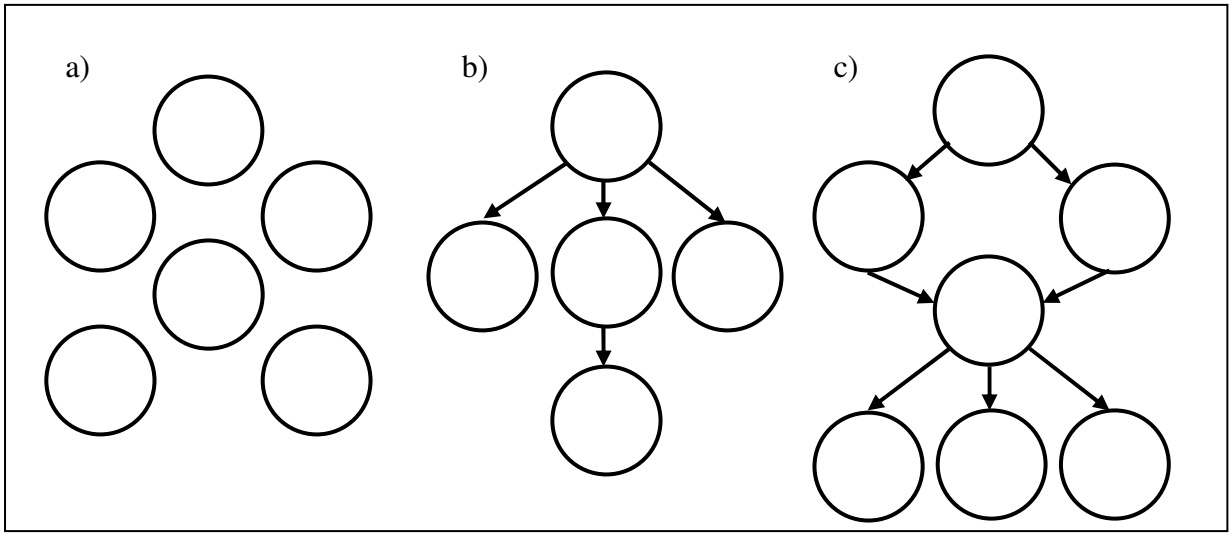


Figura 3: Exemplos de conexões do modelo de distribuição da probabilidade dos AEDs. No modelo *a* é sem dependência entre as variáveis. No modelo *b* é um exemplo com dependência bivariável. Um exemplo de modelo com múltiplas dependência é ilustrado em *c*.

Neste trabalho é tratado do AGC (Algoritmo Genético Compacto) (HARIK, LOBO e GOLDBERG, 1998), que é um AED com modelo discreto e sem dependência. Podem-se citar outros algoritmos na mesma classificação, como o PBIL (*Population Based Incremental Learning*) (BALUJA, 1994) e (BALUJA e CARUANA, 1995) e o UMDA (*Univariate Marginal Distribution Algorithm*) (MÜHLENBEIN, 1997). As principais diferenças entre estes algoritmos são apresentadas na tabela 1. O UMDA é muito tratado em estudos de AEDs sem dependência. Contudo, o AGC oferece características intrínsecas que permitem uma melhor utilização em implementações com limitações de *hardware* e memória.

Tabela 1. Principais diferenças entre os AEDs sem dependência

Algoritmo	População	Reprodução
UMDA	Conjunto de cromossomos	Baseada na freqüência que os alelos aparecem nos melhores indivíduos selecionados
PBIL	Conjunto de cromossomos e vetor de probabilidades	Baseada no vetor de probabilidades que é atualizado pela freqüência que os alelos aparecem nos melhores indivíduos selecionados
AGC	Vetor de probabilidades	Baseada no vetor de probabilidades que é atualizado na direção do melhor indivíduo

2.1.3 Algoritmo Genético Compacto

Dentre os parâmetros de um AG, a importância do tamanho da população no desempenho do AG foi assunto de diversas pesquisas. A primeira equação procurando resolver o problema da escolha do tamanho da população foi apresentada por David E. Goldberg e Mike Rudnick (1991). Esta equação se baseou em um método de cálculo da variância do *fitness* de um *schema*, também denominada como ruído colateral. Mais tarde, esta equação foi melhorada considerando outras fontes de ruídos, para aumentar a precisão estatística considerando as decisões feitas nas discriminações entre BCs (GOLDBERG, DEB e CLARK, 1992). Este modelo de decisão ainda foi estendido por George Harik, Erick Cantus-Paz, David E. Goldberg *et al* (1999), pois se BCs errados forem escolhidos nas primeiras gerações, BCs corretos podem nunca mais ser recuperados (HARIK, CANTU-PAZ, GOLDBERG *et al*, 1999 e SASTRY, GOLDBERG, 2001). A equação com este novo modelo de decisão ainda foi melhorada e generalizada por Chang W. Ahn e Rudrapatna S. Ramakrishna (2002) utilizando o modelo *gambler ruin*. O estudo por Jun He e Xin Yao (2002) procurou entender a importância real da população em EEs, mostrando que a inserção de população aumenta a probabilidade de primeiro acerto (*first hitting probability*), e que a população pode causar forte impacto no tempo de computação de um EE. Estas pesquisas deixam evidente a importância da população no desempenho de um AG. Normalmente um tamanho maior de população resulta em melhor desempenho na qualidade da solução, mas aumenta o custo de computação e utilização de memória.

Devido a sua importância no desempenho de um AG, diversos trabalhos procuram aperfeiçoar a representação da população. No trabalho de Ackley (1987) foi apresentado um

AG híbrido que usa uma representação baseada em um vetor de genes. Este vetor é usado em um algoritmo de busca local, o qual é manipulado através de uma realimentação positiva ou negativa proveniente dos membros da população. Para explicar este algoritmo, o autor usou uma metáfora política onde os eleitores (membros da população) expressam sua satisfação ou insatisfação para cada um dos *l* membros do governo (cada elemento do vetor de genes). Em outro trabalho proposto por Syswerda (1993), o autor introduz o operador chamado BSC (*Bit-Based Simulated Crossover*), onde a idéia é gerar descendentes através de informações estatísticas da população. O BSC usa um vetor com médias ponderadas para cada bit do cromossomo dos indivíduos na população. No trabalho apresentado por Eshelman e Schaffer (1993), os autores também discutem variações do BSC para investigação de como AGs diferem de *Population-Based hillclimbers*.

Nos trabalhos apresentados por Baluja (1994) e por Baluja e Caruana (1995) foi proposto o PBIL. Diferentemente do BSC, a população no PBIL é representada através de um vetor de probabilidades, onde o valor de cada elemento é a probabilidade de um gene ser 1. Inicialmente, o valor de cada elemento deste vetor é igual 0.5, ou seja: a probabilidade de cada gene ser 1 é igual à probabilidade de ser 0. Em cada geração um certo número de soluções é gerado baseado neste vetor de probabilidades. Uma regra de atualização usa estas soluções para mover cada probabilidade do vetor na direção de 1 ou 0. Esta regra de atualização é semelhante à usada no LVQ (*Learning Vector Quantization*) (HERTZ, KROGH e PALMER, 1993), sendo seu objetivo fazer com que a probabilidade se mova na direção dos melhores indivíduos. No entanto, foram feitas tentativas sem sucesso de relacionar os parâmetros do AG e PBIL, isto é: o número de indivíduos gerados, números de indivíduos para ser atualizado, critério de parada e taxa de modificação do vetor de probabilidade. Isto se deve ao fato de que o AG manipula sua distribuição de um jeito diferente (HARIK, LOBO e GOLDBERG, 1999).

O AGC foi porposto por Harik, Lobo e Goldberg (1999). O AGC é um AED que gera descendentes através de um modelo estatístico da população de pais, em vez de usar os tradicionais operadores de recombinação e mutação (TSUTSUI, 2002 e LARRAÑAGA e LOZANO, 2002). Este modelo estatístico foi inspirado pelo modelo *random walk* apresentado em (HARIK, CANTU-PAZ e GOLDBERG *et al*, 1999). Neste trabalho, os autores fizeram uma estimativa precisa do tempo de convergência para uma classe especial de problemas no AG: BC não sobrepostos de dificuldade delimitada (*non-overlapping BCs of bounded*

difficult). Nestes problemas não existem iterações entre BCs. Desta forma, os autores especularam que os BCs poderiam ser resolvidos independentemente, focando em um BC por vez. O modelo *random walk* foi utilizado como uma ferramenta matemática que modela o processo de espalhamento dos BCs corretos pela população durante a execução de um AG. Neste modelo existe uma variável *random walk* que corresponde ao número de BCs corretos em uma população em um determinado tempo, o que equivale dizer que possui valores entre 0 e o tamanho da população. A probabilidade do AG cometer um erro de decisão em uma competição entre dois *schemata* é a probabilidade de transição do *random walk*. Este erro de decisão existe porque cada *schema* é avaliado em um contexto de indivíduo longo- o AG pode cometer um erro de decisão por ruído gerado pelos outros *schema*s. Assim, a população neste modelo é uma memória do contador destes erros de decisões.

A idéia do AGC foi simular *l* destes modelos *random walks* independentes para cada bit do cromossomo, onde *l* é o tamanho do cromossomo. Desta forma, o AGC pode imitar o comportamento de um AGS com *crossover* uniforme usando uma quantidade reduzida de memória.

Apesar dos resultados com AGC mostrarem que este imita o funcionamento do AGS para problemas de BCs de ordem baixa, o mesmo não se pode constatar para ordens mais altas. Neste caso, o motivo é a forma que o AGC desconecta totalmente os diferentes bits do cromossomo, que é semelhante ao *crossover* uniforme. Com a forma compacta de representar a população no AGC, as informações sobre correlações de alta ordem entre bits não conseguem sobreviver através das gerações, diferentemente do que acontece no AGS. Para problemas reais, estas perturbações geradas pelo *crossover* uniforme devem ser superadas, pois muitos possuem ótimos locais (multimodal) e genes interdependentes (BCs de ordem alta).

No entanto, exercer um aumento da pressão seletiva no AGC ameniza estes efeitos ruidosos, pois aumenta a probabilidade de BCs de alta ordem sobreviverem através das gerações (HARIK, LOBO e GOLDBERG, 1999) e (AHN e RAMAKRISHNA, 2003). Ou seja, o aumento da pressão seletiva pode realizar a função da memória e abrindo espaço para melhorias no desempenho do AGC.

2.1.3.1 Detalhes de implementação do AGC

Resumidamente, a cada geração do AGC um vetor de probabilidades é atualizado na direção do vencedor de um torneio entre dois indivíduos. Como o AGC utiliza a representação binária para o cromossomo, cada um dos elementos deste vetor representa a probabilidade de um *alelo* ser 1. Por sua vez, esta probabilidade é usada para gerar os genes do cromossomo dos indivíduos que participam do torneio. Tradicionalmente, o critério de parada utilizado nos AGCs é a convergência do vetor de probabilidades. Desta forma, no decorrer das gerações, este vetor de probabilidades evolui até convergir, isto é: quando todos os elementos deste vetor atingirem probabilidade 0% ou 100%.

Na verdade, o vetor de probabilidade representa toda a população usada no AG, reduzindo a demanda de memória usada para armazenar a população. As abordagens convencionais do AG demandam um tamanho de memória de $n*l$ bits, onde n é o tamanho da população e l é o comprimento do cromossomo (número de bits). No caso do AGC, os elementos do vetor de probabilidades é discretizado com resolução de $1/n$. Assim, a sua demanda por memória é somente $l*\log_2(n)$. Esta característica torna o AGC bastante vantajoso para implementações em *hardware*, em especial no que diz respeito a recursos físicos necessários e velocidade de processamento.

A seleção e a reprodução no AG são imitadas pelos efeitos da geração de indivíduos e atualização do vetor de probabilidades no AGC (HARIK, LOBO e GOLDBERG, 1999). Desta forma, pode-se relacionar a seleção com a etapa de atualização do vetor de probabilidades e a reprodução com a geração de indivíduos.

Em um AG convencional fica evidenciado na seleção o princípio Darwiniano da sobrevivência do indivíduo mais bem adaptado, sendo privilegiados os que têm melhor *fitness*. Mas não significa que isto sempre vale para o gene, porque o gene é avaliado sob um contexto de um indivíduo com um cromossomo longo. Para exemplificar esta afirmação considera-se um problema *OneMax* com indivíduos com um cromossomo de comprimento de 5 alelos (5 bits) e dois indivíduos com cromossomo 10101 e 01000. A função de *fitness* deste problema é a soma de alelos iguais a 1. Desta forma, o valor de *fitness* do indivíduo com cromossomo 10101 é 3 e do indivíduo com cromossomo 01000 é 1. No problema *OneMax* procura-se maximizar o número de alelos com valores iguais a 1. No entanto, considerando estes dois indivíduos, a seleção no AG decidirá erroneamente por dar vantagem ao indivíduo com o segundo alelo igual a 0. Neste momento que a população é importante no AG, pois

pode armazenar indivíduos de forma a ser robusto a certo número de erros de decisões no decorrer da seleção.

A regra de atualização do vetor de probabilidades no AGC é equivalente a um esquema de seleção *steady-state binary tournament* (HARIK, LOBO e GOLDBERG, 1999). Considere um AG onde são realizados torneios entre dois indivíduos retirados aleatoriamente de uma população para gerar a população descendente. Nesta população descendente serão mantidas duas cópias do vencedor de cada torneio na população de pais. Logo, em uma população de tamanho *n* o número de genes do vencedor aumentará a uma taxa de *1/n*. A regra de atualização do vetor de probabilidade funciona da mesma forma. A probabilidade de cada elemento do vetor será alterada com uma taxa de *1/n*, de forma a imitar a substituição os genes do cromossomo do indivíduo perdedor pelos do vencedor na população.

Outra etapa do algoritmo é a reprodução, tipicamente representada no AG pela aplicação de operadores genéticos, usualmente *crossover* e mutação. O *crossover* é um operador que recombina partes dos cromossomos pais para gerar os descendentes. Aplicando repetidamente os operadores de *crossover* mais comuns leva-se a uma desconexão dos genes da população. Neste estado desconectado, a população pode ser mais compactamente representada como um vetor de probabilidades (HARIK, LOBO e GOLDBERG, 1999). Neste caso, a etapa de geração funciona de maneira análoga a um operador de *crossover* uniforme. A mutação, operador comum em AGs, não é utilizada no AGC original.

A condição de parada típica para um AG é o número máximo de gerações. No AGC a condição de parada típica é a convergência do vetor de probabilidades. O pseudocódigo do AGC é mostrado na figura 4.

```
1) Inicialização: Iniciar o vetor de probabilidade
Para i:= 1 até l faça p[i]:=0.5;

2) Geração: Gerar dois indivíduos através do vetor de probabilidade
a:=gerar(p);
b:=gerar(p);

3) Competição: Os indivíduos são avaliados e competem segundo um valor de fitness
a_fitness:=avaliar(a);
b_fitness:=avaliar(b);
Se a_fitness é melhor que b_fitness Então
   vencedor:=a;
   perdedor:=b;
Senão
   vencedor:=b;
   perdedor:=a;

4) Atualização: Atualizar o vetor de probabilidade na direção do vencedor
Para i:= 1 até l faça
  Se vencedor[i] ≠ Perdedor[i] Então
    Se vencedor[i] = 1 Então  p[i] := p[i] + 1/n;
                      Senão p[i] := p[i] - 1/n;

5) Verificação: Verificar se o vetor convergiu
Para i:=1 até l faça
  Se p[i]>0 e p[i]<1 Então
    Vá para o passo 2;

6) Finalização: p representa a solução final

Parâmetros:
n: Tamanho da população
l: Comprimento do cromossomo em bits
```

Figura 4: Pseudocódigo do AGC.

2.1.3.2 Extensões do AGC

O AGC se mostrou uma proposta interessante de solução para redução de custo de memória e custo computacional para os AGs, critérios normalmente críticos em aplicações em *hardware*. No entanto, sua aplicabilidade a problemas reais é limitada. Desta forma foram feitos diversos trabalhos com a intenção de melhorar o sem desempenho e torná-lo aplicável em problemas reais.

No trabalho apresentado por Harik (2000) foi proposto o ECGA (*Extended Compact Genetic Algorithm*), onde o AGC foi melhorado usando *Linkage Learning* (HARIK e GOLDBERG, 1997). Neste trabalho a população foi substituída por um modelo probabilístico chamado *marginal product model* (MPM) que é formado utilizando o modelo de busca

minimum description length (MDL). O MPM possui a vantagem de poder representar a distribuição probabilística de mais de um gene por vez. Os produtos das distribuições marginais em uma partição podem prover um *direct linkage map* com todos os BC separando os genes que são firmemente ligados (SASTRY e GOLDBERG, 2000). O ECGA alcança um melhor desempenho do que o AGS para problemas difíceis, e com menor tempo de convergência, isto é: um menor número de avaliações de *fitness*. Mas para isto ele exige uma maior utilização de memória e um maior custo computacional por avaliação de *fitness*, pois o MPM exige a identificação de um problema restrito de otimização em cada geração. Desta forma, uma implementação em *hardware* do ECGA é complicada exigindo mais recursos de *hardware* (APORNTEWAN e CHONGSTITVATANA, 2001).

No trabalho de Baraglia, Hidalgo e Perego (2001) foi introduzido um algoritmo heurístico híbrido que combina AGC com um algoritmo de busca local LK (*Lin-Kerninghan*), que foi denominado AGC-LK. A idéia foi lidar com problemas de otimização com BC de ordem alta, como o problema do caixeiro viajante, sem necessitar de um aumento da utilização da memória. No AGC-LK, o algoritmo LK refina as soluções geradas pelo AGC, que, por sua vez, são exploradas para melhorar a qualidade do vetor de probabilidades. Apesar de melhorar a qualidade de solução, superando o AGC e o AGS, o AGC-LK tem um custo computacional muito alto empregando o algoritmo LK, se comparado com o AGC original.

O MBBCGA (*Mutated by Bit Compact Genetic Algorithm*) apresentado por Zhou, Meng e Qiu (2002) propôs uma extensão do AGC que utiliza técnicas de mutação e elitismo. O objetivo é extrair mais informações dos melhores indivíduos através de uma mutação bit a bit, e obter uma melhor convergência. A cada geração do MBBCGA, um indivíduo pai é gerado através do vetor de probabilidades. Este indivíduo pai gerará um descendente para cada mutação feita em cada alelo (bit) de seu cromossomo e participará de um torneio contra cada um destes descendentes, onde o vencedor deste torneio atualizará o elemento do vetor de probabilidades correspondente ao alelo da mutação. Depois, outro torneio será realizado entre cada descendente e o indivíduo elite, ou seja: o melhor indivíduo encontrado até então. Neste torneio, todos os elementos do vetor de probabilidades atualizarão com uma razão de $2/(n*l)$ na direção do vencedor. Apesar de mostrar empiricamente que esta regra de atualização aumenta a taxa de obtenção do ótimo global, não foi feita nenhuma justificativa formal do uso desta razão de atualização. O MBBCGA mostrou, através de resultados experimentais, que

alcança melhor desempenho em comparado com o AGC e o AGS usando seleção *binary tournament* e *crossover* uniforme com probabilidade de 0,5. Apesar de incentivar a investigação do uso de técnicas de mutação e elitismo, houve um aumento da complexidade de implementação no MBBCGA. Esta complexidade é suficiente para não torná-lo interessante em aplicações em *hardware*.

O trabalho de Hidalgo, Prieto e Lanchares *et al* (2003) apresentou uma implementação de AGC híbrido para resolver o problema de particionamento em vários FPGAs (*Field Programmable Gate Array*). Para melhorar o desempenho do AGC, foram usados AGCs em paralelo e um hibridismo com um algoritmo de busca local semelhante ao LK usado no AGC-LK (BARAGLIA, HIDALGO e PEREGO, 2001), mas adaptado ao problema tratado. Enquanto os usos dos AGCs em paralelo diminuem o tempo de execução, o algoritmo de busca aumenta a pressão seletiva. Resumidamente, este algoritmo executa os AGCs em paralelo, interrompendo a cada determinado número de iterações, quando um torneio é realizado. Neste torneio, dois indivíduos competem, onde um indivíduo é gerado através do vetor de probabilidades (que possui os alelos mais usados na população) e outro pelo algoritmo de busca local. Então o vetor de probabilidades é atualizado na direção do vencedor deste torneio.

Ahn e Ramakrishna (2003) apresentaram uma proposta de relação entre AGC e a técnica de elitismo. Os autores apresentaram dois AGCs baseados em elitismo que pertencem a uma mesma classe de AEDs, o peCGA (*persistent elitism Compact Genetic Algorithm*) e neCGA (*nonpersistent elistism Compact Genetic Algorithm*). O peCGA mostra uma relação entre AED e (1+1)EE, incorporando um modelo equivalente à (1+1)EE com mutação auto-adaptativa. Em síntese, o peCGA executa um torneio entre o melhor indivíduo encontrado até então, isto é o indivíduo elite, e um gerado através do vetor de probabilidades. Então, o vetor de probabilidades é atualizado na direção do vencedor. No entanto, o elitismo pode gerar uma pressão seletiva muito forte e levar a uma convergência prematura (DUMITRESCU, LAZZERINI, JAIN *et al*, 2000). Para evitar esta convergência prematura, o neCGA apresenta um controle de elitismo parametrizável. A diferença entre o peCGA e o neCGA é que este último possui um parâmetro relativo ao número máximo de gerações que o indivíduo elite pode sobreviver. Após este número de gerações, o indivíduo elite é substituído por um indivíduo gerado aleatoriamente.

O trabalho de Gallagher, Vigraham e Kramer (2004) apresentou uma família de extensões, objetivando o uso de uma extensão do AGC em aplicações em *hardware* para problemas reais. A família apresentada foi o eCGA (*Compact Genetic Algorithm with elitism*), mCGA (*Compact Genetic Algorithm with elitism and Mutation*) e rmCGA (*Compact Genetic Algorithm with elitism, Mutation and Resampling*). No eCGA o torneio do AGC foi substituído por um torneio entre um indivíduo elite e um indivíduo gerado aleatoriamente a partir do vetor de probabilidades. No mCGA, o eCGA foi estendido com um operador de mutação que gera um novo indivíduo a partir de uma mutação do indivíduo elite. Então um novo torneio é realizado, onde o indivíduo elite e este novo indivíduo competem entre si, e o vencedor será usado para uma nova atualização do vetor de probabilidades. O rmCGA é uma extensão do mCGA, no qual o indivíduo elite é reavaliado a uma determinada taxa de gerações. Esta técnica de reamostragem é útil em problemas onde existe ruído na avaliação do *fitness* ou a função de *fitness* é variável no tempo.

No trabalho apresentado por Lobo, Lima e Martires (2004 e 2005), os autores procuraram melhorar o desempenho do AGC utilizando uma técnica de paralelismo. A idéia foi apresentar um estudo de implementação de paralelismo para AEDs, no qual os autores optaram por iniciar com a implementação para o AGC. Em síntese, um gerenciador envia o vetor de probabilidades para trabalhadores. Os trabalhadores são unidades que executam o AGC por um determinado número de gerações e retornam ao gerenciador a diferença entre o vetor recebido e o do final destas gerações. O gerenciador recebe esta diferença de todos os trabalhadores e atualiza o seu vetor de probabilidades. Este processo se repete até que o vetor de probabilidades convergir.

O CoCGA (*Cooperative Approach to Compact Genetic Algorithm*) (JEWAJINDA e CHONGSTIVATANA, 2006) introduziu uma técnica de paralelismo para AGC. Diferentemente da técnica apresentada em (LOBO, LIMA e MARTIRES, 2004), o objetivo foi aplicar a implementações em *hardware*, por exemplo: FPGAs. De forma simplificada, o CoCGA utiliza uma topologia baseada em AC (Autômatos Celulares), onde a interação entre os ACs é feita compartilhando o vetor de probabilidades, em vez de compartilhar diretamente os indivíduos de uma sub-população. Cada célula (*coarse grained cell*) do CoCGA possui um vetor de probabilidade, e estas células são interconectadas através de uma célula-líder. Esta célula-líder tem o objetivo de compartilhar o vetor de probabilidade entre as células vizinhas. Para ajudar na sua decisão do melhor vetor de probabilidade do grupo, é introduzido o

conceito de CC (*Confidence Counter*), um contador do número de atualizações no vetor de probabilidade de uma determinada célula. Os problemas *OneMax* e *DeJong* F1, F2 e F3 foram usados para comparar o seu desempenho em relação ao AGC. O CoCGA com duas células e uma célula-líder mostrou um número de avaliações de *fitness* menor, aproximadamente 3 vezes, e alcançou melhor qualidade de solução do que o AGC. Os autores afirmam que esta técnica pode ser utilizada em outros AGCs estendidos.

No trabalho apresentado por Mirminno, Cupertino e Naso (2007) foi proposta uma extensão do AGC que propõe uma codificação dos genes com números reais. O AGC com codificação real apresentado utiliza uma função de densidade de probabilidade Gaussiana para descrever a população. Diferentemente do AGC, o vetor de probabilidades é uma matriz, onde cada probabilidade de um alelo é descrita por um valor médio e um desvio padrão. A regra de atualização também é modificada para assumir este novo vetor de probabilidades, de forma a também imitar uma seleção de torneio binário estacionário. O objetivo desta extensão foi adequar o AGC a implementações em microcontroladores, onde a utilização de números reais é mais adequada. Foram executados dois tipos de teste, um para verificar a funcionalidade e outro para verificar sua aplicabilidade em microcontroladores. Os autores escolheram o neCGA (AHN e RAMAKRISHNA, 2003) para ser usado nestes testes. Para verificar a funcionalidade, foram usados os problemas *Rastrigin*, *Circle*, *Schwefel* e *Michalewicz* com dimensionalidade 3. Através dos resultados obtidos com a codificação real e binária, pode-se constatar que ambas as codificações funcionaram de forma semelhante. Anteriormente, no trabalho apresentado por Cupertino, Mirminno e Naso (2006), o neCGA com codificação binária se mostrou adequado a implementações em microntroladores não só com relação ao custo computacional, mas também com relação ao seu desempenho. Seguindo a mesma área de aplicações, a aplicabilidade do neCGA com codificação real foi demonstrada em problemas de Controle com Auto Sintonia para Motores de Corrente Contínua e de Indução.

2.2 COMPUTAÇÃO RECONFIGURÁVEL

Na Computação reconfigurável um agente de *hardware* de propósito geral é configurado para executar uma determinada tarefa, mas que pode ser reconfigurado para executar outras tarefas específicas[1]. Devido ao alto desempenho e baixo custo, os agentes de *hardware* podem ser uma alternativa às máquinas de Von Neumman, implementadas em microprocessadores.

Na computação convencional existem dois métodos de implementação de um algoritmo: implementação em *hardware* fixo, como por exemplo: usando-se ASICs (*Application Specific Integrated Circuits*), ou em microprocessadores programáveis por *software*. Enquanto os sistemas implementados usando o método de *hardware* fixo possuem maior desempenho de velocidade, os implementados em *software* possuem maior flexibilidade (COMPTON, 2002). As implementações em *software* não alcançam o desempenho de velocidade do *hardware* fixo, pois sua programação é executada seqüencialmente, por exemplo, usando o conceito da máquina de Von Neumman. A computação reconfigurável busca ser um método intermediário entre os métodos de *hardware* fixo e *software*, onde um *hardware* reconfigurável pode superar a velocidade do *software* e a flexibilidade do *hardware* fixo, apresentado na figura 5. Desta forma, a computação reconfigurável altera o balanço entre velocidade de operação versus generalidade de operação dos sistemas computacionais (VILLASENOR e SMITH, 1997).

[1] http://www.acm.uiuc.edu/sigarch/projects/reconf/report_1.html. Visitado em junho de 2007.

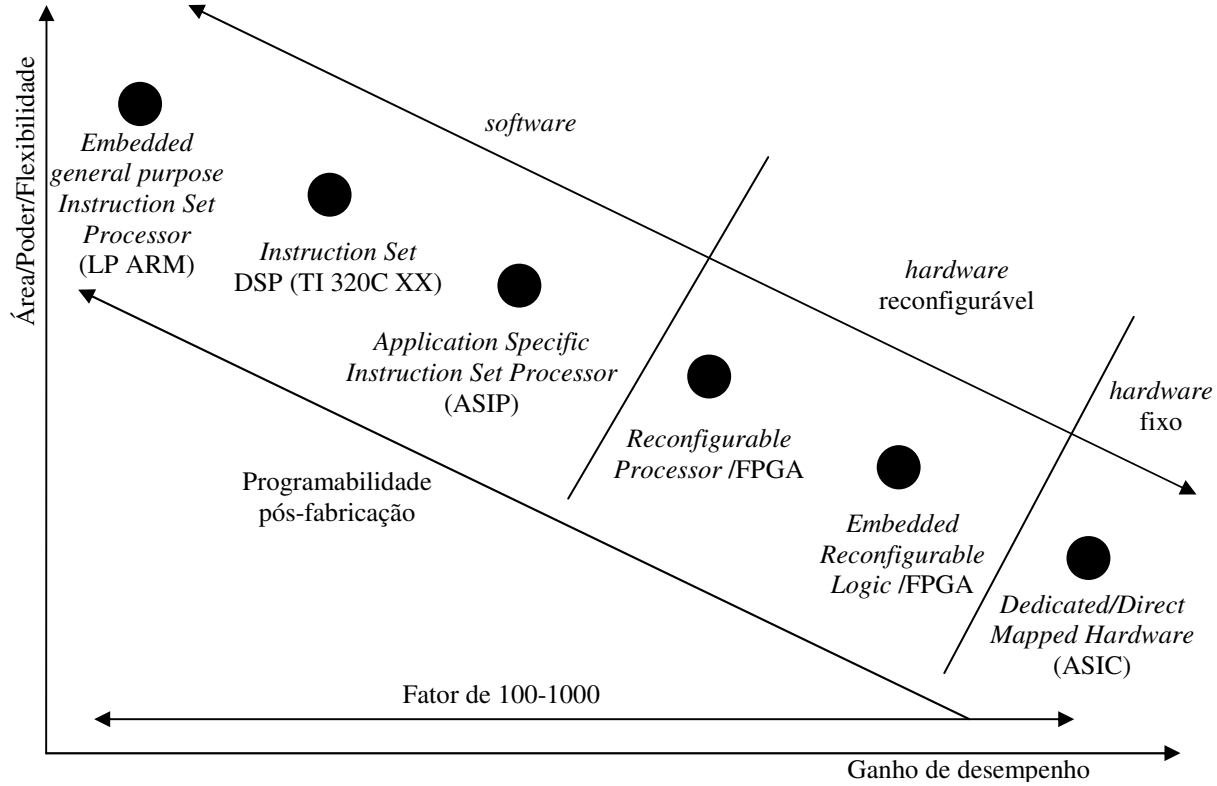


Figura 5: Posicionamento da computação reconfigurável.

Apesar do conceito de sistemas reconfiguráveis existir desde a década de 60, só se tornou comum a partir década de 90. Em (ESTRIN e VISHANATHAN, 1962a) e (ESTRIN e VISHANATHAN, 1962b) foi proposto um conceito de um computador híbrido com um processador comum e um *array* de *hardware* reconfigurável. Este *array* poderia ser configurado para executar uma tarefa, como um processamento de imagem ou reconhecimento de padrão, tão rápido quanto um *hardware* dedicado. Uma vez que a tarefa fosse terminada, o *hardware* poderia ser reconfigurado para executar uma nova tarefa. No entanto, este conceito estava muito à frente de seu tempo, com relação à tecnologia eletrônica disponível.

Nas décadas de 80 e 90, houve um renascimento de pesquisas na área de sistemas reconfiguráveis. Diversas arquiteturas de *hardware* reconfigurável foram desenvolvidas tanto em ambientes acadêmicos quanto industriais, por exemplo: Matrix, Garp, Elixent, XPP, Silicon Hive, Montium, Pleiades, Morphosys, PiCoGA. Estas arquiteturas foram possíveis devido ao rápido e constante desenvolvimento da tecnologia de silício, que possibilitou o desenvolvimento de arquiteturas complexas em *chips*. Em 1985 a Xilinx introduziu XC2064, o primeiro FPGA (*Field-Programmable Gate Array*). E em 1991 foi apresentado o primeiro

sistema reconfigurável por *hardware* comercial, o Algotronix CHS2X4. Apesar de não ser um sucesso comercial, o conceito foi promissor o suficiente para a Xilinx comprar esta tecnologia e contratar os seus inventores.

Atualmente, os FPGAs são os dispositivos mais usados na computação reconfigurável. Inicialmente o seu objetivo foi a prototipação de circuitos, mas devido a seu custo e fácil acesso, estão sendo amplamente utilizados para a implementação de *hardware* reconfigurável. Atualmente existem diversas empresas que produzem FPGAs, dentre as principais estão: Altera[2], Xilinx[3], Actel[4] e Atmel[5]. Cada fabricante possui várias famílias de FPGAs, todas as quais voltadas para tipos diferentes de aplicações.

Os trabalhos desenvolvidos em computação reconfigurável por *hardware* são referenciados por diversas denominações, entre eles lógica reconfigurável por *hardware*, *hardware* reconfigurável e *configware*[6]. Contudo, neste trabalho iremos denominar simplesmente de *hardware.*

2.2.1 HDL

O HDL (*Hardware Description Language*) é definido como qualquer linguagem computacional capaz de descrever circuitos eletrônicos, e são empregados em aplicações de *hardware* fixo e reconfigurável. Um HDL pode descrever a operação dos circuitos, seu planejamento e organização e simular esta operação para testes. Ao contrário das linguagens de *software*, o HDL possui sintaxe e semântica que descrevem simultaneidade e concorrência, atributos primários do *hardware*.

Em 1977 surgiu o ISP (*Instruction Set Processor*), o primeiro HDL, desenvolvido pela Universidade Carnegie Mellon. Esta linguagem era parecida com outras linguagens de *software*, mas possibilitava descrever as relações entre entradas e saídas de um sistema computacional. Apesar de ser possível simular o circuito projetado, não era possível

---

2 http://www.altera.com/

3 http://www.xilinx.com/

4 http://www.actel.com/

5 http://www.atmel.com/

6 http://configware.org/

sintetizar. Mais tarde, em 1983, foi introduzido ABEL (*Advanced Boolean Equation Language*) pela Data-IO, com o objetivo de descrever PLDs (*Programmable Logic Devices*) usadas, basicamente, para projetar máquinas de estados. Os PLDs são um tipo de *hardware* reconfigurável como o FPGA, mas aplicados em sistemas muito menos complexos e onde o custo é crítico.

O Verilog, o primeiro HDL moderno, foi introduzido por *Gateway Design Automation* em 1985. O Verilog-XL, que mais tarde foi adquirido por *Cadence Design*, se tornou o simulador HDL padrão para o Verilog na década seguinte. Em 1987, foi desenvolvido o VHDL (*Very high speed integrated circuit Hardware Description Language*) fomentado pelo Departamento da Defesa Norte-Americano. Outros HDLs foram propostos, tais como: ABEL (*Advanced Boolean Expression Language*), AHDL (Altera HDL) e Atom. Mas, com o tempo, tanto o Verilog quanto VHDL se tornaram dominantes na indústria eletrônica. Devido a sua origem, o VHDL se tornou o HDL mais empregado em instituições de pesquisa e governamentais. Os outros HDLs mais antigos e menos poderosos caíram em desuso. No entanto, o Verilog e VHDL possuem limitações de aplicabilidade, como por exemplo: simulação de circuitos analógicos ou de sinais mistos (*mixed-signal*). Desta forma, foram desenvolvidos HDLs com o objetivo de melhorar o Verilog e o VHDL, como o Verilog-AMS (Verilog *Analog Mixed-Signal*) e VHDL-AMS. E, da mesma forma, estas limitações impulsionaram o constante refinamento e revisões das especificações destes HDLs.

2.3 TRABALHOS RELACIONADOS

Apesar do AG ter tido sucesso em diversos problemas difíceis de otimização, sua execução consome muito tempo. Mesmo em plataforma de alto desempenho, um processo de AG pode rodar por muito tempo devido à intensa computação necessária. Desta forma diversos trabalhos foram feitos para implementar um AG em *hardware*. Os primeiros trabalhos procuraram implementar um AGS em um *hardware* empregando um controle MEF (Máquina de Estados Finita), por exemplo: HGA (*Hardware Genetic Algorithm*) (SCOTT e SETH, 1995) e SPGA (*Splash Parallel Genetic Algorithm*) (GRAHAM e NELSON, 1995). Outro método empregado para implementar AG em *hardware* foi o *systolic array* introduzido por Bland e Megson (1998). Contudo, o AGS não se mostrou adequado a implementações em *hardware*. Então outros trabalhos exploraram a implementação do SSGA (*Steady-State*

Genetic Algorithm), onde foi possível utilizar técnicas de *pipeline*. Os trabalhos mais recentes nesta área são *survival driven* SSGA (SHACKLEFORD, OKUSHI, YASUDA *et al*, 2000) e (SHACKLEFORD, SNIDER, CARTER *et al*, 2001) e PGA (*Pipelined Genetic Algorithm*) (SAENZ, IBARRA, LANCHARES *et al*, 2001). Outros trabalhos buscaram implementar uma arquitetura customizada para acelerar AGs, como o VGP-I (KAVVADIAS, GIANNAKOPOULOU e NIKOLAIDIS, 2007). Apesar do aumento do desempenho de velocidade, estes trabalhos mostraram que a implementação de um AG é difícil, muitas vezes necessitando uma estrutura de *hardware* caro e complexo.

No entanto, implementações de AG em *hardware* possuem outro gargalo além da complexidade de implementação: a memória para armazenar a população. O tamanho da população é um parâmetro crítico em AGs, podendo alcançar até milhares de indivíduos. Desta forma, existe um compromisso entre custo e desempenho ao implementar a população em *hardware*. O uso de memórias com alto desempenho de velocidade de acesso (implementação de memória usando recursos do FPGA) torna o *hardware* caro e o uso de memórias de baixo custo (como as memórias RAM) diminui a velocidade de execução. Diferentemente de um AGS, o AGC é mais apropriado para implementações em *hardware* (APORNTEWAN, CHONGSTITVATANA, 2001). Assim, é possível identificar duas vertentes em implementações de AGs em *hardware:* uma que procura implementar uma aceleração do AG usando AGS ou SSGA em um *hardware* dedicado ou processador programável, e outra que busca implementações de AGCs estendidos em *hardware* dedicados. Considerando que este trabalho visa trabalhar com a segunda vertente, são apresentados os trabalhos correlatos à implementação de extensões do AGC em *hardware*.

A implementação do AGC em *hardware* proposto em (APORNTEWAN, CHONGSTITVATANA, 2001) se mostrou 1000 vezes mais rápido do que sua versão em *software*. O sistema foi projetado na linguagem Verilog HDL e implementada em FPGA Xilinx Virtex V1000FG680 usando um *clock* de 20MHz necessitando de 15210 *equivalent gate count*. Sua versão em *software* foi escrita em C e implementada em uma plataforma Ultra Sparc II 200 MHz. O problema utilizado em ambas versões foi o *OneMax* proposto em (HARIK, LOBO, GOLDBERG, 1999).

No trabalho apresentado por Gallagher, Vigraham e Kramer (2004), os autores implementaram o mCGA baseado na implementação do AGC apresentado por Aporntewan e Chongstitvatana (2001). Com objetivo de testes iniciais, o mCGA foi utilizado para otimizar o

problema *OneMax* com cromossomo de 32 bits e uma população simulada de 255 indivíduos. O mCGA em *hardware* foi desenvolvido em VHDL e implementado nos FPGAs XC4010D BORG e VirtexII XC2V1000 da Xilinx. Estas implementações foram comparadas com uma implementação escrita em C e rodando em uma plataforma Ultra Sparc II de 200MHz. As implementações em *hardware* alcançaram um ganho de desempenho de 1000 vezes em comparação a implementação em *software*. A implementação no VirtexII XC2V1000 foi comparada com uma implementação do AGC apresentado em (APORNTEWAN, CHONGSTITVATANA, 2001). O mCGA ocupou 18732 *equivalent gate count* enquanto o AGC ocupou 15210. Desta forma, mostrou-se que, apesar do mCGA aumentar o desempenho da qualidade de solução do AGC, não aumenta significantemente a demanda por recursos de *hardware*.

O CoCGA (JEWAJINDA e CHONGSTIVATANA, 2006) implementou uma técnica de AGC paralelos para problemas de otimização. Com o objetivo de fazer uma comparação de desempenho e utilização de *hardware* com o AGC, os problemas escolhidos foram o *OneMax*, *DeJong F1*, *F2* e *F3*. O CoCGA foi desenvolvido usando Verilog-HDL e implementado em FPGA Vertex 4 VLX25 SF363-10, onde a célula normal utilizou 17034 *equivalent gate count* e a célula líder 4651. Em comparação ao AGC implementado no mesmo FPGA e baseado em (APORNTEWAN, CHONGSTITVATANA, 2001), que utilizou 12602 *equivalent gate count*, a célula normal não utiliza muito mais recursos de *hardware*. Apesar de utilizar o AGC para implementar as células normais, os autores afirmam que é possível implementar outros AGC estendidos.

Apesar de o AGC ser apropriado para implementações em aplicações em *hardware*, o número de pesquisas nesta área é limitado. Devido à forma compacta de representar a população no AGC, as informações sobre conexões de alta ordem entre bits não conseguem sobreviver através das gerações. Sem um aumento da pressão seletiva, o AGC não é apropriado para problemas reais, os quais são normalmente multimodais e de alta ordem. Contudo, recentemente alguns trabalhos introduziram extensões para o AGC, que não apenas mostraram resultados satisfatórios para problemas reais, como também não aumentaram significativamente a sua demanda por recursos de *hardware* e nem por memória. Os primeiros resultados foram apresentados em (AHN e RAMAKRISHNA, 2003) e (GALLAGHER, VIGRAHAM e KRAMER, 2004) e, mais recentemente, em (SILVA, LOPES e LIMA, 2007).

Assim, trabalhos relacionados aos AGCs estendidos aplicados em *hardware* possuem muito espaço para desenvolvimento e pesquisa.

CAPÍTULO 3

METODOLOGIA

3.1 O emCGA

Desde que o AGC foi introduzido por Harik, Lobo e Goldberg (1999) houve diversas extensões que melhoraram o seu desempenho, tornando-o mais adequado para resolver problemas reais. O foco principal destas extensões foi introduzir técnicas que permitam o AGC superar a perda de desempenho em problemas com blocos construtivos de ordem alta. Este capítulo apresenta a proposta e implementação de uma nova extensão do AGC. Esta nova proposta não aumenta significativamente o custo computacional e nem o consumo de memória, aumenta o desempenho geral em comparação com trabalhos similares.

No AGC, devido ao modelo probabilístico usado para representar a população, não ocorre o armazenamento das informações de correlações de ordem alta entre genes. Então é necessária uma técnica que permita aumentar a pressão seletiva e diminuir a deriva genética (*genetic drift*), efeito ruidoso do *crossover* uniforme. A deriva genética é um termo emprestado da genética populacional que procura explicar tendência de mudança na freqüência de aparecimento de um alelo em uma população através de amostragens aleatórias. Ou seja, mesmo que não exista nenhuma pressão seletiva, a freqüência dos alelos de uma população varia devido a erros estocásticos na aplicação dos operadores de *crossover* e mutação. Quando a variação estocástica na freqüência dos alelos supera a variação devido à seleção, então ocorre o *drift stall*. Desta forma, a população convergirá para uma solução não ótima (ROGERS e PRUEGEL-BENNETT, 1999) e (THIERENS, GOLDBERG e PEREIRA, 1998). Conseqüentemente, manter uma pressão seletiva adequada é importante para a população de um AG alcançar uma convergência para uma solução ótima ou quase-ótima. Resumidamente, a ocorrência do *drift stall* reflete a falta de pressão seletiva adequada, que pode ser aumentada no AGC através do aumento do tamanho do torneio ou do uso do elitismo.

Os trabalhos que propuseram extensões do AGC usando alguma técnica de elitismo foram os que mostraram a melhor relação custo computacional e desempenho. O elitismo permite aumentar pressão seletiva e, conseqüentemente, diminuir a deriva genética,

assegurando que o cromossomo do melhor indivíduo sobreviva e permaneça na próxima geração (DUMITRESCU, LAZZERINI, JAIN *et al*, 2000). Uma das primeiras extensões usando elitismo foi o MBBCGA (ZHOU, MENG e QIU, 2002). Contudo, a melhoria no desempenho não foi atribuída ao elitismo, mas ao operador de mutação proposto. No trabalho de (AHN e RAMAKRISHNA, 2003) foram introduzidas as extensões peCGA e neCGA. Neste trabalho houve a preocupação de inter-relacionar AEDs e EE através do estudo da equivalência de um AGC usando elitismo e um (1+1) EE com mutação auto-adaptativa. Os resultados deste trabalho mostraram que o uso do elitismo consegue compensar a falta de memória do AGC. No entanto, deve haver um controle na pressão seletiva exercida pelo elitismo para evitar uma convergência prematura. Este controle foi feito no neCGA com um parâmetro chamado comprimento de herança (*length of inheritance*), que define por quantas gerações o cromossomo do indivíduo elite pode se perpetuar. Uma nova técnica de controle de pressão seletiva foi proposto no mCGA (GALLAGHER, VIGRAHAM e KRAMER, 2004), onde foi proposto um operador de mutação que opera no cromossomo do indivíduo elite.

No AG, a mutação é um operador genético inspirado na natureza usado para manter a diversidade genética causando uma mudança em alelo. Um exemplo clássico é a mutação *flip bit*, onde bits de um cromossomo são invertidos aleatoriamente segundo uma taxa especificada, chamada de probabilidade de mutação. Podem ser encontrados diversos estudos na literatura que buscam a probabilidade de mutação ótima para os AGs; contudo, existe um consenso em recomendar uma probabilidade de $1/l$, onde l é o tamanho do cromossomo (CERVANTES e STEPHENS, 2006).

Neste trabalho é proposta uma nova extensão, o emCGA (*elitism with mutation Compact Genetic Algorithm*), que introduz um novo operador de mutação que opera na geração do novo indivíduo (SILVA, LOPES e LIMA, 2007). Desta forma, este operador de mutação mimetiza o operador de mutação do AG, pois a operação de geração no AGC é mimetização da reprodução no AGS. A diferença principal entre a implementação do emCGA e do mCGA é que no emCGA não existe uma etapa seqüencial, o torneio entre o elite e sua mutação no mCGA. Esta diferença por si só torna o emCGA mais interessante em aplicações que permitem implementar processos concorrentes, como em *hardware*. Contudo, espera-se também uma diminuição no número de avaliações de *fitness* necessários até o vetor de

probabilidades convergir, pois o mCGA desperdiça uma avaliação de *fitness* por geração sem atualizar o vetor de probabilidades.

3.1.1 Descrição do emCGA

O pseudocódigo da figura 4, representando o algoritmo do AGC, é modificado para representar o emCGA na figura 6. Muito embora todas as etapas do AGC ainda estejam presentes, ressaltam-se algumas modificações importantes:

- Na etapa Inicialização do emCGA o indivíduo Elite é gerado aleatoriamente e avaliado para calcular seu valor de *fitness*. Esta é a primeira modificação introduzida pelo emCGA, ou seja, a técnica de elitismo, já abordada em outros trabalhos.
- Na etapa Geração, diferentemente do AGC, somente um indivíduo é gerado através do vetor de probabilidades usando um operador de mutação com uma taxa μ. Esta é outra modificação introduzida pelo emCGA, ou seja, o operador de mutação. Esta é uma contribuição inédita deste trabalho.
- Na etapa Competição ocorre a troca do indivíduo Elite pelo indivíduo Novo quando este último apresentar um *fitness* melhor. Caso contrário, os indivíduos permanecem como estão.
- Na etapa de Atualização, o vetor de probabilidades é atualizado na direção do indivíduo Elite atual, ou seja, o indivíduo vencedor da etapa Competição.
- As outras etapas são idênticas ao AGC.

```
1) Inicialização
Para i:= 1 até l faça p[i]:=0.5;
Elite:=gerar (p);
Elite_fitness:=avaliar(Elite)

2) Geração: Gerar novo indivíduo através do vetor de probabilidades com mutação
Novo:=gerar_com_mutação (p, μ);

3) Competição
Novo_fitness:=avaliar(Novo);
Se Novo_fitness é melhor que Elite_fitness Então
   troca_indivíduos(Novo, Elite);

4) Atualização: Atualizar o vetor de probabilidade na direção do Elite
Para i:= 1 até l faça
  Se Elite[i] ≠ Novo[i] Então
    Se Elite[i] = 1 Então  p[i] := p[i] + 1/n;
                   Senão p[i] := p[i] - 1/n;

5) Verificação: Verificar se o vetor convergiu
Para i:=1 até l faça
  Se p[i]>0 e p[i]<1 Então
    Vá para o passo 2;

6) Finalização: p representa a solução final

Parâmetros:
n: Tamanho da população
l: Comprimento do cromossomo em bits
μ: Probabilidade de mutação
```

Figura 6: Pseudocódigo do emCGA.

Do ponto de vista funcional o algoritmo proposto pode ser visualizado usando-se o diagrama em blocos da figura 7. As etapas descritas no pseudocódigo da figura 6 são representadas por blocos funcionais relacionados com o vetor de probabilidades e os indivíduos usados em cada geração. Um bloco importante não descrito no pseudocódigo é o gerador de número pseudo-aleatório. Ele é fundamental na geração dos indivíduos. Devido à sua importância no emCGA será apresentado uma seção para discuti-lo.

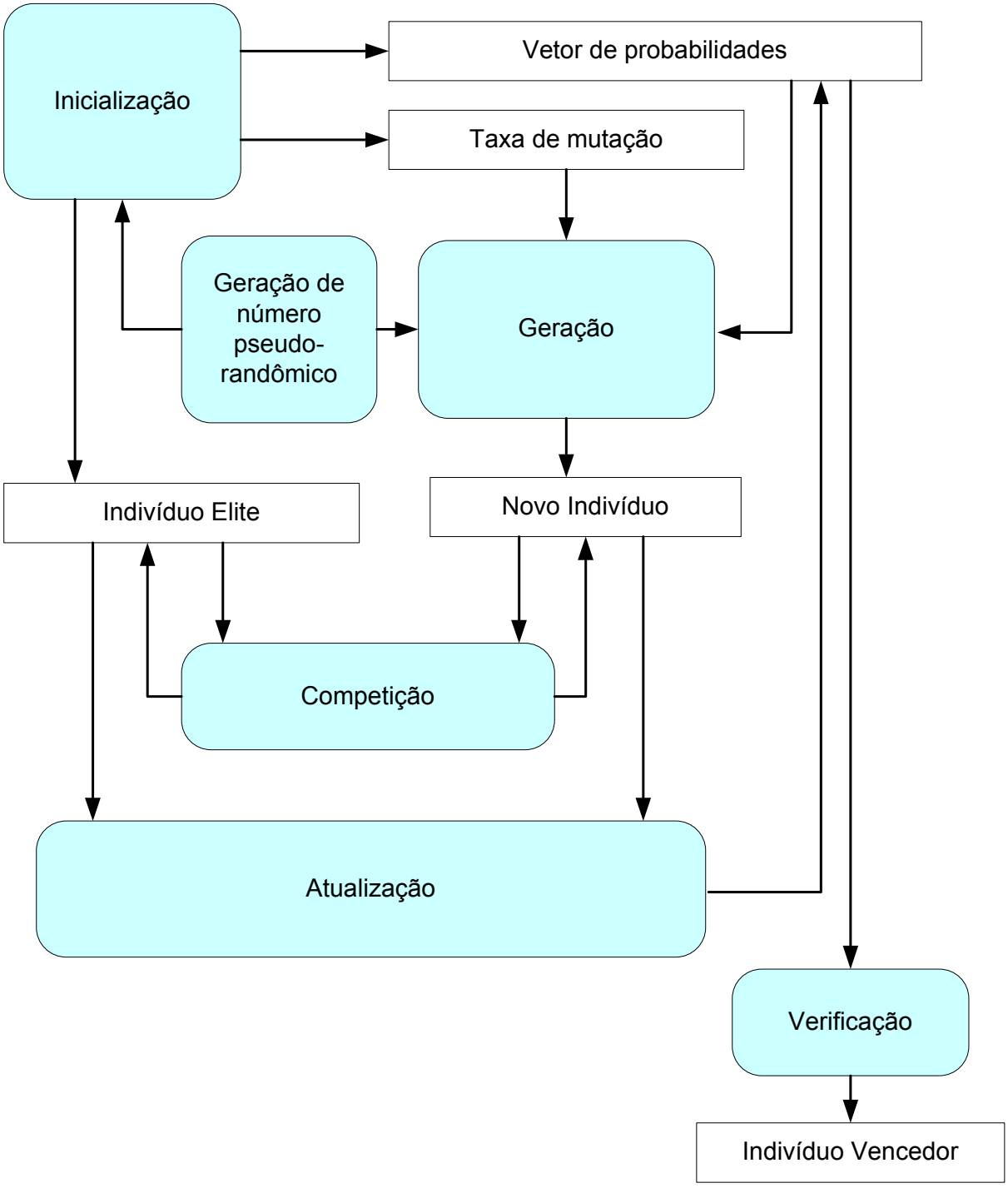


Figura 7: Blocos funcionais utilizados para implementação do emCGA.

A figura 7 sugere a máquina de estados descrita na figura 8 e tabela 2 para compreensão do seu funcionamento. De fato, esta máquina de estados será fundamental na implementação do algoritmo em lógica reconfigurável.

Tabela 2. Descrição da máquina de estados.

Estado	Descrição	Evento	Descrição	Próximo Estado
S0	**Inicialização**	**T0**	**Iniciou**	**S1**
S1	**Geração**	**T1**	**Gerou**	**S2**
S2	**Competição**	**T2**	**Competiu**	**S3**
S3	**Atualização**	**T3**	**Atualizou**	**S4**
S4	**Verificação**	**T4**	**Vetor não convergiu**	**S1**
		T5	**Vetor convergiu**	**S5**
S5	**Finalização**			

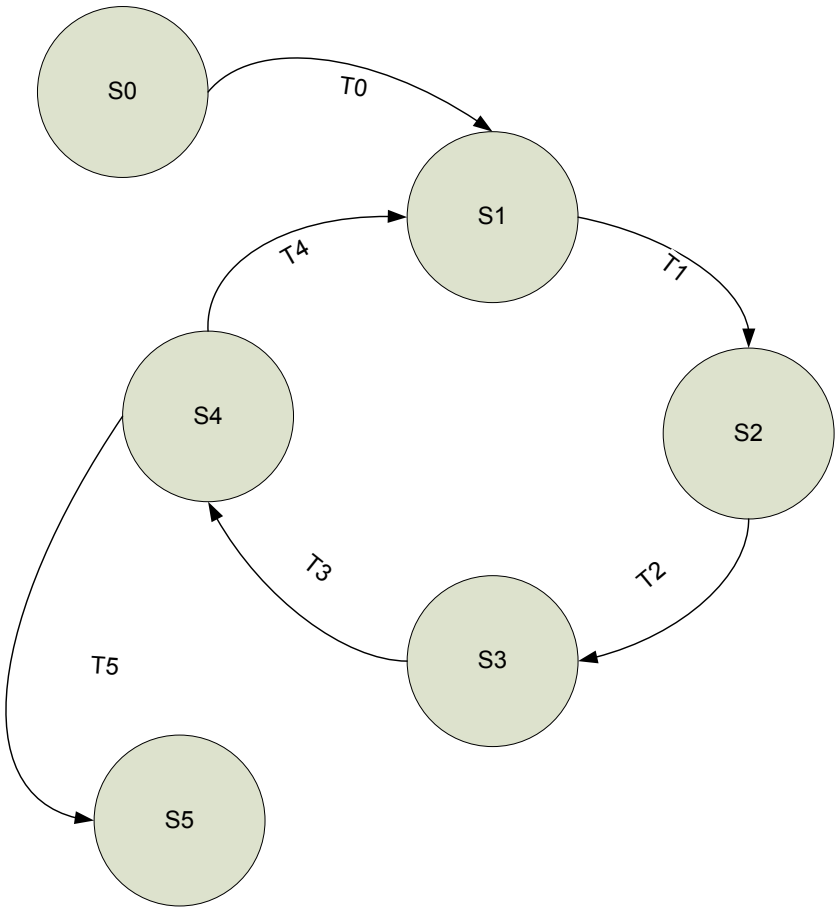


Figura 8: Máquina de estados do emCGA.

Existem 6 estados, a Inicialização, a Geração, a Competição, a Atualização, a Verificação e a Finalização. Durante o estado de Inicialização é possível atualizar os parâmetros do algoritmo, iniciar o vetor de probabilidades com a probabilidade inicial (0,5) em cada elemento e gerar o indivíduo Elite. Os parâmetros são o tamanho da população, comprimento do cromossomo, probabilidade de mutação e a semente do gerador de números pseudo-aleatórios. Uma vez iniciado, o algoritmo entra no ciclo gerar, competir, atualizar e verificar até a convergência da população; ou seja, todos os elementos do vetor de

probabilidade alcançarem 0 ou 1. No estado Geração é obtido um dos indivíduos da futura competição, usando-se o operador de mutação. No estado Competição define-se o indivíduo vencedor (Elite) em função do *fitness* dependente do problema. O estado de Atualização é responsável pela atualização do vetor de probabilidades na direção do indivíduo vencedor obtido no estado anterior. Se após esta atualização o vetor convergir, então o algoritmo pára (vai para o estado Finalização), senão gera um novo indivíduo (vai para o estado Geração).

3.1.2 Exemplo de aplicação do emCGA para o problema *OneMax*

O problema *OneMax* é um problema de maximização, no qual o objetivo é maximizar o números de bits 1 em um vetor de bits. Na definição de AG, o vetor de bits é o cromossomo e cada bit é, ao mesmo tempo, um gene e um alelo. A função a ser maximizada é mostrada a na equação 1, tal que:

$$f(x) = \sum_{i=1}^{n} x_i \quad x = \{0,1\} \tag{1}$$

Onde:

x: vetor de variáveis do problema e

n: dimensionalidade do problema.

Neste exemplo considera-se *n* igual a 10, ou seja: um cromossomo de tamanho igual a 10 bits, e uma população de 16 indivíduos. Cada indivíduo é representado por um cromossomo.

A população inicial no emCGA é representada por um vetor de probabilidades, onde cada alelo tem probabilidade de 50% de ser 1, ou, simplesmente probabilidade 0,5. A comparação da população do emCGA com uma população no AGS é mostrada nas figuras 9a e 9b. Na etapa de Competição, os indivíduos Elite e Novo competem segundo um *fitness* definido pelo problema. Como o problema *OneMax* é um problema de maximização, o indivíduo com o maior valor de *fitness* será o vencedor. Um exemplo de indivíduos é mostrado na figura 9c.

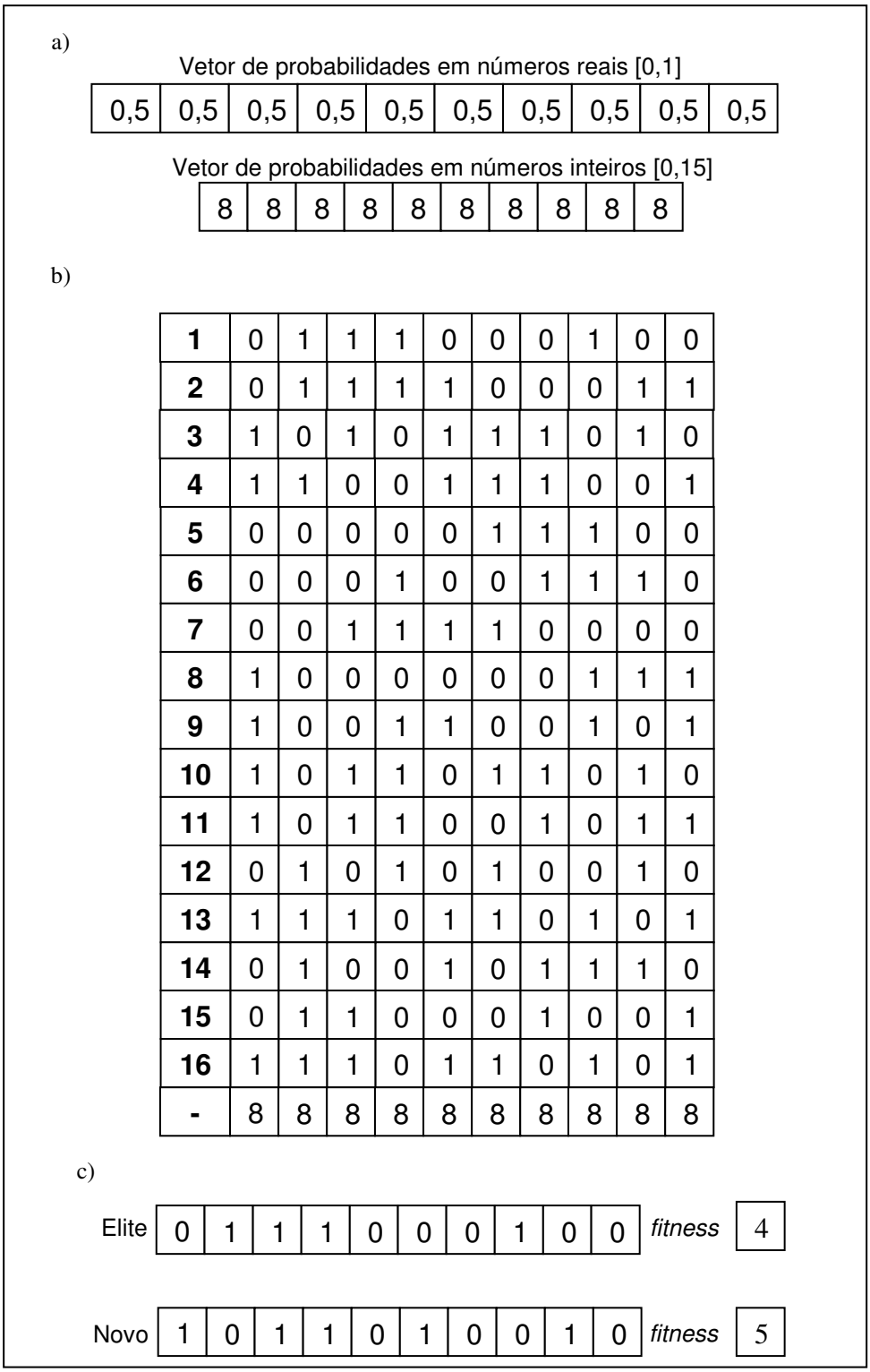


Figura 9: Representação da população inicial para o emCGA (a), para o AGS (b) e dos indivíduos Elite e Novo usados na etapa de Competição.

No caso ilustrado na figura 9c, o indivíduo Novo vence, tornando-se o indivíduo Elite. Desta forma, o indivíduo Novo antes da competição se torna Elite após a competição e vice e versa. Na etapa de Atualização, o vetor de probabilidades atualiza-se na direção do indivíduo Elite atual a uma taxa de *1/n*, onde *n* é o tamanho da população. Neste caso, a taxa é de *1/16*, ou simplesmente *1* para o vetor de probabilidade usando números inteiros de 4 bits. As etapas de Competição e de Atualização simulam a técnica de seleção por torneio de estado estacionário (*steady-state tournament*), de tamanho 2, com elitismo do AGS, utilizando um campeonato entre o indivíduo Elite e o indivíduo Novo e atualização do vetor de probabilidades na direção do vencedor. Ambas as técnicas são ilustradas na figura 10.

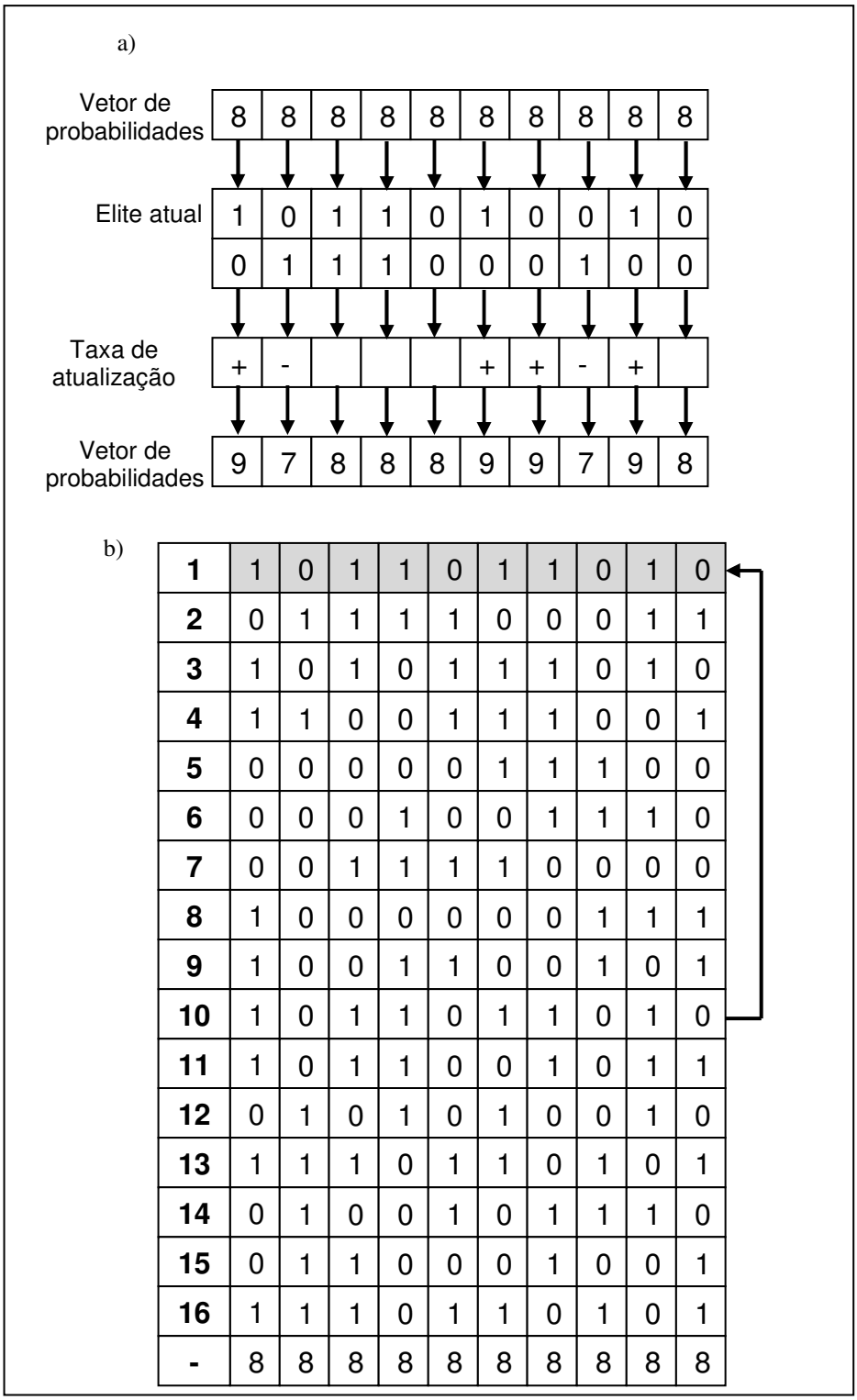


Figura 10: Representação da etapa Atualização para o emCGA (a) e da etapa equivalente de Seleção para o AGS usando seleção por torneio estacionário (*steady-state tournament*) (b).

O emCGA imita os operadores de recombinação (*crossover*) uniforme e mutação *flip-bit* do AGS. No primeiro caso, na etapa Geração no emCGA, um indivíduo é gerado segundo

um vetor de probabilidades, que recombina os bits do cromossomo. No segundo caso, ocorre a inversão dos bits deste cromossomo segundo uma certa taxa, denominada de probabilidade de mutação. Na figura 11 é mostrado como funciona a geração de um indivíduo com mutação de 10%, usando uma precisão de mutação de 16 bits.

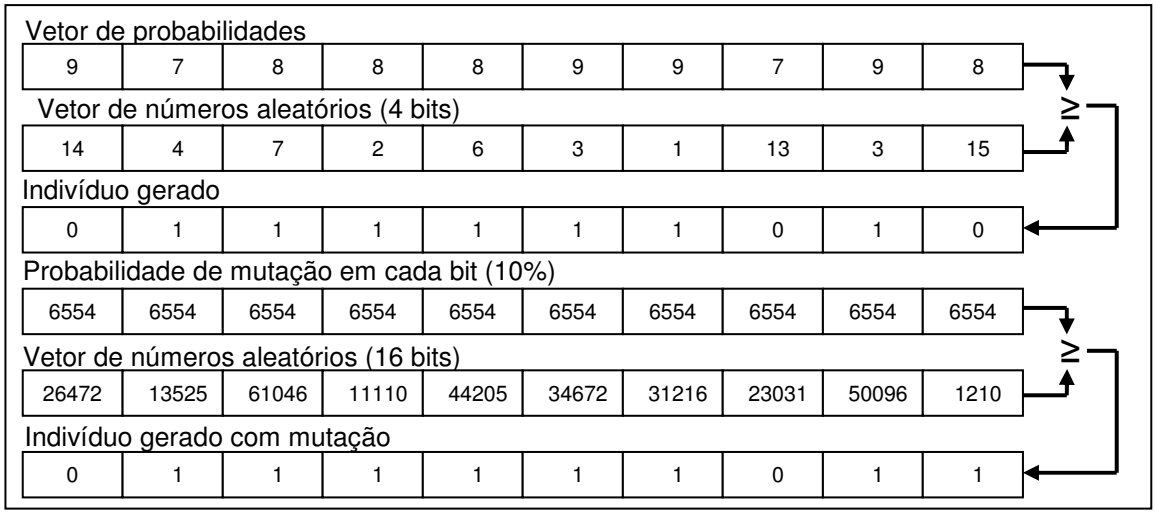


Figura 11: Representação da geração de indivíduo com mutação.

3.1.3 Gerador de números aleatórios

Como um dos objetivos deste trabalho é desenvolver e implementar uma variante do AGC em *hardware*, o RNG (*Random Number Generator*, ou Gerador de Números Aleatórios) implementado em *software* deve ser compatível com este objetivo. Logo, é possível implementar um AGC que terá o mesmo desempenho em aplicações em *hardware* ou *software*. Sendo assim, é necessário um estudo de quais técnicas de RNG estão sendo empregadas em implementações em *hardware*.

Nas implementações em *hardware* são tipicamente utilizados RNG baseados nas técnicas LFSR (*Linear Feedback Shift Registers*) e AC (Autômatos Celulares) (MARTIN, 2002). Os LFSRs também podem ser referenciados como MLS (*Maximum Length Sequence*), significando que um gerador de comprimento de *n* bits gera uma seqüência pseudo-randômica de 2^n números. Nos trabalhos (MARUYAMA, FUNATSU e HOSHINO, 1998) e (GRAHAM e NELSON, 1996) foram utilizados LFSR. No entanto no trabalho (TOMMISKA e VUORI, 1996) foi utilizado um RNG híbrido com 3 LFSRs acoplados e um gerador de ruído desenvolvido em eletrônica analógica. O ruído gerado foi usado para gerar sementes para os

LFSRs. Os fabricantes de FPGAs, como a Altera[7] e Xilinx[8], também disponibilizam *Int. Prop. cores* de LFSRs em HDL.

O trabalho de Martin (2002) estudou diversas técnicas de RNG para implementações em *hardware* e analisou seu desempenho para PG (Programação Genética). O autor concluiu que algoritmos mais complexos de RNG não mostraram desempenho muito melhor que um algoritmo simples do LFSR. Se for considerado a demanda por recursos de *hardware*, os melhores resultados foram obtidos usando LFSR simples ou usando 32 LFSRs acoplados. Apesar destas conclusões serem para PGs, nos trabalhos (MEYSENBURG e FOSTER, 1999a) e (MEYSENBURG e FOSTER, 1999b) os autores mostraram que não há diferenças estatísticas significativas entre o desempenho do AG e PG quando diferentes RNGs são empregados e, conseqüentemente, também não se espera diferenças significativas empregando um AGC.

O RNG escolhido para as implementações deste trabalho foi o LFSR simples de 32-bits. O gerador recomendado pelo Martin (2002) é o LFSR acoplado. No entanto, apesar de apresentar o pior desempenho, o desempenho do LFSR simples não é muito inferior ao LFSR acoplado, e nem aos outros apresentados no estudo. Como esta implementação do LFSR é a mais simples dentre os RNGs apresentados em (MARTIN, 2002), este foi o RNG escolhido.

O LFSR é um *shift register* onde o bit de entrada, o bit mais significativo, é uma função linear de seu estado anterior. Esta função linear é a combinação em ou-exclusivo de bits na seqüência do *shift register*. Um exemplo de funcionamento do LFSR simples de 3-bits é apresentado na figura 12, onde a sua função linear é bit 1 xor bit 0. Para o LFSR simples de 32-bits, que será empregado neste trabalho, sua função é bit 31 xor bit 21 xor bit 1 xor bit 0.

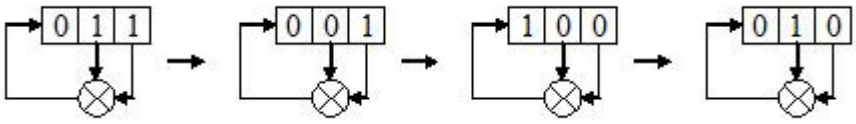


Figura 12: Exemplo de LFSR simples de comprimento de 3 bits.

7 http://www.altera.com/literature/sb/sb11_01.pdf.

8 http://www.xilinx.com/ipcenter/catalog/logicore/docs/lfsr.pdf.

3.2 IMPLEMENTAÇÃO EM *SOFTWARE* DO emCGA

A implementação em *software* do emCGA foi desenvolvida em ANSI C utilizando biblioteca MEX do MATLAB (*MATrix LABoratory*)[9]. Desta forma, o ambiente de teste pode ser desenvolvido em MATLAB, permitindo a utilização das ferramentas de visualização dos resultados deste programa. Através da biblioteca MEX, o programa é compilado como DLL (*Dynamic Link Library*) e pode ser executado no MATLAB como uma função M. Desta forma, é possível somar o potencial de visualização e análise dos resultados do MATLAB com o desempenho e flexibilidade do ANSI C.

3.2.1 Biblioteca MEX do MATLAB

MATLAB é um ambiente de computação numérica e uma linguagem para computação técnica. O MATLAB permite manipulações matemáticas de matrizes, visualizações gráficas, implementações de algoritmos, desenvolvimento de interfaces com usuários e integração com programas de outras linguagens. Estas ferramentas oferecem um atalho no desenvolvimento de experimentos e, conseqüentemente, a redução do tempo de desenvolvimento.

A biblioteca MEX possibilita programas desenvolvidos em C ou FORTRAN sejam chamados no MATLAB sem os reescrever como um arquivo M. Algumas computações, como laços, podem tomar muito tempo de execução no MATLAB e se tornar gargalos. Nestas situações, o código em MATLAB pode ser reescrito em C ou FORTRAN e ser chamado dentro do algoritmo.

O código fonte dos programas desenvolvidos com a biblioteca MEX é compilado como bibliotecas compartilhadas. No ambiente Windows estas bibliotecas são conhecidas como DLL (*Dynamic Link Library*). Esta biblioteca oferece um conjunto de funções, conhecidas como funções MEX, que permitem diversas ferramentas de integração entre estas linguagens. Além disto, a utilização destas funções automatiza a compilação, apesar de que o MATLAB não é um compilador C e nem FORTRAN, e precisa de um compilador externo. Para as linguagens C e C++ o MATLAB possui internamente o compilador LCC, mas pode ser selecionado outro compilador pelo usuário.

---

[9] www.mathworks.com/access/helpdesk/help/techdoc/ref/mex.html

O código fonte de arquivo MEX em C, denominado C-MEX, é dividido em duas partes: a rotina de interface entre MATLAB e C, e a rotina computacional, como mostrado na figura 13. Na rotina de interface, os dados de entrada e saída são convertidos em ponteiros de dados ANSI C. Por exemplo, uma matriz *mxn* de números reais é convertido para um ponteiro *double* com tamanho *m*n*. No caso dos dados de saída, eles devem ser criados dentro do C-MEX antes de serem convertidos para ponteiros. Sendo assim, as variáveis de saída só são atribuídas no C-MEX.

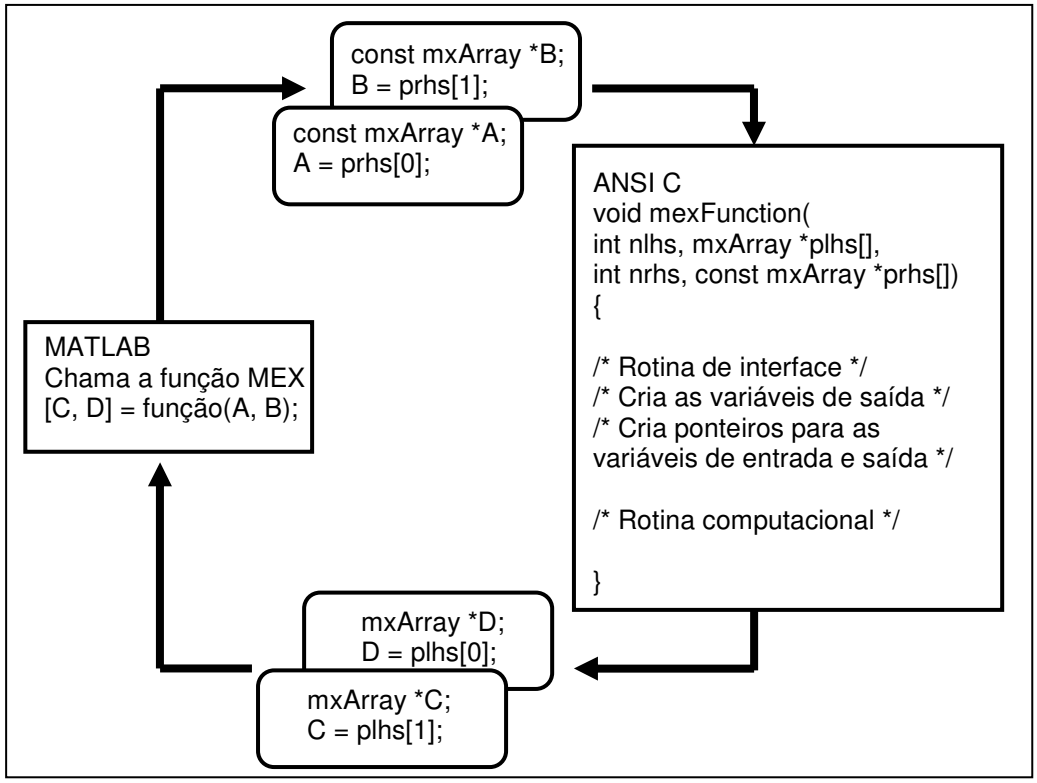


Figura 13: Ciclo do C-MEX.

O ambiente de teste é um sistema híbrido, onde uma função M do MATLAB chama o emCGA desenvolvido em ANSI C e, depois, analisa seus resultados. Resumidamente, o sistema consiste de um arquivo M e um arquivo DLL, onde o arquivo M possui código de alto nível que executa os códigos em C e analisa seus resultados.

3.3 IMPLEMENTAÇÃO EM *HARDWARE* DO emCGA

A implementação em lógica reconfigurável do emCGA apresenta algumas particularidades inerentes do projeto em *hardware* de um algoritmo. Em primeiro lugar existe a possibilidade de explorar a execução paralela de blocos funcionais que permitam este paralelismo. No caso, como cada elemento do vetor de probabilidades é independente um do outro, é possível implementar os estados de Geração e Atualização com blocos operando em paralelo. A representação de cada um dos l elementos do vetor de probabilidade é feita utilizando um número inteiro de $\log_2(n)$ bits. Para diferenciar os barramentos é adotado uma população com 256 indivíduos (barramento de 8 bits), um cromossomo de tamanho 10 bits e probabilidade de mutação representada por um inteiro de 16 bits. Para diferenciar os tamanho dos barramentos de *fitness* dos outros barramentos foi considerado um barramento de 6 bits, que simula um problema que também calcula o *fitness* com precisão de 6 bits.

No caso a probabilidade 0 ou 1 é representada por 0 ou 255, respectivamente. Isto é feito para evitar operações com ponto flutuante em *hardware,* o que seria muito custoso em termos de recursos. A utilização de um número inteiro de 8 bits reflete a necessidade de atualização do vetor de probabilidades a uma taxa de $1/n$. Isto demonstra a necessidade de configuração do projeto de *hardware* aos parâmetros do problema a ser resolvido. As figuras 14, 15 e 16 apresentam os diagramas de blocos que descrevem a implementação em *hardware* de um problema com os parâmetros definidos anteriormente. Estas figuras refletem a escolha destes parâmetros.

Na figura 14 são apresentados os três blocos necessários para operação e configuração do emCGA: o gerador de números pseudo-aleatórios (RNG - *Random Number Generator*) o parâmetro de mutação utilizado (MPM – *Mutation Probabily Memory*) e a máquina de estados descrita na figura 8 e também na tabela 2. O bloco RNG, em função do paralelismo adotado, é responsável pela geração de um barramento de 64 bits para cada bloco gerador de bit do cromossomo (BG - *Bit Generator*), mostrado na figura 15. Trata-se de um barramento de 64*10 bits (640 bits). Em cada barramento os 32 bits menos significativos são usados pelo operador de mutação e os 32 bits mais significativos são usados para geração de um novo indivíduo. Contudo, em função dos parâmetros adotados, somente uma parte de barramento é efetivamente utilizada: 16 bits para o operador de mutação (bit 0 até 15) e 8 bits para geração do novo indivíduo (bit 32 até 39). Aqui fica clara uma diferença significativa em relação à implementação em software, onde os números aleatórios são gerados de forma seqüencial.

Como um novo número é gerado a cada geração tem-se uma economia significativa de operações. A máquina de estados da figura 8 é integralmente implementada na versão em *hardware*. Inicialmente é feita a carga dos parâmetros de configuração sincronizada pelo "bit carga". Feito isto é possível gerar um "inicio bit", levando o algoritmo ao estado de inicialização, onde o vetor de probabilidade no bloco PRM (*PRobability Memory*), BM (*Bit Memory*) e EFM (*Elite Fitness Memory*) são inicializados (ver figuras 16 e 17). A máquina de estados entra no ciclo gerar, competir, atualizar e verificar até a convergência da população, ou seja, todos os PRMs alcançarem 0 ou 255.

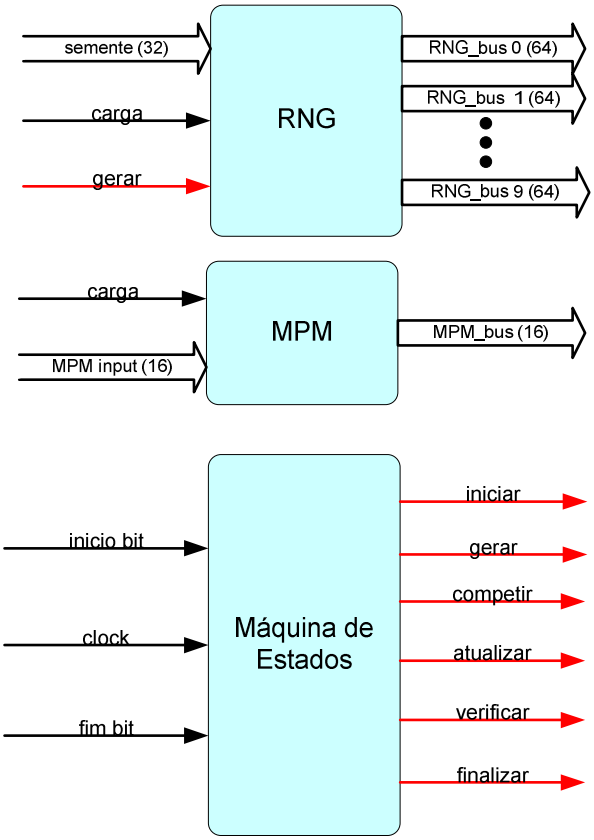


Figura 14: Blocos necessários para operação e configuração do emCGA.

A figura 15 detalha o bloco BG, responsável pela geração de um novo indivíduo e atualização do vetor de probabilidades. Esta é outra diferença significativa em comparação com a implementação em *software*. Cada bit relacionado com o cromossomo dos indivíduos, bem como cada elemento do vetor de probabilidade associado a este bit é processado em paralelo. Como no *software* estas etapas são executadas de forma seqüencial também ocorre uma redução significativa de operações. Os blocos CMP (*CoMParator*) são blocos

comparadores de números inteiros e junto com o bloco XOR (ou exclusivo) geram o bit correspondente do cromossomo Novo. Observe que é neste bloco que é inserido o operador de mutação. No bloco BM, através do sinal "resultado" é processada a técnica de elitismo, onde o melhor indivíduo permanece ao longo das gerações. Os blocos BGs acumulam uma série de etapas do algoritmo de forma proposital, buscando-se aproveitar ao máximo a construção paralela dos mesmos.

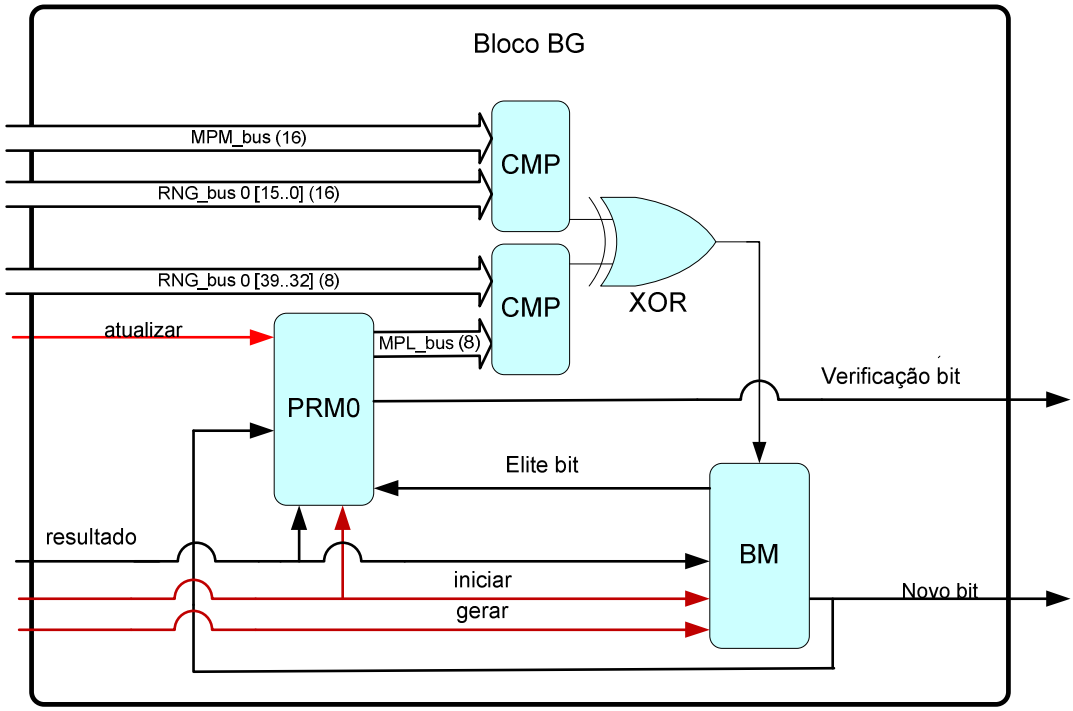


Figura 15: Representação do bloco de geração dos bits dos cromossomos Elite e Novo, denominado BG (*Bit Generator*).

Na figura 16 pode ser observada a operação paralela do algoritmo, onde são representados os 10 blocos BGs associados à geração de um novo cromossomo. No estado de competição é ativado o bloco FEV (*Fitness EValuation*), onde o *fitness* do novo cromossomo é calculado. Optou-se por um *fitness* de 6 bits para que na figura este barramento fique diferenciado dos outros. A lógica do bloco CMP para os barramentos de *fitness* depende se o problema é de maximização ou minimização. O bloco EFM durante a inicialização é carregado com o pior valor de *fitness* possível para o problema. Este bloco, juntamente com o FEV e o CMP associado geram o sinal "resultado" que será usado pelos blocos BGs para a etapa de atualização. O bloco MO (*Memory Output*) agrupa o bits de saída dos BGs para

posterior processamento. Finalmente, o bloco CV (*Convergence Verify*) verifica a convergência do vetor de probabilidades, causando a finalização do algoritmo quando for o caso. A geração do sinal "fim_bit" sinaliza à máquina de estado para entrar no estado de finalização, disponibilizando ao usuário o vetor de probabilidades que será usado para determinar a solução final.

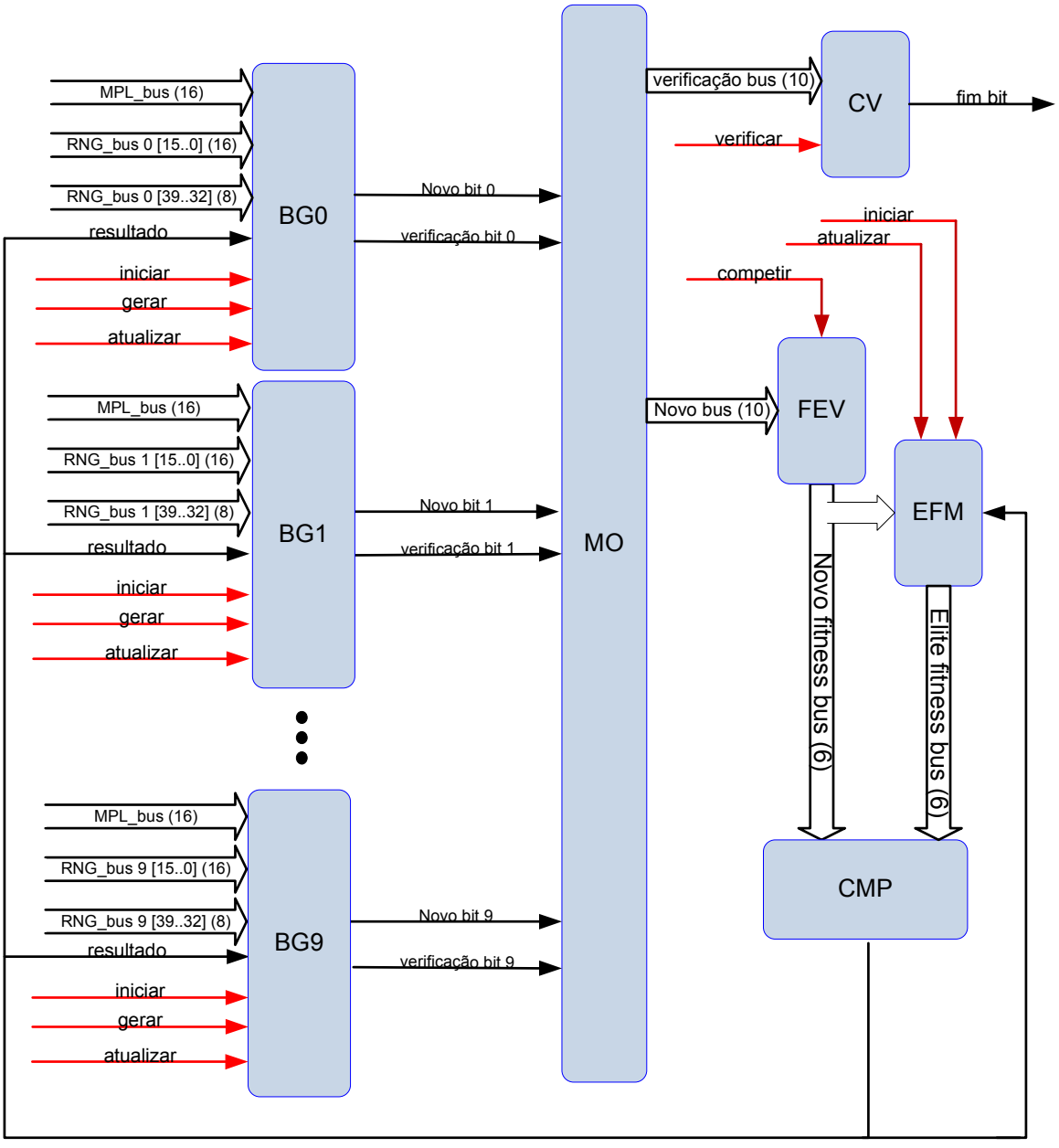


Figura 16: Diagrama de blocos representando a operação paralela do emCGA em *hardware*, bem como os blocos adicionais para operação do mesmo.

Algumas observações sobre o emCGA em *hardware*:

- O número de blocos BGs está diretamente relacionado com o comprimento do cromossomo. O mesmo não depende do tamanho da população e sim do problema.
- O parâmetro probabilidade de mutação pode utilizar até 32 bits. Contudo, foi utilizado 16 bits de precisão por ilustração.
- O barramento de cada elemento do vetor de probabilidades está relacionado com o tamanho da população. No caso, como foi usado uma população de 256 indivíduos, estes barramentos terão 8 bits.
- O barramento "Novo *fitness*" e "Elite *fitness*" são dependentes do problema. Por exemplo, para o problema *OneMax* de 10 bits, o *fitness* será um valor inteiro entre 0 e 10, o qual pode ser representado com 4 bits.
- O bloco comparador associado aos blocos FEV e EFM também é dependente do problema ser de maximização ou de minimização.

A implementação do emCGA em *hardware* é desenvolvida no kit UP3 que permite a utilização da interface paralela. Através desta interface o emCGA em *hardware* se comunica com um PC que possui uma interface de usuário. Através da interface do usuário e da interface paralela é possível dar a carga da semente do RNG e da probabilidade de mutação no MPM, iniciar a execução do emCGA través do "início bit" e receber a solução final. A interface do usuário é desenvolvido em um sistema híbrido de ANSI C compilado com Visual C++ e MATLAB versão 7 no sistema operacional Windows XP, como o emCGA em *software*.

3.4 DESEMPENHO DO emCGA EM *SOFTWARE*

Nesta seção é apresentado um estudo do desempenho do emCGA, proposto neste trabalho, através de um estudo aprofundado da relação da probabilidade de mutação no seu desempenho e comparação com outros AGCs selecionados. Devido ao fato da implementação em *hardware* ser mais complexa do que a implementação em *software* e que tais implementações alcançarão os mesmos resultados para mesmos parâmetros de entrada (parâmetros do emCGA, do problema e do gerador de números pseudo-aleatórios) optou-se pelo estudo através de uma implementação em *software*.

Feita esta consideração, devido à natureza e objetivo dos AGCs, a análise do desempenho deve considerar não somente a qualidade da solução (o melhor *fitness*), mas também o custo computacional para alcançar esta qualidade de solução. O custo computacional intrínseco (custo computacional da execução de uma geração) dos algoritmos estudados é igual ou semelhante para cata geração. Desta forma, o custo computacional pode ser mais facilmente representado pelo número de avaliações de *fitness* sem a perda de informações relevantes.

Nesta seção o desempenho do emCGA é analisado em duas etapas: estudo da relação da probabilidade de mutação no desempenho do emCGA e comparação do desempenho do emCGA com o neCGA e mCGA. O estudo da relação da probabilidade de mutação no desempenho tem o objetivo de recomendar uma probabilidade de mutação de forma que o emCGA possa alcançar um alto desempenho. Para isto procura-se entender a influência da variação da probabilidade de mutação no desempenho para diferentes complexidades de problema. Na segunda etapa, o desempenho do emCGA é comparado com outros AGCs selecionados na literatura. Estes AGCs foram selecionados de forma a restringir-se àqueles algoritmos que não apresentam um aumento na complexidade de implementação e nem no custo computacional intrínseco do algoritmo.

Devido à natureza estocástica dos AGs e, por conseqüência, dos AGCs, o resultado de uma execução não é apropriado para estimar o seu desempenho. Tradicionalmente, os AGs usam os valores médios de 100 execuções para estimar este desempenho, e algumas vezes também utilizam o desvio padrão como complemento. Este modelo também é utilizado neste trabalho.

3.4.1 Relação da probabilidade de mutação no desempenho do emCGA

O estudo da relação da probabilidade de mutação no desempenho do emCGA é feito através de um estudo empírico em diferentes variações nas funções de *fitness*. A primeira variação é no tipo do problema, onde foram selecionados quatro problemas de otimização, isto é, o *Circle*, o *Sphere*, o *Griewank* e uma função discreta simples chamada de *Discrete1*. Estas funções serão explicadas em detalhes na seção 3.4.3. Ainda, cada um destes problemas teve sua complexidade alterada através da dimensionalidade e do comprimento dos genes. O comprimento dos genes é definido como o número de alelos necessários para representar um

gene. Como é utilizada a representação binária, o comprimento dos genes é definido pelo número de bits. O aumento deste número altera o espaço de busca do algoritmo. Com este aumento é esperado que a melhor solução seja mais complicada de ser encontrar, ou seja, o problema seja mais complexo. Cada tipo de gene necessita uma codificação específica. As codificações para cada um dos comprimentos de gene são mostradas na tabela 3. Contudo, somente na alteração da dimensionalidade é alterada a complexidade da função do problema, conforme mostrado na seção 3.4.3. Com a variação da dimensionalidade é possível analisar o desempenho quando a complexidade da função também é alterada, além do espaço de busca. Desta forma, varia-se a complexidade do problema de otimização com relação à função a ser otimizada e ao espaço de busca. A Figura 17 ilustra o posicionamento de cada uma das variações com relação ao tipo de complexidade.

Tabela 3. Codificação dos cromossomos para diferentes comprimentos dos genes.

Comprimento dos genes (bits)	Faixa		Precisão
	Mínimo	Máximo	
8	1,28	-1,27	0,01
12	2,05	-2,05	0,001
16	32,77	-32,77	0,001
20	524,29	-524,29	0,001

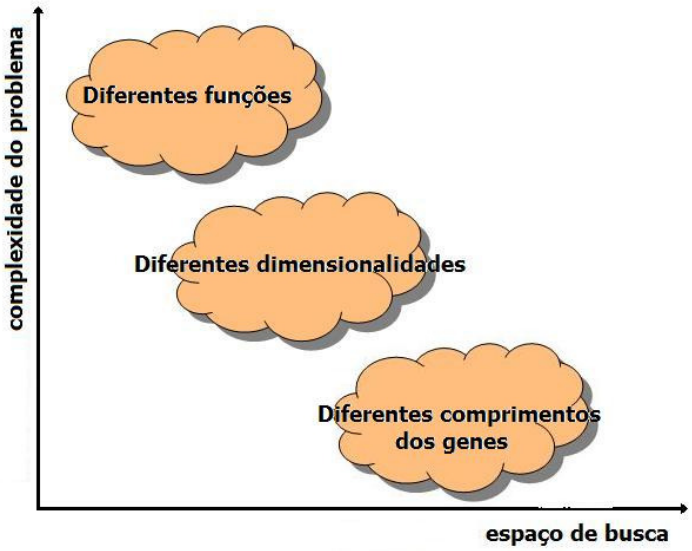


Figura 17: Posicionamento das diferentes formas de variar a complexidade dos problemas nos AGCs

Contudo, este estudo é empírico e requisita um número elevado de experimentos. Neste trabalho foram realizados 8080 experimentos, ou seja: 101 diferentes taxas de mutação para cada uma das 80 combinações de complexidades, isto é, para 4 problemas, 5 dimensionalidades e 4 comprimentos dos genes diferentes. Ressalva-se que em cada experimento são obtidos o valor de *fitness* médio e o número médio de avaliações de *fitness* para 100 execuções independentes. De cada experimento foi extraído o valor do *fitness* médio e o número médio de avaliações de *fitness*. Para facilitar a compreensão destes resultados, algumas convenções são utilizadas:

- Define-se ϕ o valor médio de *fitness* e ε como sendo o número médio de avaliações de *fitness*.
- A probabilidade de mutação é definida como sendo μ.
- O comprimento do cromossomo é definido como l.
- A diferença entre o maior e o menor *fitness* médio é denominada como altura do vale da qualidade da solução e representada por $\Delta\phi$. De maneira análoga, define-se a altura da crista de custo computacional como sendo a diferença entre o maior e o menor número médio de avaliações de *fitness,* sendo representada por $\Delta\varepsilon$.
- O melhor desempenho é definido como a solução com a melhor qualidade de solução, isto é, a solução com o menor ϕ. Pode ser observado que, em alguns casos, mais de uma probabilidade de mutação alcança a mesma qualidade de solução. Logo, nestes

casos, o melhor desempenho será quando alcançar a melhor qualidade de solução com o menor custo computacional. Define-se então, o melhor desempenho alcançado com probabilidade de mutação μ^*, *fitness* médio ϕ^*, e número médio de avaliações de *fitness* ε^*.

- A dimensionalidade do problema é definida por d.

A figura 18 mostra curvas de (ϕ,μ) e (ε,μ) sobrepostas, e através destas curvas se pode observar $\Delta\phi$, $\Delta\varepsilon$ e os pontos (ϕ^*,μ^*) e (ε^*,μ^*). Pode-se observar que curva (ϕ,μ) tem a forma de um vale. A profundidade do vale pode ser representada por $\Delta\phi$, e quanto maior for a altura deste vale, maior é a influência de μ na qualidade da solução. Da mesma forma, pode-se observar que a curva (ε,μ) forma uma crista com altura $\Delta\varepsilon$. Analogamente, quanto maior a altura desta crista, maior a influência de μ no custo computacional. No μ com valor de aproximadamente 2,8% se encontra a posição de melhor qualidade de solução, ou seja, os pontos (ϕ^*,μ^*) e (ε^*,μ^*).

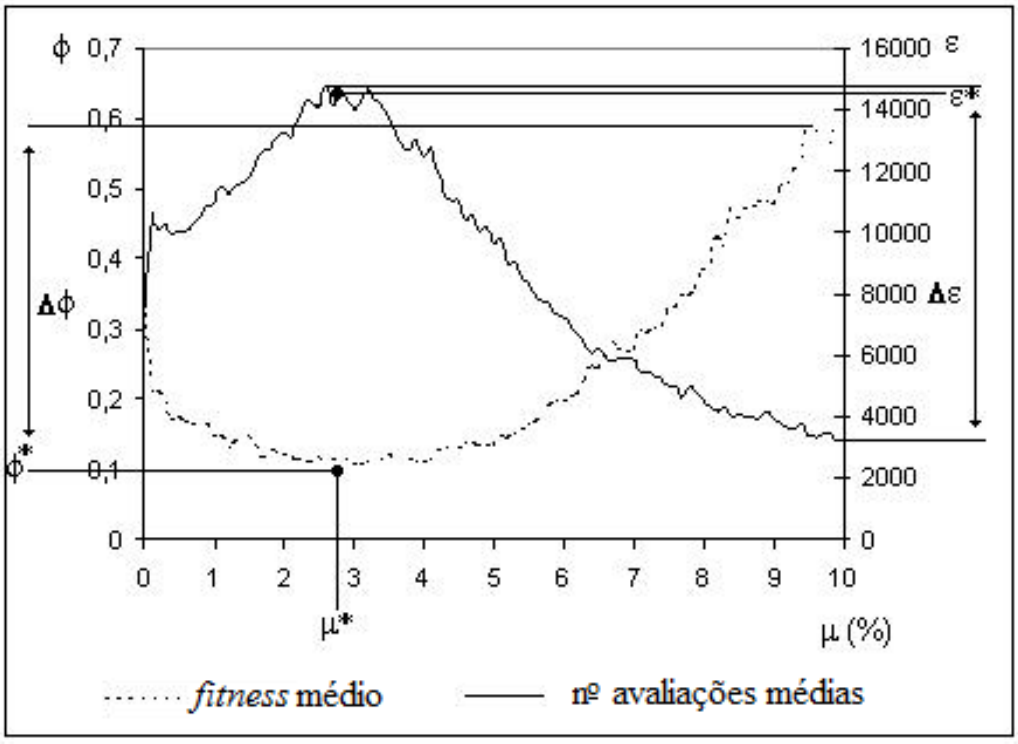


Figura 18: Resultados dos experimentos para o problema Griewank com dimensionalidade 10 e comprimento de gene de 20 bits. Os resultados para cada experimento é a média de 100 rodadas.

Além dos parâmetros estudados, o emCGA tem o tamanho da população. A escolha do tamanho da população nestes experimentos foi de 256 indivíduos. Espera-se que com este tamanho a maioria dos problemas alcancem bons resultados, e que, em alguns resultados, o desempenho do emCGA seja testado quando este tamanho é insuficiente, ou está no limiar da insuficiência, para alcançar um bom resultado.

3.4.2 Comparação do desempenho do emCGA com o neCGA e mCGA

Dentre as diversas extensões apresentadas na literatura, algumas foram selecionadas para verificar se a extensão usada no emCGA contribui com aumento no desempenho do AGC. As extensões que apresentaram resultados proeminentes e não aumentaram o custo computacional comparados com AGC original foram o neCGA (AHN e RAMAKRISHNA, 2003) e mCGA (GALLAGHER, VIGRAHAM e KRAMER, 2004). Esta comparação acrescenta mais duas funções diferentes às utilizadas no estudo da probabilidade de mutação: as funções *Schaffer F6* e *Discrete2*, uma função discreta baseada na *Discrete1*.

O parâmetro comum a todas as extensões do AGC é o tamanho da população. Assim, na comparação de desempenho todas as extensões usam o mesmo conjunto de tamanho de população, ou seja, 32, 64, 128, 256, 512, 1024 e 2048 indivíduos para os problemas *Sphere*, *Griewank*, *Discrete1* e *Discrete2*, todos de 15 dimensões, e acrescentando experimentos para 4096 e 8192 indivíduos para os problemas *Circle* e *Schaffer F6*, de 2 dimensões. Estes valores são todos potência de 2, pois somente nestes tamanhos que a taxa de atualização pode ser substituída por uma regra inteira. Utilizando esta regra inteira o AGC fica mais simples para aplicações em *hardware*, uma das contribuições deste trabalho.

Contudo, cada extensão também possui um parâmetro específico próprio. O neCGA possui o parâmetro de comprimento da herança (*length of inheritance*) e é recomendado o valor de 0,1**n* em (AHN e RAMAKRISHNA, 2003), onde *n* é o tamanho da população. Através de resultados empíricos em aplicação aos problemas *DeJong F1* a *F5*, a probabilidade de mutação recomendada no mCGA foi de 5% (GALLAGHER, VIGRAHAM e KRAMER, 2004). Para o emCGA a probabilidade de mutação será baseada no estudo para recomendar o valor deste parâmetro- contudo, pode-se adiantar que este valor é de 1/*l*. A codificação dos genes, bem como as taxas de mutação utilizadas são mostradas na tabela 4. Nos problemas de

baixa dimensionalidade foi utilizado um valor bem maior de probabilidade de mutação do que a recomendada. Quanto maior a dimensionalidade maior é a complexidade da função do problema. Através da análise apresentada na seção 5.1.1.3, espera-se que problemas de baixa complexidade não tenham forte influência da probabilidade de mutação na qualidade da solução. No entanto, espera-se existir uma significativa influência no custo computacional, onde a probabilidade de mutação pode ser recomendado por um valor próximo de 10%. Assim optou-se por usar taxas de mutação de 8,5% para problemas de dimensionalidade 2. Ainda, as taxas de mutação foram arredondadas com precisão de 0,5% por conveniência.

Tabela 4. Codificação dos problemas e parametrização do emCGA para comparação de desempenho com neCGA e mCGA.

Problema	Dimensão	Faixa		Comprimento dos Genes (bits)	Comprimento do Cromossomo	Prob. de mutação sugerida	Prob. de mutação utilizada
		Mínimo	Máximo				
Circle	2	-32,767	32,768	16	32	3,1%	8,5%
Shaffer-F6	2	-32,767	32,768	16	32	3,1%	8,5%
Discrete1	15	-32768	32767	16	240	0,4%	0,5%
Discrete2	15	-32768	32767	16	240	0,4%	0,5%
Griewank	15	-131,071	131,072	18	270	0,4%	0,5%
Sphere	15	-131,071	131,072	18	270	0,4%	0,5%

3.4.3 Problemas de *benchmark*

Funções matemáticas são freqüentemente usadas na comparação de algoritmos de otimização, incluindo AGs. Neste trabalho foram selecionadas na literatura, por suas características, algumas funções matemáticas definidas em um espaço *n*-dimensional ($\Re^n$).

A função *Circle* (MÜLLER, MARCHETTO, AIRAGHI *et al*, 2002), definida na equação 2 possui seu mínimo global em $f_c(x)=0$ quando $x=[0...0]^T$. Este problema de minimização é multimodal com diversos mínimos locais localizados concentricamente próximos ao mínimo global . Um exemplo desta função é mostrado na figura 19, para $n = 2$ e $x \in [-1{,}27,\ 1{,}28]$, isto é a resolução é de 8 bits e precisão de 0,01.

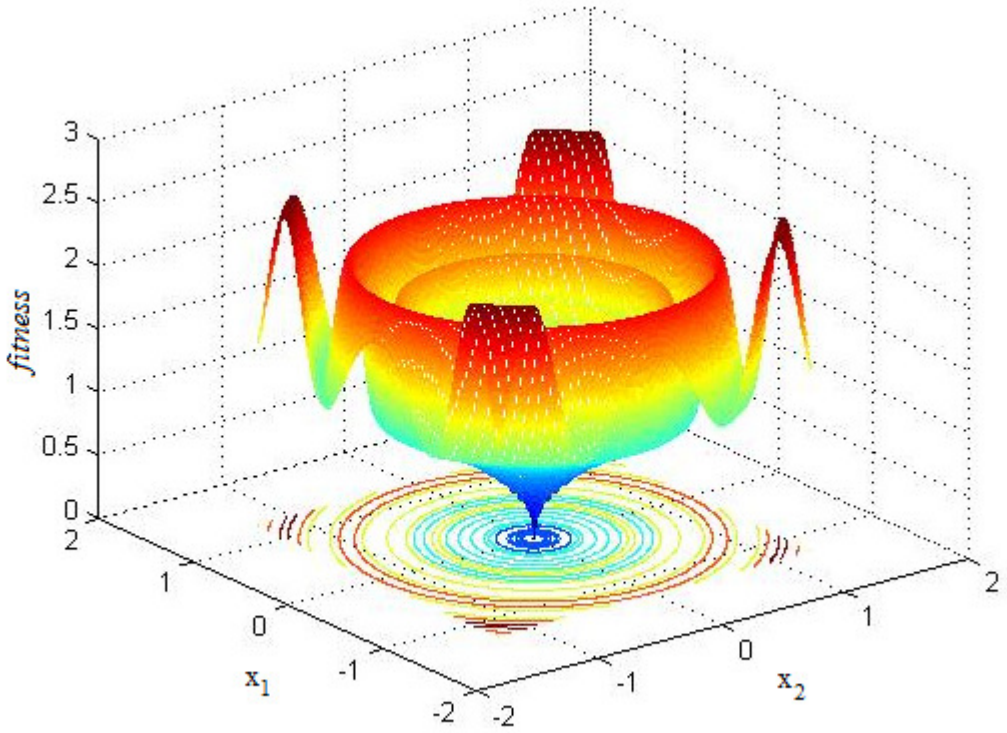


Figura 19: Função *Circle* com *n*= 2, resolução de 8 bits e precisão 0,01.

$$\min f_c(x) = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{4}} \cdot \left[\sin^2\left(50 \cdot \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{10}}\right) + 1{,}0\right] \qquad (2)$$

Onde:

n = dimensionalidade do problema.

A função *Schaffer* F6, definida pela equação 3, é mostrada na figura 20 para n = 2 e $x \in$ [-25,5, 25,6]. O seu mínimo global se encontra em $f_{F6}(x)=0$ quando $x=[0...0]^T$. Também é um problema de minimização onde os diversos mínimos locais localizados concentricamente ao global que a tornam complexa para a maioria dos algoritmos *hillclimbings*.

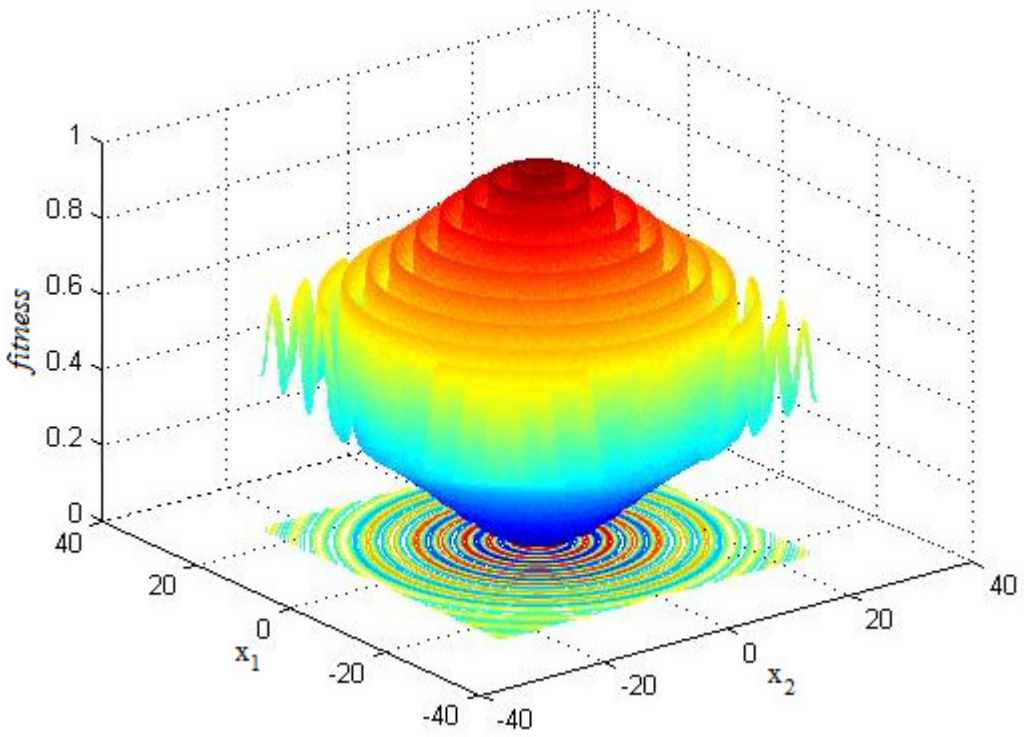


Figura 20: Função *Schaffer* F6 com n=2, resolução de 9 bits e precisão 0,1.

$$\min f_{F6}(x) = 0{,}5 + \frac{\sin^2\left(\sum_{i=1}^{n} x_i^2\right) - 0{,}5}{\left(1 + 0{,}001 \cdot \left(\sum_{i=1}^{n} x_i^2\right)\right)^2} \tag{3}$$

Onde:

n = dimensionalidade do problema.

A função de minimização *Sphere* ou *DeJong* F1 descrita na equação 4 é unimodal com mínimo global em $f_S(x)=0$ quando $x=[0...0]^T$. Esta função deve ser fácil para os AGs ou um algoritmo *hillclimbing*. Esta função para dimensionalidade 2 e $x \in [-127, 128]$ é mostrada na figura 21.

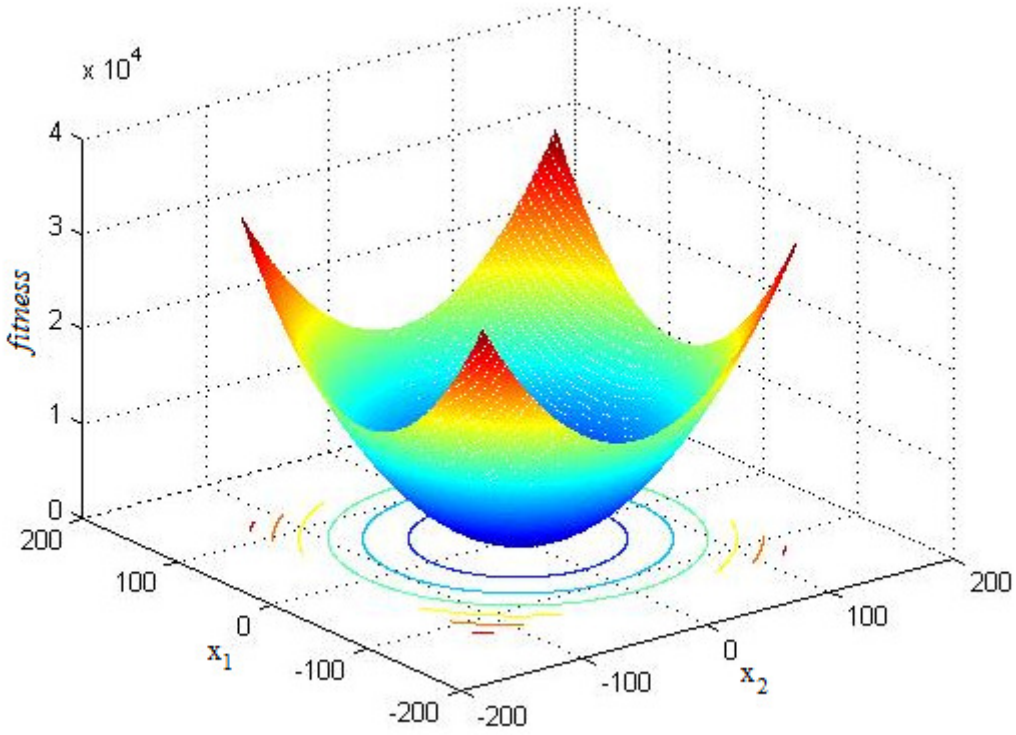


Figura 21: Função *Sphere* com *n*=2, resolução de 8 bits e precisão 1.

$$\min f_S(x) = \sum_{i=1}^{n} x_i^2 \tag{4}$$

Onde:

n = dimensionalidade do problema.

A função de minimização *Griewank*, mostrada na equação 5, possui múltiplos mínimos locais, mas com o global abaixo do nível da parábola. Esta função é considerada difícil para os AGs, pois o termo de produtos torna as variáveis muito inter-relacionadas, isto é: a variação do valor de *fitness* devido à variação de uma variável esta relacionada também com os valores das outras variáveis. Na figura 22 é mostrada esta função para dimensionalidade 2 e $x \in$ [-127, 128].

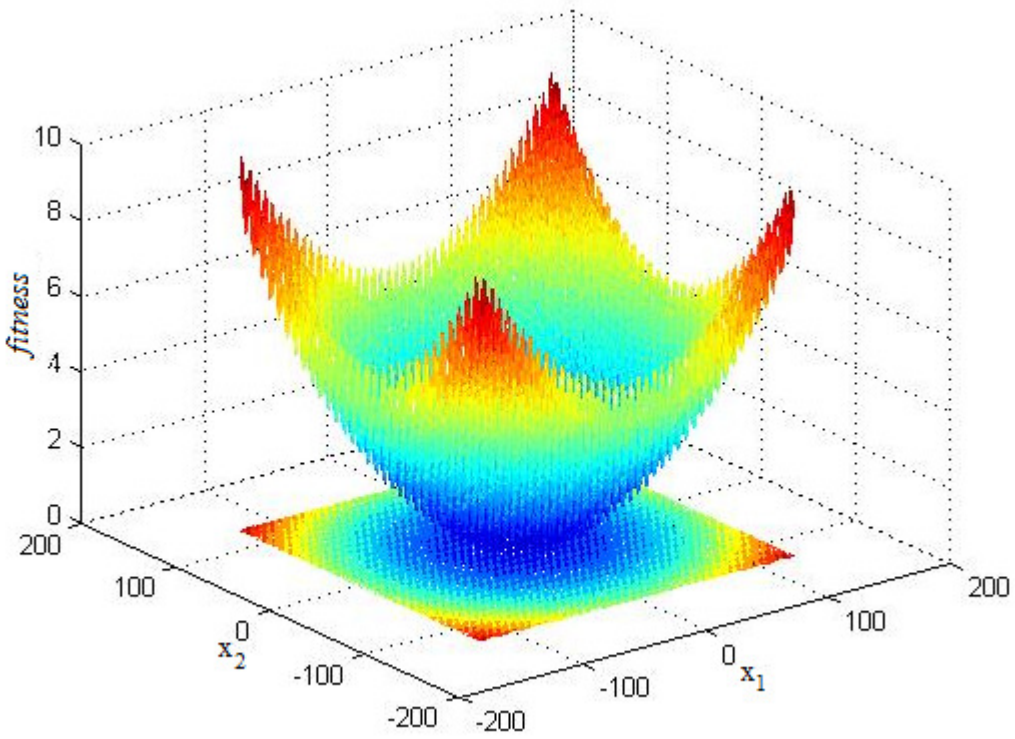


Figura 22: Função *Griewank* com *n*=2, resolução de 8 bits e precisão 1.

$$\min f_G(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos \frac{x_i}{\sqrt{i}} + 1 \qquad (5)$$

Onde:

n = dimensionalidade do problema.

Com o objetivo de observar o desempenho em funções simples, mas discretas, foram propostas as funções *Discrete1* e *Discrete2*, definidas nas equações 6 e 7, respectivamente. Estas funções possuem características semelhantes à função *Sphere*, com exceção a representação numérica das variáveis que pertencem aos números inteiros.

$$\min f_{D1}(x) = \sum_{i=1}^{n} |x_i| \tag{6}$$

$$\min f_{D2}(x) = \sum_{i=1}^{n} x_i^2 \tag{7}$$

Onde:

n = dimensionalidade do problema.

3.5 O emCGA APLICADO À DETECÇÃO DE OBJETOS EM IMAGENS

Encontrar objetos em imagens é um problema freqüente em ambientes industriais. Contudo, a detecção de objetos e a sua descrição de quão específico aparece em uma imagem usando procedimentos de busca (*matching procedure*) tradicionais normalmente envolvem um esforço computacional alto. Particularmente, este esforço aumenta quando existe translação e rotação do objeto, e aumenta mais ainda quando este objeto é parcialmente ocultado ou apresenta variações na sua escala (JAIN, KASTURI e SCHUNCK, 1995). Desta forma, é importante o desenvolvimento e implementação de algoritmos de busca rápidos para este problema. Como freqüentemente são encontradas aplicações onde estes algoritmos devem ser executados em tempo real, muitas vezes, é necessário um algoritmo com alta velocidade de execução. Nestas situações, a busca exaustiva é inadequada e métodos meta-heurísticos podem ter um papel fundamental.

Desta forma, a detecção de objetos em imagens é adequada para desempenho do emCGA em problemas reais. Também neste caso, como a implementação do emCGA em *hardware* e em *software* alcançam os mesmos resultados para os mesmos parâmetros de entrada (parâmetros do próprio emCGA, do problema e do gerador de números pseudo-aleatórios), os resultados obtidos em *software* poderão ser estendidos para futuras implementações em *hardware*. Assim, os estudos realizados com detecção de objetos em imagens usando o emCGA são feitos com a implementação em *software* deste algoritmo.

Nesta seção, são exploradas duas técnicas de processamento digital de imagens para extrapolar propriedades significativas de um objeto em uma imagem e criar um modelo. Então, o emCGA gerencia a busca deste modelo do objeto em outras imagens. Uma vez encontrado o objeto, sua posição e ângulo de rotação são determinados. Este trabalho é baseado em trabalhos anteriores (CENTENO, LOPES, FELIZBERTO *et al*, 2005) e (SCHNEIDER, LOPES e PEDRONI, 2006), os quais utilizaram AG e PSO (*Particle Swarm Optimization*), respectivamente, para solução do problema.

3.5.1 Representação do modelo do objeto

São propostos dois modelos do objeto utilizando duas técnicas diferentes de processamento de imagens. No primeiro modelo é extraída a intensidade de luz (brilho) do

objeto em uma imagem RGB (*Red, Green and Blue*) de 8 bits. Denomina-se este modelo como modelo de intensidade de luz. No segundo modelo é extraído um valor binário representando a existência ou não de luz. Denomina-se este modelo como modelo binário. Cada uma destas representações é detalhada nas seções 3.5.1.1 e 3.5.1.2.

3.5.1.1 Modelo de intensidade de luz

O modelo de intensidade de luz é um modelo com a informação de intensidade de luz da imagem do objeto a ser detectado. A intensidade de luz de um pixel deste objeto é determinada através de 30% da intensidade da cor vermelha, 59% da cor verde e 11% da cor azul[10], como mostrado na figura 23. Desta forma, é formada uma matriz M_{ilum}, onde cada elemento desta matriz é um valor de 256 tons de cinza correspondente à intensidade de luz do pixel na mesma posição na imagem. Por definição, quando o valor do elemento é 0 significa que a intensidade de luz é 0%, e 255 quando é 100%. Um exemplo de aplicação do modelo de intensidade de luz é representado por uma imagem em tons de cinza na figura 24, onde um pedaço de 11x11 *pixels* é destacado. Nesta mesma figura também é mostrada a matriz M_{ilum} dos *pixels* destacados.

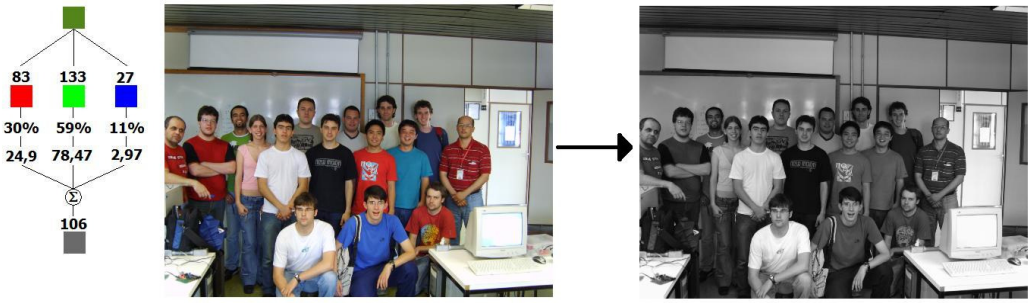
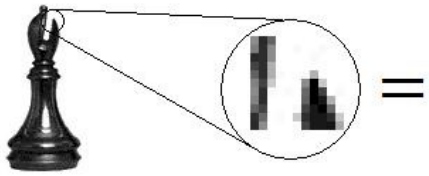


Figura 23: Exemplo da extração da intensidade de luz.

[10] http://www.mathworks.com/support/solutions/data/1-1ASCU.html



$$\begin{bmatrix}
140 & 42 & 100 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\
126 & 46 & 140 & 255 & 255 & 250 & 255 & 255 & 250 & 255 & 255 \\
42 & 46 & 140 & 255 & 255 & 255 & 255 & 250 & 255 & 255 & 255 \\
67 & 46 & 177 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\
99 & 46 & 176 & 250 & 255 & 255 & 250 & 177 & 255 & 255 & 255 \\
131 & 67 & 202 & 255 & 255 & 255 & 140 & 0 & 201 & 255 & 255 \\
140 & 47 & 250 & 255 & 250 & 255 & 105 & 0 & 67 & 255 & 250 \\
131 & 46 & 255 & 255 & 255 & 255 & 99 & 0 & 24 & 176 & 255 \\
99 & 68 & 255 & 255 & 255 & 250 & 67 & 24 & 0 & 105 & 250 \\
46 & 105 & 255 & 250 & 255 & 201 & 46 & 24 & 24 & 46 & 201 \\
42 & 140 & 255 & 255 & 255 & 250 & 67 & 24 & 24 & 22 & 168
\end{bmatrix}$$

Figura 24: Exemplo de modelo de intensidade de luz.

3.5.1.2 Modelo binário

O modelo binário utiliza o modelo de intensidade de luz, contudo, adicionando outra técnica que extrai a informação de existência ou não de luz. A informação da existência da luz é uma binarização da matriz M_{ilum}, onde cada elemento desta matriz é comparado com um valor de limiar resultando na matriz M_{bin}. Os elementos que forem maiores que o valor de limiar terão valores iguais a 1, significando existência de luz, caso contrário, terão valores iguais a 0. Um exemplo de aplicação da técnica de binarização para um valor de limiar de 150 é ilustrada na figura 25. A imagem superior é a imagem de referência usando o modelo de intensidade de luz. Uma parte da imagem com 11x11 *pixels* é destacado, isto é: os valores da matriz *M*ilum para os *pixels* das linhas 10 a 20 e colunas 28 a 38. A técnica de binarização é aplicada na matriz M_{ilum} que resulta na matriz M_{bin}. A imagem inferior apresenta o modelo binário resultante em uma imagem preto e branco, onde a parte do modelo de 11x11 *pixels* é destacado, isto é: os valores da matriz M_{bin} para os *pixels* das linhas 10 a 20 e colunas 28 a 38.

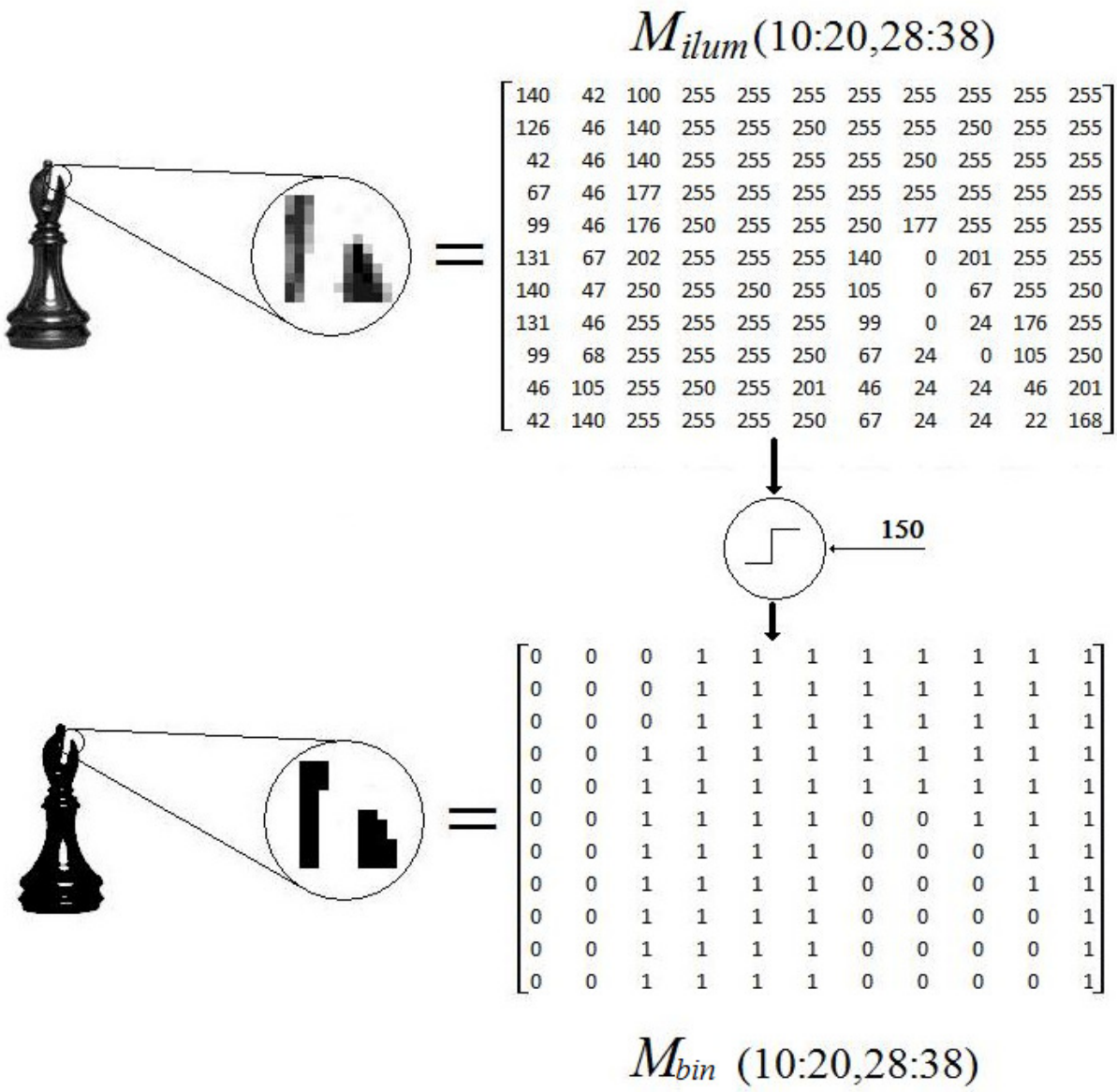


Figura 25: Exemplo de modelo binário.

3.5.2 Função de *fitness*

A função de *fitness* do emCGA mede a similaridade entre a imagem de referência e uma imagem recortada e rotacionada da imagem de entrada com o mesmo tamanho da imagem de referência. Desta forma é possível detectar se um objeto se encontra em naquela parte da imagem. O objeto é definido por uma imagem denominada de imagem de referência ou, simplesmente, referência. As imagens de entrada são previamente convertidas para um

modelo de intensidade de luz no início do algoritmo. Desta forma, é possível diminuir o custo computacional e de memória da execução da função de *fitness*.

Baseado nos trabalhos de Centeno, Lopes, Felizberto *et al* (2005) e Schneider, Lopes e Pedroni (2006), é proposta uma função que retorna a porcentagem de similaridade entre duas imagens através dos erros absolutos entre os modelos de referência e da imagem, como mostrada nas equações 8 e 9.

$$S(k) = 100\% \cdot \frac{E_{\max} - E(k)}{E_{\max}} \tag{8}$$

$$E(k) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left| F(I,k,i,j) - M_{ref}(i,j) \right| \tag{9}$$

Onde:

I = matriz com a intensidade de luz dos pixels da imagem de entrada,

F = transformada de uma imagem em um modelo, e retorna o seu elemento (i,j),

M_{ref} = matriz do modelo da imagem de referência,

$E_{\max}$ = erro absoluto máximo,

n = número de linhas na matriz do modelo de referência,

m = número de colunas na matriz do modelo de referência,

k = vetor de parâmetros da transformada,

$E(k)$ = erro absoluto entre a imagem de entrada segundo o vetor k e a imagem de referência.

Esta função de *fitness* retorna um valor entre 0 e 100%, representando a similaridade entre as imagens. Quanto maior for este valor, mais provável a existência do objeto na imagem de entrada segundo os parâmetros k. Desta forma, o objetivo é encontrar o vetor k que alcance a maior similaridade com o objeto.

Os objetos e ambientes estudados neste trabalho são tridimensionais como na figura 26a. Sendo assim, o objeto pode se deslocar nos eixos x, y e z, correspondentes aos deslocamentos horizontal, vertical e profundidade, respectivamente, além de rotacionar nos ângulos α, β e θ. Contudo, somente os deslocamentos nos eixos x e y e o ângulo θ do espaço tridimensional são representados nas imagens de referência e de entrada (ver figura 26b) e,

por conseqüência, pelos parâmetros do vetor *k*. A translação no eixo *z* e as rotações nos ângulos α e β são consideradas distorções.

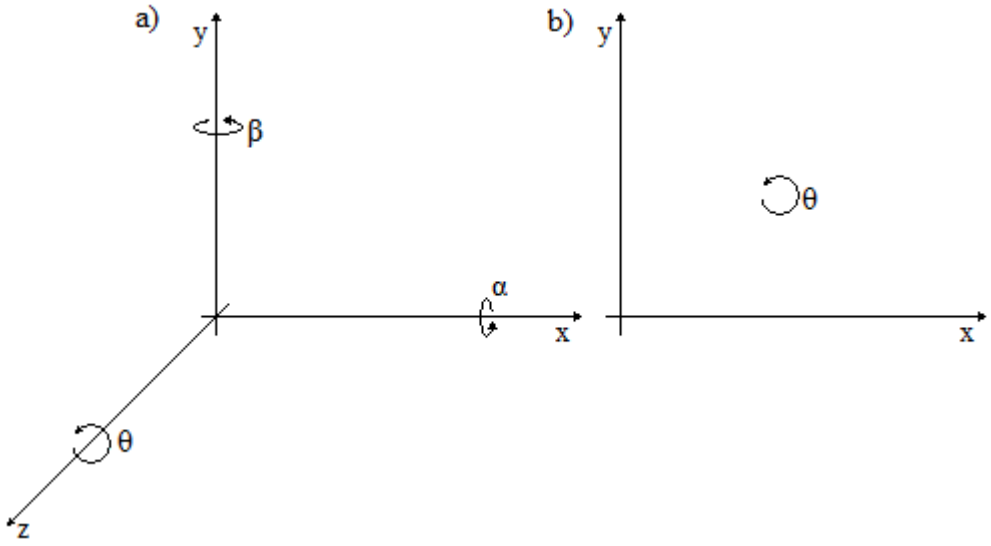


Figura 26: a) Espaço tridimensional onde são denominados os possíveis deslocamentos do objeto no ambiente da imagem. b) Plano representado na imagem de entrada.

A transformada *F* e o erro absoluto máximo E_{max} são definidos pelo seu modelo, explicado nas seções 3.5.2.1 e 3.5.2.2. A matriz M_{ref} tem *n* linhas e *m* colunas e todos os seus elementos são comparados para medir a similaridade. Esta matriz é definida no início do algoritmo através da imagem de referência.

3.5.2.1 Modelo de intensidade de luz

A imagem de entrada é sempre maior ou igual à referência. Sendo assim, o objeto contido nesta imagem pode ser encontrado com translação e rotação. Desta forma, a transformada *F* aplica translação e rotação de um ponto (i,j) e retorna o elemento correspondente de uma matriz *I* de entrada, conforme a equação 10. Como os valores de intensidade de luz variam de 0 a 255, o erro absoluto máximo de um elemento é 255. Logo, o erro absoluto máximo de todos os elementos (E_{max}) é definido na equação 11.

$$
\begin{aligned}
&F(k,i,j) = I(x',y') \\
&k = (x,y,\theta) \\
&x' = x + (i \cdot \cos\theta + j \cdot \sin\theta) \\
&y' = y + (j \cdot \cos\theta - i \cdot \sin\theta)
\end{aligned}
\tag{10}
$$

$$
E_{\max} = 255 \cdot n \cdot m \tag{11}
$$

O vetor de parâmetros k é composto pela posição (x,y) de translação e ângulo θ de rotação. A posição (x',y') é a posição transformada de (i,j) segundo este vetor de parâmetros. Contudo, como esta transformação usa funções trigonométricas que retornam números reais, o elemento retornado da matriz I é o valor do elemento mais próximo desta posição através do arredondamento de x' e y'.

No vetor k está contida a posição de translação (x,y) e ângulo θ de rotação. A posição de translação é uma posição na imagem de entrada e também é a posição central de uma possível localização do objeto. Quando se avalia uma posição próxima à borda existem algumas posições (x',y') inválidas e o valor do erro nestes pontos será sempre o erro máximo- neste caso, 255. Desta forma, é possível identificar também um objeto parcialmente oculto próximo às bordas. O cromossomo dos indivíduos no emCGA é definido por este vetor k e a sua codificação é definida pela tabela 5.

Tabela 5. Codificação para a função de *fitness* do modelo de intensidade de luz.

	Faixa			
Parâmetro	**Mínimo**	**Máximo**	**Precisão**	**Nº de bits**
$\boldsymbol{x}$	0	c	1	$\log_2 c$
$\boldsymbol{y}$	0	l	1	$\log_2 l$
$\boldsymbol{\theta}$	0	360	0,70	9

O número de bits dos parâmetros x e y são arredondados para o maior e mais próximo número inteiro. Esta limitação existe devido à codificação binária do cromossomo no emCGA. Assim, nestes casos, a busca se dará também em posições inválidas e, como para todos os outros pontos inválidos, o erro nestes pontos será 255.

3.5.2.2 Modelo binário

Neste modelo a transformada *F* aplica uma transformação translacional e rotacional no ponto (*i*,*j*) idêntica ao modelo de intensidade de luz. Desta forma, também é possível identificar objetos parcialmente ocultos nas bordas. No entanto, além destas transformações, também é aplicada a binarização dos valores da matriz *I*, conforme a equação 12. Como é mostrado na tabela 6, este modelo acrescenta mais um parâmetro no cromossomo, que define o limiar *t* de intensidade de luz. Neste caso, cada elemento terá valor igual a 0 ou 1, ou seja, o erro absoluto máximo de um elemento será igual a 1. Logo, o erro absoluto máximo de todos os elementos (E_{max}) é definido na equação 13.

$$F(k,i,j) = \begin{cases} 1 & I(x',y') > t \\ 0 & outros \end{cases} \tag{12}$$

$$k = (x, y, \theta, t)$$

$$x' = x + (i \cdot \cos\theta + j \cdot \sin\theta)$$

$$y' = y + (j \cdot \cos\theta - i \cdot \sin\theta)$$

$$E_{max} = n \cdot m \tag{13}$$

Tabela 6. Codificação para a função de *fitness* do modelo binário.

	Faixa			
Parâmetro	**Mínimo**	**Máximo**	**Precisão**	**Nº de bits**
x	0	*c*	1	$\log_2 c$
y	0	*l*	1	$\log_2 l$
θ	0	360	360/512	9
t	0	255	1	8

Neste modelo, como no modelo anterior, a função de *fitness* também considera o erro absoluto para os pontos (*x*',*y*') inválidos como o erro absoluto máximo de um elemento, neste caso, igual a 1. Estes pontos inválidos são gerados através da limitação da codificação binária do emCGA ou por busca em posições próximas às bordas da imagem.

3.5.3 Parâmetros do emCGA

O emCGA possui 2 parâmetros que devem ser previamente definidos: o tamanho da população e a probabilidade de mutação. A busca de um objeto em uma imagem é um problema complexo, pois é multimodal com muitos máximos locais. Além disto, os genes do cromossomo, isto é, os parâmetros sendo otimizados, são fortemente inter-relacionados. Desta forma, a probabilidade de mutação recomendada é $1/l$, segundo a análise feita na seção 5.1.1.3.

As imagens de entrada são todas com 1024 por 768 pixels. Assim, os parâmetros x e y terão tamanho de 10 bits cada um. No modelo de intensidade de luz adiciona-se também 9 bits para a representação do ângulo de rotação no tamanho do cromossomo. Desta forma, o tamanho do cromossomo nos problemas usando este modelo é 29 bits. Em função disto, o parâmetro de mutação recomendado é aproximadamente 3,4%. No modelo binário acrescenta-se mais 8 bits para a representação do limiar de intensidade de luz t. Logo, o comprimento do cromossomo neste caso é de 37 bits e a probabilidade de mutação recomendada é 2,7%.

Ainda não existe uma conclusão definitiva para a relação do tamanho da população e o desempenho dos AGs. Contudo, existe um consenso que quanto maior a população, melhor é o seu desempenho. Como os AGCs têm comportamento semelhante aos AGs, também é esperado que um problema mais complexo demande um tamanho maior para a população. No entanto, quanto maior é o tamanho da população também maior é o custo computacional. Portanto, existe um compromisso entre o desempenho e o custo computacional na escolha do tamanho da população. Ainda, para manter o emCGA implementável em *hardware*, o tamanho da população deve ser uma potência de 2. O tamanho da população que mostrou um compromisso razoável entre o desempenho e o custo computacional e ainda mantendo-se implementável em *hardware* foi de 16384 indivíduos.

Os AGs, bem como o emCGA, têm natureza estocástica implementada através um gerador de números pseudo-aleatórios. Assim é necessário também definir uma semente para este gerador. Nesta execução, uma semente será determinada aleatoriamente.

3.5.4 Algoritmo de Busca Exaustiva

A maioria dos experimentos utiliza imagens retiradas de fotos digitais. Portanto, a correta localização dos objetos é de difícil determinação. Contudo, é interessante a confirmação de quão correta é a localização encontrada. Assim optou-se em comparar os resultados do emCGA com um algoritmo de busca exaustiva (ABE). O ABE avalia o *fitness* igual ao usado no emCGA para todos os possíveis valores dos parâmetros. A sua solução final é o parâmetro com o melhor valor de *fitness*.

3.6 O DESEMPENHO DO emCGA EM *HARDWARE*

O objetivo do estudo do emCGA em *hardware* é mostrar que esta aplicação tem um ganho de desempenho quando comparada com a aplicação em *software*, sem um aumento significativo de demanda por recursos de *hardware*, quando comparado com o AGC. Espera-se que a aplicação do emCGA em *hardware* obtenha uma aceleração com relação ao *software* devido às técnicas de paralelismo empregadas. Além disto, o uso da técnica de elitismo e do operador de mutação empregado também não deve aumentar a demanda por recursos de *hardware* se comparado com o AGC em *hardware* baseado no trabalho de Aporntewan e Chongstitvatana (2001).

O emCGA em *hardware* funciona de forma idêntica à aplicação em *software*. Isto é, obtêm-se os mesmos indivíduos Elite e Novo na mesma geração para execuções com os mesmos parâmetros e problema. Desta forma, as conclusões de desempenho do emCGA em *software* poderão ser estendidos para o emCGA em *hardware*, ou seja, a relação da probabilidade de mutação com o desempenho, o aumento do desempenho em comparação com outros AGCs semelhantes e a aplicabilidade em problemas reais. Assim, serão mostrados três experimentos que mostram o desempenho do emCGA em *hardware*. O primeiro é a comparação do funcionamento do emCGA em *hardware* e *software*, onde mostra que na mesma geração ambas as aplicações geram os mesmos indivíduos Elite e Novo. No segundo experimento é comparada a demanda de recursos de *hardware* das implementações do emCGA e AGC. No último conjunto de resultados será determinado o ganho de desempenho do emCGA em *hardware* com emCGA em *software*.

Nestes três experimentos é utilizado o problema *Discrete1* apresentado na seção 4.4. Este problema foi escolhido pela simplicidade de implementação em *hardware* e por ter sido utilizado como função de *benchmark* no estudo do desempenho do emCGA na seção 3.4.

As implementações em *hardware* são feitas em um kit educacional UP3 da Altera, que utiliza o FPGA Cyclone EP1C6Q240C8. O código fonte é desenvolvido no Quartus II, também da Altera, usando a linguagem VHDL para o emCGA e o problema *Discrete1*. Contudo, o código da interface paralela e a sua conexão com os códigos VHDL do emCGA e *Discrete1* é utilizado o diagrama de blocos do Quartus II. Tanto as implementações do emCGA em *software* como a implementação da interface entre o PC e o kit UP3 em *software* são feitas em um PC com processador Pentium IV 2.8GHz usando código fonte ANSI C com funções MEX do MATLAB.

CAPÍTULO 4

RESULTADOS

4.1 RESULTADOS DA RELAÇÃO DA PROBABILIDADE DE MUTAÇÃO NO DESEMPENHO DO emCGA

A análise da relação entre o desempenho do emCGA e a probabilidade de mutação através de um estudo empírico requer um número elevado de experimentos. Se estes resultados forem representados por curvas de dispersão para (ϕ,μ) e (ε,μ) para cada uma das combinações de complexidades, então haveria 160 curvas, isto é: 2 curvas, (ϕ,μ) e (ε,μ), para 4 comprimentos de gene, 5 dimensionalidades e 4 problemas. A realização desta análise seria maçante e repetitiva. Desta forma, estas curvas são representadas por superfícies tridimensionais, acrescentando-se o eixo da dimensionalidade da função, ou seja, *d.* A nova representação (ϕ,μ,d) e (ε,μ,d) reduz a representação dos resultados a 32 superfícies tridimensionais. Contudo, esta representação pode ser ainda mais reduzida analisando-se somente o comprimento dos genes de 8 e de 20 bits. Neste caso são suficientes apenas 16 superfícies tridimensionais. Estas superfícies auxiliam na análise do melhor desempenho do emCGA, já que o mesmo utiliza os resultados de melhor desempenho (μ*,ϕ*,ε*), das alturas dos relevos da qualidade de solução ($\Delta\phi$) e do custo computacional ($\Delta\varepsilon$).

As superfícies de qualidade de solução são normalizadas através da subtração do melhor valor alcançado para cada problema e para cada dimensionalidade definida pela equação 14. É esperado que a qualidade de solução diminua com o aumento da complexidade do problema. Contudo, através dos pontos (μ*,ϕ*,ε*) é possível uma análise melhor desta redução da qualidade de solução, estes ilustrados na figura 18.

$$\phi^{n}(d,\mu) = \phi(d,\mu) - \phi_{\min}(d) \tag{14}$$

Onde:

$\phi^{n}(d,\mu)$: *fitness* médio normalizado para d e μ,

$\phi(d,\mu)$: *fitness* médio para d e μ,

$\phi_{\min}(d)$: menor *fitness* médio para d.

4.1.1 Gráficos para genes de resolução de 8 bits

As figuras 27 e 28 mostram, para genes de resolução de 8 bits, as superfícies tridimensionais de (ϕ^n,μ,d) e (ε,μ,d) respectivamente, para cada um dos quatros problemas. Desta forma, os resultados são obtidos para diferentes complexidades, diferenciados através da variação da dimensionalidade do problema.

Na figura 27 é possível observar que quanto maior a dimensão do problema maior é a variação na qualidade de solução quando se varia a probabilidade de mutação. Desta forma, é possível observar a formação de um vale quando a dimensão é maior. O problema Circle é a exceção nestes resultados, pois não houve o aparecimento deste vale. Contudo, outros resultados irão ajudar entender melhor a relação da probabilidade de mutação no desempenho neste problema.

A figura 28 ilustra os resultados do custo computacional devido variações na dimensão do problema e probabilidade de mutação. Conforme é aumentado a dimensão do problema, maior é o custo computacional. Além disto, existe uma formação de uma crista que nas dimensões mais altas se encontra próxima à probabilidade de mutação 5%. Novamente, estas características não são tão evidentes para o problema Circle.

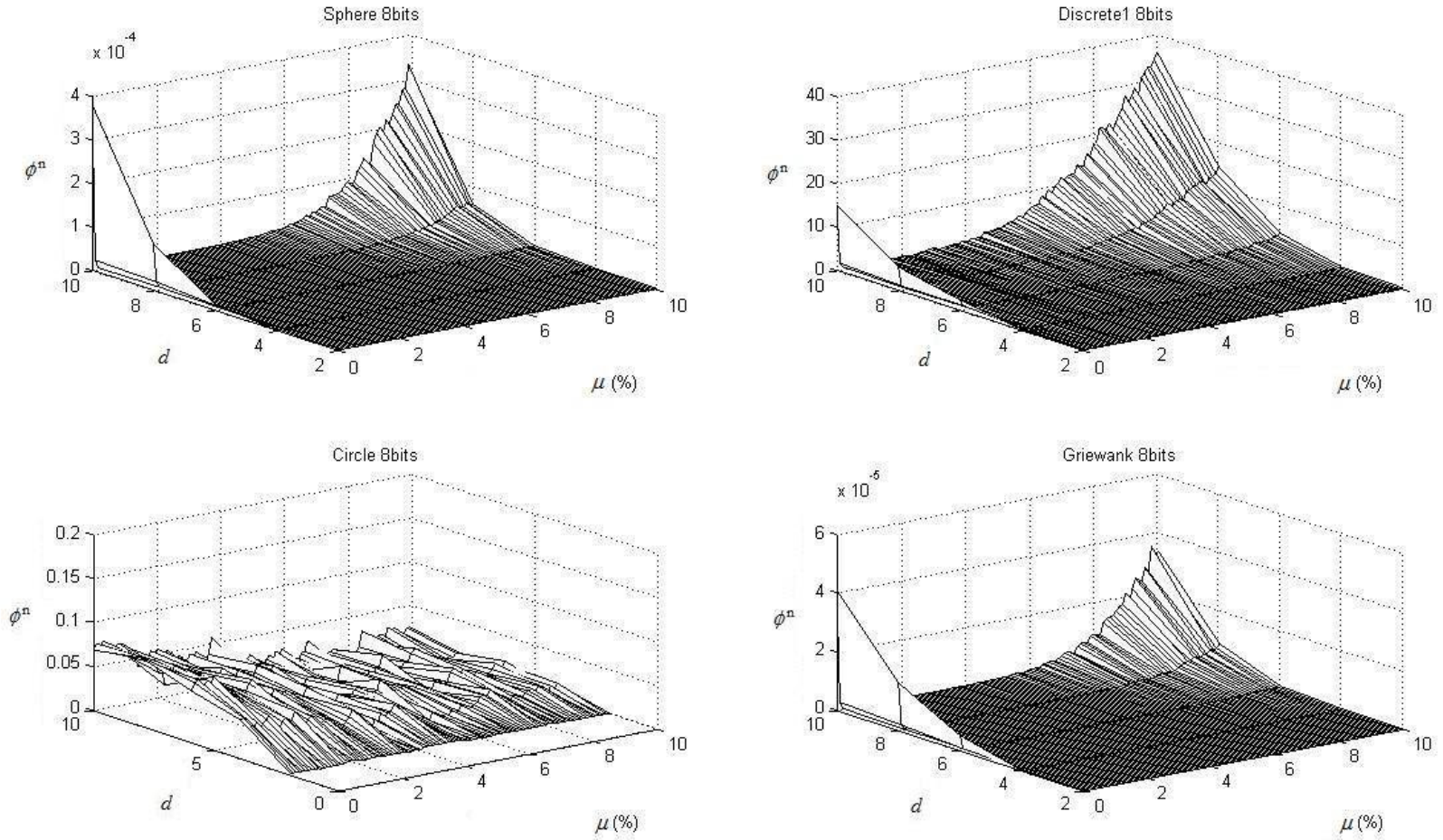


Figura 27: Superfícies de qualidade de solução mostrando os resultados de ϕ^n (*fitness* médio normalizado) para cada d (dimensão) e cada μ (probalidade de mutação) dos experimentos com comprimento de gene de 8 bits.

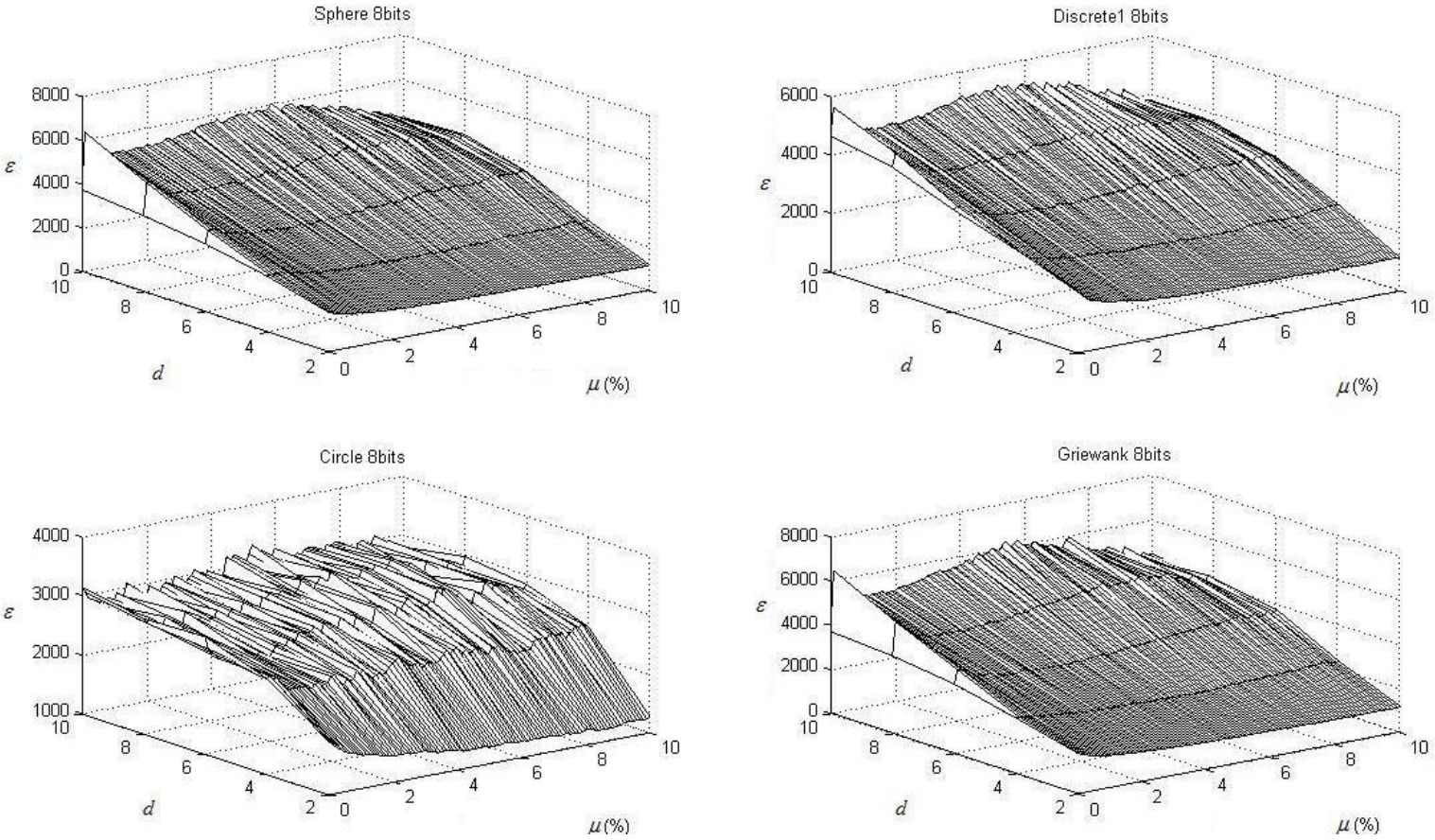


Figura 28: Superfícies de custo computacional mostrando os resultados de ε (nº médio de avaliações de *fitness*) para cada d (dimensão) e cada μ (probabilidade de mutação) dos experimentos com comprimento de gene de 8 bits.

4.1.2 Gráficos para genes de resolução de 20 bits

Os resultados das relações de (ϕ^n,μ,d) e (ε,μ,d) para genes de resolução de 20 bits são mostrados nas figuras 29 e 30, respectivamente, para cada um dos quatros problemas. Se compararmos com os problemas de genes de 8 bits, os problemas com 20 bits são mais complexos para os AGs e, conseqüentemente, para os AGCs. Isto se deve, principalmente, pelo aumento no espaço de busca.

A figura 29 mostra a relação da probabilidade de mutação na qualidade de solução do emCGA. Como na figura 27, para comprimento de genes de 8 bits, existe a formação de um vale, isto é: conforme é aumentado a dimensão do problema também aumenta-se a variação na qualidade de solução. Contudo, para os resultados apresentados na figura 29, a variação é maior. Além disto, esta variação também ocorre com problema Circle.

Na figura 30 é mostrado a relação da probabilidade de mutação no custo computacional do emCGA. Comparando com os resultados apresentados na figura 28, é possível observar que também existe uma formação de uma crista. Contudo, a variação do custo computacional é ainda maior. Além disto, a crista para a maior dimensionalidade se encontra próxima à probabilidade de mutação 3%.

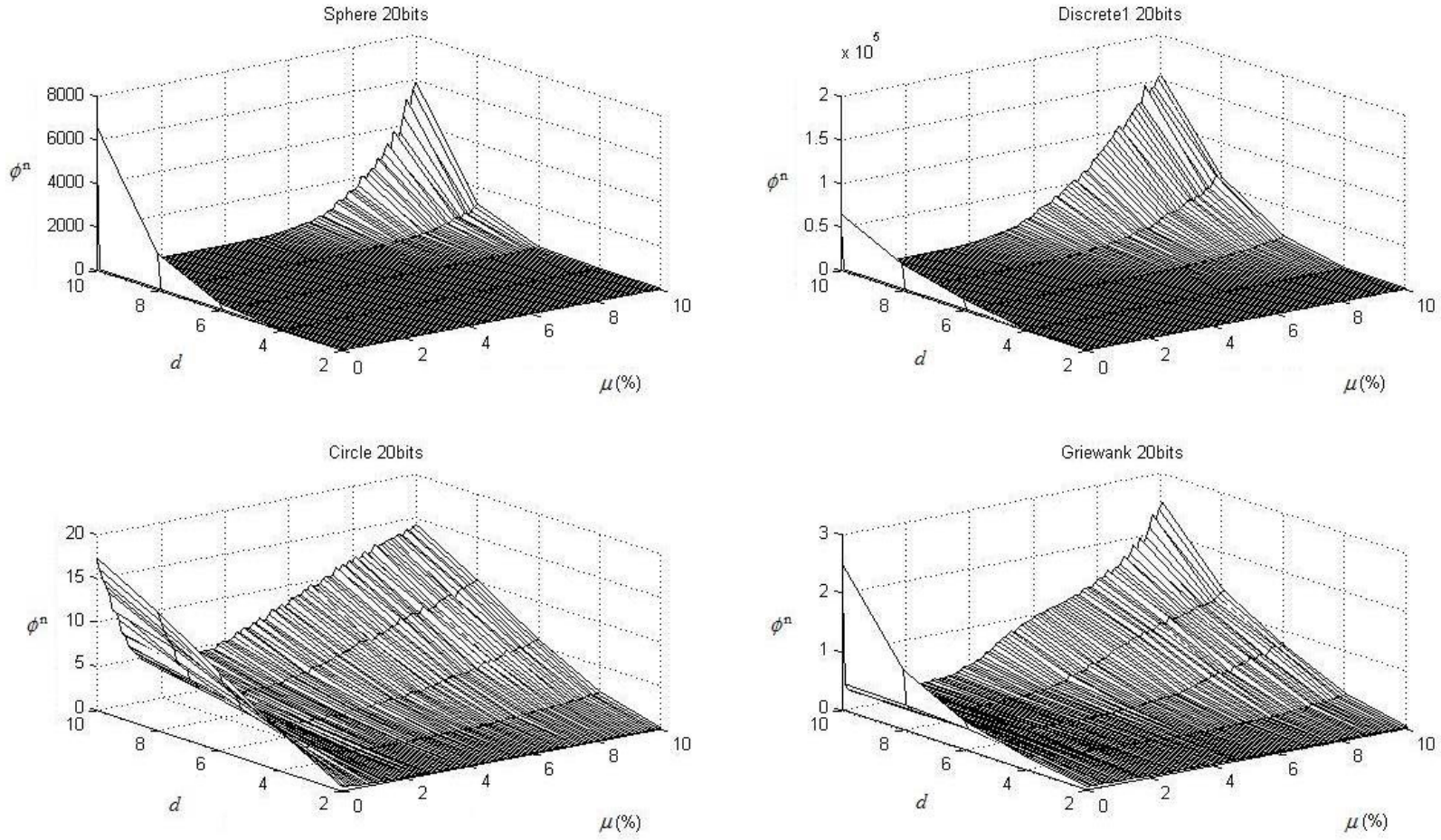


Figura 29: Superfícies de qualidade de solução mostrando os resultados de ϕ^n (*fitness* médio normalizado) para cada *d* (dimensão) e cada μ (probabilidade de mutação) dos experimentos com comprimento de gene de 20 bits.

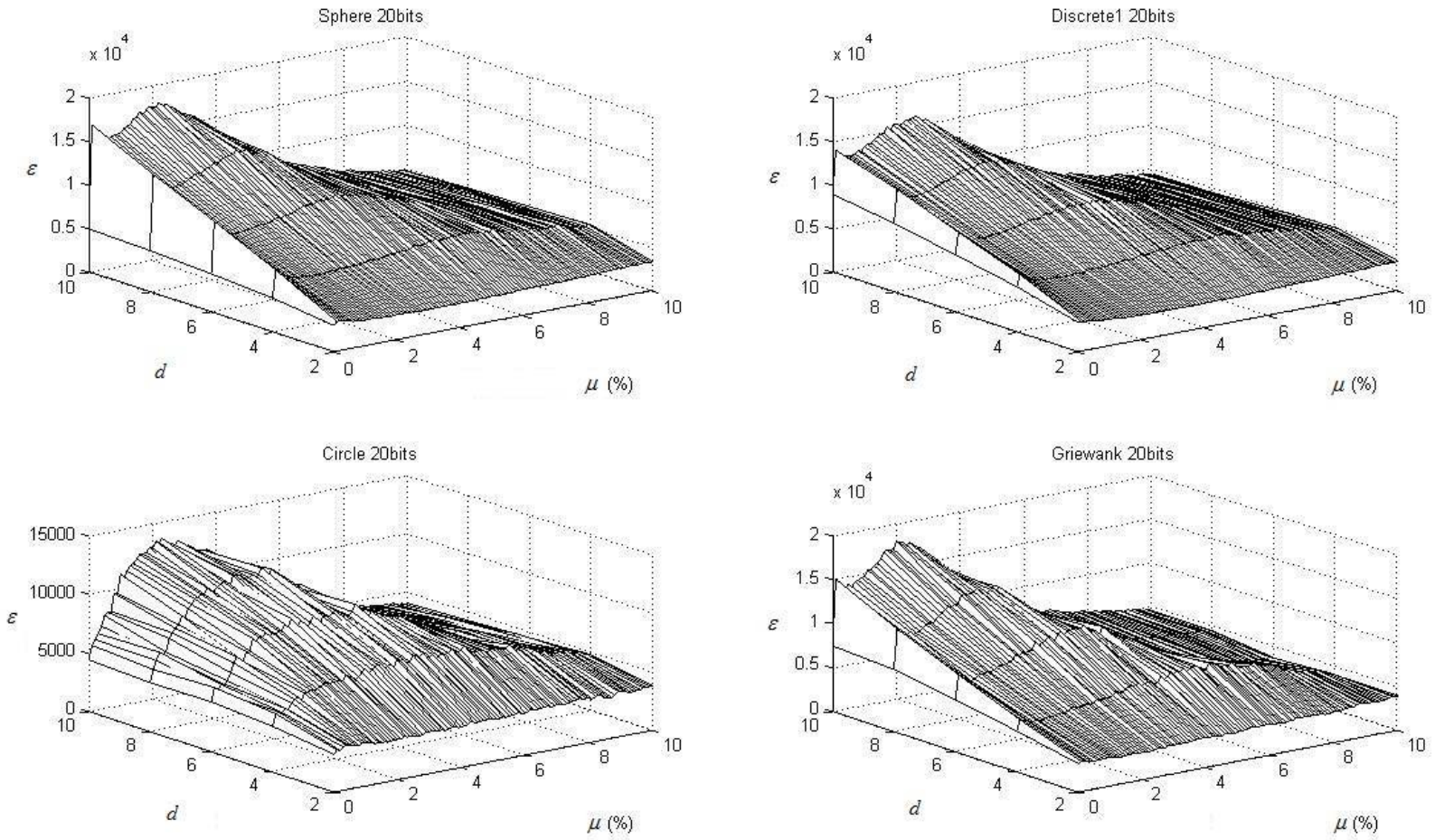


Figura 30: Superfícies de custo computacional mostrando os resultados de ε (n° médio de avaliações de *fitness*) para cada *d* (dimensão) e cada μ (probabilidade de mutação) dos experimentos com comprimento de gene de 20 bits.

4.1.2 Resultados de melhor desempenho

Os resultados de melhor desempenho são apresentados nas figuras 31, 32 e 33, as quais mostram os valores de ϕ^*, ε^* e μ^* respectivamente. Ressalta-se que cada um destes resultados é obtido através de 10100 execuções, ou seja, o melhor resultado médio de 100 execuções para um conjunto de experimentos com variação da probabilidade de mutação de 0 até 10% com precisão de 0,1% (100 x 101). Na figura 31 o valor de ϕ^* é relacionado com os dois eixos de complexidade para problemas utilizados- no caso, comprimento dos genes e dimensionalidade do problema. Na figura 32 os valores de ε^* são relacionados com l- o comprimento do cromossomo. Como a representação do cromossomo do indivíduo é binária, o comprimento do cromossomo é o número de dimensões do problema vezes o comprimento dos genes, segundo a tabela 7.

Costuma-se recomendar uma probabilidade de mutação de $1/l$ para os AGs (CERVANTES e STEPHENS, 2006) e espera-se que este resultado seja semelhante para o emCGA. Desta forma, a figura 33 mostra a relação entre o μ^* e o inverso do comprimento do cromossomo, também mostrado na tabela 7. No problema *Griewank* existe a necessidade de diferenciar os pontos dos resultados de comprimento de gene maior e menor e igual a 12 bits. Esta diferenciação é ilustrada na legenda da curva deste problema.

Na figura 34 é mostrada a distância entre o ε^* e o maior valor de número de avaliações de *fitness* do experimento (ε_{max}) com relação ao comprimento do cromossomo. Esta distância representa o quanto longe está o ε^* do pico da crista de ε, conforme apresentado na figura 18. Nos problemas *Sphere* e *Discrete1* são observados um aumento linear nesta distância com o aumento do comprimento do cromossomo. Este aumento linear é ilustrado na figura com uma curva de tendência. No problema *Circle* não existe esta tendência e, portanto, nenhuma curva é ilustrada. No entanto, no problema *Griewank* esta tendência ocorre para alguns pontos, que podem ser relacionados com a figura 33. Neste caso, estes pontos são diferenciados e uma legenda é destacada.

As alturas das superfícies da qualidade de solução ($\Delta\phi$) e do custo computacional ($\Delta\varepsilon$) alcançados nos experimentos são mostradas nas figuras 35 e 36. Quanto maiores são estas alturas, maior é a influência do operador de mutação no desempenho do emCGA. Novamente, os valores de ϕ estão relacionados com a complexidade do problema e ε com o comprimento do cromossomo. Sendo assim, a figura 35 mostra os valores $\Delta\phi$ para o comprimento dos

genes e dimensionalidade do problema e, na figura 36, $\Delta\varepsilon$ é relacionado com o comprimento do cromossomo. Contudo, a altura da superfície do custo computacional ($\Delta\varepsilon$) é normalizada dividindo-se $\Delta\varepsilon$ pelo ε_{max}.

Tabela 7. Valores do comprimento de cromossomo e probabilidade de mutação recomendada.

	Comprimento do Cromossomo			
d	**8-bits**	**12-bits**	**16-bits**	**20-bits**
2	16	24	32	40
4	32	48	64	80
6	48	72	96	120
8	64	96	128	160
10	80	120	160	200
	Probabilidade de mutação recomendada (1/*l*)			
d	**8-bits**	**12-bits**	**16-bits**	**20-bits**
2	6,3	4,2	3,1	2,5
4	3,1	2,1	1,6	1,3
6	2,1	1,4	1,0	0,8
8	1,6	1,0	0,8	0,6
10	1,3	0,8	0,6	0,5

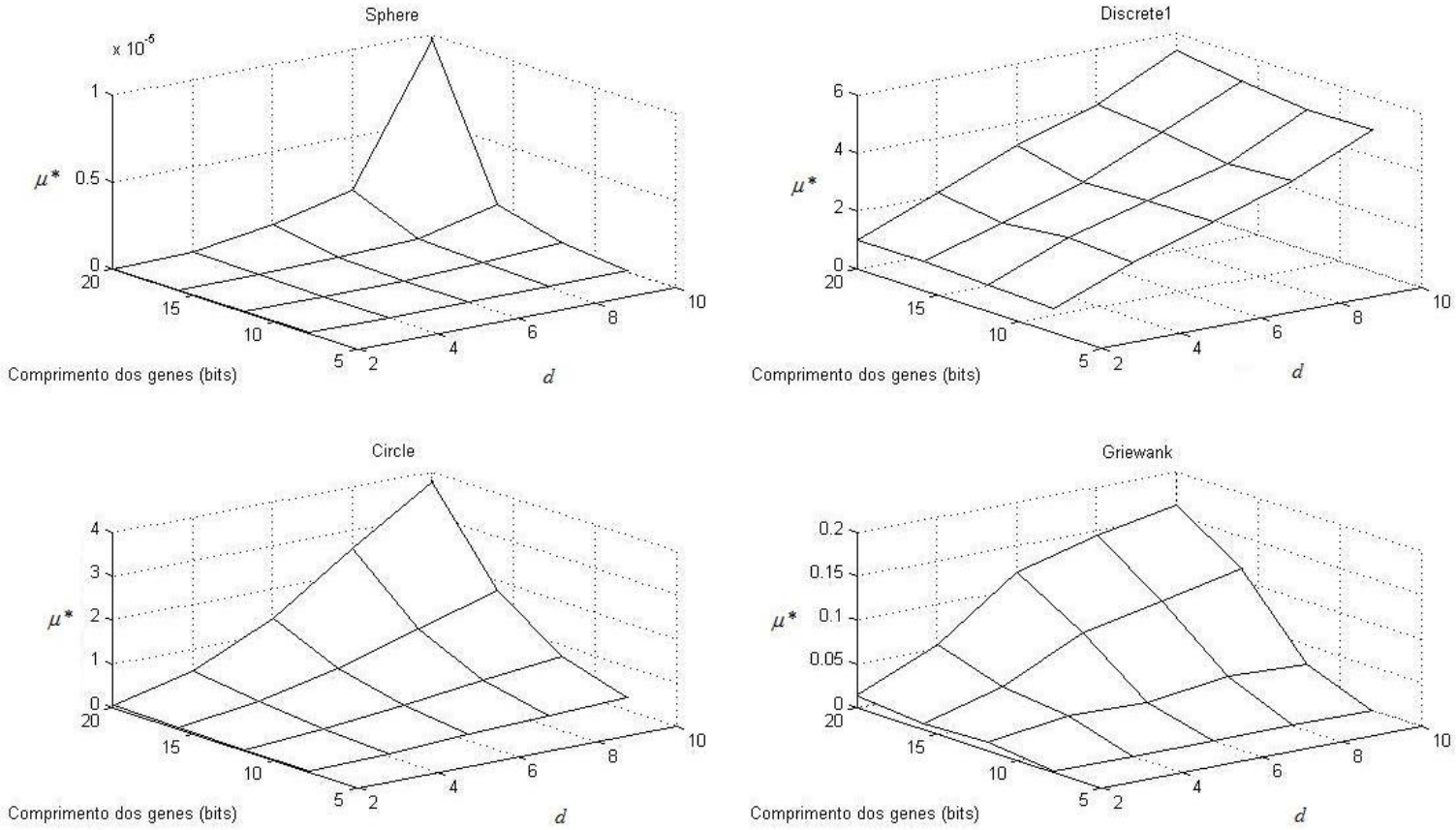


Figura 31: Relação do ϕ^* (melhor *fitness* médio) com a complexidade (comprimento do gene e *d* (dimensão) do problema.

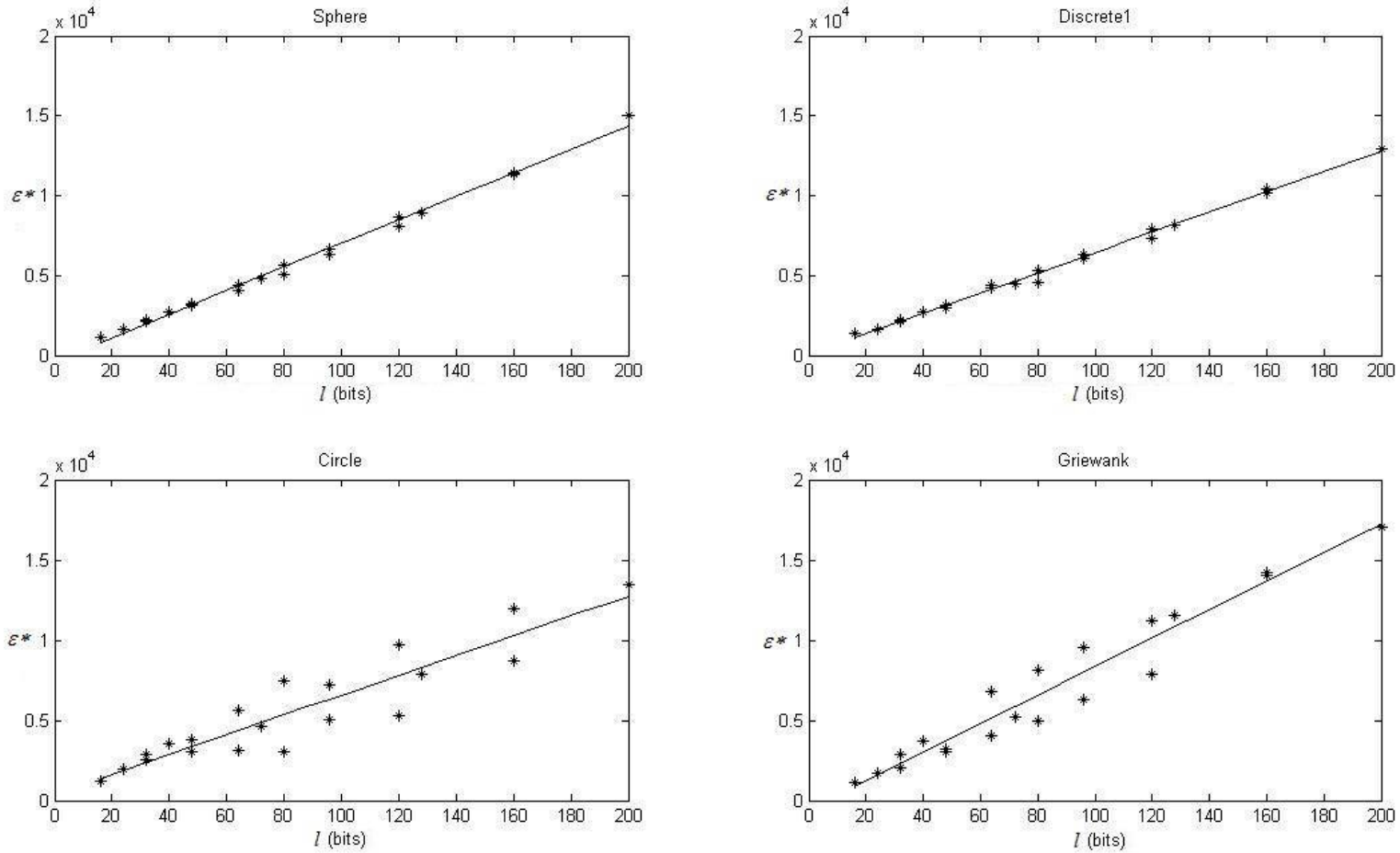


Figura 32: Relação do ε^* (número médio de avaliações de *fitness*) e o l (comprimento do cromossomo).

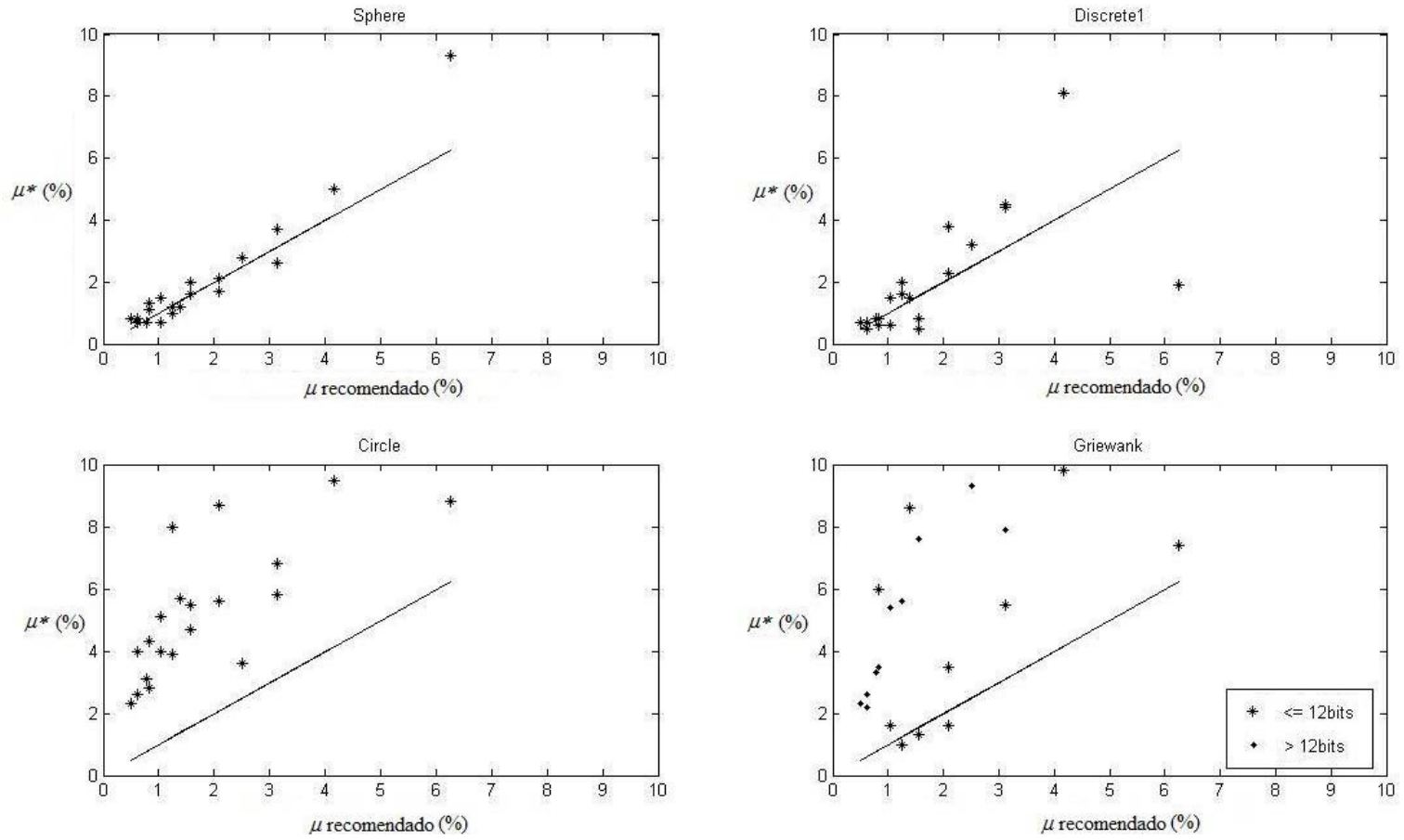


Figura 33: Relação do μ^* (melhor probabilidade de mutação) e o μ recomendado (probabilidade de mutação recomendada, isto é: $1/l$). As retas em todas as figuras representam os pontos onde o μ^* é igual ao μ recomendado.

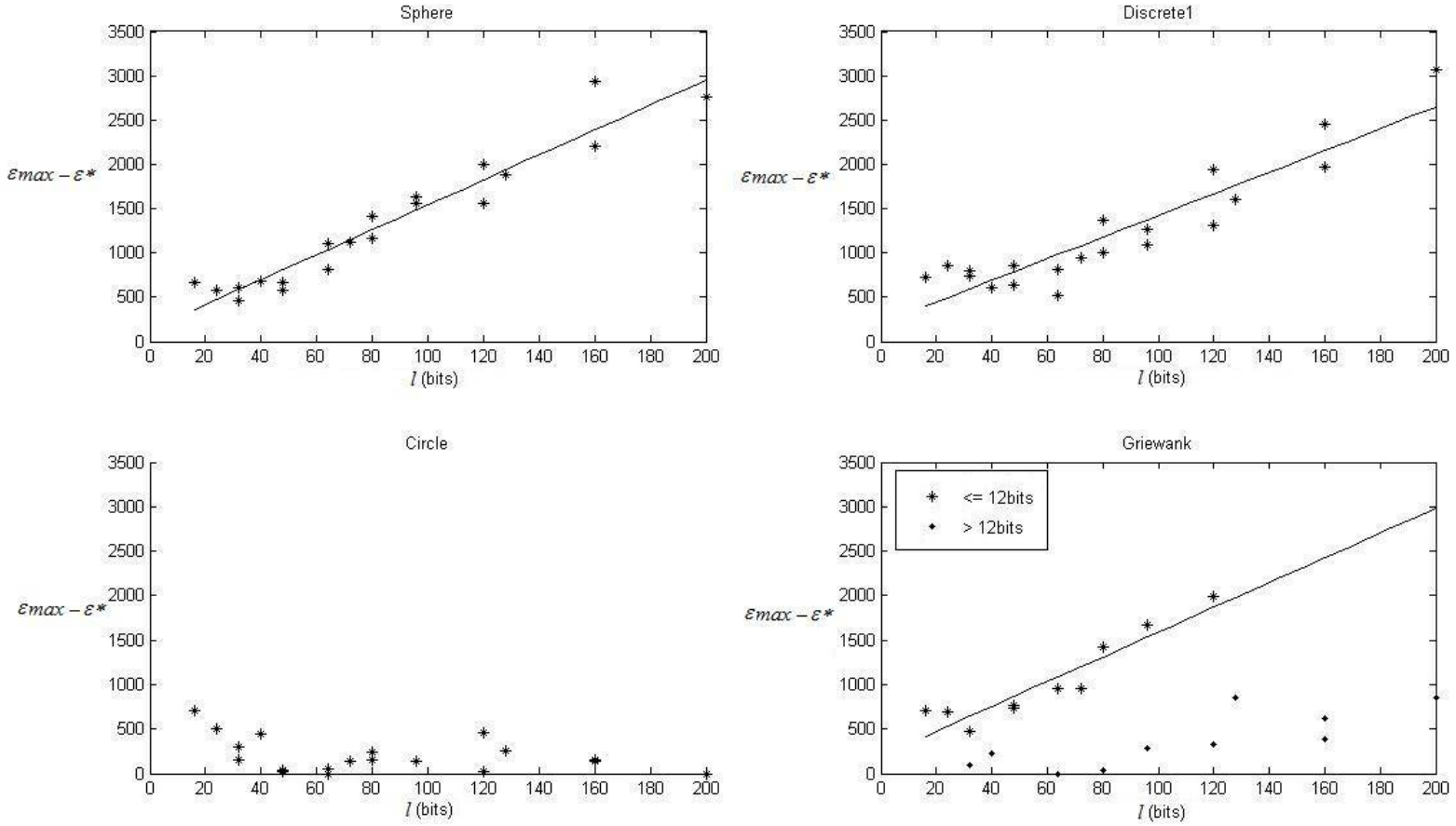


Figura 34: Distância entre ε^* e ε_{max} (número m[edio de avaliações de *fitness*) pelo l (comprimento do cromossomo). Todas as retas apresentadas representam a tendência linear dos resultados.

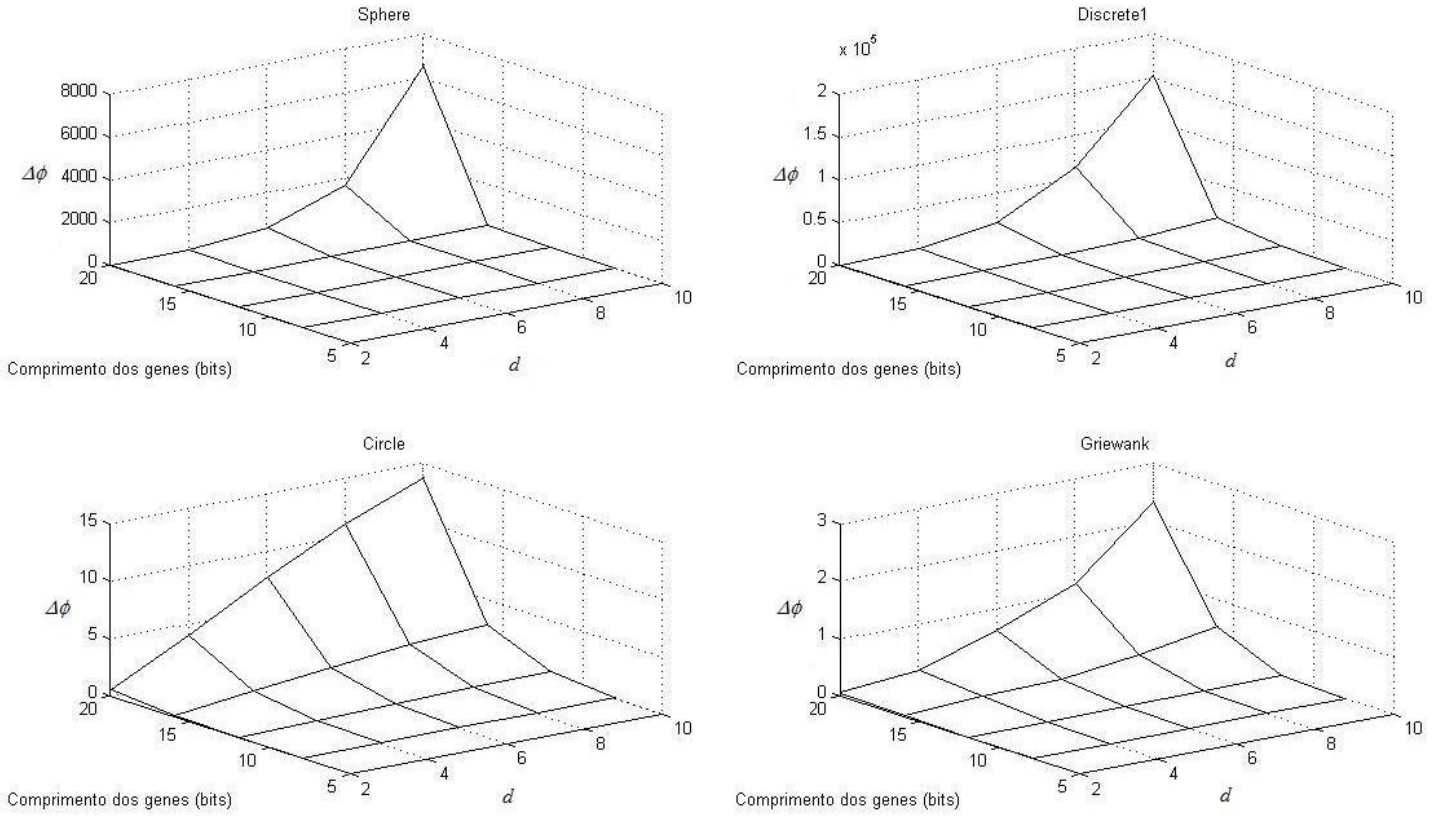


Figura 35: Relação da $\Delta\phi$ (variação do *fitness* médio) com complexidade do problema (dimensão d e comprimento dos genes).

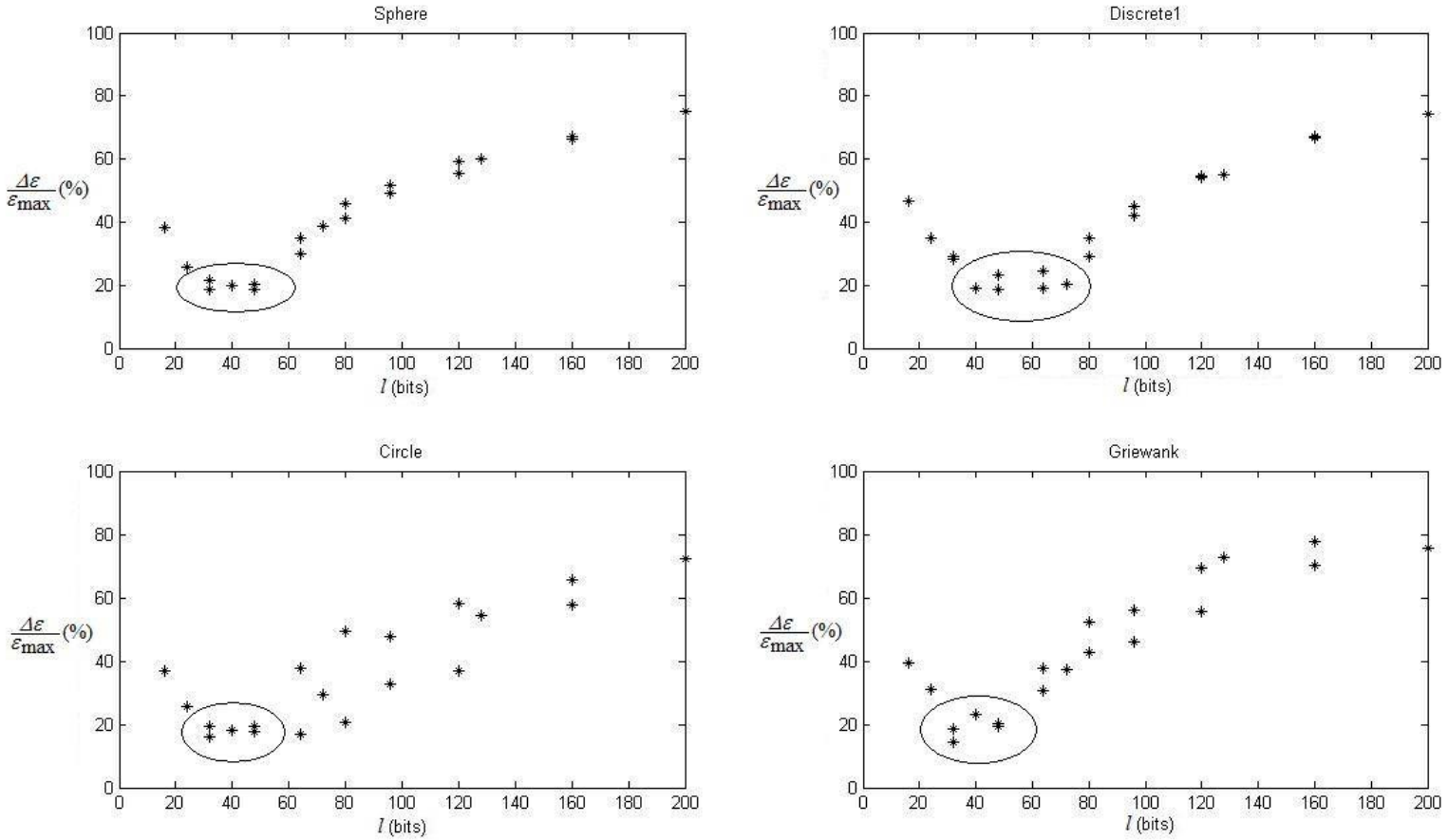


Figura 36: Variação do custo computacional ($100*\Delta\varepsilon/\varepsilon_{max}$) por l (comprimento do cromossomo). É destacada a região com o ponto de inflexão, isto é: ponte onde a tendência dos resultados muda.

4.2 RESULTADOS DA COMPARAÇÃO DO DESEMPENHO DO emCGA COM O neCGA E mCGA

Nesta seção são mostrados os resultados da comparação do desempenho entre os algoritmos emCGA (SILVA, LOPES e LIMA, 2007), neCGA (AHN e RAMAKRISHNA, 2003) e mCGA (GALLAGHER, VIGRAHAM e KRAMER, 2004). Foram calculados os valores de *fitness* médio e o número médio de avaliações de *fitness* de 100 execuções para cada experimento. Os experimentos foram feitos usando-se os diversos tamanhos de população propostos, os diversos problemas descritos na seção 3.4.2 e os três algoritmos AGC. No total são 138 experimentos (3 algoritmos x (4 problemas de 15 dimensões x 7 tamanhos de população + 2 problemas de 2 dimensões x 9 tamanhos de população)), ou seja, 13800 execuções (138 experimentos x 100 execuções). Os resultados são mostrados na figura 37, onde são destacados os melhores com um círculo e o tamanho da população empregada. No problema *Discrete2* e *Sphere* os resultados para o neCGA não aparece por serem muito inferiores aos outros algoritmos.

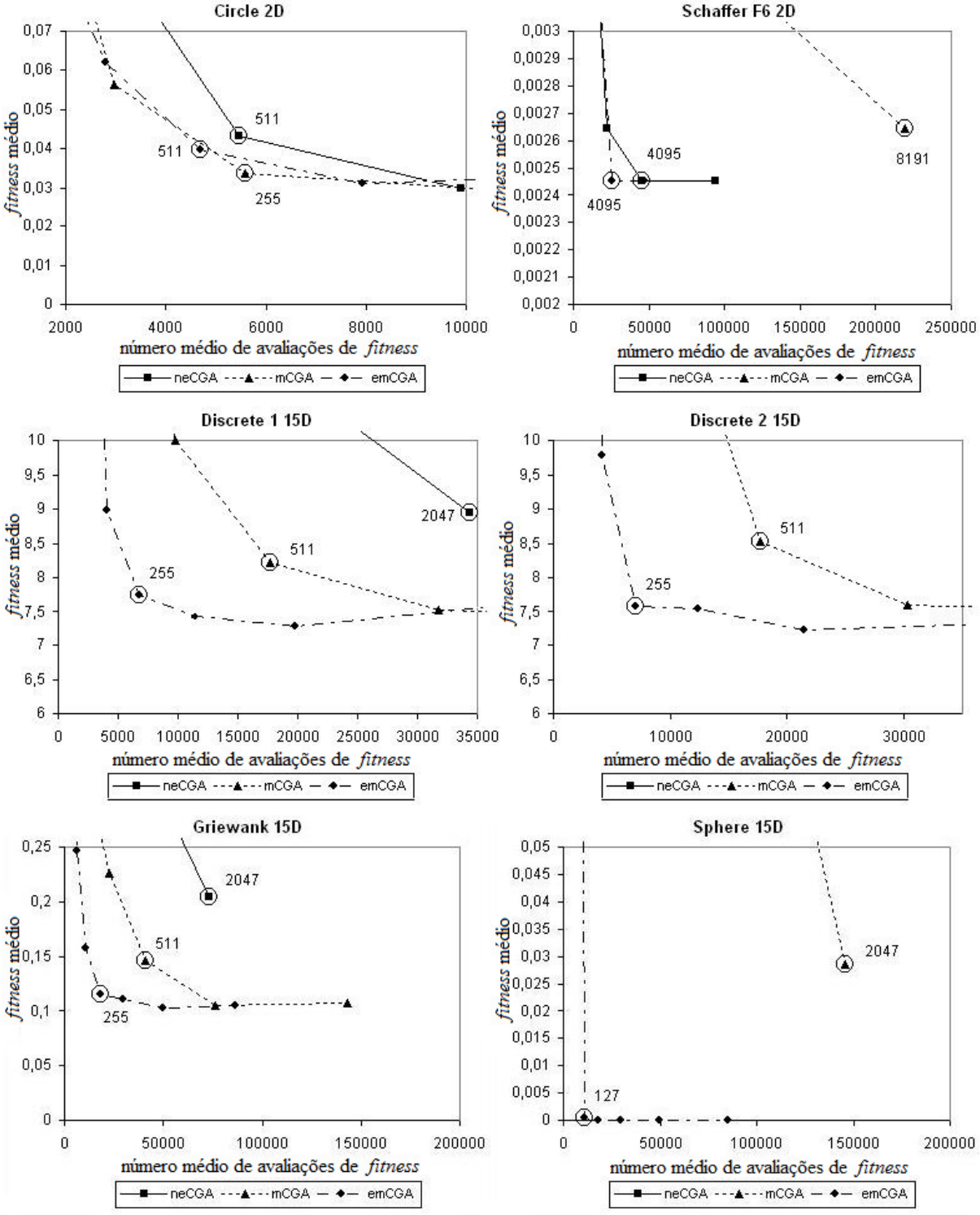


Figura 37: Comparação através da curva de Paretto do desempenho dos algoritmos emCGA, neCGA e mCGA.

4.3 RESULTADOS DO emCGA APLICADO À DETECÇÃO DE OBJETO PARA VISÃO COMPUTACIONAL

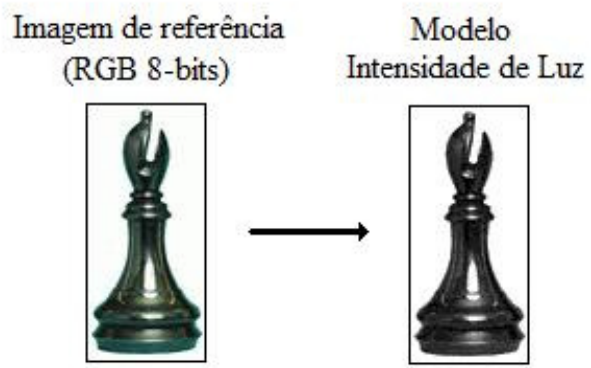
Os resultados da detecção de objetos para visão computacional usando o emCGA são apresentados em três seções separadas. Nas seções 4.3.1 e 4.3.2 são apresentados os resultados de diversos experimentos para o modelo de intensidade de luz e o modelo binário, respectivamente. Nestes experimentos busca-se testar o algoritmo próximo aos seus extremos de desempenho, adicionando-se ruídos e distorções e utilizando-se diferentes objetos em ambientes de complexidade variável. Na seção 4.3.3 são apresentados resultados de experimentos que mostram as principais diferenças do desempenho do emCGA usando os dois modelos propostos.

Os resultados são apresentados em figuras onde são mostradas as imagens de referência, a imagem de entrada e o objeto encontrado destacado nesta imagem de entrada. Optou-se por mostrar a imagem de entrada em tons de cinza para facilitar a visualização. Os resultados do emCGA são apresentados e comparados com os resultados do ABE, isto é, os valores dos parâmetros do vetor *k*, do *fitness* (*Fit.*) e número de avaliações de *fitness* (Aval.).

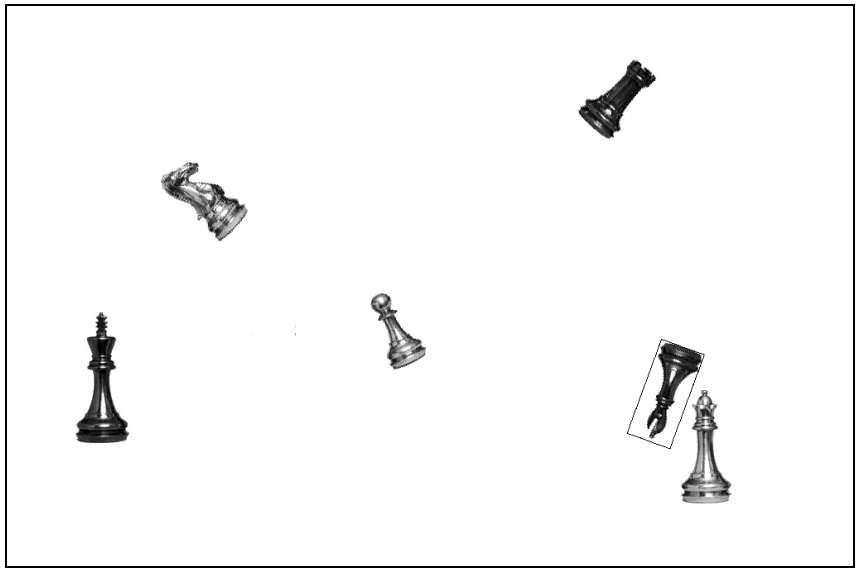
4.3.1 Resultados para o modelo de intensidade de luz

No primeiro experimento, o objeto procurado na imagem de entrada é um bispo do jogo de xadrez. A imagem de referência é usada para gerar um modelo de intensidade de luz, segundo a figura 38a. Este modelo, por sua vez, é usado pelo emCGA para encontrar o objeto na imagem de entrada. O objeto encontrado é apresentado na figura 38b, usando-se um retângulo para destacá-lo. Este artifício será usado no restante do trabalho para destacar o resultado da busca por objetos. É importante observar que o bispo na imagem de entrada se encontra transladado e rotacionado em relação à imagem de referência. Em todos os resultados apresentados a imagem de referência e a imagem de entrada não estão na mesma escala.

a)



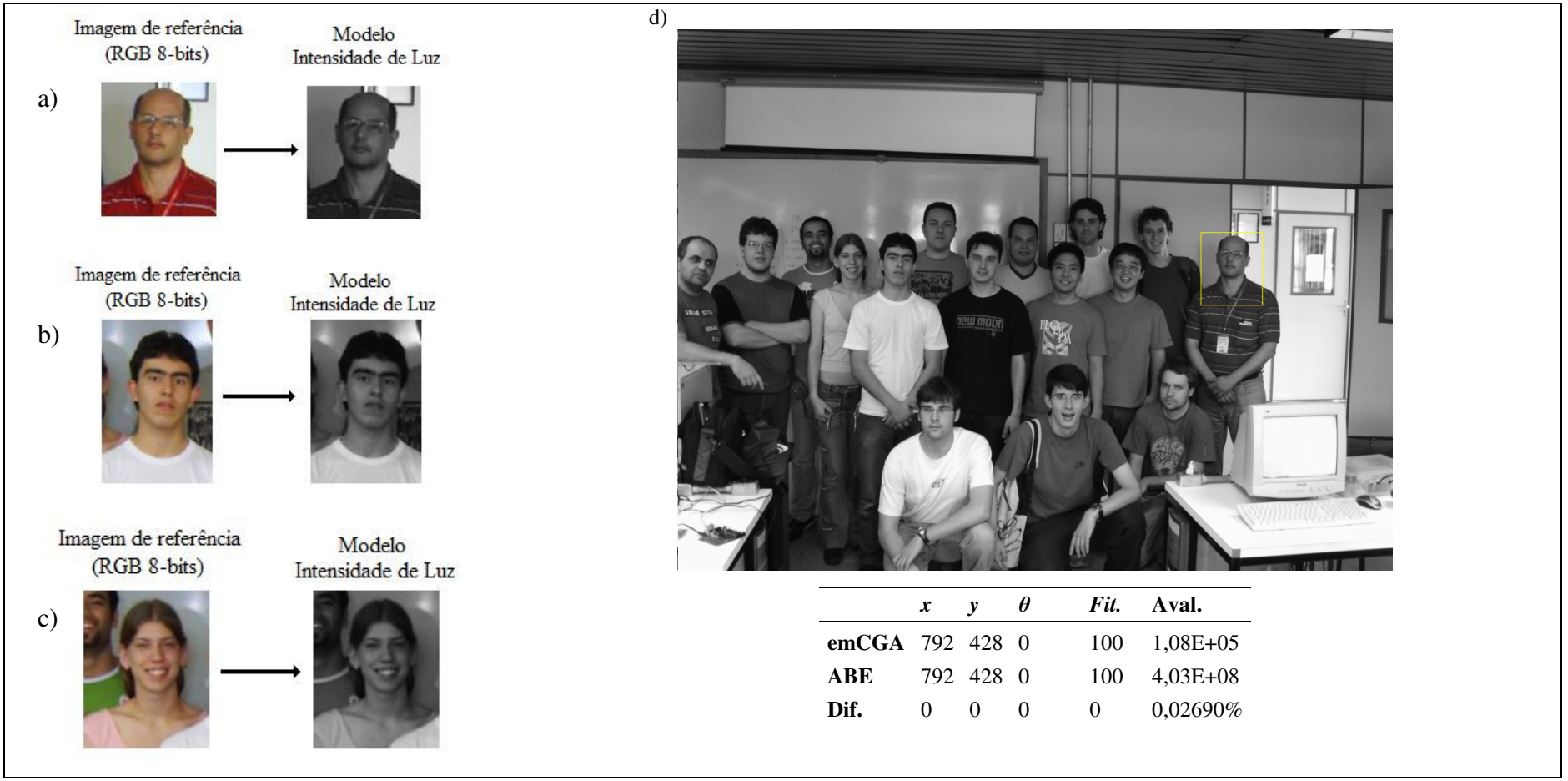
b)



	x	*y*	*Θ*	*Fit.*	**Aval.**
emCGA	795	209	200,08	97,29	8,81E+04
ABE	795	209	200,08	97,29	4,03E+08
Dif.	0	0	0	0	0,02188%

Figura 38: Busca pelo bispo do jogo xadrez. a) Imagem de referência e o seu modelo de intensidade de luz. b) Imagem de entrada e resultado da busca. A imagem de referência foi ampliada para uma melhor visualização.

No segundo experimento é realizada uma busca por faces em uma foto digital. As três faces utilizadas são representadas pelas imagens de referências mostradas nas figuras 39a, b e c. A imagem de referência é recortada da própria imagem de entrada, e esta imagem não sofre nenhum tipo de distorção. Contudo, a busca por faces em fotos é uma busca difícil, com muitos máximos locais. Os resultados das buscas para cada um das faces são mostrados nas figuras 39d, 40a e 40b.



	x	*y*	*θ*	*Fit.*	Aval.
emCGA	792	428	0	100	1,08E+05
ABE	792	428	0	100	4,03E+08
Dif.	0	0	0	0	0,02690%

Figura 39: Busca por faces em uma foto. a, b e c) Imagens de referência e seus modelos que representam as faces procuradas. d) Imagem de entrada e resultado para a busca pela face a. A imagem de referência foi ampliada para uma melhor visualização.

a)

	x	*y*	*θ*	*Fit.*	**Aval.**
emCGA	242	429	360	100	1,10E+05
ABE	242	429	360	100	4,03E+08
Dif.	0	0	0	0	0,02737%

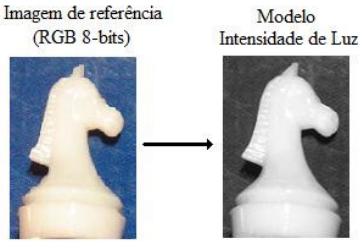
b)

	x	*y*	*θ*	*Fit.*	**Aval.**
emCGA	310	425	0	100	1,01E+05
ABE	310	425	0	100	4,03E+08
Dif.	0	0	0	0	0,02498%

Figura 40: Busca de faces em uma foto. a) Imagem de entrada e resultado para a busca pela face da figura 39b. b) Imagem de entrada e resultado para a busca pela face da figura 39c.

No último experimento é realizada a busca por uma peça de cavalo de um jogo de xadrez. A imagem de referência e seu modelo de intensidade de luz são mostrados na figura 41a. Na imagem de entrada 1, mostrada na figura 41b, o objeto encontra-se transladado e rotacionado no plano da imagem. As principais distorções nesta imagem ocorrem devido à alteração do brilho no objeto. Contudo, na imagem de entrada 2, mostrada na figura 41c, o objeto é também rotacionado nos ângulos α e β, segundo a figura 26a. Estas rotações são consideradas distorções por não serem otimizados pelo emCGA. Além destas distorções, nas duas imagens existem outras peças semelhantes que aumentam a dificuldade da busca.

a)



b)

	x	*y*	*θ*	*Fit.*	**Aval.**
emCGA	327	552	32,41	92,86	9,58E+04
ABE	327	552	32,41	92,86	4,03E+08
Dif.	0	0	0	0	0,02380%

c)

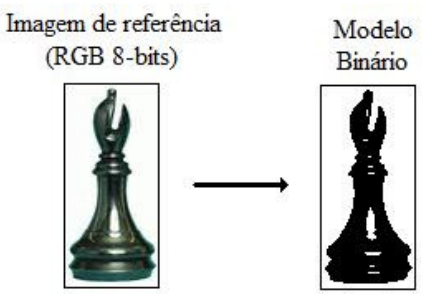
	x	*y*	*θ*	*Fit.*	**Aval.**
emCGA	292	455	345,91	85,04	8,07E+04
ABE	292	455	345,91	85,04	4,03E+08
Dif.	0	0	0	0	0,02005%

Figura 41: Busca pelo cavalo do jogo de xadrez. a) Imagem de referência e seu modelo. b) Imagem de entrada 1 e resultado da busca. c) Imagem de entrada 2 e resultado da busca. A imagem de referência foi ampliada para uma melhor visualização.

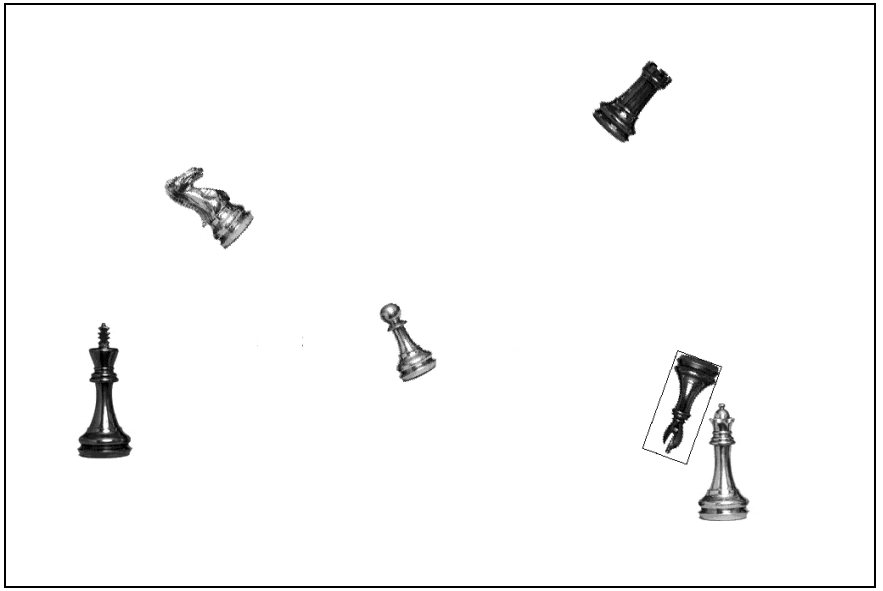
4.3.2 Resultados para o modelo binário

No primeiro experimento apresentado para o modelo binário é realizada uma busca por um bispo do jogo de xadrez. Este é o mesmo bispo do experimento mostrado na figura 38 para o modelo de intensidade de luz. Contudo, neste experimento, o objeto será representado pelo modelo binário. A imagem de referência e seu modelo binário são mostrados na figura 42a. Na imagem de entrada 1, nenhuma distorção é adicionada; no entanto, a imagem é transladada e rotacionada junto com outros objetos semelhantes, o que aumenta a dificuldade da busca. O resultado da busca nesta imagem de entrada é mostrado na figura 42b. A imagem de entrada 2 é a imagem de entrada 1 com adição de 50% de ruído gaussiano e o resultado desta busca é apresentado na figura 43a. Na figura 43b, o resultado da busca na imagem de entrada 3 é apresentado com uma redução do brilho em relação à imagem de referência. Além disto, o objeto está parcialmente oculto por outra peça.

a)



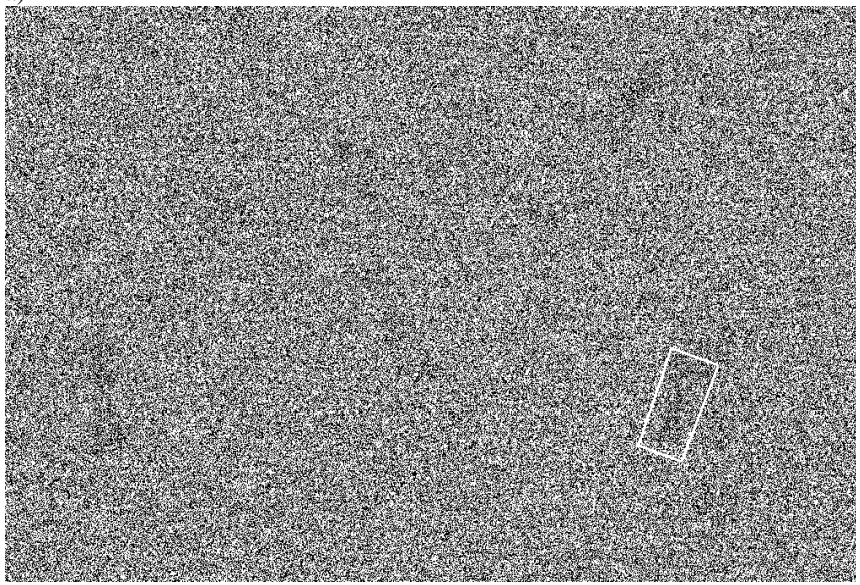
b)



	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	795	209	200,08	146	97,83	8,12E+04
ABE	795	209	200,08	140	97,83	1,03E+11
Dif.	0	0	0	6	0	0,00008%

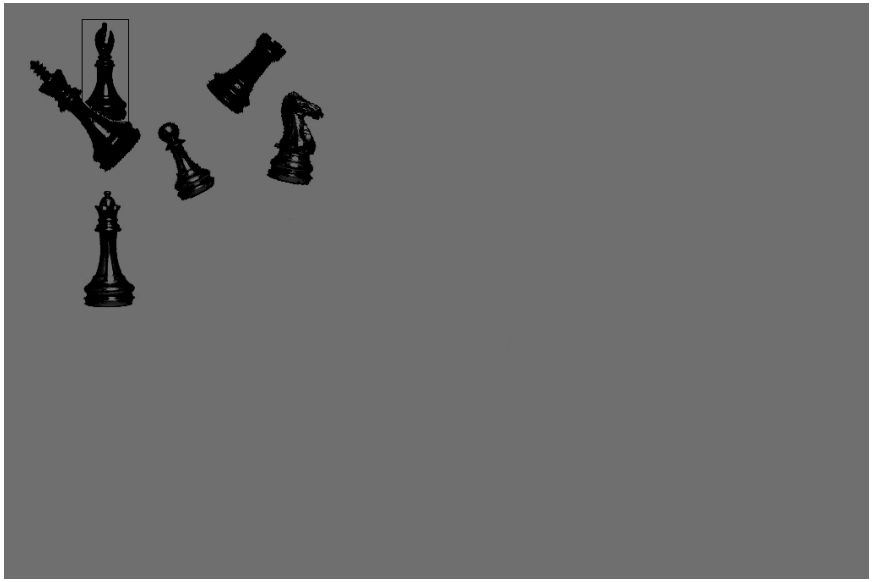
Figura 42: Busca pelo bispo do jogo de xadrez. a) Imagem de referência e seu modelo binário. b) Imagem de entrada 1 e resultado da busca. A imagem de referência foi ampliada para uma melhor visualização.

a)



	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	795	209	200,08	28	55,61	1,03E+05
ABE	795	209	200,08	27	55,53	1,03E+11
Dif.	0	0	0	1	0,080	0,00010%

b)

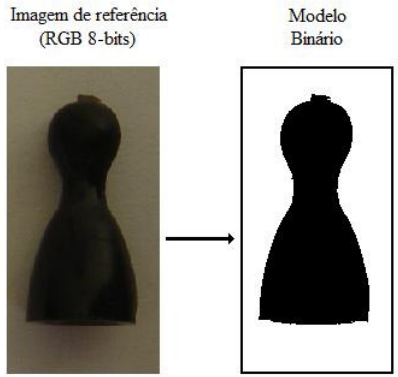


	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	119	600	360	2	97,37	8,98E+04
ABE	119	600	360	2	97,37	1,03E+11
Dif.	0	0	0	0	0	0,00009%

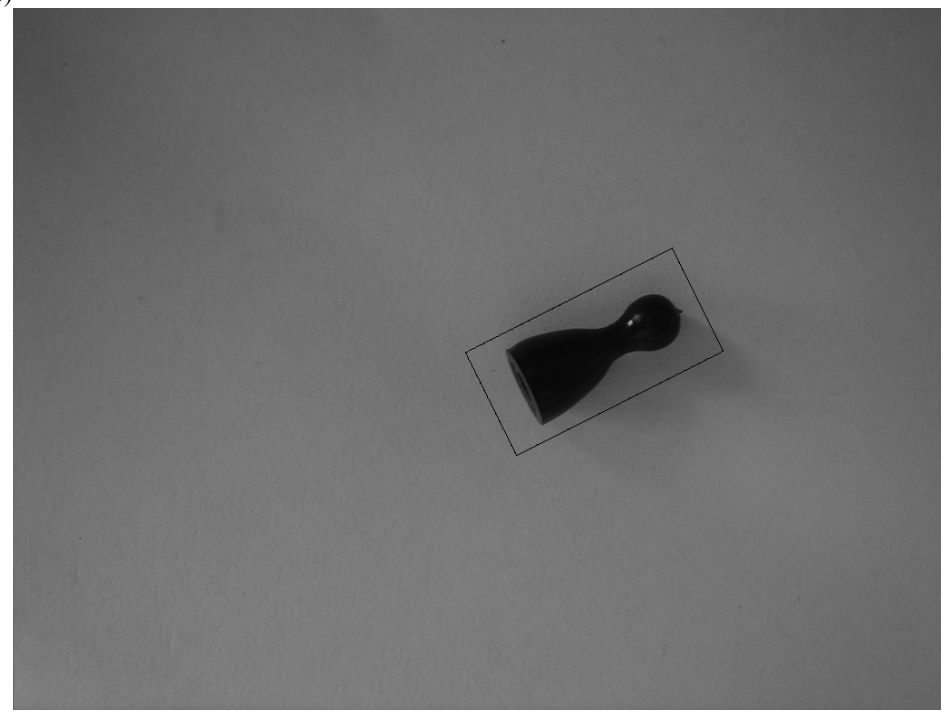
Figura 43: Busca pelo bispo do jogo de xadrez. a) Imagem de entrada 2 e resultado da busca. b) Imagem de entrada 3 e resultado da busca.

No último experimento apresentado nesta seção é realizada uma busca por uma peça do jogo ludo. A imagem de referência e seu modelo binário são mostrados na figura 44a. Na imagem de entrada 1, a peça é transladada e rotacionada e o resultado da busca é mostrado na figura 44b. A figura 45a apresenta o resultado da busca pela imagem de entrada 2, onde é realizada a busca por um objeto com cor diferente do objeto da imagem de referência. Na imagem de entrada 3, mostrada na figura 45b, existe um forte escurecimento do ambiente de onde foi tirada a foto, onde os valores da matriz M_{ilum} variam de 0 a 20 aproximadamente. O escurecimento da imagem é tão intenso que a visualização a olho nu é difícil. Desta forma, o objeto é de novo destacado, com o brilho e contraste alterados, para facilitar a visualização do resultado.

a)



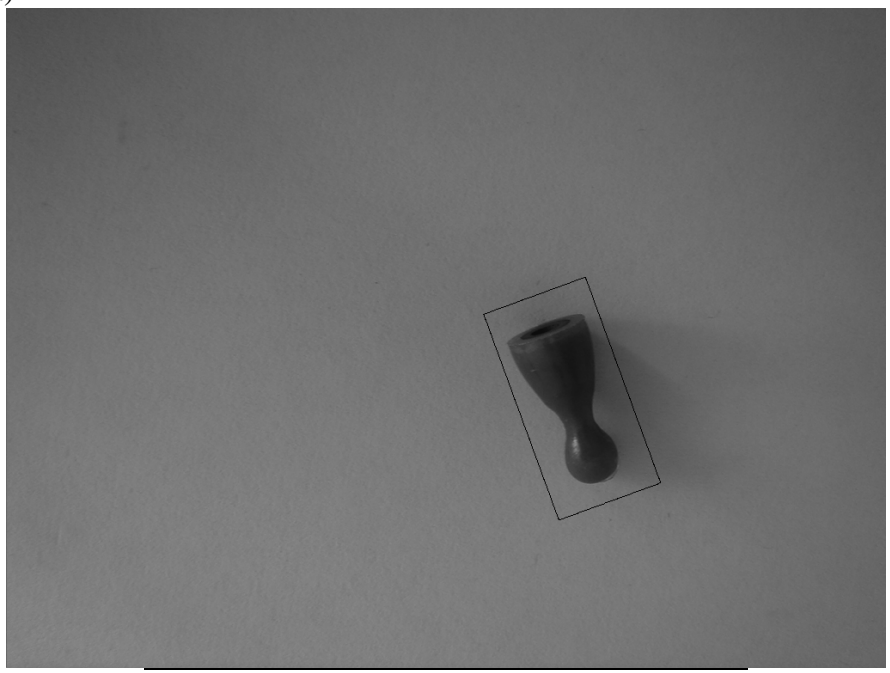
b)



	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	639	391	63,41	56	97,71	1,10E+05
ABE	641	392	63,41	50	97,89	1,03E+11
Dif.	2	1	0	6	0,028	0,00011%

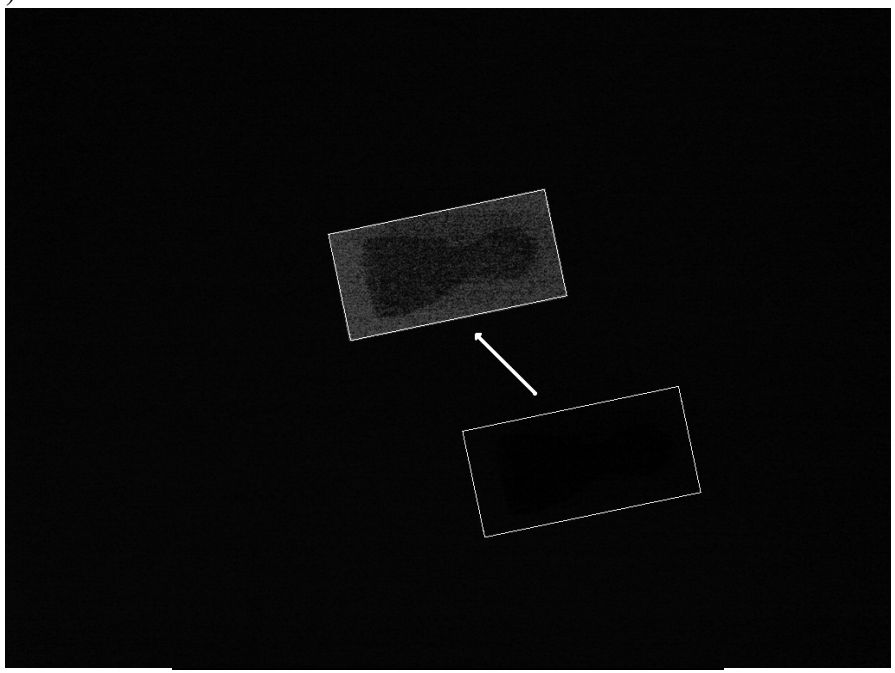
Figura 44: Busca pela peça do jogo ludo. a) Imagem de referência e seu modelo binário. b) Imagem de entrada 1 e resultado da busca. A imagem de referência foi ampliada para uma melhor visualização.

a)



	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	658	313	160,63	59	93,19	1,46E+05
ABE	658	313	159,92	59	93,17	1,03E+11
Dif.	0	0	0,71	0	0,02	0,00014%

b)



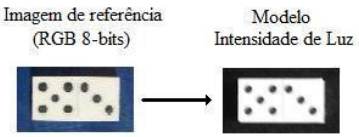
	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	666	240	78	3	79	1,14E+05
ABE	665	239	78	2	79	1,03E+11
Dif.	1	1	0,	1	0	0,00011%

Figura 45: Resultados da busca pela peça de ludo. a) Imagem de entrada 2 e resultado da busca. b) Imagem de entrada 3 e resultado da busca.

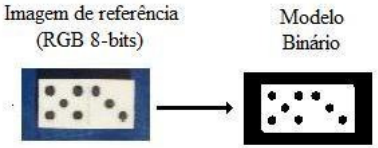
4.3.3 Comparação entre os modelos de intensidade de luz e binário

No experimento realizado nesta seção procura-se mostrar as principais diferenças das buscas realizadas pelos modelos propostos, isto é, modelo de intensidade de luz e binário. Isto é feito através da busca por uma peça do jogo dominó. A figura 46a apresenta a imagem de referência e o seu modelo de intensidade de luz e a figura 46b apresenta a imagem de referência e o seu modelo binário. Na imagem de entrada 1 o objeto se encontra sem distorções em comparação a imagem de referência. Assim, a busca é simples em ambos os modelos. O resultado da busca nesta imagem é mostrado na figura 46c usando-se o modelo de intensidade de luz e na figura 46d usando-se o modelo binário. Nas figuras 47a e 47b são apresentados os resultados da busca na imagem de entrada 2 usando-se os modelos de intensidade de luz e binário, respectivamente. Nesta imagem de entrada existe um forte escurecimento do ambiente onde a foto foi tirada, onde os valores da matriz M_{ilum} variam de 0 a 20 aproximadamente. Mais uma vez, os resultados são destacados sobre a imagem original, com brilho e contraste alterados. As figuras 48a e 48b mostram os resultados para a imagem de entrada 3 usando-se os modelos de intensidade de luz e binário, respectivamente. Nesta imagem de entrada, outras peças do jogo dominó são acrescentadas, algumas muito semelhantes ao objeto procurado.

a)



b)



c)

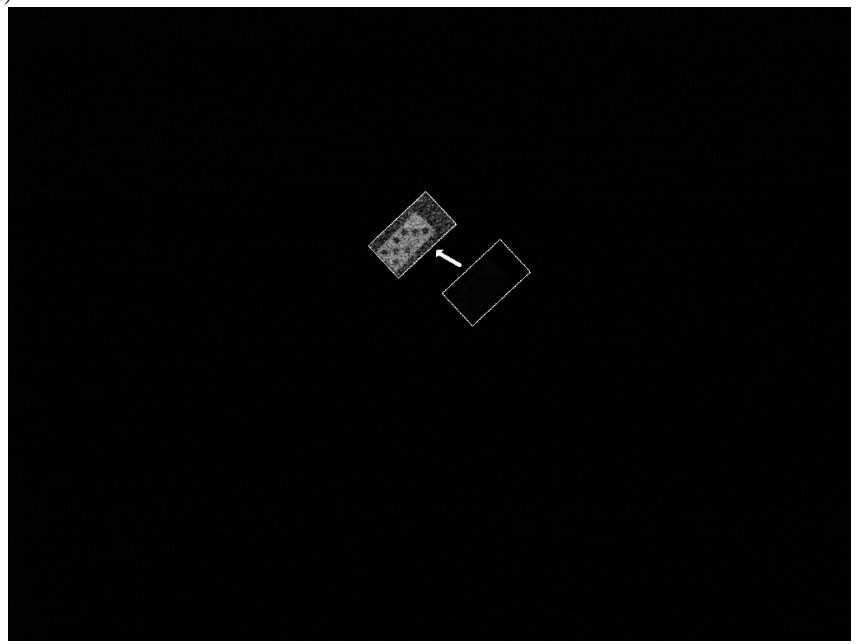
d)

	x	*y*	*θ*	*Fit.*	Aval.
emCGA	533	412	0	100	8,03E+04
ABE	533	412	0	100	4,03E+08
Dif.	0	0	0	0	0,01995%

	x	*y*	*θ*	*t*	*Fit.*	Aval.
emCGA	533	412	360	128	100	1,04E+05
ABE	533	412	360	128	100	1,03E+11
Dif.	0	0	0	0	0,000	0,00010%

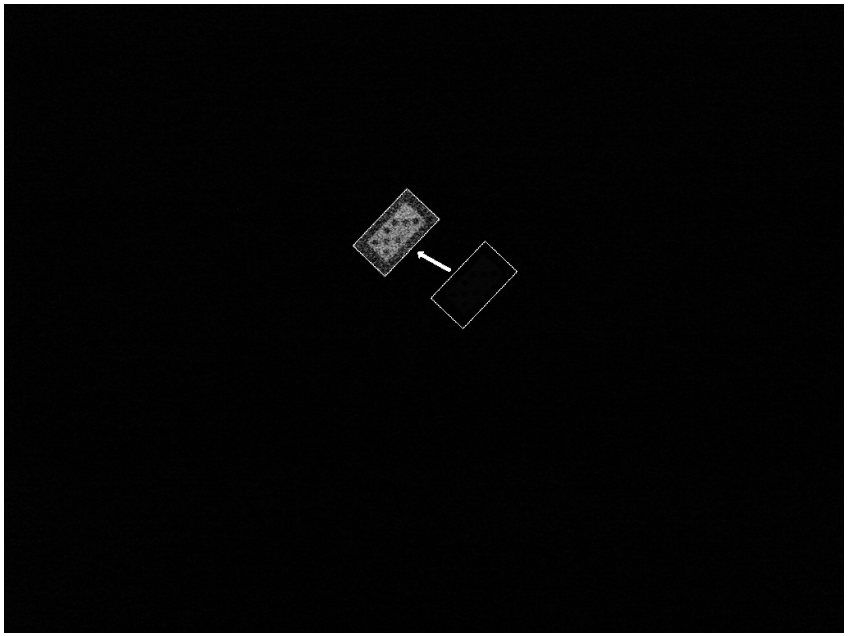
Figura 46: Busca por uma peça do dominó. a) Imagem de referência e seu modelo de intensidade de luz. b) Imagem de referência e seu modelo de intensidade de luz. c) Resultado da busca usando modelo de intensidade de luz para a imagem de entrada 1. d) Resultado da busca usando modelo binário para a imagem de entrada 1. A imagem de referência foi ampliada para uma melhor visualização.

a)



	x	*y*	*θ*	*Fit.*	**Aval.**
emCGA	580	434	317,03	50,15	8,32E+04
ABE	580	428	123,99	50,16	4,03E+08
Dif.	0	6	193,04	0,01	0,02067%

b)



	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	572	425	313,5	5	85,96	9,43E+04
ABE	572	425	313,5	5	85,96	1,03E+11
Dif.	0	0	0	0	0,000	0,00009%

Figura 47: Busca por uma peça do dominó. a) Resultado da busca usando modelo de intensidade de luz para a imagem de entrada 2. b) Resultado da busca usando modelo binário para a imagem de entrada 2.

a)

	x	*y*	*θ*	*Fit.*	**Aval.**
emCGA	575	193	241,64	89,11	8,38E+04
ABE	575	193	241,64	89,11	4,03E+08
Dif.	0	0	0	0	0,02081%

b)

	x	*y*	*θ*	*t*	*Fit.*	**Aval.**
emCGA	523	125	192,33	128	93,38	8,85E+04
ABE	574	193	241,64	128	95,13	1,03E+11
Dif.	51	68	49,31	0	1,750	0,00009%

Figura 48: Busca por uma peça do dominó. a) Resultado da busca usando modelo de intensidade de luz para a imagem de entrada 3. b) Resultado da busca usando modelo binário para a imagem de entrada 3.

4.4 RESULTADOS DO emCGA EM *HARDWARE*

Os resultados do emCGA em *hardware* são divididos em 3 experimentos: comparação do emCGA em *hardware* e em *software*, comparação da demanda por recursos de *hardware* entre o emCGA e o AGC e ganho de desempenho do emCGA em *hardware*. Todos estes experimentos foram baseados no problema *Discrete1*, com 2 ou 8 dimensões e variáveis de 8 bits.

4.4.1 Resultado da comparação do emCGA em *hardware* e em *software*

A aplicação em *hardware* do emCGA deve ter o mesmo resultado da aplicação em *software* se ambos tiverem os mesmos parâmetros e problema de entrada. Isto é, se os parâmetros de tamanho de população, probabilidade de mutação, semente do gerador de números pseudo-aleatórios, comprimento do cromossomo e problema forem idênticos.

A dimensionalidade do problema *Discrete1* foi definida neste experimento como 2 e as variáveis, representadas por números inteiros sem sinal, com tamanho de 8 bits. Para estes parâmetros, o comprimento do cromossomo é de 16 bits e, por conseqüência, a probabilidade de mutação é 6,25%, isto é, 100%*1/*l*. O tamanho de população escolhido foi de 256 indivíduos.

Os indivíduos Novo e Elite são extraídos em cada geração em *software* e em *hardware*. Na aplicação em *hardware* os resultados são extraídos da ferramenta de simulação ModelSim. A utilização desta ferramenta facilita a extração destes dados, pois são gerados relatórios com os sinais que representam os indivíduos e número de avaliações de *fitness*. Os indivíduos Elite e Novo de cada geração são comparados através de uma função de similaridade. Como cada indivíduo decodificado é representado por duas variáveis inteiras sem sinal, denominadas de *x* e *y*, a função de similaridade entre estes indivíduos é definida pela distância Euclidiana, segundo apresentado na equação 15.

$$d(I_{HW}, I_{SW}) = d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \qquad (15)$$

Onde:

I_{HW}: indivíduo gerado no emCGA aplicado em *hardware*.

I_{SW}: indivíduo gerado no emCGA aplicado em *software*.

(*x*,*y*): variáveis decodificadas do cromossomo dos indivíduos.

d: distância Euclidiana.

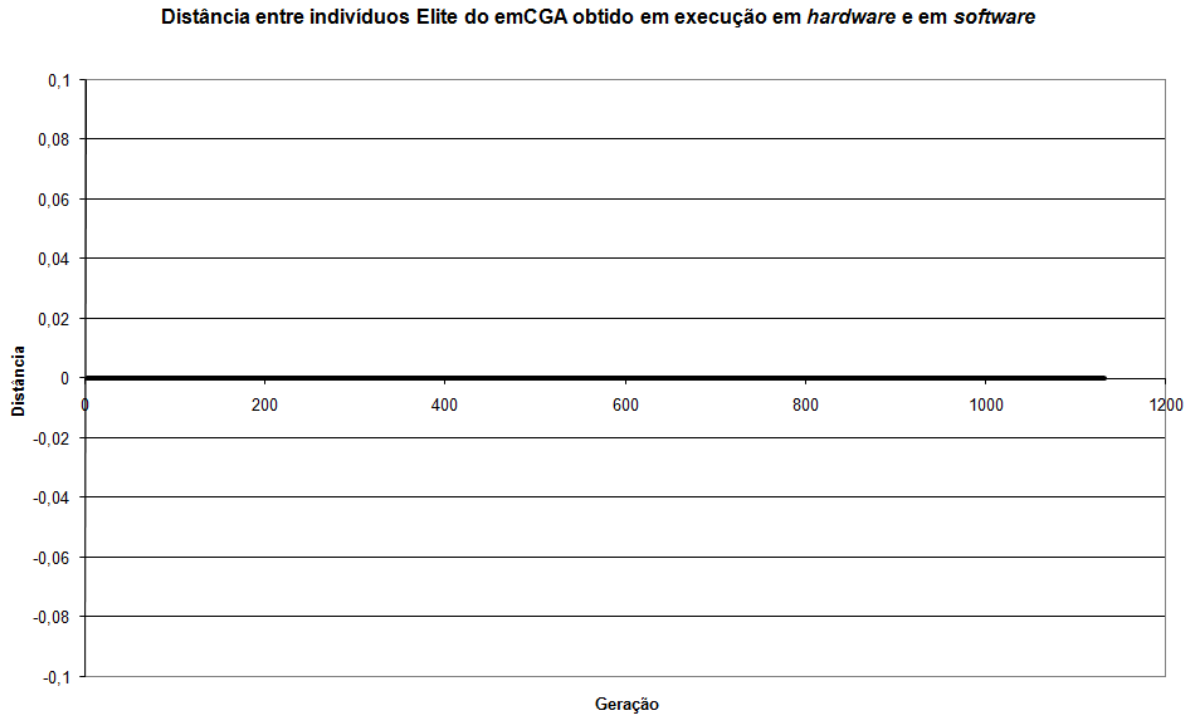


Figura 49: Distância Euclidiana entre o indivíduo Elite obtido em execução em *hardware* e em *software*.

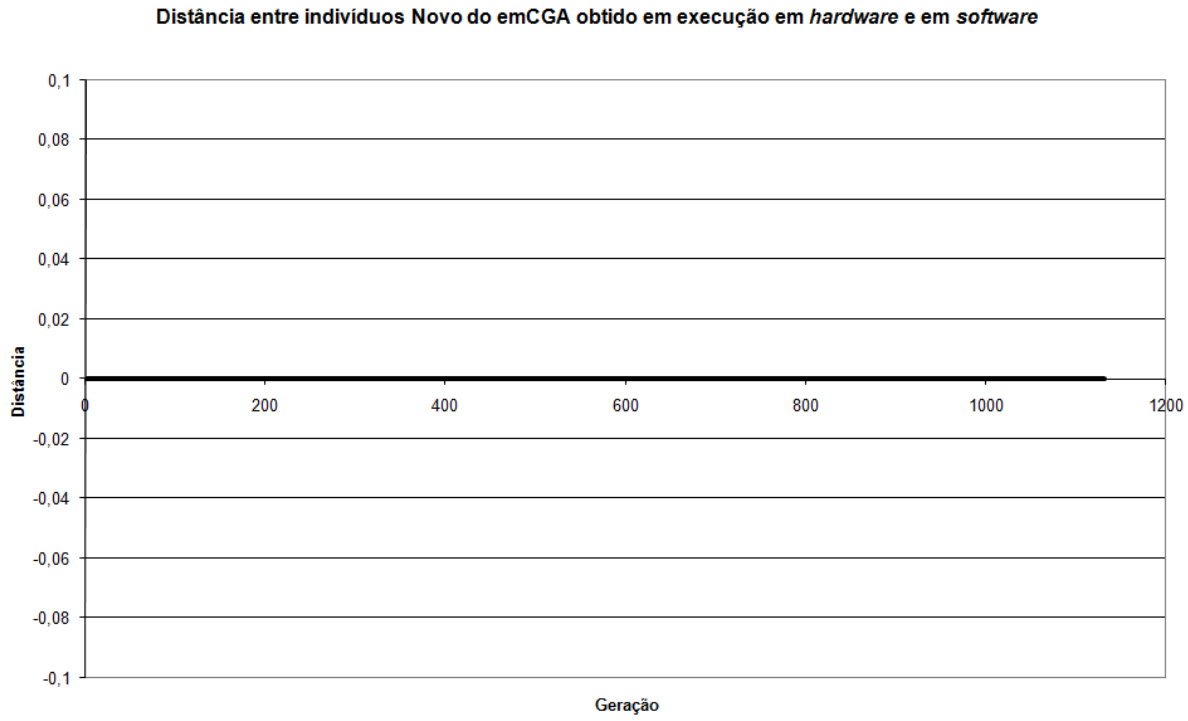


Figura 50: Distância Euclidiana entre o indivíduo Novo obtido em execução em *hardware* e em *software*.

4.4.2 Resultado da comparação da demanda por recursos de *hardware* entre o emCGA e o AGC

No segundo experimento é feita a implementação em *hardware* do emCGA e do AGC apresentado por Aporntewan e Chongstitvatana (2001), visando a comparação da demanda por recursos de hardware. O AGC em *hardware* será implementado usando o mesmo FPGA do emCGA. Como o FPGA empregado é de fabricação da Altera, esta demanda é representada numericamente pelo número de unidades lógicas. Assim, a tabela 8 mostra a demanda de recursos de *hardware*, em unidades lógicas, necessário para a implementação do emCGA e AGC para o problema *Discrete1*.

Em ambos os algoritmos, a dimensionalidade do problema é 8 e são usadas variáveis de 8 bits, inteiras e sem sinal. Desta forma, o comprimento do cromossomo é de 64 bits e a probabilidade de mutação é de 1,6%, aproximadamente 100%*1/*l*. O tamanho da população é definido como 256 indivíduos.

Tabela 8. Demanda de recursos de *hardware* para o emCGA e AGC

	Demanda por recursos de *hardware*
emCGA	4531 unidades lógicas
CGA	4212 unidades lógicas

4.4.3 Resultado do ganho de desempenho do emCGA em *hardware*

Para estimar o ganho de desempenho do emCGA será medido o tempo de execução para ambas as aplicações. O ganho de desempenho será determinado pela razão tempo de execução em *software* pelo tempo de execução em *hardware.* Ver equação 16.

$$g = \frac{t_{sw}}{t_{hw}} \tag{16}$$

Onde:

g: ganho de desempenho do emCGA em *hardware* com relação ao *software*.

t_{sw}: tempo de execução do emCGA em *software*.

t_{hw}: tempo de execução do emCGA em *hardware*.

Em ambas as aplicações o emCGA possui o mesmo problema (*Discrete1*) com os mesmos parâmetros de entrada. O problema tem dimensionalidade 8 e são usadas variáveis de 8 bits, inteiras e sem sinal. Assim, o tamanho do cromossomo é de 64 bits e probabilidade de mutação de 1,6%. O tamanho de população escolhido foi de 256 indivíduos. Estes parâmetros foram escolhidos baseados no trabalho de relação da probabilidade de mutação com o desempenho do emCGA (ver seção 3.4).

Os resultados obtidos são os tempos de execução, valores do *fitness* médio e número médio de avaliações de *fitness* para 500 execuções de cada aplicação do emCGA. A tabela 9 apresenta estes resultados. Com os tempos de execução é possível determinar o ganho de desempenho utilizando o emCGA em *hardware*. Com os resultados de *fitness* médio e número médio de avaliações de *fitness* é possível comparar estatisticamente a eficiência do emCGA em *hardware* e em *software*.

Tabela 9. Resultados de comparação entre emCGA em *hardware* e *software*.

	Fitness médio	Aval.	Tempo total (us)	Tempo médio (us)
Hardware	0,82	3300	102900	205,89
Software	0,82	3300	17781000	35562,00
	Ganho de desempenho (*g)*		173	

CAPÍTULO 5

DISCUSSÃO E CONCLUSÕES

5.1 ANÁLISE DOS RESULTADOS

5.1.1 Relação da probabilidade de mutação no desempenho do emCGA

O objetivo do estudo da relação entre o desempenho do emCGA e a probabilidade de mutação é o entendimento do funcionamento do operador de mutação neste algoritmo. Desta forma, este entendimento pode possibilitar uma melhor escolha da probabilidade de mutação, alcançando melhores resultados. Neste estudo foi feita a análise de resultados obtidos com experimentos em diferentes problemas com diferentes complexidades. Desta forma, as conclusões obtidas através desta análise podem ser extrapoladas para outros problemas.

A utilização do elitismo melhora o desempenho do AGC aumentando a pressão seletiva (AHN e RAMAKRISHNA, 2003). Em problemas mais complexos, esta pressão seletiva causa convergência prematura da população para mínimos locais. Nestes casos, o operador de mutação é uma solução para controlar a pressão seletiva e melhorar o desempenho do algoritmo. O emCGA utiliza esta técnica para melhorar o desempenho do AGC sem aumentar significativamente a complexidade, e, conseqüentemente, o custo computacional do algoritmo. Os resultados dos problemas mais complexos deste trabalho são apresentados nas figuras 29 e 30. Na figura 29 é possível se observar que a qualidade de solução é significativamente inferior quando o emCGA não usa o operador de mutação, isto é: quando a probabilidade de mutação é igual a zero. Aliado a isto, esta qualidade de solução inferior é alcançada com um menor número de avaliações de *fitness*, como apresentado na figura 30. Desta forma, é possível afirmar que, quando o AGC usa somente elitismo, a população converge prematuramente em um mínimo local.

Em um AG, o desempenho pode ser representado por dois valores numéricos, o valor de *fitness* e o número de avaliações de *fitness*. O valor de *fitness* é a qualidade da solução, ou seja: em problemas de minimização, pode-se dizer que uma solução é melhor se ela for menor do que as outras soluções comparadas. O custo computacional é proporcional ao número de avaliações de *fitness*, significando que quanto maior for o número de avaliações de *fitness* maior foi o custo computacional para alcançar a solução final. Como o critério de parada do

emCGA é a convergência da população, isto é, convergência do vetor de probabilidades, onde alcançar a solução final significa convergir o vetor de probabilidades.

Desta forma, esta análise baseia-se nos diferentes valores de *fitness* e no número de avaliações de *fitness* alcançados para diferentes valores de probabilidade de mutação. Contudo, como a probabilidade de mutação para os AGs tem relação com características intrínsecas dos problemas (CERVANTES e STEPHENS, 2006) e espera-se que o comportamento seja semelhante para o emCGA, então estes resultados serão analisados também em diferentes problemas com diferentes complexidades.

Comparando os diversos problemas quanto à complexidade para os AGCs, os problemas *Circle* e *Griewank* representam os problemas mais complexos enquanto que os problemas *Sphere* e *Discrete1* são os menos complexos. Contudo, cada um destes problemas pode ter sua complexidade modificada através da variação da dimensionalidade e do comprimento dos genes. O comprimento dos genes altera o comprimento do cromossomo e, conseqüentemente, o espaço de busca do problema sem alterar a função a ser minimizada. O aumento do espaço de busca torna o problema mais complexo para os algoritmos de busca, e também para os AGCs. Quando se aumenta a dimensionalidade também se aumenta o comprimento do cromossomo (e não o comprimento dos genes), bem como a complexidade da função a ser minimizada. Ou seja, quanto maior a dimensionalidade, mais complexa é a função a ser minimizada, e também mais complexo é o problema. Apesar de haver um consenso em recomendar uma probabilidade de mutação de 1/*l,* existem diversos trabalhos que mostram que características intrínsecas do problema alteram o valor ótimo da probabilidade de mutação para os AGs (CERVANTES e STEPHENS, 2006). Desta forma, espera-se que os resultados alcançados pelo emCGA para os diferentes problemas sejam também diferentes.

5.1.1.2 Análise gráfica para genes de resolução de 8 bits

As figuras 27 e 28 mostram, para genes de resolução de 8 bits, as superfícies tridimensionais (ϕ^n,μ,d) e (ε,μ,d), respectivamente, para cada uma das funções. Os resultados são obtidos para diferentes complexidades através da variação da dimensionalidade do problema.

Na figura 27 são mostrados os resultados de qualidade de solução para diferentes taxas de mutação e dimensionalidades do problema. Pode-se constatar que, quando a

dimensionalidade é baixa, as curvas formam uma planície, ou seja, $\Delta\phi$ é baixo. Quando se aumenta a dimensionalidade, então se aumenta o valor de $\Delta\phi$. Nesta condição, existem diversos pontos que poderiam ser (ϕ*,μ*). O ponto de melhor desempenho não pode ser indicado somente analisando a figura 27, pois dependerá também do custo computacional.

Na figura 28 é apresentada a relação entre o custo computacional e diferentes valores de probabilidade de mutação e dimensionalidade do problema. Apesar de menos evidente para o problema *Circle*, em todas as figuras é possível se observar a formação de duas cristas. A primeira crista está localizada quando a probabilidade de mutação é zero, um elitismo puro. Observando-se somente as curvas formadas para as dimensionalidades baixas, só existe esta crista. Assim, nestas dimensionalidades, o valor de ε diminui quando se aumenta μ. Contudo, quando a dimensionalidade é mais alta, a segunda crista começa a aparecer. Esta segunda crista começa mais próxima de 10%, mas, conforme a dimensionalidade aumenta, ela começa a se aproximar da origem. Como ao menos uma crista existe em todas as dimensionalidades, conclui-se que o valor de $\Delta\varepsilon$ é significativo desde dimensionalidades mais baixas.

Analisando-se em conjunto os resultados de (ϕ^n,μ,d) e (ε,μ,d) pode-se propor os pontos de melhor desempenho (ϕ*,μ*) e (ε*,μ*). Apesar dos valores de $\Delta\phi$ não serem significativos para dimensionalidades baixas, os valores de $\Delta\varepsilon$ o são. Assim, os pontos de melhor desempenho são definidos pela influência de μ no custo computacional. Para dimensionalidades altas, o valor de $\Delta\phi$ é mais significativo e a faixa de valores de μ que alcançam ϕ* fica mais estreita.

5.1.1.2 Análise para genes de resolução de 20 bits

Os resultados das relações de (ϕ^n,μ,d) e (ε,μ,d) para cada uma das funções e para genes de resolução de 20 bits são mostrados nas figuras 29 e 30, respectivamente. Comparando-se com os problemas de genes de 8 bits, os problemas com 20 bits são mais complexos para os AGCs. Isto se deve, principalmente, pelo aumento do espaço de busca. Desta forma, os resultados são obtidos para diferentes complexidades através da variação da dimensionalidade do problema.

Observa-se, na figura 29, que existe a formação de um vale em todos os problemas. Este vale é a região de melhor qualidade de solução, onde as soluções estão mais próximas do ótimo global. Pode-se observar que este vale torna-se mais estreito quanto maior a

dimensionalidade, e que este estreitamento converge para uma faixa centrada aproximadamente entre 1% e 3%.

Na figura 30, são ilustradas as superfícies que representam o custo computacional para diferentes dimensionalidades (d), para diferentes taxas de mutação (μ) e para cada um dos problemas propostos. Nestas figuras se formam duas cristas como na figura 18; contudo, elas são mais elevadas, com o valor de $\Delta\varepsilon$ maior. A primeira crista, aquela localizada no μ igual a zero, é mais baixa que a segunda crista, e não se torna tão significativa. Mas a segunda crista evidencia a correlação entre μ e ε. Esta crista começa a aparecer de forma suave próximo ao μ igual a 10%, mas enquanto d aumenta esta crista se torna proeminente e mais próxima da origem. Quando a dimensão é 10, esta crista apresenta um $\Delta\varepsilon$ maior, com μ na faixa de 1% a 3%. Esta crista é denominada como região de maior custo computacional e é mais evidente quando o problema possui maior complexidade.

Como os resultados apresentados nas figuras 29 e 30 mostram uma maior variação na qualidade de solução e custo computacional quando se aumenta a dimensionalidade, então se pode afirmar que: quanto maior a dimensionalidade do problema, mais significativo é o μ no desempenho do emCGA. Em problemas mais complexos, as regiões de melhor qualidade de solução e de maior custo computacional estão localizadas aproximadamente nos mesmos valores de μ. Ou seja, em problemas mais complexos, a escolha incorreta do μ leva a uma convergência prematura para um mínimo local. Este resultado é esperado em técnicas de CE, como explicado no trabalho de (DUMITRESCU, LAZZERINI, JAIN *et al*, 2000), onde uma pressão seletiva incorreta leva a uma convergência prematura.

5.1.1.3 Análise do melhor de desempenho

A relação do ϕ^* com a complexidade do problema é apresentada na figura 31. Pode-se observar que o valor de ϕ^* tende a aumentar com a complexidade do problema. No problema *Discrete1*, a variação do ϕ^* é menos significativa no eixo do comprimento dos genes, mas ainda existe. Apesar de este aumento ser evidente, ele não é linear. Ou seja, próximo ao extremo da complexidade, o valor de ϕ^* tende a aumentar mais rapidamente.

Como o critério de parada do algoritmo é a convergência do vetor de probabilidades, espera-se que o valor de ε^* seja fortemente influenciado pelo tamanho deste vetor, ou seja, pelo tamanho da população e o comprimento do cromossomo. Assim o valor de ε^* é

relacionado com o comprimento do cromossomo e não com a complexidade do problema, segundo a figura 32. Nesta figura pode-se observar que o valor de ε^* aumenta linearmente com o comprimento do cromossomo.

Na figura 35, é mostrada a relação da variação da qualidade da solução e da complexidade do problema. Estas figuras têm características semelhantes àquelas encontradas na figura 31, onde: quanto maior a complexidade do problema maior é a variação da qualidade da solução. Isto significa que a escolha correta do μ é mais importante em problemas com complexidades mais altas, permitindo alcançar um desempenho melhor.

Nos experimentos deste estudo, é esperado que o custo computacional seja fortemente influenciado pelo comprimento do cromossomo. Sendo assim, na figura 36 pode-se observar que o custo computacional aumenta conforme aumenta o comprimento do mesmo. Contudo, existe um ponto de inflexão para complexidades muito baixas, onde a variação do custo computacional tende a diminuir conforme aumenta o comprimento do cromossomo. Esta inflexão não ocorre para a variação da qualidade da solução. Dessa maneira, em baixas complexidades a escolha do μ influencia principalmente no custo computacional, o que se observa na análise dos resultados para comprimento de genes de 8 bits. No entanto, analisando os resultados das figuras 35 e 36, para complexidades mais altas, μ influencia mais significativamente a qualidade de solução.

Espera-se que, como os AGs, se possa recomendar uma probabilidade de mutação de $1/l$ para o emCGA. Sendo assim, a figura 33 mostra a relação do μ^* e a probabilidade de mutação recomendada $1/l$. Nos problemas *Sphere* e *Discrete1* os valores de μ^* são próximos aos valores da probabilidade de mutação recomendada, principalmente para comprimentos de cromossomos maiores. No entanto, no problema *Circle* os valores de μ^* estão afastados. Para o problema *Griewank* nem todos os valores de μ^* estão afastados, mas somente para os resultados dos experimentos com comprimentos de gene maiores que 12 bits, isto é: comprimentos de gene de 16 e de 20 bits. Na figura 34 é fornecida a evidência do que ocorre nestes casos. A distância entre o valor de ε^* e o ε_{max} é significativamente menor para os experimentos que os valores de μ^* estão afastados da probabilidade de mutação recomendada do que são próximos. Desta forma, é possível afirmar que se alcançam melhores resultados nestes experimentos, onde os parâmetros permitem um maior número de avaliações de *fitness*. Isto sugere que o algoritmo está convergindo prematuramente. Nestes casos, espera-se que o aumento do tempo de convergência do emCGA através do aumento no tamanho da população

permita a melhora do desempenho e, conseqüentemente, a probabilidade de mutação ótima seja próxima à recomendada.

5.1.2 Comparação do desempenho do emCGA com o neCGA e mCGA

A análise do desempenho leva em consideração a qualidade de solução e o custo computacional. Ou seja, define-se que o melhor desempenho é o resultado com o menor valor de *fitness* (para problemas de minimização, como os utilizados neste trabalho) e com o menor número de avaliações de *fitness*. Contudo, estes dois objetivos são conflitantes e a análise na curva de Pareto é mais útil do que a análise separada destes objetivos. Na curva de Pareto, o melhor compromisso entre a qualidade de solução e custo computacional é localizado próximo à origem, isto é: ponto (0,0). Desta forma, a figura 37 mostra os resultados somente para os pontos onde os resultados se aproximaram mais da origem. Os pontos com o melhor desempenho são destacados e mostrados os tamanhos da população usada, tornando possível uma melhor visualização. Contudo, os resultados do neCGA para os problemas *Discrete2* e *Sphere* não aparecem. Isto significa que estes resultados estão muito afastados da origem em comparação com os resultados dos outros algoritmos, não sendo, assim, representados.

Comparando-se o emCGA com o neCGA e o mCGA na figura 37, o emCGA alcança tanto a melhor qualidade de solução quanto o melhor custo computacional dos três algoritmos. Isto é observado para todos os problemas, exceto para o *Circle*, onde emCGA alcançou melhor custo computacional e uma qualidade de solução próxima aos outros algoritmos. Pode-se ainda ressaltar que o problema *Circle* utiliza uma função que é uma das mais complexas experimentadas. Logo, diferentemente do esperado, este problema mesmo com somente duas dimensões e comprimento de genes de 16 bits já poderia ser considerado de complexidade alta e a probabilidade de mutação mais adequada poderia ser mais próxima à recomendada, isto é, $1/l$. Apesar de que, no problema *Schaffer F6* o emCGA também alcançou o melhor desempenho entre os algoritmos, esta conclusão também poderia ser estendida para ele e resultados ainda melhores poderiam ser encontrados.

Os resultados na figura 37 mostram que o emCGA, de maneira geral, apresenta um melhor compromisso entre qualidade de solução e custo computacional que os outros algoritmos. Observa-se também que, para todos os problemas nos quais o emCGA alcançou o melhor desempenho, este foi alcançado com um menor tamanho de população. Desta forma, o

emCGA pode ser considerado uma alternativa interessante para problemas que exigem um AG compacto, como em implementações usando FPGA.

5.1.3 O emCGA aplicado na detecção de objeto para visão computacional

O problema de detecção de objeto para visão computacional facilmente recai em busca exaustiva (JAIN, KASTURI e SCHUNCK, 1995). O custo computacional do emCGA e do ABE é fortemente influenciado pelo número de avaliações de *fitness*. Logo, a comparação do custo computacional baseou-se na comparação entre o número de avaliações de *fitness* dos dois algoritmos.

Neste trabalho são apresentados dois modelos diferentes de representação da imagem. Em cada modelo espera-se que o emCGA possa buscar diferentes informações dos objetos procurados. Desta forma, os resultados são apresentados divididos em três seções. Na primeira e na segunda seção os resultados de cada modelo são apresentados separadamente. Nestes experimentos busca-se mostrar imagens de entrada complexas para mostrar a robustez da detecção de objeto usando o emCGA. Na última seção são apresentados resultados que mostram as diferenças de desempenho nas buscas utilizando os dois modelos propostos.

5.1.3.1 Análise dos resultados para o modelo de intensidade de luz

Na figura 38 é mostrado o resultado para uma detecção de um bispo do jogo de xadrez em uma imagem de entrada com outras peças de xadrez semelhantes. Como as outras peças de xadrez são semelhantes à peça bispo, o *fitness* alcançado em imagens representando estes objetos é alto e semelhante ao *fitness* alcançado com imagem representando o bispo. Desta forma, é possível afirmar que estas peças adicionam diversos máximos locais distribuídos no espaço de busca. Contudo, observa-se na figura 38b que o emCGA alcança o máximo global com um custo computacional de aproximadamente 0,022% do custo computacional do ABE.

No próximo experimento, diversas faces são procuradas em uma foto digital, como mostrado nas figuras 39 e 40. Neste experimento também não é adicionado nenhuma distorção significativa, pois a referência é um rosto recortado na imagem de entrada. No entanto, esta busca tem muitos máximos locais distribuídos, pois existem diversos objetos semelhantes. Portanto, este experimento explora a diferenciação de detalhes entre diferentes

objetos na imagem de entrada. Nas figuras 39d, 40a e 40b é possível observar que o emCGA alcança o máximo global com custo computacional de 0,027% do custo computacional do ABE. Ou seja, o emCGA usando o modelo de intensidade de luz consegue diferenciar o objeto de referência entre outros objetos semelhantes.

O último experimento mostra a busca por outra peça do jogo xadrez, o cavalo. Contudo, as imagens de entradas têm distorções de brilho e existem outras peças semelhantes que dificultam a detecção do objeto. Ainda, na imagem de entrada 2 o objeto procurado é rotacionado nos ângulos α e β, definidos na figura 26. O ângulo α é de aproximadamente 180º, ou seja, o objeto está invertido, e o ângulo β é gerado apoiando o objeto (o cavalo) sobre uma peça de ludo. Como mostrado nas figuras 41b e 41c, para as imagens de entrada 1 e 2 o emCGA alcança o máximo global com um custo computacional de aproximadamente 0,023% do custo computacional do ABE. Isto é, o emCGA usando o modelo de intensidade de luz é capaz de diferenciar o objeto de referência mesmo com distorções de brilho e de disposição do objeto no espaço tridimensional.

5.1.3.2 Análise dos resultados para o modelo binário

O primeiro experimento apresentado para o modelo binário é mostrado nas figuras 42 e 43. Neste experimento foi realizada uma busca por um bispo do jogo de xadrez. Na imagem de entrada 1, o bispo se encontra transladado e rotacionado em uma imagem com outras peças semelhantes, mas sem distorções. A imagem de entrada 2 é a imagem de entrada 1 adicionando um ruído gaussiano de 50%. Na imagem de entrada 3, o bispo se encontra parcialmente oculto por outra peça. Também ocorre o escurecimento da imagem devido à alteração do brilho da mesma. Nas figuras 42b, 43a e 43b é mostrado que o emCGA alcança um resultado próximo ou igual ao máximo global com um custo computacional de aproximadamente 0,0001% do custo computacional do ABE. Desta forma, o emCGA usando o modelo binário é capaz de detectar o objeto de referência entre outros objetos semelhantes e é robusto a fortes distorções de brilho e ruídos.

No experimento mostrado nas figuras 44 e 45 é mostrada uma busca por uma peça do jogo ludo. A imagem de entrada 1 apresenta o objeto transladado e rotacionado. Na imagem de entrada 2, o objeto detectado tem cor mais clara do que o objeto da imagem de referência. Na imagem de entrada 3 existe um forte escurecimento do ambiente. Este escurecimento é suficientemente forte para dificultar a detecção a olho nu. Desta forma, o objeto é destacado

também com o brilho e contraste alterados. Neste experimento o emCGA alcança novamente resultados próximos ao máximo global com um custo computacional de aproximadamente 0,0001% do custo computacional do ABE. Isto é, o emCGA usando o modelo binário é robusto a variações de brilho e de cor.

5.1.3.3 Análise os resultados comparação entre os modelos de intensidade de luz e binário

Nas figuras 45, 46 e 47 é apresentado um experimento que procura mostrar as principais diferenças entre a utilização dos dois modelos propostos. O objeto a ser detectado é uma peça do jogo dominó. A imagem de entrada 1 é de baixa complexidade para os dois modelos utilizados. Portanto, ambos os modelos alcançam bons resultados. O modelo binário tem um parâmetro a mais para otimizar, o limiar *t*. Desta forma, o custo computacional aumenta aproximadamente 30% em comparação com o modelo de intensidade de luz, o que significa um aumento de uma ordem de magnitude aproximadamente. Contudo, o aumento do ABE usando modelo binário é de 25458% em comparação com o modelo de intensidade de luz, ou seja: um aumento de 3 ordens de magnitude aproximadamente. O aumento do custo computacional do emCGA por usar o modelo binário é bem menor do que o custo computacional do ABE.

O segundo experimento tem a imagem adquirida em um ambiente com baixa luminosidade. Na última imagem, as outras peças do dominó estão espalhadas na imagem junto com a imagem a ser detectada.

Na imagem de entrada 2, o objeto também se encontra em um ambiente baixa luminosidade. Na figura 46a e 46b é mostrado que somente o modelo binário alcançou a solução global. O modelo de intensidade de luz alcançou qualidade de solução próxima à alcançada pelo ABE, muito embora ambas as soluções estejam erradas usando este modelo. Os valores de *fitness* das duas soluções são próximos apesar de que a localização e, principalmente, o ângulo de rotação serem muito diferentes. As soluções encontradas pelo ABE nos dois modelos também são diferentes. Se observarmos a solução destacada nas imagens, fica claro que a solução encontrada pelo modelo binário é a mais correta. Portanto, o modelo de intensidade de luz não se mostra apropriado na detecção de imagem quando existe uma forte alteração na luz do ambiente. No entanto, o modelo binário se mostra robusto para grandes variações de luz no ambiente, alcançando bons resultados mesmos nestas condições críticas.

Na última imagem de entrada, a peça de dominó está no meio das outras peças semelhantes de dominó. O emCGA com o modelo de intensidade de luz alcançou a mesma solução alcançada pelo ABE. No modelo binário, o ABE encontra o objeto procurado, apesar de que o emCGA encontrou um objeto semelhante, mas não o procurado, com uma qualidade de solução bem próxima à do ABE. O modelo binário não consegue diferenciar os detalhes do objeto, pois a limiarização do modelo binário compacta as informações disponíveis na imagem de referência. Isto não acontece com o modelo de intensidade de luz, pois o mesmo consegue diferenciar com sucesso entre objetos semelhantes.

5.1.4 Análise do emCGA em *hardware*

O emCGA em *hardware* foi implementado no *kit* UP3 da Altera que possui o FPGA Cyclone EP1C6Q240C8. A aplicação ocupou 4531 unidades lógicas, enquanto a aplicação usando AGC também implementado neste *kit* e baseado no algoritmo apresentado no trabalho (APORNTEWAN, CHONGSTITVATANA, 2001) ocupa 4212 unidades lógicas. Pode-se concluir, então, que a extensão elitismo com mutação proposta no trabalho de Silva, Lopes e Lima (2007) não aumenta consideravelmente o consumo de recursos de *hardware*.

O emCGA é quase tão compacto quanto o AGC, pois não existe mudança considerável nos blocos lógicos utilizados. Isto se deve primeiramente pela lógica da mutação ser praticamente comparadores de números inteiros, e também pelo elitismo acrescentar somente a memória para armazenar seu cromossomo e *fitness*, que requerem pouco recurso de *hardware*. Utilizando o elitismo não é necessário gerar dois indivíduos por geração, como no AGC. Assim, os blocos lógicos utilizados para a geração do segundo indivíduo se equivalem aos blocos lógicos usados na implementação da mutação. O aumento do consumo ainda pode ser atribuído ao bloco RNG, pois o emCGA requer o dobro de números pseudo-aleatórios por geração. Além disso, o emCGA possui ainda outra vantagem: a utilização de somente um avaliador de *fitness* (FEV). Assim, em aplicações onde a avaliação de *fitness* demanda muitos recursos de *hardware* a aplicação completa com o emCGA pode ser ainda mais compacta que a do AGC.

O objetivo do emCGA em *hardware* é ter a mesma eficiência da implementação em *software*. Ou seja, para uma mesma semente do RNG, ambas as implementações devem funcionar de forma idêntica, gerando os mesmos indivíduos a cada geração. Assim, o seu desempenho final também será idêntico. Dois experimentos procuraram demonstrar que o

funcionamento da aplicação em *hardware* a em *software* são iguais. O primeiro é utilizando ModelSim, a ferramenta de simulação disponibilizada pelo QuartusII da Altera. Nesta simulação se pode observar a cada geração o desempenho do emCGA em *hardware* e comparar com a implementação em *software*. Os resultados deste experimento são mostrados nas figuras 48 e 49 através da distância Euclidiana entre os indivíduos gerados pelo emCGA em *hardware* e em *software*. Esta distância sempre se mostrou igual a zero, demonstrando que os indivíduos são sempre iguais na mesma geração. Desta forma, além de não aumentar significativamente a demanda por recursos de *hardware*, o emCGA aumenta consideravelmente seu desempenho se comparado com outras extensões do AGC similares. Isto se espera porque se observou que emCGA em *software* mostrou melhor resultado que outras implementações de AGC, e o emCGA em *hardware* tem a mesma eficiência.

No segundo experimento, o emCGA foi executado 500 vezes e os resultados médios obtidos comparados com a versão em *software*. Assim, os resultados obtidos podem ser considerados também estatisticamente idênticos se alcançarem os mesmo resultados médios de *fitness* e número de avaliação de *fitness*. Na tabela 9 se pode constatar que os resultados médios também são iguais, comprovando, novamente, que o desempenho das duas implementações é também igual.

Uma das grandes vantagens de se utilizar uma aplicação em *hardware* é o ganho de desempenho no tempo de execução. Desta forma, é determinado o tempo de execução para as aplicações em *software* e em *hardware* durante 500 execuções. Deste tempo total é gerado o tempo médio de execução. Comparando com um emCGA em *software* executado em um computador de 2,8 GHz, a aplicação em *hardware* alcança ganho de desempenho de 173 vezes usando um *clock* de somente 48MHz.

5.2 CONCLUSÕES

O emCGA é uma nova proposta do já conhecido AGC, apresentado algumas propriedades que o torna interessante para a solução de diversas classes de problemas. Neste trabalho, além da proposta e implementação deste novo algoritmo, apresenta-se como contribuição importante o estudo detalhado da probabilidade de mutação. Em problemas muito simples, a probabilidade de mutação influencia mais significativamente o custo computacional do que a qualidade da solução. Por outro lado, para problemas mais complexos ocorre uma significativa influência da probabilidade de mutação na qualidade da

solução. De forma geral, observa-se que os valores de $\Delta\phi$ (associado à qualidade de solução) tendem a crescer com o aumento da dimensionalidade e do comprimento dos genes do problema. Por outro lado, para os valores de $\Delta\varepsilon$ (associado ao custo computacional) existe um ponto de inflexão, ou seja, até certo ponto ocorre uma diminuição do custo computacional com o aumento da dimensionalidade e do comprimento dos genes do problema. A partir deste ponto esta tendência se inverte, caracterizando este ponto de inflexão.

De forma intuitiva, é esperado que em problemas mais complexos a qualidade da solução decaia, isto é, para os problemas de minimização abordados o valor de $\phi*$ aumenta. Isto é confirmado com os resultados encontrados. Contudo, este aumento não é linear com o aumento da complexidade. Observa-se que somente nos extremos da complexidade é que a qualidade da solução começa a decair significativamente.

Como nos AGCs o critério de parada típico é a convergência do vetor de probabilidades, o emCGA estudado também utilizou este critério. Desta forma, o custo computacional (ε*) é influenciado fortemente pelo comprimento do cromossomo. Observa-se que o custo computacional do resultado com melhor desempenho aumenta linearmente com o comprimento do cromossomo. Desta forma, pode-se concluir que o custo computacional e a qualidade da solução são afetados de forma diferente pela complexidade do problema.

Finalmente, pode-se concluir que a probabilidade de mutação recomendada para o emCGA é 1/*l*, como nos AGs. Uma exceção é observada nos problemas *Circle* e *Griewank*, onde a probabilidade de mutação com o melhor desempenho (μ*) é normalmente maior do que 1/*l*. Contudo, nestes problemas os seus resultados mostraram que o melhor desempenho é alcançado próximo à região de maior custo computacional. Desta forma, conclui-se que estes problemas estavam no limiar da classe de problemas onde o emCGA com população de 256 indivíduos consegue obter um bom resultado. Assim, se escolhido um tamanho de população adequado, espera-se que a probabilidade de mutação recomendada seja 1/*l* para qualquer problema.

O emCGA apresenta um compromisso entre qualidade de solução e custo computacional melhor do que outros algoritmos AGC. Observou-se também que, para todos os problemas tratados nos quais o emCGA alcançou o melhor desempenho, este também foi alcançado com um menor tamanho de população. Ainda, o emCGA alcança o melhor desempenho sem um aumento significativo na complexidade de implementação. Desta forma, o emCGA pode ser considerado uma alternativa interessante para problemas que exigem um AG compacto, como no caso de implementação em *hardware* reconfigurável.

Para validação do emCGA foram apresentadas implementações em *software* e em *hardware.* Para implementação em *software* optou-se por usar o problema de detecção de objetos em imagens através de dois modelos diferentes. Ambos modelos mostraram uma significativa redução do custo computacional em relação ao ABE. No modelo de intensidade de luz, que tem menos parâmetros a serem otimizados, o custo computacional foi aproximadamente 0,025% do custo computacional do ABE. No modelo binário, que tem mais parâmetros, alcançou-se uma redução ainda maior, sendo o custo computacional de 0,0001% do ABE.

Observou-se que os dois modelos utilizados extraem informações diferentes do objeto, nem sempre alcançando o mesmo desempenho nos mesmos tipos de imagens de entrada. Enquanto o modelo de intensidade de luz se mostra apropriado na diferenciação de detalhes dos objetos, o modelo binário se mostra apropriado nas imagens com ruído ou baixa intensidade de luz. O modelo de intensidade de luz permite detectar objetos complexos no meio de outros objetos semelhantes, desde que os objetos apresentem alguma diferença de intensidade de luz que os diferencie - como ter cor diferente da cor predominante na imagem, por exemplo. Por outro lado, no modelo binário, os objetos devem ter apenas uma cor, ou poucas variações de cores. Ainda, é importante que o objeto se destaque, isto é: a cor do objeto ser diferente da cor predominante na imagem, e que a imagem de referência possua *pixels* da paisagem de fundo. Os *pixels* da paisagem de fundo na imagem de referência é importante para a otimização do parâmetro de limiar *t*.

De maneira geral, o emCGA alcançou qualidade de solução semelhante ao ABE e com custo computacional muito inferior, em todos os experimentos. Desta forma, conclui-se que as técnicas utilizadas no emCGA permitem utilizar o AGC em problemas reais sem aumentar significativamente sua complexidade de implementação e nem seu custo computacional.

Para implementação em *hardware* optou-se por usar um problema de menor complexidade de implementação: o *Discrete1*. Os resultados obtidos são absolutamente iguais aos encontrados na abordagem por *software*. Observou-se que o emCGA pode ser implementado em *hardware* sem um significativo aumento de recursos em comparação ao AGC, muito embora, com significativo ganho de desempenho se comparado com o emCGA implementado em *software.*

Por fim, através deste trabalho é possível concluir que emCGA é apropriado para aplicações que demandem um algoritmo compacto e eficiente, seja para aplicações que requeiram alto desempenho ou aplicações que sejam implementadas em sistemas com

capacidade limitada de processamento. Possui vantagens tanto por sua forma compacta, quanto por seu ganho no tempo de execução. Como o emCGA em *hardware* funciona como o emCGA em *software*, então espera-se que as conclusões de desempenho em *software* possam também ser estendidas para o emCGA em *hardware*.

5.3 TRABALHOS FUTUROS

Para a continuidade desta dissertação, sugere-se os seguintes trabalhos:

- Melhorar o algoritmo em *hardware*, aumentando o seu ganho de desempenho do mesmo através da implementação de uma técnica de *pipeline* nos operadores do AGC;
- Estudar e propor novas técnicas com populações em paralelo;
- Aplicar o emCGA em *hardware* ao problema de detecção de objetos em imagens;
- Utilizar outros modelos na detecção de imagem;
- Procurar novos problemas que demandem a utilização do emCGA em *software*;
- Procurar novos problemas que demandem a utilização do emCGA em *hardware*.

REFERÊNCIAS BIBLIOGRÁFICAS

ACKLEY, D. H. **A Connectionist Machine for Genetic Hill Climbing**. Boston: Kluwer Academic, 1987.

AHN, C. W., RAMAKRISHNA, R. S. A genetic algorithm for shortest path routing problem and the sizing of populations. **IEEE Transactions on Evolutionary Computation**, v. 6, n. 6, p. 566–579, Dec. 2002.

AHN, C. W., RAMAKRISHNA, R. S. Elitism-based compact genetic algorithms. **IEEE Transactions on Evolutionary Computation**, v. 7, n. 4, p. 367-385, August 2003.

APORNTEWAN, C., CHONGSTITVATANA, P. A hardware implementation of the compact genetic algorithm. In: **Proceedings of the 2001 IEEE Congress Evolutionary Computation** , Seoul, Korea, v. 1, p. 624–629, 2001.

BALUJA, S. Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning. **Technical Report No. CMUCS-94-163**, Carnegie Mellon University, USA, 1994.

BALUJA, S., CARUANA, R. Removing the genetics from the standard Genetic Algorithm. **Technical Report No. CMU-CS-95-141**. Carnegie Mellon University, USA, 1995.

BARAGLIA, R., HIDALGO, J. I., PEREGO, R. A hybrid heuristic for the traveling salesman problem. **IEEE Transactions on Evolutionary Computation**, v. 5, n. 6, p. 613–622, Dec. 2001.

BLAND, I. M., MEGSON, G. M. The systolic array Genetic Algorithm, an example of systolic arrays as a reconfigurable design methodology. In: **Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines**, Los Alamitos, USA, p. 260-261, 1998.

CENTENO, T. M., LOPES, H. S., FELIZBERTO, M. K., ARRUDA, L. V. R. Object detection for computer vision using a robust genetic algorithm. In: **Proceedings of the 7th European Workshop on Evolutionary Computation in Image Analysis and Signal**, LNCS, v. 3449, p. 284-293, Lausanne, Switzerland, 2005.

CERVANTES, J., STEPHENS, C. R. "Optimal" mutation rates for genetic search. In: **Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation**, Seattle, Washington, USA, p. 1313-1320, 2006.

COMPTON, K. Reconfigurable computing: a survey of systems and software. **ACM Computing Surveys**, v. 34, n. 2, p. 171 – 210, 2002.

CUPERTINO, F., MININNO, E., NASO, D. Elitist compact genetic algorithms for induction motor self-tuning control. In: **IEEE Congress on Evolutionary Computation CEC 2006**, Vancouver, Canada, p. 3057-3063, 2006.

DUMITRESCU, D., LAZZERINI, B., JAIN, L. C., DUMITRESCU, A. **Evolutionary Computation**. Boca Raton, FL: CRC Press, 2000.

ESHELMAN, L. J., SCHAFFER, J. D. Crossover's niche. In: Forrest, S. (Ed.), **Proceedings of the Fifth International Conference on Genetic Algorithms**, San Mateo, CA: Morgan Kaufmann, p. 9-14, 1993.

ESTRIN, G., VISHANATHAN, C. R. Organization of a "fixed-plus-variable" structure computer for computation of eigenvalues and eigenvectors of real symmetric matrices. **Journal of the ACM**, v. 9, n. 1, p. 41-60, January 1962a.

ESTRIN, G., VISHANATHAN, C. R. Correction and addendum: organization of a "fixed-plus-variable" structure computer for computation of eigenvalues and eigenvectors of real symmetric matrices. **Journal of the ACM**, v. 9, n. 4, p. 522, October 1962b.

GALLAGHER, J. C., VIGRAHAM, S., KRAMER, G. A family of compact genetic algorithms for intrinsic evolvable hardware. In: **IEEE Transactions on Evolutionary Computation**, v. 8, n. 2, p. 111–126, April 2004.

GOLDBERG, D. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Reading: Addison Wesley, 1989.

GOLDBERG, D. E., DEB, K., CLARK, J. H. Genetic algorithms, noise, and the sizing of populations. **Complex Systems**, v. 6, n. 4, p. 333–362, 1992.

GOLDBERG, D. E., RUNDNICK, M. Genetic algorithms and the variance of fitness. **Complex Systems**, v. 5, n. 3, p. 265–278, 1991.

GRAHAM, P., NELSON, B. A hardware genetic algorithm for the traveling salesman problem on SPLASH 2. In: **Proceedings of the 5th Int. Workshop on Field Programmable Logic and Applications**, Oxford, England, p. 352-361, 1995.

GRAHAM, P., NELSON, B. Genetic algorithms in software and in hardware - a performance analysis of workstation and custom computing machine implementations. In: Kenneth, L. *et al*. (Eds.)**, Proceedings of the Fourth IEEE Symposium of FPGAs for Custom Computing Machines**, p. 216–225, Napa Valley, Califormia, USA, 1996.

HARIK, G. Linkage learning via probabilistic modeling in the ECGA. **IlliGAL Report No. 99010,** University of Illinois at Urbana-Champaign, USA, 2000.

HARIK, G., CANTU-PAZ, E., GOLDBERG, D. E., MILLER, B. L. The gambler's ruin problem, genetic algorithms, and the sizing of populations. **IEEE Transactions on Evolutionary Computation**, v. 7, n. 3, p. 231–253, 1999.

HARIK, G., GOLDBERG, D. E. Learning linkage. In: Belew, R. e Vose, M. (Eds.), **Foundations of Genetic Algorithms**, v. 4, San Francisco: Morgan Kaufmann, p. 247-262, 1997.

HARIK, G., LOBO, F., GOLDBERG, D. E. The compact genetic algorithm. In: **IEEE Transactions on Evolutionary Computation**, v. 3, n. 4, p. 287-297, 1999.

HE, J., YAO, X. From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms. **IEEE Transactions on Evolutionary Computation**, v. 6, n. 5, p. 495–511, Oct. 2002.

HERTZ, J., KROGH, A., PALMER, G. **Introduction to the Theory of Neural Computation**. Addison-Wesley, 1993.

HIDALGO, J. I., PRIETO, M., LANCHARES, J., BARAGLIA, R., TIRADO F. Hybrid parallelization of a Compact Genetic Algorithm. In: **Proceedings of the 11th Euromicro Conference on Parallel Distributed and Network Processing (Congress)**, Genoa, Italy, 2003.

HOLLAND, J. H. Hierarchical descriptions of universal spaces and adaptive systems. **Technical Report ORA Projects 01252 and 08226**, Ann Arbor: University of Michigan, Department of Computer and Communication Sciences, 1968.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems.** Ann Arbor: The University of Michigan Press, 1975.

JAIN, R., KASTURI R., SCHUNCK B. G. **Machine Vision**. New York: McGraw-Hill, 1995.

JEWAJINDA, Y., CHONGSTIVATANA, P. A cooperative approach to compact genetic algorithm for evolvable hardware. In: **Proceedings of IEEE Congress on Evolutionary Computation CEC 2006**, Vancouver, Canada, p. 2779-2786, 2006.

KAVVADIAS, N., GIANNAKOPOULOU, V., NIKOLAIDIS, S. Development of customized processor architecture for accelerating genetic algorithm. In: **Microprocessors and Microsystems**, v. 31, p. 347-359, 2007.

LARRAÑAGA, P., LOZANO, J. A. **Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation**. Boston, MA: Kluwer, 2002.

LOBO, F. G., LIMA, C. F., MARTIRES, H. An architecture for massive parallelization of the compact genetic algorithm. In: K. Deb *et al*, (Eds.), **Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)**, Seattle, Washington, USA, Part II, LNCS, v. 3103, p. 412-413. Springer, 2004.

LOBO, F. G., LIMA, C. F., MARTIRES H. Massive parallelization of the compact genetic algorithm. In: R. Ribeiro *et al* (Eds.) **Proceedings of the International Conference on Adaptive and Natural Computing Algorithms (ICANNGA'05)**, Coimbra, Portugal, p. 530-533, December 2005.

MARTIN, P. An analysis of random number generators for a hardware implementation of genetic programming using FPGAs and Handel-C, **Technical Report CSM-358**, University of Essex, 2002.

MARUYAMA, T. FUNATSU, T., HOSHINO T. A field-programmable gate-array system for evolutionary computation. In: **Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm FPL'98**, Tallinn, Estonia LNCS, v. 1482, p. 356-365, 1998.

MIRMINNO, E., CUPERTINO, F., NASO, D. Real-valued compact genetic algorithms for embedded microcontroller optimization, In: **IEEE Transactions on Evolutionary Computation**, v. 12, n. 8, p. 203-219, 2008.

MEYSENBURG, M. M., FOSTER J. A. Random generator quality and GP performance. In: Banzhaf, W. *et al*. (Eds.), **Proceedings of the Genetic and Evolutionary Computation Conference**, Orlando, Florida, USA, v. 2, p. 1121–1126, 1999a.

MEYSENBURG, M. M., FOSTER J. A. Randomness and GA performance, revisited. In: Banzhaf, W. *et al*. (Eds.), **Proceedings of the Genetic and Evolutionary Computation Conference**, Orlando, Florida, USA, v. 1, p. 425–432, 1999b.

MÜHLENBEIN, H. The equation for response to selection and its use for prediction, In: **Evolutionary Computation**, v. 5, n. 3, p. 303-346, 1997.

MÜHLENBEIN, H., PAAß, G. From recombination of genes to the estimation of distributions I: binary parameters. In**: Proceedings of Parallel Problem Solving from Nature IV**, Berlin, Germany, p. 178 –187, 1996.

MÜLLER, S. D., MARCHETTO, J., AIRAGHI, S., KOUMOUTSAKOS, P. Optimization based on bacterial chemotaxis. In: **IEEE Transactions on Evolutionary Computation**, v. 6, n. 1, p. 16-29, 2002.

ROGERS, A., PRUEGEL-BENNETT, A. Genetic drift in genetic algorithm selection schemes. In: **IEEE Transactions on Evolutionary Computation**, v. 3, n. 4, p. 298 – 303, 1999.

SAENZ, F., IBARRA, A., LANCHARES, J., HIDALGO, J. I. Pipelined genetic architecture with fitness on the fly. In: **Proceedings Euromicro Symposium on Digital Systems Design DSD'2001**, Warsaw, Poland, p. 382 – 385, 2001.

SASTRY, K., GOLDBERG, D. E. On extended compact genetic algorithm. **IlliGAL Report No. 2000026,** University of Illinois at Urbana-Champaign, USA, 2000.

SASTRY, K., GOLDBERG, D. E. Modeling tournament selection with replacement using apparent added noise, In: **Proceedings of the Genetic and Evolutionary Computation Conference**, San Francisco, California, USA, p. 7–11, 2001.

SCHNEIDER, M., LOPES, H. S., PEDRONI, V. A. Object detection in a computer vision system using particle swarm optimization. In: **Proceedings of the XXVII Iberian Latin-American Congress on Computational Methods in Engineering - CILAMCE**, Belém, PA, Brazil, [s.p.], 2006.

SCOTT, S., SETH, A. HGA: A hardware-based Genetic Algorithm. In: **Proceedings of the ACM/SIGDA Third International Symposium on Field-Programmable Gate Arrays**, Monterey, California, USA, p. 53-59, 1995.

SHACKLEFORD, B., OKUSHI, E., YASUDA, M., KOIZUMI, H., SEO, K., IWAMOTO, T., YASUURA, H. An FPGA-based genetic algorithm machine, In: **Proceedings of the ACWSIGDA International Symposium on Field-Programmable Gate Arrays**, Monterey, California, USA, p. 218, 2000.

SHACKLEFORD, B., SNIDER, G., CARTER, R.J., OKUSHI, E., YASUDA, M., SEO, K., YASUURA, H. A high-performance, pipelined, FPGA-based genetic algorithm module, **Genetic Programming and Evolvable Machines**, v. 2, n. 1, p. 33-60, 2001.

SILVA, R. R., LOPES, H. S., LIMA, C. R. E. A new mutation operator for the elitism-based compact genetic algorithm. In: Beliczynski, B. *et al*. (Eds.), **Proceeding of Adaptive and Natural Computing Algorithms**, Warsaw, Poland, parte I, LNCS, v. 4331, p. 159-166, 2007.

SYSWERDA, G. Uniform crossover in genetic algorithms. In: J. D. Schaffer (Eds.), **Proceedings of the Third International Conference on Genetic Algorithms**, USA, Morgan Kaufmann, 1989.

SYSWERDA, G. Simulated crossover in Genetic Algorithms. In: Whitley, L. D. (Ed.), **Foundations of Genetic Algorithms 2**, San Mateo, CA: Morgan Kaufmann, p. 239-255, 1993.

THIERENS, D., GOLDBERG, D. E., PEREIRA, A. Domino convergence, drift and the temporal salience structure of problems. In: **Proceedings of IEEE World Congress on Computational Intelligence**, Anchorage, Alaska, USA, p. 535 – 540, 1998.

TOMMISKA, M., VUORI, J. Hardware implementation of GA. In: Alander, J. T. (Ed.)**, Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)**, Vaasa, Finland, p. 71-78, 1996.

TSUTSUI, S. Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram, In: Guervós, J. J. M. *et al*. (Eds.), **Parallel Problem Solving from Nature - PPSN VII**, Granada, Spain, LNCS, v. 2439, Berlin, Germany: Springer-Verlag, p. 224–233, 2002.

VILLASENOR, J., MANGIONE-SMITH, W. H., Configurable computing. **Scientific American**, USA, June 1997.

WRIGHT, A. H., VOSE, M. D., ROWE, J. E, Implicit parallelism. In Foster, J. (Ed.), **Proceedings of the Genetic and Evolutionary Computation Conference** (GECCO), Chicago, USA, LNCS, v. 2724, p. 1505-1517, Springer, 2003.

ZHOU, C., MENG, K., QIU, Z. Compact genetic algorithm mutated by bit. In: **Proceedings of the 4th World Congress on Intelligent Control and Automation**, Shanghai, China, v. 4, p. 1836-1839, 2002.

RESUMO:

O AG (Algoritmo Genético) é uma ferramenta eficiente para problemas de otimização, mas, em muitas aplicações, o custo computacional do AG pode ser alto demais ou pode demandar muitos recursos de *hardware*. O AGC (Algoritmo Genético Compacto) imita um AGS (Algoritmo Genético Simples) reduzindo a demanda por recursos através da utilização um vetor de probabilidades no lugar da população. Aliado a isto, o AGC é um algoritmo recente, introduzido em 1999, e não completamente explorado. Desta forma, este trabalho propôs um novo operador de mutação para o AGC com elitismo, denominado emCGA (*elitism with mutation Compact Genetic Algorithm*). Observou-se o desempenho deste novo algoritmo usando diferentes valores de probabilidade de mutação em diferentes problemas de *benchmark*. O emCGA pode alcançar melhor qualidade de solução com um menor custo computacional e utilizando um tamanho menor de população, se comparado com outros algoritmos semelhantes encontrados na literatura, isto é: neCGA (*Nonpersistent Elitism Compact Genetic Algorithm*) e mCGA (*Compact Genetic Algorithm with elitism and Mutation*). Ainda, o emCGA foi aplicado ao problema de detecção de objetos em imagens, onde pode-se observar que este algoritmo pode alcançar bons resultados em problemas reais, considerando o compromisso entre qualidade de solução e custo computacional. Aliando-se ao ganho de desempenho e à reduzida demanda por hardware na implementação em lógica reconfigurável, recomenda-se o uso do emCGA em problemas reais que demandem um algoritmo compacto e eficiente.

PALAVRAS-CHAVE

Computação Evolucionária, Algoritmo Genético Compacto, Algoritmos de Estimação de Distribuição, Registro de Imagens, Lógica Reconfigurável.

ÁREA/SUB-ÁREA DE CONHECIMENTO

1.03.00.00-7 – Ciência da Computação

1.03.03.04-9 – Engenharia de Software

1.03.04.01-0 – *Hardware*

3.04.99.00-7 – Engenharia Elétrica

3.04.03.03-0 Circuitos Eletrônicos

ano
2008

N°: 473