Sistemas imunológicos artificiais no teste de agentes inteligentes

Sávio Mota Carneiro [1]
Thiago Allisson Ribeiro da Silva [2]
Ricardo de Andrade Lira Rabêlo [2]
Francisca Raquel de V. Silveira [3]
Gustavo Augusto Lima de Campos [3]

Resumo: Agentes inteligentes consistem em uma tecnologia da computação promissora para o desenvolvimento de sistemas distribuídos complexos. Apesar dos referenciais teóricos disponíveis para orientar o projetista desses agentes, existem poucas técnicas de testes propostas para validar esses sistemas. Sabe-se que essa validação depende dos casos de teste selecionados, os quais devem providenciar informações a respeito dos componentes na estrutura do agente que estão com desempenho insatisfatório. Este artigo apresenta a aplicação de sistemas imunológicos artificiais (SIA), por meio do algoritmo de seleção clonal (Clonalg), para o problema de otimização de seleção de casos de teste para o teste de sistemas computacionais baseados em agentes inteligentes. A fim de validar o uso do Clonalg, foram realizadas comparações entre as técnicas de algoritmos genéticos (AG) e algoritmos de otimização por colônia de formigas (ACO). Nos experimentos com a abordagem testando agentes inteligentes com diferentes tipos de arquitetura em ambientes parcialmente e totalmente observáveis, a abordagem selecionou um conjunto de casos de teste satisfatório em termos das informações geradas sobre o desempenho irregular do agente. Com base nesse resultado, a abordagem possibilita a identificação dos episódios problemáticos, permitindo ao projetista realizar mudanças objetivas na estrutura interna do agente de forma a melhorar seu desempenho.

***Palavras-chave*:** Agentes inteligentes. Seleção de casos de teste. Sistemas imunológicos artificiais.

Abstract: *Intelligent agents consist of a computing technology promising to develop complex distributed systems. Despite the theoretical frameworks available to guide the designer of these agents, there are few techniques proposed tests to validate these systems. It is known that this validation depends on the selected test cases, which should provide information on the components in the structure of the agent that are underperforming. This paper presents the application of artificial immune systems (AIS), through clonal selection algorithm (Clonalg) for the optimization problem of selecting test cases for testing computer systems based on intelligent agents. In order to validate the use of Clonalg comparisons between techniques of aenetic algorithms (GA) and algorithms for ant colony optimization (ACO) were performed. In the experiments with the approach testing intelligent agents with different architecture types in partially and fully observable environments, the approach selected a set of test cases satisfying in terms of information generated on the uneven performance of the agent. From this result, the approach enables the identification of problematic episodes, allowing the designer to make objective changes in the internal structure of the agent in order to improve its performance.*

***Keywords*:** *Artificial immune systems. Intelligent agents. Selection of test cases.*

[1]Instituto Federal do Piauí, Teresina-PI, Brasil.
{saviod2@gmail.com}
[2]Universidade Federal do Piauí, Teresina-PI, Brasil.
{allissonribeiro02@gmail.com}, {ricardor_usp@yahoo.com.br}
[3]Universidade Estadual do Ceará, Fortaleza-CE, Brasil.
{raquel_silveira@ifce.edu.br}, {gustavo@larces.uece.br}

http://dx.doi.org/10.5335/rbca.2015.4540

1 Introdução

Agentes inteligentes são sistemas capazes de, autonomamente, perceber e modificar o ambiente no qual estão inseridos, buscando alcançar o melhor resultado, ou, quando existe incerteza, o melhor resultado esperado [1]. A ideia inicial de agentes de software foi proposta em meados de 1950 por John MacCarthy [2], e, desde então, foi possível observar o crescimento da sua aplicação nas mais diversas áreas, tais como Smart Grid [3], comércio eletrônico [4], jogos [5], sistemas de recomendações [6], sistemas de detecção de intrusão [7], controle de tráfego[8], controle de tráfego aéreo [9], educação a distância [10], monitoramento de pacientes [11], robótica [12], entre outras. Mesmo com toda essa ampla aplicabilidade, testar a eficiência de sistemas baseados em agentes não tem sido uma tarefa fácil. Um dos motivos possíveis para a incipiência dessa área é a difícil aplicação de técnicas de testes de software tradicionais que sejam capazes de garantir a confiabilidade de sistemas baseados em agentes. A dificuldade existente na realização de testes de agentes se dá pelo fato de estes serem projetados para trabalharem de forma distribuída, autônoma e deliberativa, o que pode criar o efeito de irreprodutividade [13], ou seja, não há garantia de que duas execuções do sistema levem a estados iguais, mesmo se as entradas utilizadas forem idênticas. Como consequência, procurar um erro específico pode ser difícil, já que não é possível reproduzi-lo a cada nova execução [14].

Na literatura de engenharia de software orientada a agentes, as pesquisas têm se concentrado no estudo de como os agentes interagem, no desenvolvimento de arquiteturas e de protocolos para os sistemas de agentes enquanto um processo de teste estruturado ainda está ausente [15]. Para se obter um panorama sobre a área de pesquisa de teste de agentes de software, alguns trabalhos relacionados são apresentados a seguir:

O autor, em [16], apresenta um agente especial testador, usado para testar os agentes individualmente ou na comunidade à qual pertencem, de modo que possa garantir a confiança dos sistemas baseados em agentes. O agente de teste pode ler a especificação de cada agente e produzir casos de teste para executar os agentes individualmente e em conjunto, avaliando o desempenho dos agentes por meio da comunicação estabelecida entre eles. O trabalho descrito em [17] propõe um framework para testes de sistemas multiagentes. Esse framework facilita a derivação de casos de teste através da geração semiautomática de esqueletos de teste a partir de diagramas de análise de objetivos. Os autores usam a combinação de testes evolucionários [18] e mutação [19], para a geração de casos de testes executados pelo agente testador, no intuito de verificar o uso do Multi-Agents Systems (MAS) em um grande número de diferentes cenários e condições. O autor, em [13], descreve uma metodologia de teste orientada a objetivos que realiza a derivação de casos de testes baseada na análise de requisitos e nos artefatos de design. Essa metodologia fornece uma forma sistemática de gerar casos de testes para a modelagem de artefatos durante o processo de desenvolvimento. Os casos de teste são gerados automaticamente e evoluem continuamente guiados pela mutação e função de qualidade, a fim de detectar erros desde o início do processo de desenvolvimento. Uma abordagem evolucionária para a realização dos testes de agentes autônomos também é adotada por [15]. A metodologia representa os objetivos dos stakeholders como função de qualidade e faz uso dos algoritmos genéticos para gerar uma variedade de casos de teste com alto nível de dificuldade para o agente. A abordagem propõe aplicar um recrutamento dos melhores casos de teste para evoluir os agentes. Para cada agente, é dado um período experimental em que os testes com diferentes níveis de dificuldade são executados. Os agentes são recrutados apenas quando passam pelo critério de qualidade definido.

Este artigo apresenta a aplicação de sistemas imunológicos artificiais (SIA) [20], por meio do algoritmo de seleção clonal [21], para o problema de otimização de seleção de casos de teste para o teste de sistemas computacionais baseados em agentes inteligentes [22]. Os testes consistem na verificação dinâmica do comportamento de um programa com base em um conjunto finito de casos de teste. Na prática, há um número muito grande de casos de teste possíveis, portanto, faz-se necessária uma seleção adequada dos casos de teste, a fim de se verificar o comportamento do sistema. Dessa forma, o texto descreve a utilização de algoritmos de otimização baseados em Inteligência Computacional (IC) para a seleção dos melhores casos de teste para um sistema baseado em agentes inteligentes. As técnicas (algoritmos) de otimização baseada em inteligência computacional [23] têm as seguintes vantagens:

- não requerem propriedades especiais sobre o espaço de busca (função objetivo e restrições de igualdade/desigualdade), tais como convexidade, existência de derivadas, continuidade, unimodalidade;
- são baseadas em população. Dessa forma, os algoritmos de otimização evoluem uma população de soluções

candidatas, o que permite um compartilhamento de informações sobre o espaço de busca a fim de melhorar a convergência e a qualidade das soluções;

- incluem componentes estocásticos (aleatórios) para atualização das soluções entre as iterações. Portanto, a evolução da população segue regras de transição probabilísticas (estocásticas), o que reduz a dependência da solução inicial. Adicionalmente, as regras de transição estocásticas reduzem as chances de o processo de busca ficar estagnado em mínimos locais.

Especificamente, pretende-se mostrar a aplicação do algoritmo de seleção clonal na seleção dos melhores casos de teste, por meio da identificação de cenários que levam o agente testado a um baixo desempenho, fornecendo ao projetista informações necessárias para a melhoria do projeto do agente inteligente. De forma a validar os resultados obtidos pela aplicação do algoritmo de seleção clonal, esse artigo apresenta a aplicação de algoritmos Genéticos (GA) [24, 25] e algoritmos de otimização por colônia de formigas (ACO) [26, 27] para o problema em questão.

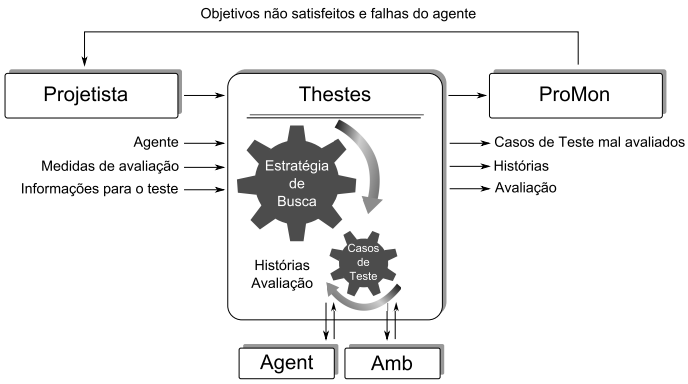
O restante do trabalho foi dividido em quatro seções. A abordagem utilizada é detalhada na seção 2. Em seguida, a seção 3 expõe os detalhes dos experimentos realizados. A seção 4 mostra os resultados obtidos. A seção 5 apresenta as conclusões, juntamente com os trabalhos futuros.

2 Abordagem utilizada

2.1 Visão geral

A abordagem utilizada fundamenta-se na noção de agentes inteligentes e utiliza casos de teste, gerados de acordo com os objetivos do agente testado, para avaliar o seu desempenho. Essa avaliação é realizada por meio da análise das interações do agente testado com seu ambiente (histórias). Cada história recebe uma pontuação, determinada pelas medidas de avaliação de desempenho, que são utilizadas para encontrar casos de teste e histórias correspondentes em que o agente não foi bem avaliado.

Figura 1: Visão geral da abordagem utilizada apresentando as interações entre o *Projetista*, *ProMon*, *Thestes*, *Agent* e *Amb*.



A Figura 1 representa uma visão geral da abordagem utilizada. Ao observá-la, é possível verificar a existência de cinco componentes: o agente *Thestes*, responsável pela geração de casos de testes por meio das estratégias de buscas empregadas (AG, ACO e SIA); o agente *ProMon*, responsável pela identificação de casos de testes mal avaliados, falhas do agente e pela identificação de objetivos não satisfeitos; o agente *Agent*, que é o agente a ser testado; o agente *Amb*, que simula o ambiente onde o *Agent* atua; e, por fim, o *Projetista*, pessoa responsável pela construção dos agentes *Agent* e *Amb*, além das definições das medidas de avaliação e informações dos parâmetros necessários para a realização dos testes.

A seguir, são apresentadas algumas definições necessárias para um melhor entendimento da abordagem descrita.

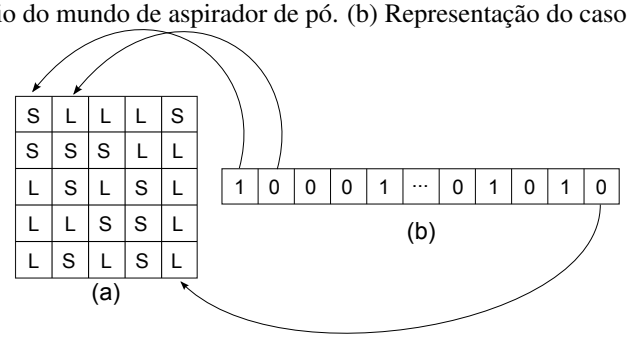
2.2 Casos de teste

Teste de software é uma atividade crucial do desenvolvimento do software, por meio da qual é possível a identificação de defeitos (bugs). O teste é considerado o último reduto no qual a qualidade pode ser avaliada e, mais pragmaticamente, erros podem ser descobertos [28].

Tradicionalmente, um caso de teste de software é definido como uma especificação mais detalhada que concentra quais informações serão empregadas durante os testes e quais os resultados esperados [29]. No contexto deste trabalho, considera-se um caso de teste a representação do cenário do ambiente em que o agente deverá atuar para alcançar seus objetivos. Dessa forma, um caso de teste não visa avaliar a corretude do agente desenvolvido, e sim fornecer ao projetista informações sobre cenários em que o agente teve um comportamento inadequado.

Por exemplo, vale considerar o teste de um agente aspirador de pó projetado para operar em um ambiente que consiste em uma área contendo *N* salas, que podem estar sujas (S) ou limpas (L), conforme Figura 2 (a). Existem várias alternativas possíveis para a representação desse tipo de ambiente. Neste trabalho, adotou-se um esquema de representação direta do ambiente em uma string formada por *N* bits, em que o estado da sala é representado por 1 quando o agente está em uma sala que contém sujeira, e 0 quando se encontrar em uma sala limpa, conforme Figura 2(b).

Figura 2: (a) Um cenário do mundo de aspirador de pó. (b) Representação do caso de teste correspondente.



2.3 História

Seja um agente a ser testado, *Agent*: $P^* \rightarrow A$, em um ambiente, *Amb*: $PxA \rightarrow P(P)$, tal que *P* representa um percepção; *P** representa uma sequência de percepções; *A* representa uma ação; e *P(P)*, um subconjunto de percepções obtidas a partir de *P*. A história de um agente em um ambiente descreve a experiência do agente no ambiente em uma sequência de episódios da forma:

$$h(Caso_i) : (\textit{Percepção}^0, \textit{Ação}^0) \rightarrow (\textit{Percepção}^1, \textit{Ação}^1) \rightarrow ... \rightarrow (\textit{Percepção}^k, \textit{Ação}^k) \quad (1)$$

tal que $Ep^k = (\textit{Percepção}^k, \textit{Ação}^k)$ representa o *k*-ésimo episódio da história $h(Caso_i)$. Portanto, a partir de uma história, é possível realizar a avaliação do comportamento de um determinado agente, tendo em vista que todos os dados necessários, para tal, encontram-se nela.

2.4 Sistemas imunológicos artificiais

Sistemas imunológicos artificiais podem ser definidos como sistemas computacionais inspirados pela teoria imunológica, na qual as funções, os princípios e os mecanismos de imunidade são utilizados para resolver problemas. Seu desenvolvimento e os domínios de aplicação seguem os paradigmas de soft computing [30], tais como redes neurais e algoritmos evolucionários [20].

Seguindo esse princípio, foi proposto um algoritmo de seleção clonal, chamado Clonalg, para aprendizado e otimização [21, 31]. O Clonalg gera uma população de *N* anticorpos, cada um especificando uma solução para o

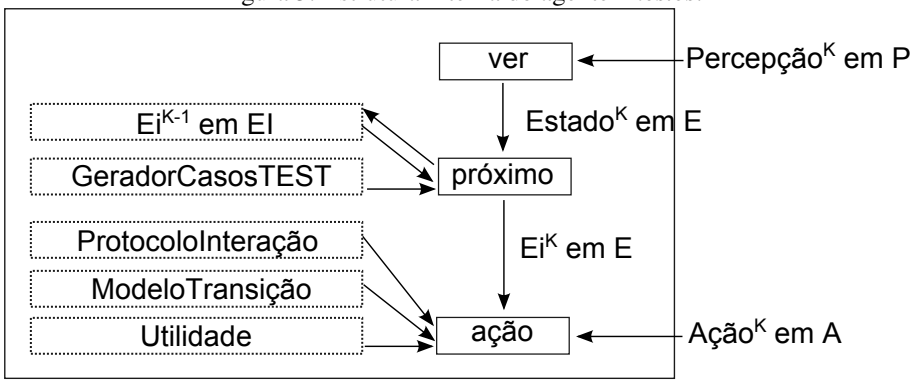
problema. Durante cada iteração do algoritmo, algumas das melhores soluções são selecionadas, clonadas e sofrem mutação no intuito de produzirem uma nova população candidata. Os novos anticorpos são, então, avaliados, e uma certa porcentagem dos melhores anticorpos é adicionada à população original. Finalmente, uma percentagem dos piores indivíduos da geração anterior é trocada por novos indivíduos criados aleatoriamente.

2.5 Agente *Thestes*

O agente *Thestes* foi concebido como um agente de resolução de problemas de seleção de caso de testes. Seu esqueleto fundamenta-se na estrutura do programa de agentes orientado por utilidade especificada por [1] e na arquitetura abstrata do agente com estado interno de [32].

Na Figura 3, pode-se observar que, internamente, o agente *Thestes* possui um subsistema de percepção, *ver*, responsável pelo mapeamento das informações necessárias em uma representação computacional, $Estado^K$. Um subsistema de atualização de estado interno, *próximo*, é responsável por gerar novos *CasosTEST* a partir das informações contidas em um $Estado^K$. Inicialmente, um *CasosTEST* pode ser gerado de forma aleatória, ou utilizar um conjunto de casos de testes criados manualmente pelo projetista. Por fim, um subsistema de atuação, *ação*, utiliza informações a respeito de um modelo de transição de estados para indicar a próxima $Ação^K$, que será executada pelo *Agent* e avaliada pela função utilidade.

Figura 3: Estrutura interna do agente *Thestes*.

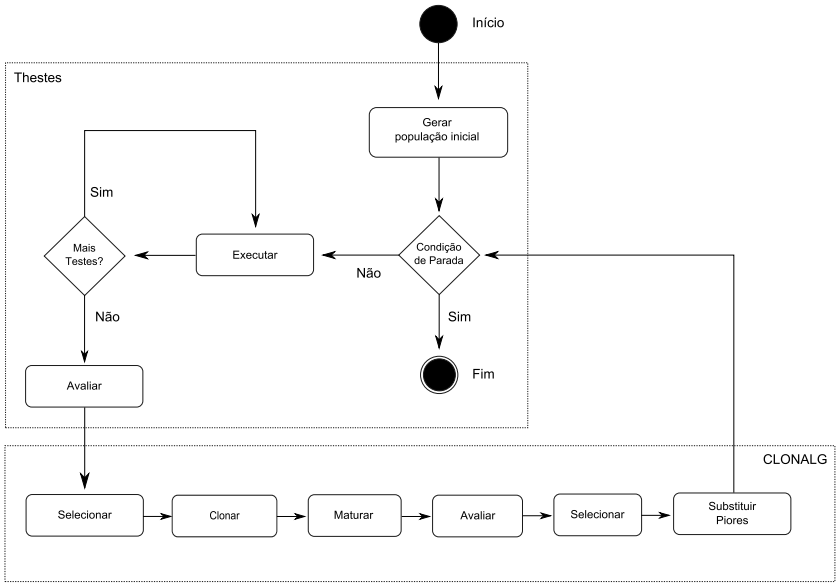


O funcionamento do agente *Thestes* em conjunto com o algoritmo de seleção clonal é apresentado por meio do fluxograma presente na Figura 4, e seu funcionamento é descrito a seguir: O processo começa com a geração da população inicial de casos de testes (correspondente aos indivíduos da população), que pode ser de forma aleatória ou definida pelo projetista. Portanto, a geração da população inicial implica que os cenários serão inicializados e as sujeiras e o agente serão posicionados no cenário. No próximo passo, todos os casos de testes da geração são executados, o que significa submeter o agente *Agent* a cada um dos cenário configurado no passo anterior. Durante a execução, as histórias são armazenadas para serem avaliadas em seguida. Na avaliação das histórias, o agente *Thestes* calcula o valor de cada medida de avaliação de desempenho. As histórias avaliadas são submetidas ao Clonalg, que seleciona os melhores casos de testes e gera clones proporcionalmente à sua avaliação (quanto maior a avaliação, maior o número de clones gerados). Os clones gerados são submetidos a um processo de maturação, em que cada indivíduo irá sofrer modificações com uma taxa inversamente proporcional à sua avaliação (quanto maior a avaliação, menor a taxa de modificação). Os novos indivíduos são avaliados, e os melhores são selecionados. Na fase final do Clonalg, os piores indivíduos da geração são substituídos pelos melhores clones e por novos indivíduos gerados de forma aleatória, para induzir a diversidade da população.

A fim de tornar mais clara a forma como foi implementado o algoritmo de seleção clonal utilizado, o pseudocódigo das rotinas implementadas é apresentado na Figura 5 e suas subfunções são descritas a seguir:

- *calcularAfinidade(pop)* - calcula a afinidade da população *pop* em relação à solução atual, gerando uma afinidade *f*, baseada nas medidas de avaliação de desempenho;

Figura 4: Fluxograma do funcionamento do agente *Thestes* utilizando o CLONALG como estratégia de busca.



- *selecionarNMelhores(pop, n)* - seleciona as *n* melhores soluções proporcionalmente à afinidade *f*, gerando uma subpopulação *nMelhores*;
- *gerarClones(nMelhores, b)* - clona os elementos da população *nMelhores* proporcionalmente à afinidade f e ao fator multiplicativo β, gerando uma população *clones*. A quantidade de clones de cada solução (*nClones*) é obtida por meio da fórmula (2), em que *nPop* é o tamanho da população *pop* e *i* é a posição da solução em relação à afinidade f;

$$nClones = round\left(\frac{\beta \times nPop}{i}\right) \tag{2}$$

- *mutação(clones)* - muta os elementos da população *clones* proporcionalmente à afinidade *f*, gerando uma população *mutados*;
- *adicionar(melhoresClones, pop)* - adiciona à população *pop* os elementos de *melhoresClones*;
- *gerarAleatórios(d, L)* - gera uma população *aleatórios*, contendo *d* elementos, em que cada elemento representa um indivíduos com o tamanho igual a *L*;
- *substituirPiores(pop, aleatórios)* - substitui *d* elementos da população *pop* pelos elementos contidos na população *aleatórios*.

3 Experimentos

Para a realização dos experimentos, a abordagem utilizada foi implementada por meio do *framework* de desenvolvimento de agentes JADE[33]. Foram testados dois tipos de agentes aspiradores de pó: um reativo simples, que atua baseado apenas na sua percepção atual, e um reativo baseado em modelos, que apresenta uma representação interna do cenário e atua de acordo com as informações armazenadas internamente. Para ambos os tipos de agentes, dois ambientes foram utilizados: um parcialmente observável, em que somente o estado atual da sala está disponível para o agente, e outro totalmente observável, em que o agente conhece o estado de todas as salas do cenário.

Os casos de testes iniciais foram gerados de forma aleatória, e os critérios de energia e limpeza, que representam a quantidade de energia gasta para realizar uma ação e a limpeza obtida, respectivamente, foram utilizados

Figura 5: Pseudo código do Clonalg.

```
 1. função Clonalg(pop, L, max, n, b, d)
 2. início
 3.  entrada
 4.   pop //População inicial
 5.   L   //Comprimento dos anticorpos
 6.   max //Núm. máximo de gerações
 7.   n   //Núm. de anticorpos selecionados
 8.         para clonagem
 9.   b   //Fator multiplicativo usado na definição
10.         da quantidade de clones
11.   d   //Quantidade de anticorpos de baixa
12.     afinidade que serão substituídos
13.
14.  variáveis
15.   contador, nMelhores, clones,
16.   melhoresClones, aleatórios
17.  para contador = 1 até max faça
18.  início
19.   calcularAfinidade(pop)
20.   nMelhores = selecionarNMelhores(pop, n)
21.   clones = gerarClones(nMelhores, b)
22.   mutados = mutação(clones)
23.   calcularAfinidade(mutados)
24.   melhoresClones = selecionarNMelhores(mutados, n)
25.   adicionar(melhoresClones, pop)
26.   aleatórios = gerarAleatórios(d, L)
27.   substituirPiores(pop, aleatórios)
28.  fim
29.  calcularAfinidade(pop)
30.
31.  retorne pop
32. fim
```

como medidas de desempenho. Os experimentos realizados visaram à validação e à análise de desempenho do algoritmo de seleção clonal, comparada a algoritmos genéticos e algoritmo de otimização por colônia de formigas, na abordagem para o teste de agentes inteligentes.

3.1 Ambiente

O ambiente do aspirador de pó considerado para os experimentos possui 25 salas, dispostas em forma de uma matriz 5 x 5, em que cada sala pode conter ou não sujeira, Figura 2 (a). O ambiente é estático, sendo assim, somente as ações dos agentes modificam o ambiente, e determinístico, ou seja, o próximo estado do ambiente é completamente determinado pelo estado atual e pelas ações executadas pelo agente, conforme o modelo definido na Tabela 1.

A tabela apresenta informações sobre as leis que governam as mudanças de estado de qualquer ambiente descrito em um caso de teste. Com base nela, dela, pode-se constatar que a percepção do próximo estado, $Percepção^{K+1}$, presente na terceira coluna, é obtida quando a $Ação^{K}$ é executada sobre a $Percepção^{K}$, segunda e primeira colunas, respectivamente.

3.2 Agentes testados

Internamente, os agentes possuem um programa de agente que é responsável pelo mapeamento das percepções em ações. Segundo os autores em [1], há quatro tipos básicos de programas de agentes:

Tabela 1: Modelo determinístico do mundo aspirador de pó.

Percepção^K	Ação^K	Percepção^{K+1}
Limpa	Não-operar	Limpa
Limpa	Sugar	Limpa
Limpa	Cima	Norte
Limpa	Esquerda	Oeste
Limpa	Direita	Leste
Limpa	Baixo	Sul
Suja	Não-operar	Suja
Suja	Sugar	Limpa
Suja	Cima	Norte
Suja	Esquerda	Oeste
Suja	Direita	Leste
Suja	Baixo	Sul

- reativo simples - seleciona ações com base na percepção atual, ignorando o histórico de percepções;
- reativo baseado em modelos - mantém um estado interno que depende do histórico das percepções;
- baseado em objetivos - além do estado atual, mantém informação sobre os objetivos que descrevem situações desejáveis; e
- baseado em utilidade - possui uma função utilidade que mapeia um estado em um grau de felicidade (utilidade) associado.

Neste trabalho, foram implementados dois tipos de programas de agentes: o reativo simples e o reativo baseado em modelos. O modelo determinístico do ambiente foi considerado na composição das regras condição-ação no programa de agente do ambiente (*Amb*).

O primeiro agente desenvolvido (*Agent_RS_Parcial*) foi do tipo reativo simples, no qual a seleção da ação a ser executada baseia-se apenas na percepção atual do ambiente parcialmente observável, de forma que o agente consegue identificar apenas a sua localização e a presença ou não de sujeira. As regras condição-ação do programa de agente são apresentadas na Figura 6.

Figura 6: Regras condição-ação *Agent_RS_Parcial*.

Se o estado da sala é S **então faça** Sugar
Se o estado da sala é L **então faça** movimento aleatório (Cima, Esquerda, Direita, Baixo, Não-Operar)

O segundo agente (*Agent_RS_Parcial_alterado*) foi desenvolvido com o objetivo de verificar a sensibilidade da abordagem proposta a possíveis falhas do *Agent_RS_Parcial*. Para isso, foi realizada uma alteração nas regras de condição-ação do *Agent*, de forma que a ação escolhida independe do estado da sala na qual o agente se encontra. Assim, as regras foram definidas conforme a Figura 7.

Figura 7: Regras condição-ação *Agent_RS_Parcial_alterado*.

Se o estado da sala é L ou S **então faça** ação aleatória (Sugar, Cima, Esquerda, Direita, Baixo, Não-Operar)

O terceiro agente (*Agent_RS_Total*) diferencia-se do primeiro agente em somente um aspecto: a observabilidade do ambiente. Nesse agente, foi considerada a capacidade de uma visibilidade total do ambiente, com isso, o agente movimenta-se em direção à sujeira mais próxima. A Figura 8 apresenta as regras condição-ação do *Agent_RS_Total*.

O quarto agente desenvolvido (*Agent_RM_Parcial*) foi do tipo baseado em modelo com visibilidade parcial do ambiente, no qual uma representação interna do ambiente é mantida com todas as salas já visitadas, a fim de

Figura 8: Regras condição-ação *Agent_RS_Total*.

Se o estado da sala é S **então faça** Sugar
Se o estado da sala é L e PróximaSalaSuja(norte) **então faça** Cima
Se o estado da sala é L e PróximaSalaSuja(sul) **então faça** Baixo
Se o estado da sala é L e PróximaSalaSuja(leste) **então faça** Esquerda
Se o estado da sala é L e PróximaSalaSuja(oeste) **então faça** Direita

evitar que o agente visite salas já percorridas. As regras condição-ação do *Agent_RM_Parcial* são apresentadas na Figura 9.

Figura 9: Regras condição-ação *Agent_RM_Parcial*.

Se o estado da sala é S **então faça** Sugar
Se o estado da sala é L e NãoVisitou(norte) **então faça** Cima
Se o estado da sala é L e NãoVisitou(sul) **então faça** Baixo
Se o estado da sala é L e NãoVisitou(leste) **então faça** Esquerda
Se o estado da sala é L e NãoVisitou(oeste) **então faça** Direita
Se o estado da sala é L e visitou todas **então faça** ação aleatória (Sugar, Cima, Esquerda, Direita, Baixo, Não-Operar)

O quinto e último agente desenvolvido (*Agent_RM_Total*) foi do tipo baseado em modelo, no qual uma representação interna do ambiente é mantida com todas as salas já visitadas. Nesse agente, foi considerada a capacidade de uma visibilidade total do ambiente. Com isso, o agente movimenta-se em direção à sujeira mais próxima. As regras condição-ação do *Agent_RM_Parcial* são apresentadas na Figura 10.

Figura 10: Regras condição-ação *Agent_RM_Total*.

Se o estado da sala é S **então faça** Sugar
Se o estado da sala é L e NãoVisitou(norte) e PróximaSalaSuja(norte) **então faça** Cima
Se o estado da sala é L e NãoVisitou(sul) e PróximaSalaSuja(sul) **então faça** Baixo
Se o estado da sala é L e NãoVisitou(leste) e PróximaSalaSuja(leste) **então faça** Esquerda
Se o estado da sala é L e NãoVisitou(oeste) e PróximaSalaSuja(oeste) **então faça** Direita

Em um funcionamento ideal, um agente aspirador de pó deve limpar a quantidade máxima de sujeira presente no seu ambiente, utilizando a menor quantidade de energia possível. Visando avaliar esses dois critérios, foi utilizada a medida de avaliação de desempenho da Tabela 2 para todos os agentes desenvolvidos. Essa tabela representa a relação entre a percepção obtida, a ação realizada e a avaliação de energia e limpeza do *Agent*, na qual valores negativos representam uma penalidade ao *Agent* por ter tomado uma ação inadequada.

3.3 *Thestes*

O *Thestes* parte de um conjunto inicial de casos de testes e utiliza estratégia de busca para encontrar cenários em que os agentes testados apresentam um baixo desempenho. Portanto, nesses experimentos, o agente *Thestes* tem como principal objetivo encontrar situações em que o *Agent* funcione de maneira inversa à qual foi projetado, em outras palavras, em que o *Agent* gaste cada vez mais energia e limpe cada vez menos sujeiras. Dessa forma, a obtenção de cenários em que os agentes apresentam baixo desempenho permite ao projetista realizar mudanças na estrutura do agente, a fim de melhorar o seu desempenho.

Para o estudo realizado, foi utilizado como função utilidade o inverso do somatório da multiplicação de cada uma das *M* medidas de avaliação de desempenho, f_m, obtida em uma história *H* de um caso de teste *CasoTEST*, por um peso definido para a respectiva medida de avaliação, w_m, somado à quantidade de salas sujas S_j existentes no ambiente ao final de cada execução, conforme fórmula definida a seguir:

Tabela 2: Medidas de avaliação de desempenho.

Percepção	Ação	Energia	Limpeza
L	Sugar	-1.0	0.0
L	Esquerda, Direita, Cima, Baixo	-2.0	1.0
L	Não-operar	0.0	0.0
S	Sugar	-1.0	2.0
S	Esquerda, Direita, Cima, Baixo	-2.0	-1.0
S	Não-operar	0.0	-1.0

$$Utilidade = -\left(\sum_{m=1}^{M} w_m \times f_m(H(CasoTEST)) + S_j\right) \quad (3)$$

A utilização da quantidade de salas sujas S_j ao final de cada execução visa privilegiar a seleção de casos de testes em que o cenário permaneceu com uma maior quantidade de sujeira. Nos experimentos realizados, os atributos de energia e limpeza foram considerados de igual importância. Assim, os valores dos pesos utilizados para esses atributos foram iguais a 0.5.

Os parâmetros utilizados nas estratégias de buscas utilizadas são apresentados nas subseções a seguir.

3.3.1 Estratégia de busca com GA

O algoritmo genético implementado foi do tipo GA Simples [34]. Para a escolha dos ambientes de tarefas candidatos a sofrerem alterações, foi empregada uma estratégia simples de seleção de pares. Para a escolha do subconjunto de salas candidatas às mudanças, por sua vez, foram utilizados operadores genéticos simples de cruzamento e mutação.

Tabela 3: Parâmetros utilizados no GA Simples.

NumCromo	Ocros	Pcros	Omut	Pmut	Osel
10	SinglePointCrossover	0.9	Uniform	0.6	Roleta

A Tabela 3 apresenta a melhor configuração obtida dos parâmetros utilizados, durante a realização de várias simulações computacionais. Essas informações descrevem a quantidade de cromossomos utilizados *NumCromo*, a probabilidade e o operador de cruzamento (*Pcros* e *Ocros*) entre os casos de teste, a probabilidade e o operador de mutação (*Pmut* e *Omut*), bem como o método para seleção dos melhores casos de teste (*Osel*). Mais especificamente, o operador de cruzamento *SinglePointCrossover* indica que apenas um ponto de cruzamento é selecionado; o operador de mutação *Uniform* substitui o valor de um gene escolhido por um valor aleatório uniforme selecionado de um intervalo de valores pré-determinados; e o operador de seleção *Roleta* atribui uma probabilidade aos indivíduos proporcionalmente à sua afinidade.

3.3.2 Estratégia de busca com ACO

Para a estratégia de busca utilizando algoritmos de otimização por colônia de formigas foi implementado o algoritmo Ant System [35] com os parâmetros especificados na Tabela 4, em que α é uma constante positiva usada para amplificar a concentração de feromônio; β^{aco} representa o peso dado à atratividade de uma solução (heurística); *m*, a quantidade de formigas (número de casos de teste gerados a cada iteração); ρ, a taxa de evaporação do feromônio depositado em cada uma das salas (valor real entre 0 e 1); τ^0, a quantidade inicial de feromônio depositado em cada sala (que deve ser maior que zero) e a constante *Q* indica a importância relativa da trilha de feromônio para a probabilidade de escolha.

Tabela 4: Parâmetros utilizados no Ant System.

α	β^{aco}	m	ρ	τ^0	Q
1	5	10	0.5	0.3	1

3.3.3 Estratégia de busca com Clonalg

Para a estratégia de busca por meio do Clonalg, os parâmetros apresentados na Tabela 5, em que *NumAnticorpos* representa a quantidade de anticorpos (Soluções) em cada geração, *NumClones* representa a quantidade de anticorpos que serão selecionados para a clonagem (proporcionalmente à utilidade do caso de teste), *NumSubst* representa a quantidade de anticorpos que serão substituídos por novos anticorpos e β^{sia} representa um fator multiplicativo utilizado para a determinação da quantidade de clones gerados.

Tabela 5: Parâmetros utilizados no CLONALG.

NumAnticorpos	NumClones	NumSubst	β^{sia}
10	3	1	0.3

3.4 Simulações

As simulações realizadas para a demonstração e avaliação da abordagem utilizada consideraram os agentes descritos na subseção 3.2 atuando no ambiente descrito na subseção 3.1. O *Thestes* utilizou algoritmos genéticos, algoritmo de otimização por colônia de formigas e algoritmo de seleção clonal para a estratégia de busca com os parâmetros especificados na subseção 3.3. Em todas as simulações realizadas, foi considerado como critério de parada o número máximo de trinta gerações (iterações). Cada geração (iteração) é composta por dez casos de testes, os quais são representados pelos indivíduos (cromossomos, formigas e anticorpos) da população. A avaliação de um caso de teste é obtida pela média de cinco repetições de cada simulação realizada. Somente as 25 primeiras ações realizadas pelo *Agent* em *Amb* são levadas em consideração para cada simulação, limitação que se deve à grande quantidade de interações que podem ocorrer entre agentes do tipo reativo simples atuando em ambientes parcialmente observáveis, em que a repetição de ações, tais como CIMA, BAIXO ou ESQUERDA, DIREITA, levaria a uma execução infinita do agente, o que inviabilizaria a execução dos testes.

A Tabela 6 apresenta um resumo dos experimentos realizados com a indicação dos nomes dos experimentos, dos tipos de agentes e dos ambientes empregados.

Tabela 6: Experimentos realizados.

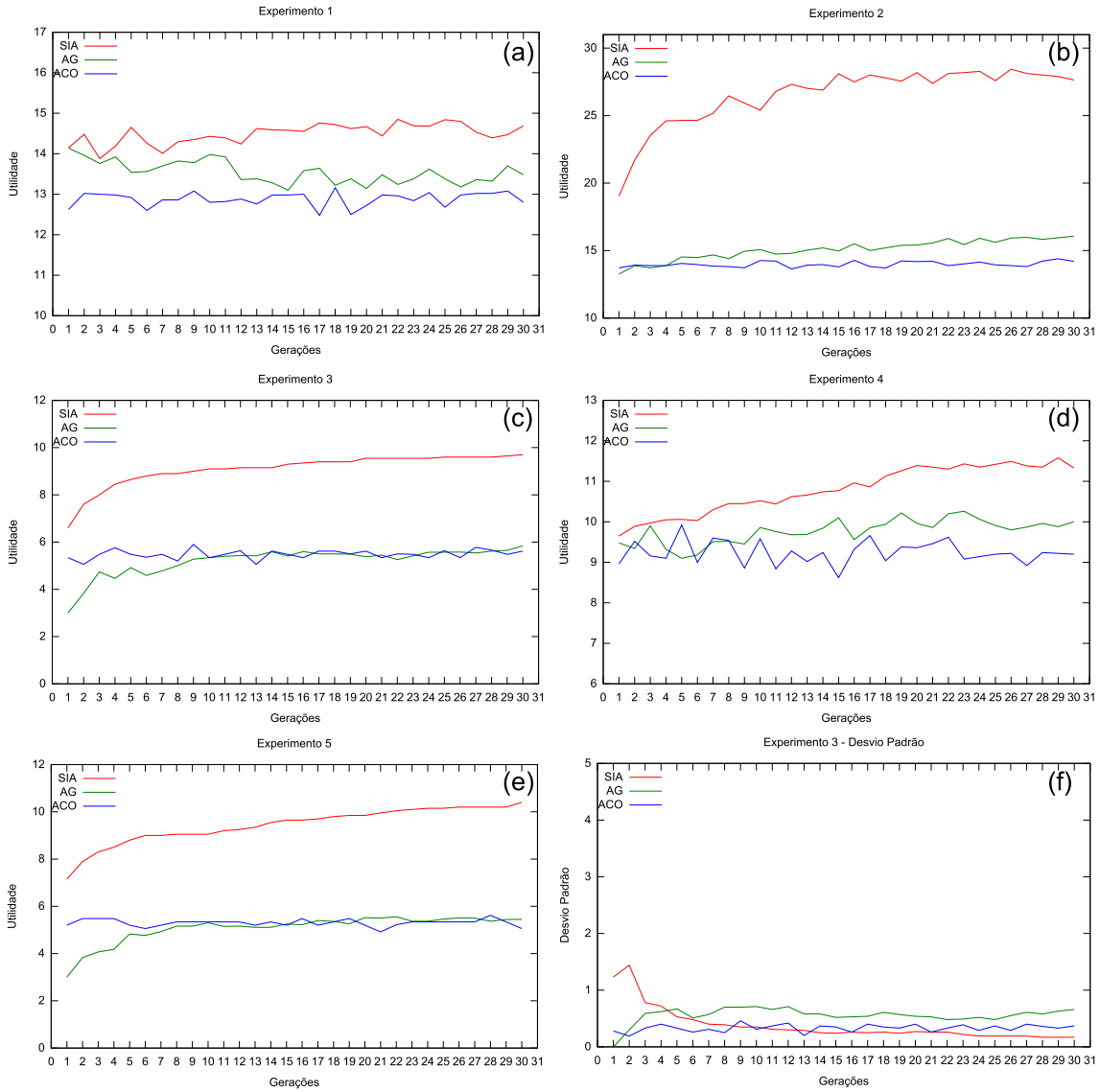
Exp	Nome	Tipo de agente	Ambiente
1	Agent_RS_Parcial	Reativo simples	Parcialmente observável
2	Agent_RS_Parcial_alterado	Reativo simples	Parcialmente observável
3	Agent_RS_Total	Reativo simples	Completamente observável
4	Agent_RM_Parcial	Baseado em modelo	Parcialmente observável
5	Agent_RM_Total	Baseado em modelo	Completamente observável

4 Resultados

Os resultados obtidos nos experimentos são apresentados a seguir em forma de gráficos que sistematizam os valores da utilidade linear do agente *Thestes* nas trinta gerações do experimento realizado. A utilidade linear é a métrica adotada para a avaliação dos experimentos, pois permite verificar o quão bom foi o desempenho do agente em um determinado cenário.

No Experimento 1, não foi possível verificar uma evolução significativa da utilidade linear do *Thestes*.

Figura 11: Resultados obtidos nos experimentos realizados.



Como pode ser visto na Figura 11 (a), a estratégia de busca baseada no Clonalg apresentou um pequeno crescimento na utilidade linear (diferença entre a última geração e a primeira) de 0.54, o ACO teve um crescimento de 0.18, enquanto a estratégia de busca AG apresentou um decrescimento de 0.66. Isso aconteceu devido à natureza randômica do agente testado, pois, embora o ambiente seja determinístico, os movimentos realizados pelo *Agent* no Experimento 1 são aleatórios, dessa forma, o cenário que obteve a melhor avaliação na geração atual pode terminar como o cenário de pior avaliação na geração seguinte. Esse foi o único caso de decrescimento observado.

A Figura 11 (b) apresenta os resultados do Experimento 2. Nesse experimento, os valores obtidos na geração final pelo ACO, pelo AG e pelo Clonalg foram, respectivamente, 14.20, 16.06 e 27.64, o que demonstra que o Clonalg obteve quase o dobro do valor das demais técnicas. Ao se observar o crescimento da utilidade linear nesse cenário, a diferença entre as técnicas fica ainda mais evidente, tendo em vista que o Clonalg obteve um valor de 8.6, enquanto o AG e o ACO obtiveram, respectivamente, 2.8 e 0.48, o que equivale a quase dezoito vezes no primeiro caso e a três vezes no segundo.

Em comparação ao Experimento 1, pôde-se verificar que a utilidade linear do agente *Thestes* foi bem superior no segundo experimento. Esse fato deve-se à modificação realizada no programa do *Agent* visando à introdução

de falhas, que levaram a que o agente passasse a realizar ações sem analisar o estado da sala na qual se encontra, conforme definido na subseção 3.2.

A Figura 11 (c) apresenta os resultados do Experimento 3. Nesse experimento, observou-se uma tendência de estabilização dos valores, o que pode ser explorado melhor por meio da análise do valor do desvio padrão dos indivíduos de cada geração, conforme o gráfico presente na Figura 11 (d). Esse gráfico permite verificar que o Clonalg apresentou uma convergência das soluções encontradas. Assim, a semelhança entre os indivíduos foi aumentando a cada geração até chegar ao desvio padrão de 0.17, obtido na última geração e que significa um baixo grau de diferenciação entre os seus indivíduos.

A Figura 11 (e) apresenta os resultados do Experimento 4. Nesse experimento, mais uma vez foi possível observar um melhor desempenho do Clonalg diante das demais técnicas utilizadas. O crescimento da utilidade linear do Clonalg foi de 1.68, enquanto a do ACO e a do AG cresceram, respectivamente, 0.24 e 0.52. Com esse experimento, foi possível confirmar que o agente *Thestes* obteve um melhor desempenho ao testar agentes reativos simples do que agente baseado em modelo, quando ambos atuam em um ambiente parcialmente observável. Isso aconteceu devido a uma característica presente no agente baseado em modelo: a representação interna do ambiente. Por meio dela, o agente verifica, por exemplo, se uma determinada sala já foi visitada antes de realizar uma ação. Dessa forma, ações desnecessárias, como retornar a uma sala já visitada, são evitadas, o que não acontece nos agentes reativos simples.

No Experimento 5, os resultados obtidos – Figura 11 (f) – foram praticamente os mesmos do Experimento 3. Esse comportamento já era esperado, pois a observabilidade completa do ambiente sobrepõe-se à principal característica de um agente baseado em modelo: a representação interna do ambiente. Assim, ao escolher as ações, o agente *Agent* não precisa verificar a representação interna do ambiente, pois já conhecê-lo completamente.

A análise dos resultados mostrou que, dentre os experimentos realizados, o Experimento 2 obteve o melhor desempenho; ou seja, nesse experimento, *Thestes* levou o desempenho de *Agent* em *Amb* para os valores mais baixos registrados, enquanto o Experimento 5 ficou com o pior desempenho. Esse comportamento já era esperado, no primeiro caso, devido à escolha das ações do agente ser independente do estado da sala em que ele se encontra e, no segundo, devido à observabilidade total do ambiente, o que permite objetividade/eficiência na escolha das ações.

Nos experimentos realizados, o agente baseado em modelo só não foi mais adequado do que os agentes que atuaram em um ambiente completamente observável, pois a capacidade de percepção de todo o ambiente leva a que o agente possa ir diretamente às salas que apresentam sujeiras. Por fim, constatou-se que, em todos os experimentos, o Clonalg conseguiu atingir um desempenho acima das demais técnicas implementadas, o que o credencia como uma boa técnica de otimização para o problema estudado. Entretanto, não se pretende descartar a utilização das demais técnicas, tendo em vista que esse desempenho pode variar de acordo com o problema a que são aplicadas.

5 Conclusões

Este artigo apresentou a aplicação de SIA, por meio do Clonalg, para o problema de otimização de seleção de casos de teste para o teste de sistemas computacionais baseados em agentes inteligentes. A fim de verificar a viabilidade da abordagem utilizada, foram realizadas implementações de dois tipos de agentes aspiradores de pó: um agente reativo simples e um agente reativo baseado em modelo. Além disso, foram considerados ambientes parcialmente e completamente observáveis. Com isso, pretendeu-se cobrir diversos cenários possíveis no teste de agentes. Para a validação do uso do Clonalg, foram estabelecidas comparações entre as técnicas de algoritmos genéticos e algoritmos de otimização por colônia de formigas. A avaliação do desempenho das técnicas deu-se mediante a mensuração da utilidade do agente *Thestes*. Os resultados mostraram a viabilidade da utilização da abordagem proposta na identificação de cenários que levam o agente testado a um baixo desempenho, fornecendo para o projetista informações necessárias para a melhoria do agente. Os resultados mostraram, ainda, que o Clonalg conseguiu atingir um desempenho acima das demais técnicas implementadas, o que a credenciou como uma boa técnica de otimização para o problema estudado. A principal limitação dessa pesquisa está na simplicidade e natureza aleatória da maioria dos agentes testados. Acredita-se que a aplicabilidade da abordagem utilizada possa ser melhor avaliada por meio do emprego de um agente com maior complexidade e cujas ações realizadas sejam

baseadas na racionalidade em vez da aleatoriedade. No entanto, ressalta-se a importância dessa pesquisa, tendo em vista as contribuições trazidas para a área de teste de agentes inteligentes.

Como trabalhos futuros, sugere-se: extensão de *Thestes* para contemplar os testes da simulação da avaliação de desempenho das interações entre os agentes presentes em sistemas multiagentes e o ambiente de tarefa; investigação de outros aspectos que possam ser incluídos nas medidas de avaliação de desempenho, ou um critério mais apropriado, que contribuam na identificação de episódios problemáticos do agente; e aplicação de outras meta-heurísticas multiobjetivos baseadas em população para avaliação dos resultados obtidos.

Referências

[1] RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. [S.l.]: Prentice Hall, 2009.

[2] KAY, A. Computer software. *Scientific American*, v. 251, n. 3, p. 53 – 59, 1984.

[3] DIVENYI, D.; DAN, A. Agent-based modeling of distributed generation in power system control. *IEEE Transactions on Sustainable Energy*, v. 4, n. 4, p. 886–893, Oct 2013.

[4] HUANG, W. et al. Technology and application of intelligent agent in electronic commerce. In: *Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on*. [S.l.: s.n.], 2010. v. 3, p. 730–733.

[5] FARIMANI, F.; YEKTAY, N.; MASHHADI, H. A novel adaptive agent-based algorithm to find mixed nash equilibrium in static games. In: *21st Iranian Conference on Electrical Engineering (ICEE)*. [S.l.: s.n.], 2013. p. 1–5.

[6] TAIR, H. et al. Pro-active multi-agent recommender system for travelers. In: *International Conference for Internet Technology and Secured Transactions (ICITST)*. [S.l.: s.n.], 2011. p. 466–471.

[7] CAN, O. Mobile agent based intrusion detection system. In: *Signal Processing and Communications Applications Conference (SIU), 2014 22nd*. [S.l.: s.n.], 2014. p. 1363–1366.

[8] BHADRA, S.; KUNDU, A.; GUHA, S. An agent based efficient traffic framework using fuzzy. In: *Fourth International Conference on Advanced Computing Communication Technologies (ACCT)*. [S.l.: s.n.], 2014. p. 464–470.

[9] TORRES, S. Swarm theory applied to air traffic flow management. *Procedia Computer Science*, v. 12, n. 0, p. 463 – 470, 2012. ISSN 1877-0509. Complex Adaptive Systems 2012.

[10] CAO, M.; LI, B.; WANG, C. Modeling intelligent distance education system based on agent. In: *First IEEE International Symposium on Information Technologies and Applications in Education (ISITAE)*. [S.l.: s.n.], 2007. p. 348–353.

[11] SALEEM, R.; MUHAMMAD, A.; MARTINEZ-ENRIQUEZ, A. Remote patient monitoring and healthcare management using multi-agent based architecture. In: *Ninth Mexican International Conference on Artificial Intelligence (MICAI)*. [S.l.: s.n.], 2010. p. 118–123.

[12] RAZA, A.; SHARIF, U.; HAIDER, S. On learning coordination among soccer agents. In: *IEEE International Conference on Robotics and Biomimetics (ROBIO)*. [S.l.: s.n.], 2012. p. 699–703.

[13] NGUYEN, C. D.; PERINI, A.; TONELLA, P. Goal-oriented testing for mass. *International Journal of Agent-Oriented Software Engineering*, Inderscience Publishers, v. 4, n. 1, p. 79–109, 2010.

[14] HOUHAMDI, Z. Test suite generation process for agent testing. *Indian Journal of Computer Science and Engineering (IJCSE)*, v. 2, n. 2, 2011.

[15] NGUYEN, C. D. et al. Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 25, n. 2, p. 260–283, 2012.

[16] ROUFF, C. A test agent for testing agents and their communities. In: IEEE. *IEEE Aerospace Conference*. [S.l.], 2002. v. 5, p. 5–26.

[17] NGUYEN, C. D. et al. Automated continuous testing of multi-agent systems. In: CITESEER. *The fifth European workshop on Multi-agent systems*. [S.l.], 2007.

[18] PARGAS, R. P.; HARROLD, M. J.; PECK, R. R. Test-data generation using genetic algorithms. *Software Testing Verification and Reliability*, Chichester, Sussex, England: J. Wiley, c1992-, v. 9, n. 4, p. 263–282, 1999.

[19] DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. *Computer*, IEEE, v. 11, n. 4, p. 34–41, 1978.

[20] CASTRO, L. N. de; TIMMIS, J. Artificial immune systems as a novel soft computing paradigm. *Soft computing*, Springer, v. 7, n. 8, p. 526–544, 2003.

[21] CASTRO, L. N. D.; ZUBEN, F. J. V. The clonal selection algorithm with engineering applications. In: *Genetic and Evolutionary Computation Conference*. [S.l.: s.n.], 2000. p. 36–39.

[22] SILVEIRA, F. R. V.; CAMPOS, G. A. L.; CORTÉS, M. I. Rational agents for the test of rational agents. *IEEE Latin America Transactions*, Fortaleza-CE, v. 11, p. 651 – 657, 2013.

[23] ENGELBRECHT, A. *Computational Intelligence: An Introduction*. [S.l.]: Wiley, 2007. ISBN 9780470512500.

[24] HOLLAND, J. H. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* [S.l.]: U Michigan Press, 1975.

[25] GOLDBERG, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. [S.l.]: Addison-Wesley, 1989. (Artificial Intelligence). ISBN 9780201157673.

[26] DORIGO, M.; STÜTZLE, T. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In: GLOVER, F.; KOCHENBERGER, G. (Ed.). *Handbook of Metaheuristics*. [S.l.]: Springer US, 2003, (International Series in Operations Research and Management Science, v. 57). p. 250–285. ISBN 978-1-4020-7263-5.

[27] DORIGO, M.; BONABEAU, E.; THERAULAZ, G. Ant algorithms and stigmergy. *Future Generation Computer Systems*, v. 16, n. 8, p. 851 – 871, 2000. ISSN 0167-739X.

[28] PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. [S.l.]: McGraw Hill, 2009.

[29] BASTOS, A. et al. *Base de Conhecimento em Teste de Software*. [S.l.]: Martins Fontes, 2007.

[30] DOTE, Y.; OVASKA, S. J. Industrial applications of soft computing: a review. *Proceedings of the IEEE*, IEEE, v. 89, n. 9, p. 1243–1265, 2001.

[31] CASTRO, L. N. D.; ZUBEN, F. J. V. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 6, n. 3, p. 239–251, 2002.

[32] WOOLDRIDGE, M. *An introduction to multiagent systems*. [S.l.]: Wiley, 2008.

[33] BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. *Developing Multi-Agent Systems with JADE*. [S.l.]: Wiley, 2007. ISBN 9780470058404.

[34] IBA, H.; NOMAN, N. *New Frontier in Evolutionary Algorithms: Theory and Applications*. [S.l.]: Imperial College Press, 2012. ISBN 9781848166813.

[35] DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. *Part B: Cybernetics, IEEE Transactions on Systems, Man, and Cybernetics*, IEEE, v. 26, n. 1, p. 29–41, 1996.