Diagnóstico em Nível de Sistema Baseado em Computação Evolucionária

Bogdan Tomoyuki Nassu, Aurora T. Ramirez Pozo, Elias Procópio Duarte Jr.

Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.018 – CEP 81.531-990 – Curitiba – PR – Brasil

{bogdan, aurora, elias)@inf.ufpr.br

Abstract. *The size and complexity of systems based on multiple processing units asks for the employment of techniques for automatic diagnosis of these units. System-level diagnosis consists in determining which units in a system are faulty and which are fault-free. This work describes evolutionary algorithms that can be used to accomplish diagnosis. A simple and a specialized genetic algorithm, as well as variants of the PBIL and Compact GA algorithms were implemented. Experimental results show a comparison of the performances of these algorithms.*

Resumo. *O aumento no tamanho e complexidade dos sistemas compostos por múltiplas unidades de processamento torna necessário o uso de técnicas para o diagnóstico automático destas unidades. O diagnóstico em nível de sistema consiste em determinar quais unidades do sistema estão falhas e quais não estão. Este trabalho descreve algoritmos evolutivos que podem ser usados para realizar o diagnóstico. Foram implementados um algoritmo genético simples e um especializado, além de variantes dos algoritmos PBIL e AG compacto. Resultados experimentais mostram uma comparação entre o desempenho destes algoritmos.*

1. Introdução

Um sistema composto por múltiplas unidades de processamento é definido como sendo uma coleção de n unidades heterogêneas, representadas por um conjunto $U = \{u_1, ..., u_n\}$. Estas unidades podem falhar, comprometendo o funcionamento do sistema. O aumento no tamanho e complexidade de sistemas deste tipo torna necessário o uso de técnicas para o diagnóstico automático de falhas em suas unidades. O diagnóstico em nível de sistema consiste em determinar quais unidades do sistema estão falhas e quais não estão. A partir desta informação, pode-se tomar providências como a substituição ou conserto das unidades falhas.

O modelo clássico para considerar as falhas em nível de sistema é aquele apresentado em [Preparata, Metze e Chien 1967], o modelo PMC. Neste modelo, cada unidade é capaz de testar outras unidades de forma a determinar seu estado de maneira inequívoca. Em um sistema em situação de falha existe um certo subconjunto F de unidades permanentemente falhas, chamado conjunto de falhas. Assume-se que o estado das unidades em F não muda durante o diagnóstico, e que os resultados dos testes feitos por estas unidades são indefinidos, ou seja, uma unidade falha pode "mentir", afirmando que uma outra unidade falha não está falha ou vice-versa. O diagnóstico consiste na determinação do estado das unidades do sistema a partir dos testes realizados. Este

diagnóstico é feito por uma unidade central que nunca falha, para a qual os resultados dos testes feitos são enviados.

No modelo PMC, define-se um grafo direcionado G(U,*E*), no qual cada unidade $u_i \in U$ é um vértice, e cada aresta (u_i, u_j) pertence a E se e somente se u_i testa u_j.

A cada unidade $u_i \in U$, associa-se dois conjuntos:

$$\Gamma(u_i) = \{u_j : (u_i, u_j) \in E\} \text{ e}$$

$$\Gamma^{-1}(u_i) = \{u_j : (u_j, u_i) \in E\}$$

O primeiro contém as unidades u_j testadas por u_i, e o segundo as unidades uj que testam u_i. A estes conjuntos, associam-se os valores $d_{in}(u_i) = |\Gamma^{-1}(u_i)|$ e $d_{out}(u_i) = |\Gamma(u_i)|$, que definem, respectivamente, o número de unidades que testam u_i e o número de unidades testadas por u_i.

Após os testes, a cada aresta $(u_i, u_j) \in E$ associa-se um resultado de teste w_{ij}. Se a unidade u_i testa u_j como sendo falha, o valor 1 é atribuído a w_{ij}, do contrário o seu valor é 0. Dadas as asserções do modelo PMC, w_{ij} é um resultado confiável se e somente se u_i é uma unidade sem falhas. O conjunto de todos os resultados dos testes feitos é a síndrome *S* do sistema. A síndrome é definida como uma função de mapeamento *W*, definida tal que $W(u_i, u_j) = w_{ij}$. O diagnóstico é realizado pela unidade central a partir de *S*. Uma síndrome *S* é dita compatível com um conjunto de falhas *F* se, para toda aresta $(u_i, u_j) \in E$ tal que u_i não é falha, $W(u_i, u_j) = 1$ se e somente se u_j é falha. São definidos também os subconjuntos $S(u_i)$ e $S^{-1}(u_i)$ de *S* como sendo os conjuntos dos resultados dos testes feitos por u_i e os resultados dos testes feitos sobre u_i, respectivamente.

Um sistema é dito t-diagnosticável se todas as unidades falhas do sistema podem ser identificadas sem erro desde que o seu número não ultrapasse *t*. É provado que, se duas unidades não se testam mutuamente, um sistema é t-diagnosticável se $n \geq 2t + 1$ e, para toda unidade $u_i \in U$, $d_{in}(u_i) \geq t$, ou seja, se cada unidade é testada por pelo menos t outras unidades [Preparata, Metze e Chien 1967]. O diagnóstico é um problema bem conhecido [Masson, Blough e Sullivan 1996], e diversas soluções eficientes já foram desenvolvidas. Porém, o problema de se determinar o estado de todas as unidades de um sistema t-diagnosticável a partir dos resultados dos testes feitos carece de métodos eficientes. Por este motivo, em [Elhadef e Ayeb 2000] é apresentada uma solução para este problema baseada no uso de algoritmos genéticos.

Algoritmos genéticos, ou AGs, são algoritmos de busca baseados nos princípios da seleção natural e recombinação gênica [De Jong 1975], bastante usados na otimização de funções. Existem também diversas técnicas evolutivas que derivam dos AGs, e que usam a distribuição probabilística das soluções como ferramenta para guiar a busca. Entre estas, pode-se destacar o *Population-Based Incremental Learning* - PBIL [Baluja 1994, Baluja e Caruana 1995] e os AGs compactos [Harik, Lobo e Goldberg 1998].

O objetivo deste trabalho é comparar o desempenho de quatro tipos de algoritmos evolutivos quando os mesmos são usados para resolver o diagnóstico em nível de sistema. O primeiro tipo é um algoritmo genético simples, cujos resultados são usados apenas como base para a comparação. O segundo tipo é um algoritmo genético especializado, baseado naquele proposto em [Elhadef e Ayeb 2000]. Os outros dois algoritmos são implementações especializadas do PBIL e de um AG compacto, com otimizações desenvolvidas especificamente para este trabalho.

A seção 2 apresenta um algoritmo genético simples que pode ser usado para resolver o diagnóstico em nível de sistema. A seção 3 apresenta outros algoritmos evolutivos que podem ser usados para resolver este problema: um AG especializado, o PBIL e o AG compacto. A seção 4 descreve os experimentos realizados e os resultados obtidos nos mesmos, e discorre sobre tais resultados. A seção 5 traz as conclusões e considerações finais do artigo.

2. Um Algoritmo Genético Simples para Diagnóstico em Nível de Sistema

Algoritmos genéticos, ou AGs, são algoritmos de busca baseados nos princípios biológicos da seleção natural e da recombinação gênica [De Jong 1975]. Eles são bastante usados na otimização de funções. Os AGs são capazes de encontrar rapidamente regiões do espaço de busca com alto desempenho, e são resistentes a ruídos nos dados de entrada, mas como outros métodos heurísticos por vezes não conseguem encontrar uma solução ótima.

O funcionamento de um AG é baseado na sobrevivência dos indivíduos mais aptos no decorrer de uma série de gerações. Cada indivíduo, ou cromossomo, é uma possível solução para o problema que se quer resolver. Em um AG padrão, as soluções são codificadas como vetores binários de tamanho fixo, nos quais cada bit é chamado de gene e cuja interpretação varia de acordo com o problema. Cada indivíduo faz parte de uma população, que contém diversas outras soluções. A população inicial é definida de forma aleatória. Ela evolui por um determinado número de gerações, de forma que os indivíduos tendam a se aproximar da solução ótima. Para guiar esta evolução, uma função que calcula o *fitness* de cada indivíduo é definida. O *fitness* é uma medida da qualidade de uma solução, e é usado para selecionar os indivíduos sobre os quais serão aplicados certos operadores genéticos, que podem modificar características de uma solução ou combinar partes de duas soluções. Os indivíduos gerados por estes operadores não possuem necessariamente *fitness* maiores que os dos seus "pais", mas a pressão seletiva faz com que haja uma tendência nesta direção. Após a passagem de um certo número de gerações, se a solução ótima não foi encontrada, a melhor solução da população é tomada como resultado final.

Os operadores genéticos normalmente usados são a reprodução, a seleção elitista, o *crossover* e a mutação. A reprodução é simplesmente a escolha de alguns indivíduos da população que são replicados de uma geração para a outra. Os indivíduos são selecionados com base no seu *fitness*: quando mais apto é o indivíduo, maior a sua chance de sobrevivência. A seleção elitista consiste na manutenção do indivíduo com melhor *fitness* de cada geração na geração seguinte. Garantindo a permanência desta solução na população, obtém-se uma ascensão monotônica do *fitness* do melhor indivíduo através das gerações. O *crossover* consiste na combinação de características de dois indivíduos selecionados de forma aleatória da população. Assim como ocorre na reprodução, indivíduos com valores mais altos de *fitness* possuem maiores chances de serem escolhidos. Existem várias formas de se realizar o *crossover*, com resultados que variam de acordo com o problema. Uma mutação é a inversão de um bit de um indivíduo. Geralmente, a chance de ocorrer uma mutação é pequena, pois do contrário os resultados do algoritmo poderiam se tornar completamente aleatórios.

Uma diferença fundamental entre os algoritmos genéticos e outras heurísticas é o seu paralelismo [Baluja 1994]: enquanto muitas heurísticas trabalham com um único ponto no espaço de busca, em um AG cada indivíduo da população representa um ponto.

Portanto, para que o AG faça uma busca exaustiva, deve-se tomar precauções para garantir que haja uma certa diversidade genética. Quando esta diversidade é perdida, o algoritmo pode ficar preso em um grupo de soluções sub-ótimas. Dois dos mecanismos acima descritos são usados com este propósito. Um deles é a seleção probabilística dos indivíduos usados no *crossover*. Indivíduos com *fitness* baixo possuem chances menores de serem escolhidos, mas ainda assim podem ser usados e suas características podem se perpetuar em gerações futuras. O outro é a mutação, que introduz variações aleatórias que ajudam a população a escapar de ótimos locais.

Quando se trabalha com algoritmos genéticos, diversas questões precisam ser tratadas, tais como a correta definição da função que calcula o *fitness*, a certificação de que informações importantes não são perdidas por conta de escolhas aleatórias e a representação eficiente do problema. O tamanho da população também é importante: populações maiores aumentam as chances de convergência, mas também aumentam o custo computacional do algoritmo. Portanto, para que se resolva o diagnóstico em nível de sistema através do uso de um AG, certas definições devem ser feitas. As definições aqui apresentadas para a represetnação do problema e para o cálculo do *fitness* foram propostas em [Elhadef e Ayeb 2000].

2.1. Representação do Problema

No AG considerado por este trabalho, um cromossomo representa uma possível situação de falha do sistema. Cada cromossomo v é representado por uma *string* de bits, com cada gene representando o estado de uma unidade do sistema (falha ou sem falha). Assim, um sistema com n unidades é representado por uma *string* de comprimento n. No exemplo abaixo, o conjunto de unidades falhas é $F = \{u_1, u_4, u_8\}$:

(10010001)

Para o problema do diagnóstico em nível de sistema, certos cromossomos são considerados ilegais. Estes são aqueles que não satisfazem as condições necessárias de um sistema t-diagnosticável. Por exemplo, os seguintes cromossomos são ilegais para um sistema t-diagnosticável com $t = 4$:

(000000000) e (111001110)

No primeiro caso, o cromossomo é ilegal porque em uma situação de falha pelo menos uma das unidades deve estar falha, e o segundo porque o número de unidades falhas é maior que t.

2.2. Cálculo do *Fitness*

Para este problema, uma boa medida do *fitness* de uma solução em potencial é a probabilidade da mesma estar correta. Para um cromossomo ilegal, esta probabilidade é nula. Para os demais, esta probabilidade pode ser obtida gerando-se uma síndrome S, compatível com a solução v, e comparando-se esta síndrome com a síndrome S^*, obtida nos testes realizados pelas unidades do sistema. A geração da síndrome S deve ser feita de tal forma que a mesma seja idêntica a S^* se e somente se v representa a solução ótima, ou seja, a situação real de falhas do sistema.

Sejam $F(v)$ o conjunto de unidades falhas de um cromossomo v (aquelas unidades cujos genes possuem valor 1); $v[i]$ o i-ésimo gene de v e $W_S(u_i, u_j)$ o resultado do teste (u_i, u_j) em uma síndrome S. A síndrome S compatível com v é gerada segundo as regras:

1) para todo $u_i \in F(v)$ e todo $u_j \in \Gamma(u_i)$, $W_S(u_i, u_j) = W_{S*}(u_i, u_j)$; e

2) para todo $u_i \in U - F(v)$ e todo $u_j \in \Gamma(u_i)$, $W_S(u_i, u_j) = w_{ij}$.

A regra 1 diz que se uma unidade é falha em v, em S os seus testes possuem os mesmos resultados que na síndrome "real" S^*. A regra 2 diz que se uma unidade u_i não é falha em v, em S o resultado do teste $W_S(u_i, u_j)$ é 1 se u_j é falha em v, ou 0 do contrário. Desta forma, se o conjunto $F(v)$ corresponde ao conjunto de falhas F do sistema, S e S^* são idênticos e o cromossomo v é consistente com o estado real do sistema.

Por exemplo, supondo um sistema com $n = 3$ e $t = 1$ tal que a síndrome S^* tem $W_{S*}(u_1, u_2) = 1$, $W_{S*}(u_2, u_3) = 0$ e $W_{S*}(u_3, u_1) = 1$. A síndrome S gerada pelo cromossomo (001) tem $W_S(u_1, u_2) = 0$, $W_S(u_2, u_3) = 1$ e $W_S(u_3, u_1) = 1$. O cromossomo (100) gera a síndrome S com $W_S(u_1, u_2) = 1$, $W_S(u_2, u_3) = 0$ e $W_S(u_3, u_1) = 1$. Neste caso, S e S^* são idênticos, e o cromossomo (100) corresponde à solução ótima.

A função de *fitness* usada neste trabalho considera que cada gene de um cromossomo possui seu próprio *fitness*. O *fitness* do cromossomo é a soma normalizada dos *fitness* dos seus genes. A função f que calcula o *fitness* de $v[i]$, o i-ésimo gene de v, é definida como:

$$f(v[i]) = \frac{f_{out}(v[i]) + f_{in}(v[i])}{2}, \text{ onde}$$

$$f_{in}(v[i]) = \frac{\left|S^{-1}(u_i) \cap (S^*)^{-1}(u_i)\right|}{d_{in}(u_i)} \text{ e}$$

$$f_{out}(v[i]) = \begin{cases} 1, \text{se } d_{out}(u_i) = 0, \\ \dfrac{\left|S(u_i) \cap S^*(u_i)\right|}{d_{out}(u_i)} \text{ do contrário.} \end{cases}$$

$f_{in}(v[i])$ computa o número normalizado de testes efetuados sobre a unidade u_i cujos resultados na síndrome gerada S são idênticos aos seus correspondentes na síndrome S^*. Da mesma forma, $f_{out}(v[i])$ computa o número normalizado de testes efetuados pela unidade u_i com resultados iguais em S e S^*. Se u_i não efetua testes, $f_{out}(v[i]) = 1$. Assim, o *fitness* do i-ésimo bit considera a unidade u_i como testadora e testada. $f(v[i])$ pode ser vista como a probabilidade do estado de u_i em v estar correto.

O *fitness FT* do indivíduo v é a soma normalizada dos *fitness* dos seus bits, ou seja:

$$FT(v) = \frac{\sum_{i=1}^{n} f(v[i])}{n}$$

Se o cromossomo v corresponde à solução ótima, ou seja, se v representa o estado real do sistema, então $FT(v) = 1$. Portanto, a condição de término do algoritmo é a obtenção de um indivíduo cujo *fitness* é igual a 1.

No exemplo anterior, se $v = (001)$, $f(v[1]) = 0.5$, $f(v[2]) = 0$ e $f(v[3]) = 0.5$, então $FT(v) = 0.33$. Se $v = (100)$, o *fitness* de todos os cromossomos é igual a 1, e $FT(v) = 1$.

2.3. População Inicial e Operadores Genéticos

Para este AG simples, a população inicial é definida de forma completamente aleatória. Indivíduos ilegais podem ser gerados, mas seu *fitness* será igual a 0. Esta é uma abordagem ingênua, que pode ser melhorada de outras formas, como é mostrado na seção 3.1.

Os operadores genéticos utilizados também são simples. Em uma população de tamanho *p*, são selecionados *p* indivíduos que serão usados para gerar a nova população. Estes indivíduos formam o conjunto de reprodutores, e são escolhidos através de um método probabilístico conhecido como técnica da roleta. Este método define uma chance de escolha para cada indivíduo a partir do seu *fitness*. Como o sorteio é feito sempre sobre toda a população atual, cada indivíduo pode ser selecionado mais do que uma vez. O conjunto de reprodutores é então usado para gerar a nova população: uma proporção deles é reproduzida sem alterações e uma outra parte é usada na operação de *crossover*. A chance de ocorrer um *crossover* é dada por uma taxa que permanece constante durante a execução do algoritmo. Para o *crossover*, toma-se um par de cromossomos de tamanho *n* e um valor aleatório *pos* entre 1 e *n*, que define a posição onde os cromossomos são "rompidos", gerando um novo par de cromossomos. Por exemplo, tendo *pos* = 3, os cromossomos *v1* = (011|001000) e *v2* = (101|110000) são substituidos por *v1*' = (011|110000) e *v2*' = (101|001000). Por fim, cada bit de cada indivíduo da nova população tem uma pequena chance de sofrer uma mutação, que inverte o seu valor. O *crossover* e a mutação podem gerar indivíduos ilegais. Estes cromossomos terão *fitness* igual a 0.

3. Outros Algoritmos Evolutivos para Diagnóstico em Nível de Sistema

Na sessão 2, foi apresentado um algoritmo genético simples que pode ser usado para o diagnóstico em nível de sistema. Este AG pode ser melhorado através de técnicas que tiram proveito do conhecimento do domínio do problema, criando um AG especializado. Pode-se também fazer uso de outros algoritmos evolutivos, como o *Population-Based Incremental Learning* - PBIL [Baluja 1994, Baluja e Caruana 1995], e o AGs compacto [Harik, Lobo e Goldberg 1998]. Esta seção mostra como estes algoritmos podem ser usados para realizar o diagnóstico. Os indivíduos gerados pelos algoritmos descritos nesta seção possuem a mesma representação daqueles gerados pelo AG simples, e são avaliados pela mesma função de *fitness*. O AG especializado é baseado naquele apresentado em [Elhadef e Ayeb 2000], com algumas pequenas melhorias. As implementações do PBIL e do AG compacto foram criadas especificamente para este trabalho.

3.1. Um AG Especializado para Diagnóstico em Nível de Sistema

Abaixo, são apresentadas algumas técnicas que melhoram aspectos do AG usado para o diagnóstico em nível de sistema.

3.1.1. População Inicial

O AG simples cria uma população inicial completamente aleatória. Esta população inicial pode ser gerada de maneira mais eficiente, diminuindo o número total de gerações necessárias para a obtenção da solução ótima. O AG especializado melhora a inicialização aleatória usando o seguinte algoritmo para gerar cada indivíduo da população inicial.

1. Uma unidade u_i é sorteada e definida como sendo livre de falhas.

2. Unidades testadas por u_i têm o seu estado definido pelos resultados dos testes feitos por u_i. Estes resultados são obtidos na síndrome do sistema, S^*.
3. Unidades que testem u_i em S^* como sendo falha são definidas como falhas.
4. Se o número de unidades falhas for maior que t ou todas as unidades tiverem o seu estado definido como sendo sem falha, u_i não pode estar sem falhas. O cromossomo é descartado e u_i não pode mais ser sorteado no passo 1 para gerar outros indivíduos.
5. Se nem todas as unidades tiverem o seu estado definido e não há unidades falhas no sistema, uma das unidades com estado indefinido é definida como sendo falha, e as demais são consideradas sem falha. Do contrário, todas as unidades com estado indefinido são consideradas sem falha.

Esta estratégia é capaz de evitar a geração de indivíduos ilegais.

3.1.2. Mutação

No AG simples, todos os bits de um cromossomo possuem uma chance igual de sofrerem mutação. O algoritmo especializado modifica a escolha dos bits que são invertidos, usando uma estratégia adaptativa. A função de *fitness* usada define um valor de *fitness* para cada bit do cromossomo. O AG especializado pode usar este valor para determinar quais bits sofrem mutação: se o *fitness* do bit for inferior à taxa de mutação, o mesmo é invertido. Se não houverem bits para se inverter por este critério, aquele com menor *fitness* tem uma chance igual à taxa de mutação de ser invertido.

Para evitar a geração de indivíduos ilegais, são usadas duas estratégias. Sejam v um cromossomo e i o bit que será invertido. Se $|F(v)| = 1$ e $v[i] = 1$, i é a única unidade falha do cromossomo, e a inversão do i-ésimo bit irá gerar um cromossomo que não possui unidades falhas. Se isto ocorrer, o j-ésimo bit, correspondente ao testador de i com menor *fitness*, também é invertido. Se $|F(v)| = t$ e $v[i] = 0$, a inversão do i-ésimo bit irá gerar um cromossomo com mais que t unidades falhas. Neste caso, o estado da unidade falha com menor *fitness* é invertido. Desta forma, pode-se inverter o bit i sem que se gere um indivíduo ilegal.

3.1.3. Crossover

A geração de cromossomos ilegais através do *crossover* é resolvida da seguinte forma: se o *crossover* de dois cromossomos gerar um indivíduo ilegal, a operação é ignorada e o indivíduo é gerado como uma cópia do seu primeiro pai. Nos testes feitos em [Elhadef e Ayeb 2000], os melhores resultados foram obtidos quando as menores taxas de *crossover* foram usadas. Desta forma, pode-se questionar a importância do *crossover* para a evolução neste problema. Por este motivo, a implementação feita para este trabalho prevê também a possibilidade de uma taxa de *crossover* nula.

3.2. PBIL – *Population-Based Incremental Learning*

A população de um AG guarda informações relativas aos pontos do espaço de busca já visitados pelo algoritmo. Os operadores de seleção e *crossover* podem ser vistos como maneiras de fazer uso desta informação. A compreensão do papel e do funcionamento da população e dos operadores genéticos dos AGs possibilitou a criação de uma outra classe de algoritmos, que substitui a população, o *crossover* e a seleção por outra técnica: a distribuição probabilística dos genes. O PBIL (*Population-Based Incremental Learning*)

[Baluja 1994, Baluja e Caruana 1995], apresentado nesta seção, é um algoritmo evolutivo baseado neste conceito.

O PBIL cria um vetor de valores reais, que reprentam a probabilidade de cada gene de um cromossomo ter valor igual a 1. Estas probabilidades são iniciadas com um valor de 0.5 ou 50%, ou seja, cada gene tem uma chance igual de assumir um valor 0 ou 1. Com o passar das gerações, este vetor é atualizado de forma a representar indivíduos com *fitness* altos. A cada geração, uma nova população é criada a partir do vetor de probabilidades.

O PBIL é caracterizado por três parâmetros. O primeiro é o tamanho da população gerada a cada iteração do algoritmo. O segundo é a taxa de aprendizado, que diz o quanto cada bit do vetor de probabilidades é deslocado na direção das boas soluções a cada geração. O terceiro define quantos indivíduos são usados na atualização do vetor de probabilidades a cada geração. A escolha destes indivíduos pode ser estendida, de forma que o algoritmo aprenda a partir de exemplos negativos. Assim, além de ser atualizado na direção dos x melhores exemplos, o vetor de probabilidades pode ser atualizado na direção contrária aos y piores exemplos. O PBIL também pode fazer uso da mutação: uma mutação acrescenta uma pequena variação positiva ou negativa a um bit no vetor de probabilidades.

Quando se usa o PBIL para o diagnóstico em nível de sistema, o vetor de probabilidades pode ser gerado de uma forma otimizada, que exclui logo de início certos indivíduos ilegais. Isto é feito através do seguinte algoritmo, executado para cada posição i no vetor de probabilidades.

1. A unidade $u_i \in U$ é tomada como sendo livre de falhas.
2. Unidades testadas por u_i têm o seu estado definido pelos resultados dos testes feitos por u_i. Estes resultados são obtidos na síndrome do sistema, S^*.
3. Unidades que testem u_i em S^* como sendo falha são definidas como falhas.
4. Se o número de unidades falhas for maior que t ou todas as unidades tiverem o seu estado definido como sendo sem falha, a unidade u_i não pode estar sem falhas, ou seja, é certo que está falha, e o valor da posição i do vetor de probabilidades é definido como sendo 1. Do contrário, este valor é igual a 0.5.

Para evitar a geração de cromossomos ilegais a partir de um vetor de probabilidades, usa-se a seguinte estratégia: após a criação do indivíduo, se este não possui unidades falhas, a unidade cujo bit possui menor *fitness* é definida como sendo falha. Se exitirem mais que t unidades falhas, aquelas cujos bits possuirem os menores valores de *fitness* são consideradas como sendo sem falhas, até que o número de unidades falhas seja menor ou igual a t.

3.3. AG Compacto

Um outro algoritmo que, assim como o PBIL, substitui os operadores de seleção e *crossover* pela distribuição probabilística dos genes, é o Algoritmo Genético Compacto [Harik, Lobo e Goldberg 1998], ou AGc. Este algoritmo reduz de forma significativa os requisitos de memória pois, ao contrário dos outros algoritmos até aqui considerados, não tem a necessidade de manter uma população.

Assim como o PBIL, o AGc faz uso de um vetor de valores reais que representam probabilidades. No início do algoritmo, todas as posições possuem o valor 0.5, ou 50%.

Este vetor é usado para gerar de forma probabilística 2 indivíduos. Estes indivíduos são então comparados, e o vetor de probabilidades é atualizado na direção do "vencedor", ou seja, daquele que possuir *fitness* mais alto. Cada bit é atualizado independentemente: se o vencedor possui um bit em 1 e o perdedor em 0, o bit correspondente é incrementado no vetor de probabilidades. Da mesma forma, se o vencedor possui um bit em 0 e o perdedor em 1, o bit é decrementado no vetor de probabilidades. Se um bit possui valor igual no vencedor e no perdedor, seu valor não é alterado no vetor de probabilidades. Os bits do vetor de probabilidades são incrementados ou decrementados de um valor *E*. Estudos mostram que um AGc com $E = 1/n$ se comporta de forma similar a um AG simples com população de tamanho n [Harik, Lobo e Goldberg 1998 e Harik 1999].

O AGc pode ser usado para realizar o diagnóstico em nível de sistema de forma similar ao PBIL. Os procedimentos para inicialização do vetor de probabilidades e para evitar a criação de indivíduos ilegais podem ser repetidos sem alterações.

4. Experimentos

Os algoritmos apresentados nas seções 2 e 3 foram implementados para ter o seu desempenho comparado quando usados para resolver o diagnóstico em nível de sistema. Foram avaliadas algumas variações dos algoritmos, resultando em um total de seis diferentes configurações: AG simples, AG especializado com e sem *crossover*, PBIL com e sem o uso de exemplos negativos e AG compacto.

4.1. Descrição dos Experimentos

Para os experimentos foram usados grafos de teste para sistemas t-diagnosticáveis com diferentes tamanhos. Estes grafos de teste foram gerados com cada unidade $u_i \in U$ testando as unidades $u_{(i+1) \bmod n} .. u_{(i+t) \bmod n}$. Como $n \geq 2t + 1$, cada unidade será testada por t outras unidades e nenhum par de unidades se testará mutuamente. A cada teste, um conjunto de unidades falhas foi definido selecionando-se aleatoriamente n_f unidades como sendo falhas, tal que $n_f \leq t$. A partir do grafo de testes e do conjunto de unidades falhas, a síndrome S^* do sistema foi gerada, com os resultados dos testes executados pelas unidades falhas sendo definido aleatoriamente.

Para evitar possíveis problemas com arredondamentos e melhorar o desempenho dos algoritmos, a função para cálculo do *fitness* foi modificada de forma a não fazer normalizações, ou seja, as divisões foram removidas. Desta forma, pode-se trabalhar apenas com valores inteiros, e o *fitness* varia de 0 a $2nt$, ao invés de 0 a 1.

Os algoritmos foram experimentados com diferentes configurações. Foram escolhidas aquelas configurações que apresentaram o melhor desempenho médio em 10 execuções com $n = 81$ e $t = 40$ (o caso mais complexo considerado nestes experimentos). As configurações escolhidas foram:

- AG simples: população = 7, taxa de *crossover* = 0.1 e taxa de mutação = 0.01.
- AG especializado com *crossover*: população = 7, taxa de *crossover* = 0.05 e taxa de mutação = 0.01.
- AG especializado sem *crossover*: população = 7 e taxa de mutação = 0.01.
- PBIL sem uso de exemplos negativos: população = 7, taxa de atualização = 0.2, taxa de mutação = 0.08 e número de indivíduos usados na atualização = 1.

- PBIL com uso de exemplos negativos: população = 7, taxa de atualização positiva = 0.2, taxa de atualização negativa = 0.1, taxa de mutação = 0.08 e número de indivíduos usados na autalização = 1.
- AG compacto: taxa de atualização = 0.15 e taxa de mutação = 0.08.

Seja (*n, t*) um grafo de testes para um sistema *t*-diagnosticável de *n* unidades. Foram definidos os seguintes grafos de teste, com tamanho e número de falhas bastante variado: (81, 5), (81, 10), (81, 10), (81, 40), (41, 5), (41, 10), (41, 20), (21, 5), (21, 10), (11, 5) e (8, 3). Cada algoritmo foi executado 100 vezes para cada grafo de testes, e o seu desempenho médio foi computado.

4.2. Resultados dos Experimentos

As tabelas 1 a 4 apresentam os resultados obtidos nos experimentos realizados. A solução obtida corresponde àquela com melhor *fitness* após 500 gerações, ou à solução ótima, caso a mesma tenha sido obtida.

Tabela 1: Número de gerações para a obtenção da solução.

	AG Simples	AG Espec. c/ X-Over	AG Espec. s/ X-over	PBIL s/ ex. neg.	PBIL c/ ex. neg.	AGc
n = 8, t = 3	99.1	2.4	16.81	18.04	18.19	68.82
n = 11, t = 5	66.33	2.01	8.37	19.7	30.06	75.81
n = 21, t = 5	354.66	9.96	14.63	33.11	41.53	41.41
n = 21, t = 10	88.12	5.06	9.85	56.16	47.88	165.93
n = 41, t = 5	500	6.91	8.68	38.44	48.88	410
n = 41, t = 10	475.32	7.33	7.14	55.42	70.8	450
n = 41, t = 20	177.83	14.09	11.16	88.16	80.87	307.97
n = 81, t = 5	500	0.86	0.84	35.14	71.26	390
n = 81, t = 10	500	0.88	0.89	67.59	104.29	450
n = 81, t = 20	500	0.95	0.86	126.38	124.02	444
n = 81, t = 40	421,62	0.85	0.82	164.76	142.2	500

Tabela 2: *Fitness* médio da solução após no máximo 500 gerações.

	Máx. Possível	AG Simples	AG Espec. c/ X-Over	AG Espec. s/ X-over	PBIL s/ ex. neg.	PBIL c/ ex. neg.	AGc
n = 8, t = 3	48	47.34	48	47.94	48	47.98	48
n = 11, t = 5	110	110	110	109.76	110	110	110
n = 21, t = 5	210	94.5	210	210	210	210	201.12
n = 21, t = 10	420	420	420	420	420	420	420
n = 41, t = 5	410	0	410	410	410	410	398.46
n = 41, t = 10	820	113.38	820	820	820	820	763.38
n = 41, t = 20	1640	1639.38	1640	1640	1640	1640	1634.18
n = 81, t = 5	810	0	810	810	810	810	797.86
n = 81, t = 10	1620	0	1620	1620	1620	1620	1573
n = 81, t = 20	3240	0	3240	3240	3240	3240	3030.89
n = 81, t = 40	6480	6457.18	6480	6480	6480	6480	6132.8

Tabela 3: Porcentagem das rodadas que obtiveram a solução ótima.

	AG Simples	AG Espec. c/ X-Over	AG Espec. s/ X-over	PBIL s/ ex. neg.	PBIL c/ ex. neg.	AGc
n = 8, t = 3	92%	100%	97%	100%	99%	100%
n = 11, t = 5	100%	100%	99%	100%	100%	100%
n = 21, t = 5	45%	100%	100%	100%	100%	30%
n = 21, t = 10	100%	100%	100%	100%	100%	100%
n = 41, t = 5	0%	100%	100%	100%	100%	18%
n = 41, t = 10	9%	100%	100%	100%	100%	10%
n = 41, t = 20	99%	100%	100%	100%	100%	91%
n = 81, t = 5	0%	100%	100%	100%	100%	22%
n = 81, t = 10	0%	100%	100%	100%	100%	10%
n = 81, t = 20	0%	100%	100%	100%	100%	11%
n = 81, t = 40	62%	100%	100%	100%	100%	0%

Tabela 4: Tempo médio para a obtenção da solução (em segundos – 0 indica tempo inferior a 0.01 s).

	AG Simples	AG Espec. c/ X-Over	AG Espec. s/ X-over	PBIL s/ ex. neg.	PBIL c/ ex. neg.	AGc
n = 8, t = 3	0	0	0	0	0	0
n = 11, t = 5	0	0	0	0	0	0
n = 21, t = 5	0	0	0	0	0	0.25
n = 21, t = 10	0	0	0	0	0	0
n = 41, t = 5	0	0	0	0.01	0.04	5.52
n = 41, t = 10	0	0	0	0.12	0.29	6.76
n = 41, t = 20	0.71	0	0	0.96	0.93	2.23
n = 81, t = 5	0	0	0	0.76	2.06	43.58
n = 81, t = 10	0	0	0	2.48	4.87	81.05
n = 81, t = 20	0	0.02	0	10.15	12.45	116.2
n = 81, t = 40	26.31	0.85	0.21	16.3	15.74	50.16

Ao se analisar os resultados obtidos nos testes, é possível chegar a diversas conclusões sobre o desempenho dos algoritmos. O AG Simples é bastante eficiente no que diz respeito ao tempo de execução. Porém, os resultados obtidos foram os piores entre todos os algoritmos. Quando há uma grande diferença entre os valores de *n* e *t*, o algoritmo foi, na maior parte dos casos, incapaz de encontrar sequer uma solução que se aproximasse da solução ótima. Isto provavelmente ocorre porque nestes casos o número de indivíduos inválidos que é gerado é muito grande, o que dificulta a convergência dos resultados.

O AG Compacto teve os piores tempos de execução. Nos testes mais complexos, o número de gerações para a obtenção da solução foi bastante alto, mas diferente do AG Simples, o *fitness* médio das soluções se aproximou da solução ótima. Portanto, pode-se concluir que o AG Compacto teve uma convergência razoável, mas teve dificuldades para encontrar a solução ótima propriamente dita.

Na quase totalidade dos casos, o PBIL conseguiu encontrar a solução ótima após um certo número de gerações. Seu desempenho foi superado apenas pelo AG especializado. Pode-se verificar que, para este problema, o uso de exemplos negativos no aprendizado foi prejudicial, visto que na maioria dos testes a complexidade adicional não foi compensada por uma convergência mais rápida. De fato, no geral, a conversão foi mais lenta quando foram usados exemplos negativos.

O AG Especializado foi aquele que apresentou o melhor desempenho entre os algoritmos testados. Além de ter encontrado a solução ótima na grande maioria dos casos, o AG Especializado o fez com um número médio de gerações muito baixo. De fato, isto demonstra a importância de um bom algoritmo para a inicialização da população, já que em muitos casos a solução ótima foi encontrada na população inicial, ou após poucas gerações. Pode-se dizer que o AG Especializado sem *crossover* apresentou desempenho similar ao com *crossover*. Nos exemplos mais complexos, o número de gerações necessário para a obtenção da solução ótima foi bastante parecido para as duas abordagens, e a retirada do *crossover* fez o tempo médio de execução do algoritmo ser reduzido de forma discreta. Isto leva a crer que existe uma boa possibilidade de o *crossover* não ser o principal responsável pela convergência neste problema. Porém, visto que em grande parte dos exemplos a população inicial já continha a solução ótima, ou uma solução próxima a ela, o número de gerações dos testes foi bastante reduzido. Por este motivo, não é possível de se analisar de forma conclusiva o papel do *crossover* na convergência para este problema.

Os algoritmos testados podem ser ordenados de acordo com o seu desempenho nos experimentos, do melhor para o pior, da seguinte forma: AG Especializado sem

crossover; AG Especializado com *crossover*, PBIL sem exemplos negativos, PBIL com exemplos negativos, AG Compacto e AG Simples.

5. Conclusões e Trabalhos Futuros

Neste trabalho, foram apresentados algoritmos evolutivos que podem ser usados para realizar o diagnóstico em nível de sistema. Estes algoritmos foram implementados e testados, de forma a terem comparados os seus desempenhos. A partir dos resultados dos testes, pode-se dizer que a complexidade adicional de algoritmos mais sofisticados muitas vezes é compensada por uma convergência rápida dos resultados para a solução ótima. Nos casos mais complexos, esta compensação fica bastante evidente, com as abordagens mais simples (AG simples, AG compacto) não conseguindo encontrar a solução ótima ou mesmo convergir para uma boa solução, enquanto a abordagem mais complexa (AG Especializado) encontra rapidamente a solução ótima.

Entre trabalhos futuros, pode-se destacar a execução de mais experimentos que usem configurações diferentes para os algoritmos aqui apresentados ou outros algoritmos evolutivos, tais como o AG compacto extendido [Harik 1999] e o BOA [Peikan, Goldberg e Cantú-Paz 1999]. Também pode-se realizar trabalho semelhante com a aplicação de algoritmos evolutivos para o diagnóstico distribuído, no qual cada unidade do sistema realiza o diagnóstico.

Referências

Baluja, S. (1994) "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning", Tech. Rep. No. CMU-CS-94-163, Pittsburgh, PA, Carnegie Mellon University.

Baluja, S. & Caruana, R. (1995), "Removing the Genetics from the Standard Genetic Algorithm", Proceedings of ML-95, Twelfth International Conference on Machine Learning, A. Prieditis and S. Russel (Eds.), 1995, Morgan Kaufmann, pp. 38-46.

De Jong, K. (1975), "An Analysis of the Behavior of a Class of Genetic Adaptive Systems", University of Michigan, Tese de Ph.D.

Elhadef, M. & Ayeb, B. (2000), "Efficient Fault Identification in Diagnosable Systems: An Evolutionary Approach". University of Sherbrooke, Quebec, 2000.

Harik, G. (1999), "Linkage Learning via Probabilistic Modeling in the ECGA", IlliGAL Technical Report 99010, Urbana, IL: University off Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Harik, G.R.; Lobo, F.G. & Goldberg, D.E. (1998), "The Compact Genetic Algorithm", In of Electrical, I., & Engineers, E. (Eds.), Proceedings of 1998 IEEE International Conference on Evolutionary Computation (pp. 523-528).

Peikan, M.; Goldberg, D.E. & Cantú-Paz, E. (1999), "BOA: The Bayesian Optimization Algorithm", Preceedings Genetic and Evolutionary Computation Conference 1999.

Preparata, F.P.; Metze, G. & Chien, R.T (1967), "On the Connection Assignment Problem of Diagnosable Systems", IEEE Trans. on Eletron. Comput., 16.

Masson, G.; Blough, D. & Sullivan, G (1996), "System Diagnosis", in Fault-Tolerant Computer System Design, ed. D. K. Pradhan, Prentice-Hall.