UNIVERSIDADE ESTADUAL DO CEARÁ - UECE
CENTRO DE CIÊNCIAS E TECNOLOGIA – CCT
MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

FRANCISCA RAQUEL DE VASCONCELOS SILVEIRA

UMA ABORDAGEM FUNDAMENTADA EM AGENTES RACIONAIS PARA O TESTE DE AGENTES RACIONAIS

FORTALEZA - CEARÁ
2013

FRANCISCA RAQUEL DE VASCONCELOS SILVEIRA

UMA ABORDAGEM FUNDAMENTADA EM AGENTES RACIONAIS PARA O TESTE DE AGENTES RACIONAIS

Dissertação submetida à Comissão Examinadora do Programa de Pós-Graduação Acadêmica em Ciência da Computação da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientação: Professor Dr. Gustavo Augusto Lima de Campos

FORTALEZA – CEARÁ

2013

Dados Internacionais de Catalogação na Publicação

Universidade Estadual do Ceará

Biblioteca Central Prof. Antônio Martins Filho

Bibliotecário (a) Leila Cavalcante Sátiro – CRB-3 / 544

S587u Silveira, Francisca Raquel de Vasconcelos.

Uma abordagem fundamentada em agentes racionais para o teste de agentes racionais/Francisca Raquel de Vasconcelos Silveira. — 2013.

CD-ROM 123f. : il. (algumas color.) ; 4 ¾ pol.

"CD-ROM contendo o arquivo no formato PDF do trabalho acadêmico, acondicionado em caixa de DVD Slin (19 x 14 cm x 7 mm)".

Dissertação (mestrado) – Universidade Estadual do Ceará, Centro de Ciências e Tecnologia, Mestrado Acadêmico em Ciência da Computação, Fortaleza, 2013.

Área de Concentração: Ciências da Computação.

Orientação: Prof. Dr. Gustavo Augusto Lima de Campos.

1. Teste de agentes. 2. Seleção de casos de teste. 3. Metaheurística baseada em população. 4. Agentes racionais. 5. Avaliação de desempenho. I. Título.

CDD: 001.6

UNIVERSIDADE ESTADUAL DO CEARA – UECE
PRO-REITORIA DE POS-GRADUACAO E PESQUISA - PRPGPq
CENTRO DE CIENCIAS E TECNOLOGIA – CCT
Mestrado Acadêmico em Ciência da Computação – MACC

Av. Paranjana, 1700. Bairro: Serrinha. Campus do Itapery Fone: (85) 3101 9776

ATA DA SEXAGÉSIMA DEFESA PÚBLICA DE DISSERTAÇÃO DE MESTRADO

Aos dezoito dias do mês de outubro de dois mil e treze, no miniauditório do prédio de Pesquisa e Pós-Graduação em Computação, do **Mestrado Acadêmico em Ciência da Computação – MACC,** realizou-se a sessão de defesa pública da dissertação de **FRANCISCA RAQUEL DE VASCONCELOS SILVEIRA,** aluna regularmente matriculada no Mestrado Acadêmico em Ciência da Computação – MACC, tendo como título: **"UMA ABORDAGEM FUNDAMENTADA EM AGENTES RACIONAIS PARA O TESTE DE AGENTES RACIONAIS"**. A Banca Examinadora reuniu-se no horário de 9:40 às 12:55 horas, sendo constituída pelos professores: Dr. Gustavo Augusto Lima de Campos, orientador e presidente da Banca, Paulo Henrique Mendes Maia, Ph.D. e Dra. Mariela Inés Cortés, todos da Universidade Estadual do Ceará – UECE e Ricardo de Andrade Lira Rabêlo, da Universidade Federal do Piauí - UFPI. Inicialmente a mestranda expôs seu trabalho e a seguir foi submetida à arguição pelos membros da Banca, dispondo cada membro de tempo para tal. Finalmente a Banca reuniu-se em separado e concluiu por considerar a mestranda APROVADA, por sua dissertação e sua defesa pública. Eu, Professor Dr. Gustavo Augusto Lima de Campos, orientador da dissertação e presidente da Banca, lavro a presente Ata que será assinada por mim e demais membros da Banca. Fortaleza, 18 de outubro de 2013.

Prof. Dr. Gustavo Augusto Lima de Campos
Orientador - UECE

Prof. Dr. Ricardo de Andrade Lira Rabêlo
Membro Externo - UFPI

Profa. Dra. Mariela Inés Cortés
UECE

Prof. Paulo Henrique Mendes Maia, Ph.D.
UECE

A meus pais, a meus irmãos e a meu esposo por serem meus maiores incentivadores.

AGRADECIMENTOS

Inicialmente, gostaria de agradecer a Deus, por ter dado a oportunidade de realizar este trabalho, guiando-me para vencer as dificuldades do percurso.

Aos meus pais, João e Ivanira, que sempre incentivaram e apoiaram minhas decisões. Agradeço ao carinho, à educação e aos conselhos que recebi durante toda a minha vida. Aos meus irmãos Willamy e Wesley pela solidariedade com que sempre convivemos. Obrigada a todos vocês pela confiança e compreensão.

Ao meu esposo, Alberto, que sempre me apoiou e me incentivou a enfrentar os desafios e a superar os obstáculos que surgiram. Agradeço pela ajuda, paciência, companhia e amor doados em todo esse tempo que estamos juntos.

Aos grandes amigos que inúmeras vezes ajudaram e colaboraram no desenvolvimento desse trabalho. À sabedoria e amizade provada nos momentos de reflexão e companheirismo.

Ao orientador, Prof. Dr. Gustavo Augusto, pela orientação, pela confiança depositada para a realização do presente trabalho e pelo estímulo e atenção dedicados ao longo do processo.

Agradeço a Profa. Dra. Mariela pela atenção, conselhos e contribuições dadas a este trabalho.

Aos professores Drs. Paulo Henrique e Ricardo Lira, por aceitarem o convite para participação em minha Banca Examinadora.

Aos funcionários do MACC, pela amizade e colaboração durante o mestrado.

A todos aqueles que de uma forma ou de outra colaboraram para a realização dessa conquista.

"Se vi mais longe foi por estar de pé sobre ombros de gigantes." (Isaac Newton)

RESUMO

Agentes racionais 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 trabalho apresenta uma formulação para o problema de seleção de casos de teste de agentes racionais através de uma abordagem solução fundamentada em um agente orientado por utilidade para selecionar casos de teste através de metaheurísticas baseadas em população para testar o desempenho dos agentes racionais e em um agente de monitoramento e diagnóstico das falhas. Para cada caso de teste são realizadas as interações entre Agente e Ambiente para obter a avaliação de desempenho correspondente. Nos experimentos com a abordagem testando agentes racionais com diferentes tipos de arquitetura em ambientes parcialmente e totalmente observável, a abordagem selecionou um conjunto de casos de teste satisfatório em termos das informações geradas sobre o desempenho insatisfatório do agente. A partir desse resultado, a abordagem possibilita a identificação dos objetivos que não estão sendo satisfeitos e as falhas apresentadas pelo agente na execução das ações e dos planos associados aos objetivos, permitindo ao projetista realizar mudanças objetivas na estrutura interna do agente de forma a melhorar o seu desempenho.

Palavras–Chave: Teste de agentes. Seleção de casos de teste. Metaheurísticas baseadas em população. Agentes racionais. Avaliação de desempenho.

ABSTRACT

Rational agents consist of a promising computer technology aiming at the development of complex distributed systems. In spite of the theoretical references available to guide the designer of those agents, there are few test techniques proposed to validate the aforementioned systems. It is known that this validation depends on the selected test cases which should arrange for information concerning the components in the agent's structure that are underperforming. This paper presents a formulation for the problem of selecting test cases of rational agents through of an agent-oriented solution approach by utility to select test case, through metaheuristics based on population in order to test the performance of the rational agents and in a monitoring agent and diagnosis of flaws. For each test case, interactions between agent and environment are held in order to obtain the corresponding performance evaluation. In test experiments focusing on rational agents with different architecture types in partially and totally watchable environments, the approach selected a set of satisfactory test cases in terms of information generated on the unsatisfactory targets that are not being fulfilled and flaws presented by the agent in the execution of actions and plans related to the goals, allowing the designer to make objective changes in the internal structure of the agent in order to improve its performance.

Keywords: Test agents. Selection of test cases. Metaheuristics based on population. Rational agents. Performance evaluation.

LISTA DE FIGURAS

FIGURA 1 – Agente com seus sensores e atuadores interagindo com o ambiente 22
FIGURA 2 – Subsistemas ver e ação 26
FIGURA 3 – Regras condição-ação do agente aspirador de pó 27
FIGURA 4 – Subsistemas ver, próximo e ação 28
FIGURA 5 – Agente reativo baseado em modelo 28
FIGURA 6 – Agente baseado em objetivos 29
FIGURA 7 – Agente orientado por utilidade 31
FIGURA 8 – Fluxograma geral de um algoritmo genético 37
FIGURA 9 – Ilustração do algoritmo NSGA-II 38
FIGURA 10 – Análise dos requisitos iniciais do agente aspirador 44
FIGURA 11 – Processo de evolução dos casos de teste 45
FIGURA 12 – Representação de um caso de teste para o mundo aspirador de pó 48
FIGURA 13 – Visão geral da abordagem 51
FIGURA 14 – Estrutura do programa agente *Thestes* 55
FIGURA 15 – Esqueleto de *Thestes* 55
FIGURA 16 – Esqueleto da função ação de *Thestes* 56
FIGURA 17 – Simulação das interações entre *Agent* e *Amb* 59
FIGURA 18 – Exemplo de curvas de indiferença considerando dois objetivos 61
FIGURA 19 – Regras condição-ação de agente *ProMon* para agentes reativos 66
FIGURA 20 – Ambiente de Tarefa/Caso de Teste 70
FIGURA 21 – Regras condição-ação de *RS_Parcial* 73
FIGURA 22 – Regras condição-ação de *RS_Parcial_alterado* 73
FIGURA 23 – Regras condição-ação de *RS_Total* 73
FIGURA 24 – Regras condição-ação de *RM_Parcial* 74
FIGURA 25 – Codificação do caso de teste 75
FIGURA 26 – Valores de utilidade de *CasosTEST* em 30 gerações (Experimentos 1, 3 e 4) . 80
FIGURA 27 – Box-plot dos valores de utilidade de 10 casos *CasosTEST* em 30 gerações 82
FIGURA 28 – Valores de inadequação de energia (1) e limpeza (2) (Experimentos 1, 3 e 4) 84
FIGURA 29 – Valores de inadequação de energia e limpeza (Experimentos 1, 3 e 4) 86
FIGURA 30 – Percentual de salas sujas nos *CasosTEST* (Experimentos 1, 3 e 4) 88

FIGURA 31 – Valores de utilidade de *CasosTEST* em 30 gerações (Experimento 2) 101
FIGURA 32 – Valores de inadequação de energia (1) e limpeza (2) (Experimento 2) 102
FIGURA 33 – Valores de inadequação de Energia e Limpeza (Experimento 2)..... 103
FIGURA 34 – Percentual de salas sujas nos *CasosTEST* em 30 gerações (Experimento 2) . 104
FIGURA 35 – Valores de utilidade de *CasosTEST* em 30 gerações (Experimentos 6 e 7)... 107
FIGURA 36 – Box-plot dos valores de utilidade dos Experimentos 5 e 6 em 30 gerações... 109
FIGURA 37 – Valores de inadequação de energia (1) e limpeza (2) (Experimentos 5 e 6) .. 110
FIGURA 38 – Percentual de salas sujas nos *CasosTEST*(Experimentos 6 e 7)..... 112

LISTA DE TABELAS

TABELA 1 – Comparativo entre os Trabalhos Relacionados 46
TABELA 2 – Medida de Avaliação de Desempenho..... 49
TABELA 3 – História parcial de *Agent* em *Amb* 50
TABELA 4 – Modelo Determinístico do Ambiente 71
TABELA 5 – Medida de Avaliação de Desempenho..... 72
TABELA 6 – Informações *ParâmetrosSimulação*..... 75
TABELA 7 – Informações em *ParâmetrosBUSCA* considerando GA e NSGA-II 76
TABELA 8 – Episódios com falhas para o agente aspirador de pó 77
TABELA 9 – Falhas possíveis de serem detectadas pelo Projetista 77
TABELA 10 – Experimentos 79
TABELA 11 – Geração e Utilidade do melhor caso em *CasosTEST* 81
TABELA 12 – Atributos energia e limpeza nos casos selecionados 87
TABELA 13 – Percentual de salas sujas nos casos selecionados 89
TABELA 14 – Quatro primeiros episódios na história de RS_Parcial 90
TABELA 15 – Conjunto de episódios ideais produzidos por *Agent** 90
TABELA 16 – Significado dos símbolos 91
TABELA 17 – Primeiro episódio com falha na história de *RS_Parcial* 92
TABELA 18 – Segundo episódio com falha na história de *RS_Parcial* 92
TABELA 19 – Episódios na história de *RS_Parcial*..... 92
TABELA 20 – Episódios ideais produzidos por *Agent** 93
TABELA 21 – Nono episódio da história de *RS_Parcial* 93
TABELA 22 – Seis primeiros episódios na história de *RS_Total* 94
TABELA 23 – Conjunto de episódios ideais produzidos por *Agent** para os seis episódios iniciais de *RS_Total*..... 94
TABELA 24 – Seis últimos episódios na história de *RS_Total* 95
TABELA 25 – Conjunto de episódios ideais produzidos por *Agent** para os seis últimos episódios de *RS_Total* 95
TABELA 26 – Seis primeiros episódios na história de *RM_Parcial* 97
TABELA 27 – Conjunto de episódios ideais gerados por *Agent** para os seis primeiros episódios de *RM_Parcial* 97

TABELA 28 – Seis últimos episódios na história de *RM_Parcial* 99
TABELA 29 – Ep^{18} com falha na história de *RM_Parcial* 99
TABELA 30 – Ep^{19} com falha na história de *RM_Parcial* 99
TABELA 31 – Ep^{20} com falha na história de *RM_Parcial* 100
TABELA 32 – Energia e limpeza nos casos selecionados de *RS_Parcial_alterado* 103
TABELA 33 – Percentual de salas sujas nos casos selecionados do *RS_Parcial_alterado* .. 104
TABELA 34 – Quatro primeiros episódios na história de *RS_Parcial_alterado* 105
TABELA 35 – Primeiro episódio com falha na história de *RS_Parcial_alterado* 105
TABELA 36 – Segundo episódio com falha na história de *RS_Parcial_alterado* 105
TABELA 37 – Ep^{6} com falha na história de *RS_Parcial_alterado* 106
TABELA 38 – Ep^{7} com falha na história de *RS_Parcial_alterado* 106
TABELA 39 – Geração e Utilidade do melhor caso em *CasosTEST* (Experimentos 5 e 6).. 108
TABELA 40 – Energia e limpeza nos casos selecionados dos Experimentos 5 e 6 111
TABELA 41 – Percentual de salas sujas nos casos selecionados dos Experimentos 5 e 6.... 112
TABELA 42 – Quatro primeiros episódios na história de *RS_Parcial_NSGA_II* 113
TABELA 43 – Primeiro episódio com falha na história de *RS_Parcial_NSGA_II* 114
TABELA 44 – Segundo episódio com falha na história de *RS_Parcial_NSGA_II* 114
TABELA 45 – Terceiro episódio com falha na história de *RS_Parcial_NSGA_II*..... 114
TABELA 46 – Quatro primeiros episódios na história de *RM_Parcial_NSGA_II*..... 115
TABELA 47 – Ep^{1} com falha na história de *RM_Parcial_NSGA_II*..... 115
TABELA 48 – Ep^{3} com falha na história de *RM_Parcial_NSGA_II*..... 116

LISTA DE ABREVIATURAS E SIGLAS

AE	Algoritmos Evolucionários
AOSE	*Agent-Oriented Software Engineering*
ES	Engenharia de Software
ET	*Evolutionary Testing*
GA	*Genetic Algorithm*
IA	Inteligência Artificial
NSGA	*Nondominated Sorting Genetic Algorithms*
NSGA-II	*Nondominated Sorting Genetic Algorithms II*
PEAS	*Perfomance, Environment, Actuators, Sensors*
PMO	Programação Multi-Objetivo
SBSE	*Search Based Software Engineering*
SMA	Sistema Multi-Agente
TDMC	Tomada de Decisão Multicritério

SUMÁRIO

1 INTRODUÇÃO 17
1.1 Contextualização 17
1.2 Motivação 18
1.3 Objetivos 19
1.4 Organização do Trabalho 20

2 FUNDAMENTAÇÃO TEÓRICA 22
2.1 Agentes Racionais 22
2.2 Ambientes 24
2.3 Arquiteturas Internas de Agentes Racionais 25
2.3.1 Agentes Reativos Simples 26
2.3.2 Agentes Reativos Baseados em Modelos 27
2.3.3 Agente Baseado em Objetivos 29
2.3.4 Agente Orientado por Utilidade 30
2.4 Testes de Agentes 31
2.5 Metaheurísticas 34
2.5.1 Algoritmos Genéticos 35
2.5.2 NSGA-II 38

3 TRABALHOS RELACIONADOS 40
3.1 Abordagem de Testes Orientada a Objetivos 40
3.2 Proposta do Componente Agente Testador 41
3.3 Testes Evolucionários para Agentes Autônomos 43
3.4 Comparativo entre os Trabalhos Relacionados 46
3.5 Conclusão 46

4 ABORDAGEM PARA O TESTE DE AGENTES RACIONAIS 48
4.1 Aspectos Considerados 48
4.1.1 Casos de Teste 48
4.1.2 História de um Agente em um Ambiente de Tarefa 49
4.1.3 Avaliação de Desempenho 49
4.2 Visão Geral da Abordagem 50

4.3 Agente *Thestes* **51**
4.3.1 Formulação do Problema de Seleção de Casos de Teste para Agentes Racionais 52
4.3.2 Estrutura do Programa Agente *Thestes* para Seleção de Casos de Teste 54
4.4 Modelo de Transição **58**
4.5 Mecanismo de Simulação de Interações *Agent-Amb* **58**
4.6 Função Utilidade **59**
4.7 Agente ProMon **62**
4.7.1 Noção de Agentes Racionais em *ProMon* 62
4.7.2 Funcionalidades do Agente 63

5 AVALIAÇÃO EXPERIMENTAL DA ABORDAGEM **69**
5.1 Plano Experimental **69**
5.1.1 Ambiente de Tarefa 70
5.1.2 Medida de Avaliação de Desempenho 72
5.1.3 Agentes Testados 72
5.1.4 Concretização de *Thestes* 74
5.1.5 Concretização do Agente *ProMon* 76
5.1.6 Experimentos 78
5.2 Apresentação e Análise dos Resultados **79**
5.2.1 Experimentos 1, 3 e 4 – Resultados *Thestes* 80
5.2.2 Experimentos 1, 3 e 4 – Funcionalidades *ProMon* 90
5.2.3 Experimentos 2 – Resultados *Thestes* 101
5.2.4 Experimentos 2 – Funcionalidades *ProMon* 105
5.2.5 Experimentos 5 e 6 – Resultados *Thestes* 107
5.2.6 Experimentos 5 e 6 – Funcionalidades *ProMon* 113

6 CONCLUSÃO **117**
6.1 Contribuições e Limitações da Pesquisa **117**
6.2 Trabalhos Futuros **118**
6.3 Conclusão **118**
6.4 Publicações Resultantes **119**

REFERÊNCIAS **121**

1 INTRODUÇÃO

Neste capítulo são apresentados os principais fatores que contextualizaram e motivaram o desenvolvimento desta pesquisa, assim como os objetivos a serem alcançados e estabelece como o trabalho está organizado.

1.1 Contextualização

À medida que a tecnologia evolui, somos levados à abstração e a generalização. O uso crescente da internet como a coluna vertebral de todos os serviços e dispositivos interligados faz com que amplie o surgimento de sistemas de software altamente complexos e praticamente ilimitados. Esses sistemas, em algumas circunstâncias, precisam ser adaptáveis, autônomos e dinâmicos para atender diferentes usuários e plataformas (NGUYEN, 2008; HOUHAMDI, 2011a).

Neste contexto, agentes inteligentes são vistos como uma tecnologia promissora para atender às necessidades empresariais modernas e oferecem também uma metodologia conceitual eficiente para projetar tais sistemas complexos. Na prática, a pesquisa sobre o desenvolvimento de agentes de software tornou-se muito grande e utilizada em diferentes áreas focando principalmente em arquiteturas, protocolos, estruturas, infraestrutura de mensagens e interações com a comunidade (HOUHAMDI, 2011a). As propriedades peculiares dos agentes (propriedades reativas, de memória, de aprendizagem, orientação por metas e por utilidade) satisfazem as necessidades empresariais modernas, oferecendo um paradigma conceitual eficaz para modelar sistemas complexos (NGUYEN, 2008).

Uma vez que esses sistemas estão cada vez mais assumindo as operações e controles na gestão da organização, os veículos automatizados e sistemas financeiros, garantias de que esses sistemas complexos funcionam corretamente precisam ser dadas aos usuários. Isto exige uma investigação de estruturas de engenharia de software apropriadas, incluindo requisitos de engenharia, arquitetura e técnicas de teste, para proporcionar processos adequados de desenvolvimento de software e ferramentas de apoio (NGUYEN et al., 2011).

Apesar de existirem alguns esforços no sentido de facilitar o desenvolvimento de sistemas baseados em agentes, pouco tem sido feito na direção de propor métodos e técnicas para testar a eficiência desses sistemas (NGUYEN, 2008). Em particular, devido às suas propriedades peculiares e à natureza específica dos agentes de software, que são projetados para serem distribuídos, autônomos e deliberativos, torna-se difícil a aplicação de técnicas de

testes de software que sejam capazes de garantir a confiabilidade desses sistemas. Portanto, testar agentes inteligentes é uma tarefa desafiadora (NGUYEN et al., 2009). As técnicas de testes precisam ser efetivas e adequadas para avaliar todos os comportamentos e as características dos agentes (NGUYEN, 2008).

O agente racional seleciona suas ações objetivando o melhor resultado possível, ou na presença de incertezas, o melhor resultado esperado conforme uma medida de desempenho estabelecida para avaliar seu comportamento. Idealmente, para cada sequência de percepções possível, um agente racional deve selecionar uma ação que se espera maximizar a sua medida de desempenho, dada a evidência fornecida pela sequência de percepções e por qualquer conhecimento interno do agente (RUSSELL; NORVIG, 2004). Devido às propriedades peculiares dos agentes racionais (propriedades reativas, de memória, de aprendizagem, orientação por metas e por utilidade) e de seus ambientes de tarefa, o teste destes sistemas não é uma tarefa trivial. A maioria dos trabalhos desenvolvidos para testar agentes adaptou as técnicas de testes de software para o caso dos agentes de software. No caso dos agentes racionais, sabe-se que estas adaptações devem avaliar a racionalidade das ações e dos planos executados pelo agente em seu ambiente (NGUYEN et al., 2009).

Baseados nesses princípios, Russell e Norvig (2004) descrevem quatro tipos básicos de programas de agentes racionais: (i) agentes reativos simples (selecionam ações com base na percepção atual, ignorando o histórico de percepções), (ii) agentes reativos baseados em modelos (o agente mantém um estado interno que depende do histórico das percepções), (iii) agentes baseados em objetivos (além do estado atual, mantém informação sobre os objetivos que descrevem situações desejáveis); (iv) agentes baseados em utilidade (possuem uma função utilidade que mapeia um estado em um grau de felicidade associado).

1.2 Motivação

Considerando que o agente racional deve ser capaz de realizar seus objetivos, testes adequados devem ser desenvolvidos para avaliar as ações e os planos executados pelo agente na realização destes objetivos em seu ambiente de tarefa (RUSSELL; NORVIG, 2004). Para a realização de testes em agentes racionais, é necessário que as técnicas existentes para testes de softwares sejam adaptadas e combinadas visando a detecção de diferentes tipos de falhas, tornando os agentes de software mais confiáveis (HOUHAMDI, 2011b).

Na literatura sobre o assunto, podemos evidenciar uma lacuna em termos de técnicas de testes efetivas aplicadas especificamente para testar agentes inteligentes. Nguyen

(2008) aponta que pouco tem sido estudado no sentido de propor métodos e técnicas eficazes que garantam a qualidade dos sistemas baseados em agentes. Como estes sistemas estão cada vez mais comuns, é de fundamental importância apresentar uma abordagem que assegure a eficiência da execução dos agentes.

Devido ao alto nível de dificuldade em testar os sistemas com agentes, novas técnicas de testes que tratam da natureza peculiar e específica dos agentes de software são pesquisadas. As técnicas precisam ser eficazes e adequadas para avaliar o comportamento autônomo do agente e avaliar sua confiança (HOUHAMDI, 2011a; HOUHAMDI, 2011b). Em outra perspectiva enquanto esse campo de pesquisa está se tornando mais avançado, há uma necessidade emergente para orientações detalhadas durante o processo de teste (HOUHAMDI, 2011a).

Na maioria dos trabalhos sobre o assunto, o pressuposto é que uma boa avaliação do agente depende dos casos de testes selecionados. Bons casos de teste devem gerar informações sobre o desempenho insatisfatório dos componentes na estrutura do agente e do funcionamento destes componentes de maneira integrada (HOUHAMDI, 2011b). Algumas abordagens focam a produção de artefatos de teste para dar suporte às metodologias de desenvolvimentos de sistemas de agentes (NGUYEN, 2008). Mais próximas da proposta deste trabalho, outras abordagens focam a seleção automática de casos para testar os agentes como, por exemplo, abordagens evolucionárias para gerar casos com níveis de dificuldades crescentes (NGUYEN et al., 2009).

Sabe-se que, nem sempre os melhores casos estão disponíveis inicialmente e, dependendo do ambiente de tarefa do agente, pode existir uma grande quantidade de casos a serem observados. De acordo com este ponto de vista, a abordagem proposta neste trabalho considera que a seleção de um conjunto de casos de teste é um problema de busca em um espaço de estados composto por uma grande família de conjuntos de casos possíveis. Um caso de teste ótimo é aquele em que o agente obtém o valor mínimo possível de desempenho. A abordagem proposta considera também o monitoramento e o diagnóstico das falhas dos agentes racionais.

1.3 Objetivos

A partir da motivação descrita, este trabalho tem como principal objetivo conceber uma abordagem baseada em um agente de resolução de problemas de seleção de casos de teste (*Thestes*) e um agente de monitoramento e diagnóstico das falhas do agente

testado (*ProMon*) que seja capaz de contribuir com o teste de programas de agentes racionais, gerando para o projetista informações relevantes sobre o desempenho do agente, que lhe permitam realizar melhorias nos programas dos agentes. O esqueleto principal de *Thestes* fundamenta-se na estrutura do programa agente orientado por utilidade. O programa emprega uma estratégia de busca local, baseada em populações e orientada por uma função utilidade, para encontrar conjuntos de casos de teste satisfatórios, ou seja, ambientes específicos em que as histórias associadas do agente testado em seu ambiente têm baixo desempenho. Durante o processo de busca de um conjunto de casos de teste, o agente *Thestes* utiliza um protocolo de interação entre o agente testado e seu ambiente para realizar simulações, anotar e avaliar por meio de uma função utilidade as histórias correspondentes. O esqueleto principal de *ProMon* fundamenta-se na arquitetura de agente reativo baseado em modelos com regras condição-ação.

De forma complementar, tem-se como objetivos específicos:

- Conceber um agente de resolução de problemas de seleção de casos de teste que realiza busca local no espaço de estados de casos de teste;
- Esboçar um agente de monitoramento dos testes e diagnóstico das falhas;
- Realizar a modelagem matemática do problema, definindo as variáveis envolvidas, suas dependências e restrições;
- Aplicar metaheurísticas na resolução do problema modelado e demonstrar a eficácia desses algoritmos na sua resolução;
- Projetar e desenvolver experimentos realizados em cenários do problema, bem como os resultados desses experimentos, para demonstrar a aplicabilidade e a viabilidade da abordagem proposta;
- Analisar as interações do agente testado com seu ambiente de tarefa (histórias) para identificar se a abordagem é capaz de demonstrar o desempenho insatisfatório e as falhas do agente na escolha de suas ações.

1.4 Organização do Trabalho

O conteúdo deste trabalho é apresentado ao longo de seis capítulos, incluindo a presente introdução, os quais são descritos sucintamente a seguir.

Capítulo 2 – Fundamentação Teórica: Discorre sobre os conceitos relacionados a agentes racionais, bem como suas arquiteturas. Adicionalmente, são apresentados os testes

de agentes comparados aos testes de software tradicionais. Fornece também uma visão geral sobre as metaheurísticas baseadas em Algoritmos Genéticos.

Capítulo 3 – Trabalhos Relacionados: Apresenta uma revisão bibliográfica acerca dos trabalhos relacionados que consideram os conceitos de testes de agente autônomos, destacando suas principais características e limitações.

Capítulo 4 – Abordagem para o Teste de Agentes Racionais: Descreve em detalhes a abordagem proposta, apresentando o agente de resolução de problemas de seleção de casos de teste para programas agentes racionais e o esboço do agente de monitoramento e diagnóstico das falhas. Ainda, discute importantes aspectos levados em consideração durante a definição da abordagem.

Capítulo 5 – Avaliação Experimental da Abordagem: Apresenta um estudo de caso que ilustra o funcionamento e uma avaliação da abordagem proposta, resolvendo um problema de seleção de casos de teste para programas agentes racionais e discorrendo a cerca do monitoramento e o diagnóstico das falhas identificadas.

Capítulo 6 – Conclusões: Relaciona as principais contribuições desta pesquisa, assim como as suas limitações. Apresenta também oportunidades para trabalhos futuros e as conclusões deste trabalho.

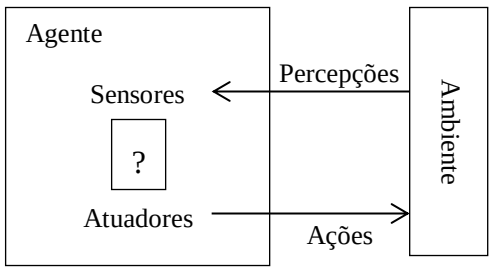
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece a fundamentação teórica sobre as áreas relacionadas ao escopo deste trabalho. Inicialmente, são contextualizados os agentes racionais. Em seguida, são apresentadas as arquiteturas dos agentes racionais propostas por Russell e Norvig (2004), necessárias para a compreensão da abordagem proposta e aplicadas nos agentes presentes na abordagem e nos experimentos. Também são contextualizados os testes de agentes, realizando uma analogia com os testes de softwares tradicionais. Por fim, o capítulo aborda as metaheurísticas utilizadas internamente pela abordagem para a geração e seleção dos casos de teste.

2.1 Agentes Racionais

Um agente (do latino *agere*, que significa fazer) é um tipo de programa de computador, aqui denominado programa agente, que tem a capacidade de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores (RUSSELL; NORVIG, 2004). A FIGURA 1 representa esquematicamente a interação entre um agente e seu ambiente. Na figura, o agente desempenha ações capazes de alterar o ambiente além de possuir sensores capazes de perceber alterações no ambiente.

FIGURA 1 – Agente com seus sensores e atuadores interagindo com o ambiente



Fonte: Adaptado de Russell e Norvig (2004)

De acordo com Wooldridge (2002), agentes são sistemas computacionais situados em algum ambiente onde são capazes de realizar atividades de maneira autônoma para atingir seus objetivos, ou seja, ele deve existir dentro de um ambiente, agir de maneira independente e tomar suas próprias decisões.

As definições acima se referem ao termo “agente” e não a agentes inteligentes. Segundo Wooldridge e Jennings (1995), para que um agente seja considerado “inteligente”, além de fazer parte de um ambiente, o perceber (através de sensores) e ter uma lista de ações possíveis, o agente deve possuir outras características:

- **Autonomia**: o agente deve ser capaz de operar sem nenhuma intervenção humana direta, ou de outros agentes, possuindo algum tipo de controle sobre suas ações e sobre seu estado interno.
- **Habilidade Social**: um agente pode interagir com outros agentes do ambiente, através de um protocolo de comunicação de agentes, para compartilhar conhecimentos ou concorrer para realizar seus objetivos.
- **Reatividade**: um agente é reativo quando é capaz de perceber, processar e responder (em tempo hábil) às alterações do ambiente.
- **Pró-atividade**: ser pró-ativo significa ser capaz de exibir um comportamento direcionado a objetivos, ou seja, um agente deve tomar a iniciativa de escolher quais ações tomar para alcançar seus objetivos em determinadas situações.

Do ponto de vista do projetista, um programa de agente é racional se, baseado nas informações percebidas por seus sensores, em suas crenças e no conjunto de ações possíveis de ser executado por seus atuadores, o programa for capaz de tomar decisões corretas, ou seja, que realizem os objetivos estabelecidos pelo projetista ou, quando não for possível realizar todos os objetivos, tomar as decisões de maior sucesso, definido de acordo com algum critério objetivo. Este critério é denominado medida de avaliação de desempenho. Quando um agente se baseia nas suas próprias percepções, dizemos que o agente tem autonomia. Um agente racional deve ser autônomo – deve aprender o que puder para possuir um conhecimento prévio. Sistemas de agentes racionais bem-sucedidos podem ser adequadamente chamados de sistemas de agentes inteligentes (RUSSELL; NORVIG, 2004).

A medida de avaliação considera um ou mais aspectos do ambiente, mais especificamente, considera o resultado desejado no ambiente. O programa agente racional é ideal se, para cada sequência de percepções possível, o agente atuar da maneira esperada para a maximização de sua medida de desempenho, conforme as evidências providenciadas pela sequência perceptiva e o conhecimento interno embutido no agente ou aprendido por meio de suas experiências (RUSSELL; NORVIG, 2004).

O processo de projeto de um programa agente racional deve iniciar com uma boa especificação dos componentes PEAS (*Perfomance, Environment, Actuators, Sensors* – desempenho, ambiente, atuadores, sensores). Esta especificação (PEAS) passa uma ideia da complexidade envolvida na concepção da função agente, ou seja, uma função que descreve o mapeamento das informações sobre percepções (S) nas informações sobre as ações (A), que é necessário para um bom desempenho (P) no ambiente (E) (RUSSELL; NORVIG, 2004).

2.2 Ambientes

Os ambientes provêm informações que são captadas pelas percepções do agente (ANTUNES, 2009). Os ambientes podem ser classificados em categorias, nas quais cada categoria influencia na definição da arquitetura de agente mais apropriada, e posteriormente, na aplicabilidade de uma técnica para a implementação de agentes (RUSSELL; NORVIG, 2004). As dimensões ao longo das quais os ambientes de tarefas podem ser divididos em categorias, conforme Russell e Norvig (2004) são:

- **Completamente observável ou parcialmente observável**: se o sensor do agente permite acesso completo ao estado do ambiente a qualquer instante, conseguindo observar todos os aspectos relevantes para a escolha da ação a executar, neste caso o ambiente é completamente observável. Um ambiente pode ser parcialmente observável devido a ruídos e a sensores imprecisos ou quando somente parte do estado do ambiente pode ser acessado pelo agente.
- **Determinístico ou estocástico**: se o próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada pelo agente, então o ambiente é determinístico; caso contrário, quando não é possível determinar o estado que o ambiente assumirá após a execução de uma ação, o ambiente é estocástico.
- **Episódico ou sequencial**: em um ambiente de tarefa episódico, a experiência do agente é dividida em episódios atômicos. Cada episódio consiste na percepção do agente, e depois na execução de uma única ação. O próximo episódio não deve depender das ações executadas em episódios anteriores. Em ambientes episódicos, a escolha da ação em cada episódio depende somente do próprio episódio. Em ambientes sequenciais, a escolha da ação em um determinado episódio pode afetar as decisões futuras.
- **Estático ou dinâmico**: um ambiente é estático quando permanece inalterado enquanto um agente está deliberando; caso contrário, ele é dinâmico. Em um ambiente dinâmico, além das ações desempenhadas pelos agentes, existem processos que operam sobre o ambiente, alterando o estado do mesmo.
- **Discreto ou contínuo**: um ambiente discreto é aquele que possui um número finito de estados, além de um conjunto finito de percepções e ações. Um ambiente contínuo é aquele no qual é possível assumir incontáveis estados.

- **Agente único ou multiagente**: ambientes com apenas um agente são caracterizados como agente único; ambientes multiagente são compostos da interação de múltiplos agentes.

Dependendo do tipo de ambiente em que o agente a ser implementado irá atuar, algumas arquiteturas internas de agente podem ser mais adequadas. O tipo de arquitetura interna de um agente também pode estar relacionada ao sub-problema que o mesmo resolverá no contexto que está inserido (GONÇALVES, 2009).

2.3 Arquiteturas Internas de Agentes Racionais

Os agentes inteligentes podem ser classificados de acordo com a maneira através da qual coletam informações e agem no ambiente, ou seja, de acordo com a sua arquitetura. A arquitetura de agentes é essencialmente um mapa interno para a construção de um agente que especifica a decomposição em módulos e como esses módulos interagem, além de um mapeamento que relaciona as percepções recebidas com as ações a serem executadas (WOOLDRIDGE, 2002).

Russell e Norvig (2004) definem que o agente atua no ambiente através de seus atuadores e o percebe através de seus sensores. É possível observar na FIGURA 1 um símbolo de '?' situado entre as percepções do agente e suas ações, o qual representa a função agente que é diferente para cada arquitetura interna. A concepção da função agente que faz o mapeamento entre os canais de percepção e de ação consiste em uma das principais tarefas da Inteligência Artificial. Dependendo do ambiente de tarefa, a concepção do agente pode não ser uma tarefa trivial. É necessário que as propriedades do ambiente interno do agente sejam adequadas às propriedades do ambiente externo onde os objetivos devem ser realizados (GONÇALVES, 2009).

As arquiteturas internas do agente são basicamente determinadas a partir de dois princípios: reativo e pró-ativo. O reativo apenas responde às mudanças que ocorrem em seu ambiente. Os pró-ativos ou cognitivos podem decidir sobre suas ações, definir sequência de ações e adquirir conhecimento através das situações em que o agente se encontra (GONÇALVES, 2009; WEISS, 1999).

Russell e Norvig (2004) destacam quatro tipos básicos de programas de agentes, a saber: (i) agentes reativos simples, (ii) agentes reativos baseados em modelos, (iii) agentes baseados em objetivos e (iv) agentes baseados em utilidade. Considerando os princípios

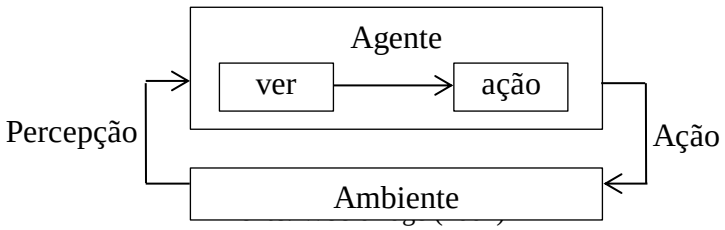
reativo e pró-ativo, os tipos e programas (i) e (ii) destacam a reatividade, enquanto os tipos (iii) e (iv) dizem respeito à pró-atividade.

2.3.1 Agentes Reativos Simples

Os agentes puramente reativos devem responder continuamente às mudanças que ocorrem no ambiente. Esses agentes decidem o que fazer sem levar em consideração o histórico de percepções obtido até o momento. A tomada de decisão é realizada de acordo com o estado atual, sem referenciar aos estados obtidos anteriormente (WOOLDRIDGE, 2002). Devido à arquitetura pouco complexa, os agentes reativos simples se caracterizam por possuir inteligência muito limitada. O agente funcionará somente se a decisão correta puder ser tomada com base na percepção atual (RUSSELL; NORVIG, 2004).

A FIGURA 2 ilustra algumas das informações e dos módulos (subsistemas) propostos para esse tipo de arquitetura. O subsistema *ver* captura a habilidade do agente em perceber o ambiente, enquanto o subsistema *ação* representa o processo de tomada de decisão do agente através do mapeamento percepção-ação (WOOLDRIDGE, 2002).

FIGURA 2 – Subsistemas ver e ação



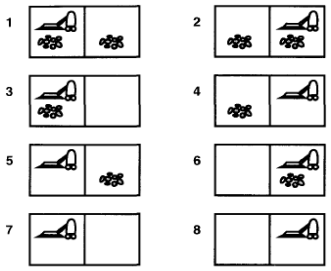
Mais especificamente, o funcionamento da arquitetura do agente envolve quatro passos, ou seja: (1) por meio dos sensores, o agente recebe informações do ambiente que são sequências de estados definidos em $S = \{s_1, ..., s_n\}$; (2) um subsistema de percepção, $Ver: S \rightarrow P$, processa cada estado de uma sequência S^* e mapeia em uma de m percepções, $P = \{p_1, ..., p_m\}$, que são representações de aspectos dos estados de S que estão acessíveis ao agente para a tomada de decisão; (3) um subsistema de tomada de decisão, $Ação: P^* \rightarrow A$, processa as sequências perceptivas P^*, resultantes de S^*, e seleciona uma de k ações em $A = \{a_1, ..., a_k\}$; e (4) por meio de atuadores o agente envia a ação selecionada para o ambiente (GONÇALVES, 2009).

Russell e Norvig (2004) incorporaram um conjunto de regras condição-ação no subsistema de tomada de decisão do agente reativo. Estas regras são um conjunto de ações especificadas para várias situações previstas, simplificando o mapeamento percepção-ação. As regras condição-ação são formalizações de associações comuns observadas entre certas

percepções e ações possíveis para o agente. Assim, o agente tem especificado um conjunto de ações para várias situações previstas, o que simplifica a implementação do mapeamento percepção-ação.

Por exemplo, vale destacar o projeto do agente reativo aspirador de pó em um mundo simplificado. O aspirador deve ser capaz de limpar um ambiente estático formado por duas salas (A e B) que podem conter sujeira. O ambiente é parcialmente observável, ou seja, o sensor do aspirador disponibiliza apenas informações locais a respeito da sala (A ou B) e do estado da sala (L-limpa ou S-suja) em que o agente está. Os atuadores do aspirador permitem que ele Aspire, vá para a Esquerda ou para Direita de uma sala (GONÇALVES, 2009). A FIGURA 3 ilustra em (a) os oito estados possíveis no mundo do aspirador de pó simplificado e em (b) um conjunto de regras condição-ação correspondente.

FIGURA 3 – Regras condição-ação do agente aspirador de pó



(a)

função AGENTE-ASPIRADOR-REATIVO([*posição,estado*])
retorna uma ação

se estado = *Sujo* **então retorna** *Aspirar*
senão se posição = *A* **então retorna** *Direita*
senão se posição = *B* **então retorna** *Esquerda*

(b)

Fonte: Russell e Norvig (2004)

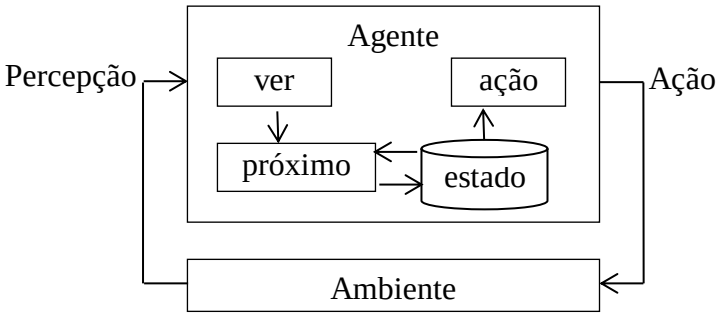
Considerando o ambiente simplificado (estático e parcialmente observável) e o objetivo do agente (limpar as duas salas), pode-se afirmar que, independentemente da configuração inicial do mundo, o agente reativo aspirador de pó simplificado esboçado na FIGURA 3 é racional. Para este mundo, não existe outro mapeamento capaz de produzir um desempenho melhor. Entretanto, em mundos mais complexos (com extensão, limites e obstáculos) uma abordagem puramente reativa pode não ser adequada. Nestes mundos, pode ser mais interessante concretizar a arquitetura do agente com estado interno e regras condição-ação, um refinamento da arquitetura do agente reativo simples (GONÇALVES, 2009).

2.3.2 Agentes Reativos Baseados em Modelos

Os agentes reativos puramente baseados em modelo (conhecimento a respeito do estado do ambiente) tem alguma estrutura de dados interna, que é normalmente usada para registrar informações sobre o estado do ambiente e da sequência de percepções obtidas. O processo de tomada de decisão (ação) deste tipo de agente baseia-se em, pelo menos, parte

dessa informação (WOOLDRIDGE, 2002). A FIGURA 4 ilustra algumas das informações e dos módulos (subsistemas) propostos para esse tipo de arquitetura.

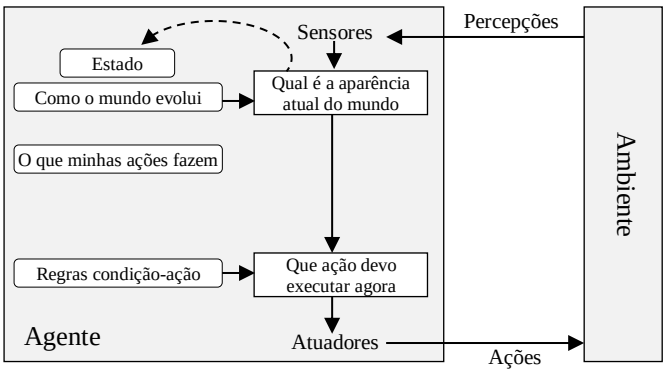
FIGURA 4 – Subsistemas ver, próximo e ação



Fonte: Adaptado de Wooldridge (2002)

Russell e Norvig (2004) salientam que o modo mais efetivo de lidar com a possibilidade de observação parcial requer um mecanismo que possibilite ao agente controlar a parte do mundo que ele não pode ver agora. Isto é, o agente deve manter um estado interno que dependa do histórico de percepções e assim reflita pelo menos alguns dos aspectos não observados do estado atual. A FIGURA 5 fornece a estrutura do agente reativo com seu estado interno, mostrando como a percepção atual é combinada com o estado interno antigo para gerar a descrição atualizada do estado atual.

FIGURA 5 – Agente reativo baseado em modelo



Fonte: Adaptado de Russell e Norvig (2004)

Comparando com o agente reativo simples, o agente baseado em modelo possui um subsistema de processamento de informação a mais, como visto na FIGURA 4: o subsistema *próximo*. Isto implica em uma nova decomposição no subsistema de tomada de decisão do agente padrão. Em termos de funcionamento, esta decomposição equivale à decomposição do passo (3) do ciclo de operação do agente puramente reativo descrito na seção anterior. Mais especificamente, depois do subsistema de percepção *ver* mapear um estado do ambiente em *S* em uma percepção em *P*, um subsistema de atualização de estado interno, Próximo: I x P → I, mapeia a percepção em *P* e o estado interno corrente em $I = \{i_1,$

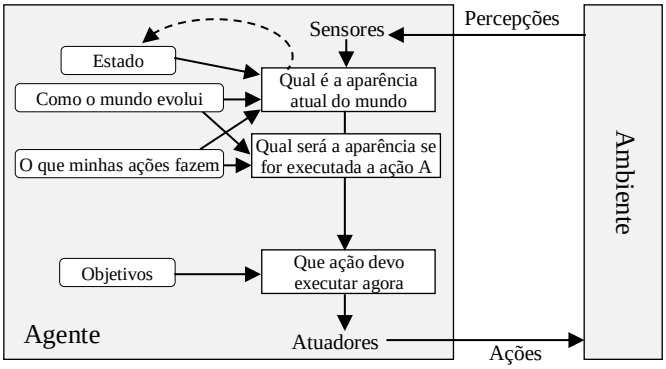
..., $i_m\}$, em um novo estado interno; que, por sua vez, é processado pelo subsistema de tomada de decisão, *Ação:* $I \rightarrow A$, para selecionar uma ação possível em *A* (GONÇALVES, 2009).

Por exemplo, voltando ao mundo simplificado do aspirador descrito na FIGURA 3, considerando que na configuração inicial, em $t = 0$, o mundo está no *estado 1* e que o estado interno inicial do agente é "vazio" (não sabe a sala em que está, e nem os estados das salas *A* e *B*), o funcionamento desta nova arquitetura pode ser resumido em ciclos compostos de cinco passos principais: (0) o agente sai de um estado interno inicial $i_0 = [\ ]$; (1) observa o estado do ambiente $s_0 = [sujo, sala\ A]$; (2) e gera uma percepção $p_0 = Ver(s_0) = [S, A]$; (3.1) seu estado interno é então atualizado por meio da função próximo, tornando-se, $i_1 = Próximo(i_0, ver(s_0)) = [A, S\ em\ A]$; (3.2) seleciona uma ação por meio de $a_0 = Ação(próximo(i_0,ver(s_0))) = Aspirar$; e (4) a ação é então executada no ambiente. Em seguida, visando realizar o objetivo o agente inicia outro ciclo de operação, em $t = t+1$, ou seja, percebendo o mundo por meio de Ver, adaptando o estado interno por meio de Próximo, escolhendo uma nova ação por meio de Ação e executando a mesma no ambiente.

2.3.3 Agente Baseado em Objetivos

Conhecer o estado do ambiente nem sempre é suficiente para se decidir o que fazer. Da mesma forma que o agente precisa do estado atual, ele precisa de informações sobre os objetivos que descrevem situações desejáveis. O programa agente pode combinar isso com informações sobre os resultados de ações possíveis (as mesmas informações que foram utilizadas para atualizar o estado interno no agente reativo), a fim de escolher ações que alcancem o objetivo (RUSSELL; NORVIG, 2004). A FIGURA 6 apresenta a estrutura do agente baseado em objetivos.

FIGURA 6 – Agente baseado em objetivos



Fonte: Adaptado de Russell e Norvig (2004)

Às vezes, a seleção da ação baseada em objetivos é direta, quando a satisfação do objetivo resulta de imediato de uma única ação. Outras vezes ela será mais complicada,

quando o agente tiver de considerar longas sequências de ações até encontrar um meio de atingir o objetivo. Busca e planejamento dedicam-se a encontrar sequências de ações que alcançam os objetivos do agente (RUSSELL; NORVIG, 2004).

Por exemplo, se um agente motorista estivesse em um entroncamento com capacidade de seguir em frente, virar à esquerda ou virar à direita, o objetivo para o motorista poderia ser o destino do cliente. Adicionalmente, o agente pode usar as informações existentes nas crenças para auxiliar a tomada de decisão, visando cumprir seu objetivo (GONÇALVES, 2009).

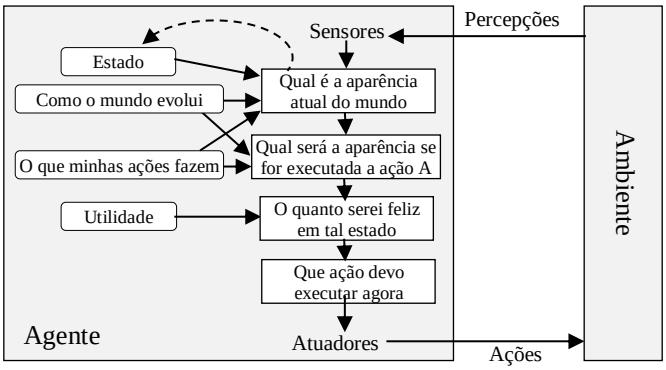
2.3.4 Agente Orientado por Utilidade

Sozinhos, os objetivos não são realmente suficientes para gerar um comportamento de alta qualidade na maioria dos ambientes. Os objetivos simplesmente permitem uma distinção entre “estados felizes” e “infelizes”, enquanto uma medida de desempenho mais geral deve permitir uma comparação entre diferentes estados do mundo, de acordo com o grau exato de felicidade que proporcionariam ao agente se pudessem ser alcançados. Se um estado for preferido em detrimento de outro, ele terá mais utilidade para o agente (RUSSELL; NORVIG, 2004).

Podemos utilizar uma função responsável por mapear um estado (ou conjunto de estados) possível com um grau de utilidade associado. Esta função é chamada Função Utilidade (GONÇALVES, 2009). A função utilidade permite decisões racionais em dois casos: a) quando objetivos contraditórios estão associados ao agente, dos quais apenas alguns podem ser atingidos, a função utilidade especifica o compromisso de cada um, e b) quando existem vários objetivos que o agente deseja alcançar e nenhum deles pode ser atingido com certeza, a utilidade fornece um meio pelo qual a probabilidade de sucesso pode ser ponderada em relação à importância dos objetivos (RUSSELL; NORVIG, 2004).

A estrutura interna do agente orientado por utilidade é apresentada na FIGURA 7FIGURA 7. O agente baseado em utilidade estende o conceito de agente baseado em objetivos, isto é, ele inclui os seguintes elementos: Percepção, Função próximo, Função de formulação de objetivo, Função de formulação de problema, Planejamento, Ações, e adiciona a Função utilidade. Esta função recebe um estado e retorna o grau de utilidade de tal estado de acordo com os objetivos atuais (GONÇALVES, 2009).

FIGURA 7 – Agente orientado por utilidade



Fonte: Adaptado de Russell e Norvig (2004)

2.4 Testes de Agentes

Na contextualização de softwares tradicionais, Tsui e Karam (2013) apontam que um dos principais objetivos do desenvolvimento de software é produzir software de alta qualidade, sendo a "qualidade" geralmente definida como o atendimento às especificações e a adequabilidade ao uso. Para atingir este objetivo, existe a necessidade de testes – mantendo um conjunto de técnicas para a detecção e correção de erros em um produto de software.

Teste de software é uma atividade que tem como objetivo avaliar a qualidade do produto e possibilitar a sua melhoria através da identificação de defeitos e problemas (BOURQUE; DUPUIS, 2004). Os testes consistem na verificação dinâmica do comportamento de um programa com base em um conjunto finito de casos de teste, selecionado de forma apropriada a partir do habitualmente infinito domínio de execuções, em relação ao comportamento esperado (TSUI; KARAM, 2013). O teste oferece efetivamente o último reduto no qual a qualidade pode ser avaliada e, mais pragmaticamente, erros podem ser descobertos (PRESSMAN, 2006).

Todos os testes requerem a definição dos critérios de teste, que são utilizados para determinar o que deve ser o conjunto apropriado de situações de teste. Uma vez que o conjunto de situações de teste selecionadas tenha sido executado, os testes poderão ser encerrados. Portanto, os critérios de teste também podem ser encarados como um meio de se determinar quando os testes podem ser encerrados, observando-se a não ocorrência de problemas no produto de software quando todas as situações de teste selecionadas tiverem sido executadas (TSUI; KARAM, 2013).

Sob vários aspectos, o teste é um processo independente e o número de tipos e técnicas diferentes de teste varia tanto quanto as diferentes abordagens de desenvolvimento.

Analogamente, diferentes métodos de teste estão começando a se agregar em várias abordagens e filosofias distintas (PRESSMAN, 2006).

A técnica de teste *Evolutionary Testing* (ET) (MCMINN; HOLCOMBE, 2003; WEGENER, 2005) é inspirada pela teoria da evolução na biologia que enfatiza a seleção natural, herança e variabilidade. Indivíduos mais aptos têm maior chance de sobreviver e de reproduzir, fazendo com que características especiais desses indivíduos sejam herdadas. Em ET, costuma-se codificar cada caso de teste como um indivíduo na população de casos de teste. Para orientar a evolução dos melhores casos de teste é utilizada a medida de *fitness*. Casos de teste com o melhor valor de *fitness* têm uma alta chance de serem selecionados para a geração de novos casos de teste. Além disso, a mutação é aplicada durante a reprodução, a fim de gerar um conjunto mais diversificado de casos de teste (NGUYEN, 2008).

O passo chave na ET é a transformação dos objetivos do teste em problemas de busca, especificamente na medida de *fitness*. Diferentes objetivos de teste geram diferentes definições da medida de *fitness*. Uma vez que a medida de *fitness* tenha sido definida, diferentes técnicas de busca, tal como busca local, algoritmos genéticos, podem ser usadas para gerar casos de teste visando otimizar a medida de *fitness* (NGUYEN, 2008).

O uso de técnicas de ET para automatizar a geração de dados de teste tem recebido um crescente interesse de muitos pesquisadores, refletindo em uma tendência crescente para a *Search-Based Software Engineering* (SBSE) (HARMAN; JONES, 2001).

Segundo Houhamdi (2011b) existe uma procura por novas técnicas de testes relacionadas à natureza particular dos agentes. As técnicas precisam ser eficazes e adequadas para avaliar os comportamentos dos agentes autônomos e construir a confiança neles.

Existem várias razões para considerar que os testes de agentes possuem um nível de dificuldade maior que os testes de softwares tradicionais (HOUHAMDI, 2011b). Agentes de software autônomos diferem de softwares tradicionais, pois eles têm seus próprios objetivos e operam de maneira autônoma. Como resultado, a resposta (também conhecida como saída dos testes) que o agente exerce pode ser diferente em diferentes casos, mesmo que o estado do agente seja o mesmo no momento da execução (NGUYEN et al., 2009).

Para ilustrar a complexidade envolvida no teste de agentes autônomos em comparação aos sistemas não autônomos, consideremos um exemplo. Um componente que fornece a funcionalidade de compressão é testado conforme dois critérios: (1) a saída da função compressão é menor que a entrada; (2) a saída da função compressão, quando dado como entrada a função descomprimir, retorna a entrada original. Neste simples exemplo de

teste em um programa não autônomo, o testador pode não conhecer o algoritmo de compressão usado pelo componente e não conhecer exatamente os dados gerados como resultado da função comprimir. No entanto, ele pode esperar que a saída seja a mesma para uma determinada entrada, isto é, o contexto da execução no qual os testes são realizados não altera o resultado do teste (NGUYEN et al., 2009).

Em comparação, o mesmo teste pode ser aplicado por um cliente com um agente autônomo para comprimir dados, mas existem variações no atual comportamento do agente. O agente pode ter seus próprios objetivos internos e conhecimentos, os quais podem ser alterados durante a execução, e estes podem afetar o resultado retornado, se houver. Por exemplo, o agente pode escolher diferentes algoritmos de compressão baseado nos recursos disponíveis, ou delegar a tarefa para diferentes subordinados, dependendo de quem atualmente fornece o melhor serviço. Para testar corretamente o agente autônomo é requerido mais que um único teste no componente, requer que o mesmo teste seja aplicado em diferentes contextos. Garantir a variedade de contextos testados para declarar que o agente se comporta corretamente é uma tarefa difícil, mas importante (NGUYEN et al., 2009).

Testar um único agente é diferente de testar uma comunidade de agentes. Ao testar um agente único, o desenvolvedor está mais interessado na funcionalidade do agente e se o agente atua por um conjunto de mensagens, entradas de contexto e condições de erro. Mas, ao testar uma comunidade de agentes, o testador está interessado em saber se os agentes operam em conjunto, são coordenados, e se a passagem de mensagens entre eles é correta (HOUHAMDI, 2011b).

Considerando os testes de agentes, Nguyen (2008) afirma que alguns trabalhos classificam os testes de agentes em diferentes níveis: unitário, agente, integração ou grupo, sistema ou sociedade e aceitação. Os objetivos dos testes, assim como as atividades de cada nível de teste são descritas abaixo:

- **Unitário:** Testa todas as unidades que compõem o agente, incluindo blocos de código, implementação de unidades do agente como objetivos, planos, base de conhecimento, raciocínio, regras, e assim por diante; para certificar se as unidades funcionam conforme projetado.
- **Agente:** Testa a integração de diferentes módulos dentro do agente; testa a capacidade dos agentes em cumprir seus objetivos e de perceber e agir no ambiente.

- **Integração ou Grupo:** Testa a integração do agente, comunicação, protocolo e semântica, interação dos agentes com o ambiente, integração dos agentes com os recursos compartilhados. Assegura que um grupo de agentes e os recursos do ambiente trabalham corretamente juntos.
- **Sistema ou Sociedade:** Testa as propriedades do Sistema Multi-Agente (SMA); testa as propriedades que o sistema deverá atingir, tais como adaptação, tolerância a falhas, desempenho.
- **Aceitação:** Testa o SMA no ambiente do cliente e verifica se atende às especificações definidas pelo cliente.

2.5 Metaheurísticas

Nem sempre é possível encontrar a melhor solução de um problema de otimização, em tempo razoável, através de algoritmos exatos. No entanto, encontrar uma solução relativamente boa pode ser suficiente para o problema em questão. Assim, existem inúmeras técnicas que produzem resultados considerados satisfatórios. Tais técnicas não garantem que a melhor solução possível seja encontrada (ou seja, a inexistência de uma solução melhor não é garantida), pois para alguns problemas, é impossível pesquisar, em tempo hábil, todo o universo de soluções existentes. O objetivo de tais técnicas é fazer uma redução do espaço de busca do problema, de forma a se obter um bom resultado em tempo aceitável. A todos os métodos aproximativos, desenvolvidos exclusivamente para resolver certo tipo de problema a um custo computacional razoável, dá-se o nome de heurísticas. Assim, os métodos heurísticos não garantem encontrar a solução ótima de um problema, mas são capazes de obter soluções de qualidade em um tempo adequado às necessidades da aplicação. As heurísticas de propósito geral, desenvolvidas para resolução de problemas difíceis, são denominadas Metaheurísticas (VIANA , 1998).

A palavra heurística, do grego *heuriskein*, significa a arte de descobrir novas estratégias (regras) para resolver problemas. O prefixo *meta*, do grego, significa “metodologia de nível superior”. O termo metaheurística foi introduzido por Glover (1986), definido como metodologias de nível superior que podem ser usadas como orientação na concepção de estratégias heurísticas subjacentes para resolver os problemas de otimização específicas (TALBI, 2009).

Na concepção de uma metaheurística, devem ser tomados dois critérios: (i) a exploração do espaço de busca (diversificação) – *exploration* e (ii) a exploração das melhores

soluções encontradas (intensificação) – *exploitation*. Na diversificação, regiões não exploradas devem ser visitadas para ter a certeza de que todas as regiões do espaço de busca são uniformemente exploradas e que a busca não se limita apenas a um reduzido número de regiões. Na intensificação, as regiões promissoras (regiões que obtém "boas" soluções) são exploradas mais a fundo na esperança de encontrar melhores soluções (TALBI, 2009).

Metaheurísticas baseadas em população (*P-metaheuristics*) permitem uma melhor diversificação em todo o espaço de busca e podem ser vistas como uma melhoria iterativa em uma população de soluções. Inicialmente, uma população é construída. Em seguida, uma nova população de soluções é gerada. Finalmente, esta nova população está integrada na atual utilizando alguns procedimentos de seleção. O processo de busca é interrompido quando uma determinada condição é satisfeita (critério de parada). Algoritmos evolutivos, algoritmos de formigas, pesquisa de dispersão, algoritmos de distribuição, de otimização por enxame de partículas, colônia de abelhas e sistemas imunológicos artificiais pertencem a esta classe de metaheurísticas (TALBI, 2009).

Algoritmos evolucionários (AE) ou evolutivos são metaheurísticas capazes de otimizar problemas que seriam difíceis de ser resolvidos com o uso de técnicas convencionais, como programação linear e não-linear. Um AE inicia a busca com uma população de soluções geralmente gerada de forma aleatória nos limites impostos às variáveis (superior e inferior). A partir da população inicial são geradas as populações descendentes utilizando operadores de seleção, mutação e cruzamento, e um operador para preservar os melhores indivíduos, ou seja, um operador de elitismo (SIMONOVIC, 2009; DEB, 2008).

A seguir são apresentadas duas importantes metaheuríticas, baseadas em algoritmos evolucionários, para a resolução de problemas de otimização, as quais foram utilizadas na seleção de casos de teste na abordagem proposta.

2.5.1 Algoritmos Genéticos

O *Genetic Algorithm* (GA), proposto por Holland (1975), foi um dos primeiros algoritmos da categoria dos AE e faz analogia aos mecanismos de seleção natural. Holland (1975) incorporou características da evolução das espécies para desenvolver um método flexível, robusto (independentemente do ponto de partida, os GAs são capazes de encontrar soluções de boa qualidade) e computacionalmente simples de implementar para resolver problemas complexos (SOUSA, 2012).

Um GA convencional é constituído por quatro etapas: Geração da população inicial, Cruzamento, Mutação e Seleção. Para gerar uma nova população, no GA são executados os três operadores: seleção, cruzamento e mutação. Através dos operadores de cruzamento, mutação e seleção é possível melhorar os resultados entre uma geração e outra. São eles que atribuem ao algoritmo a capacidade de evoluir (SCHARDONG, 2011).

O GA funciona com uma população inicial, formada por cromossomos, que por sua vez são formados pelos genes (cadeia de símbolos). Os cromossomos podem ter, dentre outras representações, representação binária ou real, definindo assim dois tipos de algoritmos, de representação real e de representação binária (SCHARDONG, 2011).

Nos GAs, a cada indivíduo deve ser atribuído um valor que indica sua adaptação, ou seja, quando se trata de um problema a ser resolvido, o valor de adaptação indica o custo da solução que o indivíduo representa. A cada interação dos GAs, também chamada de geração, um conjunto de indivíduos formará uma população. Um dos parâmetros mais importantes da população é seu tamanho, que geralmente se mantém constante durante toda a execução (SOUSA, 2012).

Uma vez que o GA é baseado no princípio da seleção natural, eles devem ser capazes de identificar os indivíduos mais adaptados, que representam soluções de boa qualidade para o problema, para que estes possam ser combinados para gerar novos descendentes. Este processo é guiado por um método de seleção (SOUSA, 2012). O operador de seleção é usado para selecionar os cromossomos aos quais serão aplicados os operadores de cruzamento. Existem diferentes operadores de seleção e em geral cromossomos com valor maior de função objetivo, em um problema de maximização, têm maiores chances de serem selecionados (DEB, 2008).

O GA proposto por Holland (1975) utiliza um método de seleção de indivíduos para a próxima geração, chamado técnica da roleta. A técnica da roleta atribui a cada indivíduo de uma população uma probabilidade de passar para a próxima geração que é proporcional ao *fitness* do indivíduo. Assim, quanto maior o *fitness* de um indivíduo, maior a probabilidade deste passar para a próxima geração. Para a seleção do indivíduo, cria-se uma roleta (virtual) que seleciona aleatoriamente um número entre 0 e a soma total do *fitness* dos indivíduos. É selecionado o indivíduo sobre o qual a roleta parar. Sendo assim, a seleção de indivíduos pela técnica da roleta pode fazer com que o melhor indivíduo da população seja perdido, ou seja, não passe para a próxima geração (MICHALEWICZ, 1996).

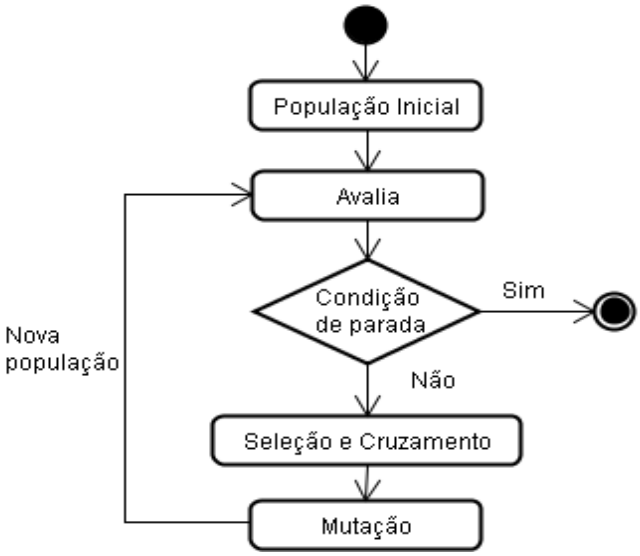
Uma alternativa é aplicar o elitismo que tem como objetivo evitar a perda dos melhores indivíduos encontrados durante a busca. Uma forma simples de implementar o elitismo é selecionar τ indivíduos da população e copiá-los para a próxima geração, sem nenhuma modificação. Os outros $N - \tau$ indivíduos são gerados normalmente considerando os operadores genéticos de seleção, cruzamento e mutação (SOUSA, 2012).

Back, Fogel e Michalewicz (1997) relatam outros mecanismos de seleção, sendo que dentre esses destacam-se a baseada em *rank* e seleção por torneio. A primeira estratégia utiliza as posições dos indivíduos ordenados de acordo com o *fitness* para determinar a probabilidade de seleção. Podem ser usados mapeamentos lineares ou não-lineares para determinar a probabilidade de seleção. A segunda, um número *m* de indivíduos da população é escolhido aleatoriamente para formar uma sub-população temporária. Deste grupo, o melhor indivíduo é selecionado. Assim, escolhe-se cada indivíduo que irá compor o grupo de *N* indivíduos selecionados (FACCIOLI, 2012).

Para gerar uma nova população, os operadores de cruzamento e mutação são importantes. O operador de cruzamento é responsável pela recombinação de características dos pais durante a reprodução, permitindo que as próximas gerações herdem essas características. O operador de mutação consiste em mudar aleatoriamente um ou mais genes de um indivíduo da população. A taxa de mutação é a probabilidade de ocorrência de mutação em um gene. O operador de mutação é usado para garantir diversificação da população e impedir, desta forma, a convergência prematura para um ótimo local (SOUSA, 2012).

A FIGURA 8 apresenta de forma simplificada as principais etapas de um GA que são executadas durante o processo de otimização.

FIGURA 8 – Fluxograma geral de um algoritmo genético



Fonte: Adaptado de Schardong (2011)

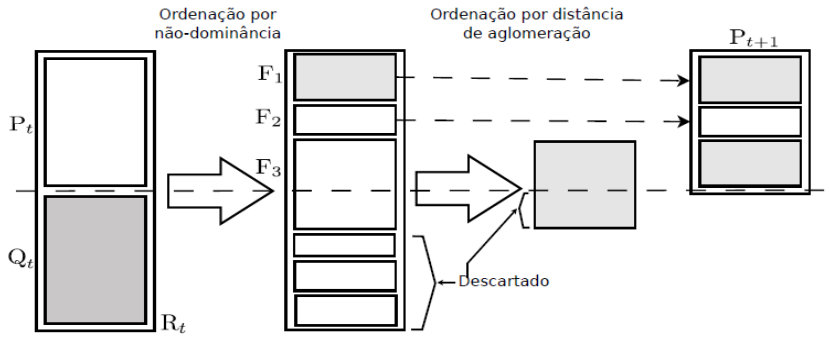
2.5.2 NSGA-II

Proposto por Srinivas e Deb (1994), o algoritmo *Nondominated Sorting Genetic Algorithms* (NSGA) é baseado na classificação dos indivíduos da população em diversos níveis de não-dominância; varia do algoritmo genético simples apenas na forma em que o operador de seleção trabalha, uma vez que os operadores de cruzamento e mutação são aplicados da mesma forma do GA tradicional.

O algoritmo NSGA-II, proposto por Deb et al. (2000), é uma modificação do NSGA que usa (i) um método rápido de ordenação baseado em não-dominância; (ii) uma abordagem elitista e (iii) um método eficiente para determinar a distância de aglomeração. O NSGA-II aplica um mecanismo de elitismo que consiste em combinar os melhores pais com os melhores filhos obtidos (DEB et al., 2000).

O NSGA-II começa com uma população inicial P_1, de tamanho M, da qual uma descendência Q_t é criada. Ambas as populações são combinadas formando uma população $R_t = P_t \cup Q_t$ de tamanho $2M$. Para as seguintes gerações, $n = 1, 2, ...,$ o algoritmo NSGA-II trabalha com a população R_t. Um algoritmo de ordenação por não-dominância é aplicado à população R_t, obtendo as fronteiras F_1, F_2, ... e todos estes conjuntos são inseridos na nova população P_{t+1}. Considerando que apenas N soluções podem ser inseridas na população P_{t+1}, N soluções de R_t são descartadas. Para preencher as P_{t+1}, começa-se com as soluções em F_1; se não forem contempladas as N soluções, prossegue-se com F_2 e, assim por diante. Cada conjunto F_i deve ser inserido na sua totalidade em P_{t+1}, isto ocorre quando $|P_{t+1}| + |F_i| \leq N$. Quando ocorre o caso de ao inserir F_j a $|F_j| > N - |P_{t+1}|$, o algoritmo NSGA-II seleciona então as soluções de F_j que estejam melhor diversificadas (SOUSA, 2012; FACCIOLI, 2012). A FIGURA 9 ilustra este procedimento de definição dos indivíduos que deverão passar para a próxima geração.

FIGURA 9 – Ilustração do algoritmo NSGA-II



Fonte: Sousa (2012)

O algoritmo NSGA-II emprega um método chamado de distância de multidão (*crowding distance*). Tendo obtidas as distâncias, os conjuntos de soluções F_j são ordenados decrescentemente em relação as suas distâncias, e copiam-se as primeiras N - $|P_{t+1}|$ soluções de F_j para P_{t+1}. Finalmente, obtém-se Q_{t+1} a partir de P_{t+1} usando os operadores de seleção por torneio, cruzamento e mutação (FACCIOLI, 2012).

A forma como é mantida a diversidade entre as soluções não dominadas é a principal vantagem do NSGA-II. O método de comparação por multidão é utilizado para a seleção por torneio e para escolher os elementos da fronteira F_j. Se o conjunto F_1 tem um tamanho maior que N, será então executado o processo de escolher apenas N soluções, pois utilizando-se a distância de multidão faz com que sejam perdidas algumas soluções. Seja um F_1 onde existam várias soluções Pareto-ótimas muito próximas e alguma solução distante não Pareto-ótima, mas não dominada no momento. Considerando que o cuboide da solução não dominada é maior, esta solução será copiada em P_{t+1}, enquanto que uma solução Pareto-ótima é eliminada. Esta situação faz com que o NSGA-II possa cair em um ciclo de gerar soluções Pareto-ótimas e não Pareto-ótimas até convergir finalmente a um conjunto de soluções Pareto-ótimas (DEB et al., 2000).

3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma revisão bibliográfica sobre as principais abordagens presentes na literatura para realizar os testes em agentes (independentemente de suas características e de sua arquitetura interna). É apresentado um resumo das abordagens descritas com suas principais características e a diferenciação entre as abordagens e a abordagem proposta.

Na literatura de engenharia de software orientada a agentes, as pesquisas tem se concentrado no estudo de como os agentes interagem, no desenvolvimento de arquiteturas e de protocolos para os sistemas de agentes. Além disso, tem sido desenvolvida uma variedade de pesquisas abordando diferentes aspectos de teste de agentes. Cada uma das quais com diferentes abordagens e perspectivas. Entretanto, o teste de agente é uma atividade desafiadora e um processo de teste estruturado para agentes de software ainda está ausente (HOUHAMDI, 2011b).

A análise dos trabalhos relacionados cobre algumas abordagens propostas na literatura para a realização dos testes nos agentes. Neste capítulo, serão verificadas se as abordagens são capazes de atender aos seguintes critérios: (i) noção de agentes racionais de Russell e Norvig (2004), (ii) utilização de casos de teste gerados de acordo com os objetivos, (iii) medida de avaliação de desempenho do agente testado, (iv) consideração dos planos que são necessários para que este agente alcance estes objetivos, (v) simulação da avaliação de desempenho das interações do agente testado com seu ambiente (histórias) e (vi) utilização de estratégias de busca local multiobjetivo orientada por utilidade para encontrar casos de teste e histórias correspondentes em que o agente não foi bem avaliado.

3.1 Abordagem de Testes Orientada a Objetivos

Uma abordagem de teste orientada a objetivos para os agentes é apresentada por Houhamdi (2011b). Na abordagem proposta é especificado um processo de teste que complementa a metodologia Tropos (MYLOPOULOS; CASTRO, 2000) e reforça a relação mútua entre a análise de objetivos e testes. A abordagem propõe a derivação de casos de teste a partir da análise de artefatos de requisitos orientado aos objetivos do agente para a realização de dois níveis de teste: unitário (certificar de que todas as unidades que fazem parte de um agente, tais como metas, planos, base de conhecimento, funcionam conforme projetado) e agente (testar a integração dos diferentes módulos dentro de um agente).

Houhamdi (2011b) fornece um processo estruturado e abrangente para a derivação de casos testes a partir de artefatos modelados conforme a análise de objetivos e planos dos agentes, produzidos juntamente com o processo de desenvolvimento. Este conjunto de casos de teste, por um lado, pode ser utilizado para aperfeiçoar a análise dos objetivos e detectar problemas no início do processo de desenvolvimento. Por outro lado, são executados para testar a realização dos objetivos dos quais foram derivados.

Especificamente, a metodologia proposta contribui para as metodologias *Agent-Oriented Software Engineering* (AOSE) existentes, fornecendo: (i) um modelo de processo de teste, que contempla a metodologia de desenvolvimento, relacionando os objetivos e os casos de teste; e (ii) uma forma sistemática para derivar casos de teste a partir da análise dos objetivos.

Apesar de utilizar a análise de objetivos, essa estratégia não apresenta: (i) a noção de agentes racionais de Russell e Norvig (2004); (ii) uma medida de avaliação de desempenho; e (iii) uma simulação que possa monitorar o comportamento do agente ao executar as ações que envolvem os objetivos do agente.

3.2 Proposta do Componente Agente Testador

Rouff (2002) aborda os desafios relacionados aos testes em sistemas multi-agentes, apresentando os testes em agentes individuais e em comunidades de agentes. Neste contexto, é proposto um agente especial testador, usado para testar os agentes individualmente ou na comunidade a qual pertencem, de modo que possa instituir a confiança de que os sistemas baseados em agentes funcionam corretamente e que os erros podem ser encontrados rapidamente.

O agente de teste pode ler a especificação de cada agente ou sistema autônomo e produzir casos de teste para executar os agentes individualmente e em conjunto, como comunidades de cooperação e coordenação de sistemas autônomos. Além disso, o agente de teste é capaz de monitorar os agentes depois que eles são implantados para garantir a funcionalidade adequada.

Mais especificamente, o agente testador proposto na abordagem é capaz de testar os seguintes aspectos de agentes individuais e de comunidades de agentes:

- testar a capacidade de um único agente para enviar ou receber uma mensagem específica;
- testar a capacidade de um único agente para lidar com mensagens inválidas;

- testar a capacidade da comunidade para lidar com mensagens válidas e inválidas;
- manter as especificações da mensagem;
- coletar as métricas;
- monitorar o sistema de agentes em relação a possíveis erros e problemas de desempenho.

A partir da análise dos critérios para avaliação das abordagens, a abordagem de Rouff (2002) não trata: (i) a noção de agentes racionais de Russell e Norvig (2004); (ii) utilização de casos de teste gerados de acordo com os objetivos; (iii) simulação da avaliação de desempenho das interações do agente testado com seu ambiente (histórias); (iv) utilização de estratégias de busca local multiobjetivo orientada por utilidade para encontrar casos de teste e histórias correspondentes em que o agente não foi bem avaliado.

Nguyen, Perini e Tonella (2007) compartilham do mesmo propósito de ter um agente testador, no entanto, apresentam novas perspectivas. Um dos principais componentes é o agente autônomo testador, que é capaz de gerar automaticamente novos testes e executá-los. Nessa abordagem, o agente testador incorpora a responsabilidade de monitoração das ações executadas pelo agente testado.

A abordagem propõe um *framework* para testes de sistemas multi-agentes que facilita a derivação de casos de teste a partir da análise dos objetivos seguindo a metodologia de teste orientada por objetivo, por meio da geração semi-automática de esqueletos de teste a partir de diagramas de análise de objetivos. O *framework* fornece uma interface que facilita a entrada de dados de teste. A partir desses, o *framework* pode evoluir e gerar mais entradas de teste por evolução e mutação, e executar essas entradas de teste para testar continuamente o SMA.

Além de apresentarem um agente testador para a realização dos testes em sistemas baseados em agentes, os autores usam a combinação de testes evolucionários (PARGAS; HARROLD; PECK, 1999) e mutação (DEMILLO; LIPTON; SAYWARD, 1978; HAMLET, 1977), para a geração de casos de testes executados pelo agente testador. A abordagem faz uso de um *score* para a mutação como uma medida de aptidão para guiar a evolução, que quantifica os mutantes inadequados. Um mutante é uma versão modificada do agente original contendo uma falha. Um mutante é considerado inadequado por uma entrada de teste se a entrada causa no mutante um comportamento diferente do desejado. Neste contexto, os casos de testes que tornam a maior quantidade de mutantes inadequados serão susceptíveis de revelar falhas reais nos agentes originais.

Nguyen, Perini e Tonella (2007) abordam o problema dos testes de autonomia indiretamente, usando restrições e ao fato de que, enquanto os agentes de software são livres para evoluir, o seu comportamento deve obedecer às normas e regras que regem o funcionamento do sistema em que os agentes estão situados, ou as restrições impostas ao comportamento.

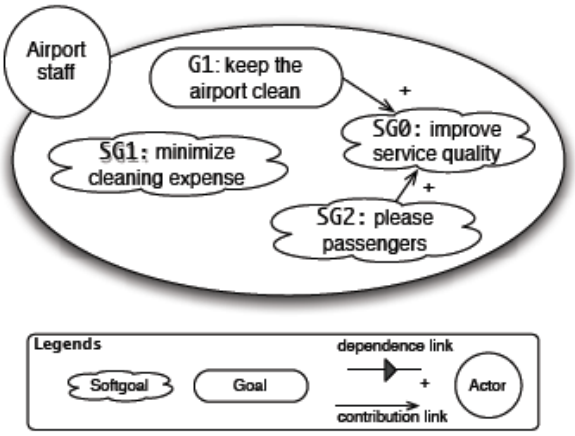
A abordagem proposta por Nguyen, Perini e Tonella (2007) não abrange os seguintes critérios: (i) a noção de agentes racionais de Russell e Norvig (2004); e (ii) simulação da avaliação de desempenho das interações do agente testado com seu ambiente (histórias).

3.3 Testes Evolucionários para Agentes Autônomos

Nguyen (2008) desenvolveu uma abordagem de análise orientada a objetivos vinculada aos testes evolucionários, visando um processo de testes sistemático e abrangente para agentes, que engloba o processo de desenvolvimento do agente de acordo com a metodologia Tropos (BRESCIANI et al., 2004) a partir da análise de requisitos iniciais até a implantação. É proposta uma metodologia de como produzir artefatos de testes, ou seja, entradas, cenários, e assim por diante, a partir das especificações dos agentes e do *design*, e usa estes artefatos para detectar problemas antecipadamente. Especificamente, a metodologia proposta contribui para as metodologias AOSE (HENDERSON-SELLERS; GIORGINI, 2005) existentes, fornecendo: (i) um modelo de processo de testes, que complementa a metodologia de desenvolvimento, traçando uma ligação entre objetivos e casos de teste e (ii) de forma sistemática para derivar casos de teste a partir da análise dos objetivos.

Os autores exemplificam a abordagem através de um sistema multi-agente composto de vários agentes aspirador que tem como objetivo manter o aeroporto limpo, além disso, os agentes do sistema precisam colaborar para otimizar o trabalho e serem gentis com os passageiros. Seguindo as diretrizes da metodologia Tropos (BRESCIANI et al., 2004), os requisitos iniciais são analisados, identificados os objetivos associados aos agentes e, então, decompostos em sub-objetivos, conforme apresentado na FIGURA 10.

FIGURA 10 – Análise dos requisitos iniciais do agente aspirador



Fonte: Nguyen (2008)

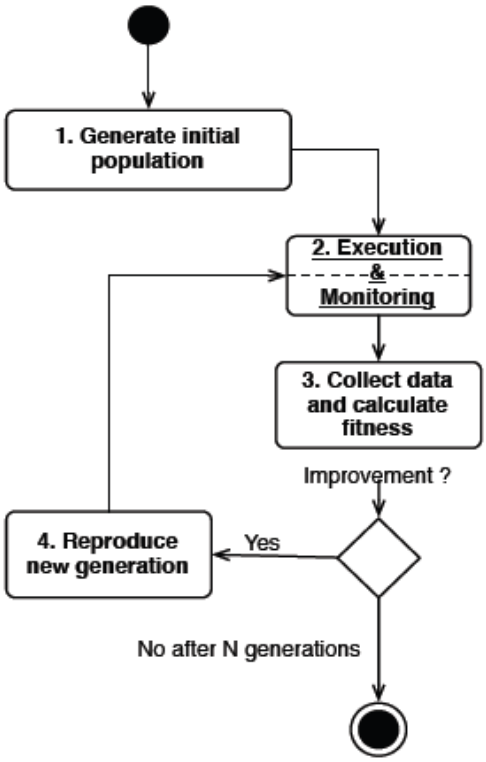
A proposta divide o teste do SMA em diferentes níveis: unitário, agente, integração, sistema e aceitação. A partir da análise dos objetivos do agente, os casos de teste são gerados automaticamente e evoluem continuamente guiados pela mutação e função de qualidade. A estratégia de evolução é conduzida pelos seguintes passos: gerar uma população (um conjunto de casos de teste chamado de população pode ser gerado randomicamente ou obtido a partir de casos de teste gerados pelos testadores); execução e monitoramento (os agentes são colocados em teste e a execução é observada); coleta dos dados e cálculo do valor de *fitness* (o valor de *fitness* é calculado cuja forma depende do objetivo e do domínio do problema); reprodução (dois indivíduos são selecionados e a operação de crossover é utilizada para produzir dois novos indivíduos).

A abordagem de Nguyen (2008) é bem estruturada, no entanto, considerando os critérios estabelecidos de avaliação, não trata: (i) a noção de agentes racionais de Russell e Norvig (2004); e (ii) uma simulação que possa monitorar o comportamento do agente ao executar uma ação que envolve os objetivos.

Uma abordagem evolucionária para a realização dos testes de agentes autônomos também é adotada por Nguyen et al. (2009). A metodologia proposta consiste em dois passos principais: (i) representação dos objetivos como função de qualidade (os objetivos que são necessários para avaliar o agente são transformados ou representados como função de qualidade para mensurar a satisfação dos *stakeholders*); e (ii) testes evolucionários (para gerar uma variedade de casos de teste com alto nível de dificuldade, a abordagem faz uso de metaheurísticas de algoritmos de busca que têm sido usadas em trabalhos de SBSE, mais especificamente, são usados algoritmos evolucionários guiados pela função de qualidade).

No processo de geração de casos de teste, realizado a partir dos testes evolucionários, são definidos quatro passos principais (conforme FIGURA 11): (i) geração da população inicial (um conjunto de casos de teste inicial é gerado randomicamente ou especificado pelo testador); (ii) execução e monitoração (a execução do teste consiste em inserir o agente no ambiente de teste; o mecanismo de monitoração é necessário para observar o comportamento do agente); (iii) coleta dos dados observados e cálculo do valor de *fitness* (os dados de todas as execuções são usados para calcular o *fitness* dos casos de teste selecionados); e (iv) reprodução (dois indivíduos são selecionados e os operadores de cruzamento e mutação são usados para produzir uma nova prole).

FIGURA 11 – Processo de evolução dos casos de teste



Fonte: Nguyen et al. (2009)

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 o número de testes com diferentes níveis de dificuldade são executados. Os agentes são recrutados apenas quando passam pelo critério de qualidade definido. Os algoritmos genéticos são utilizados para gerar uma variedade de testes com aumento do nível de dificuldade. As funções de qualidade são usadas como funções objetivo para guiar a pesquisa gerando mais desafios para os testes.

Na abordagem de Nguyen et al. (2009), considerando os critérios avaliados, não é tratado: (i) a noção de agentes racionais de Russell e Norvig (2004) e (ii) uma simulação que possa monitorar o comportamento do agente ao executar uma ação que envolve os objetivos.

3.4 Comparativo entre os Trabalhos Relacionados

A TABELA 1 apresenta o comparativo entre os trabalhos relacionados apresentados neste capítulo. Para a comparação, foram levados em consideração os critérios para avaliação das abordagens definidos no início deste capítulo.

TABELA 1 – Comparativo entre os Trabalhos Relacionados

Critérios Avaliados	**Trabalhos Relacionados**					
	(ROUFF, 2002)	(NGUYEN; PERINI; TONELLA, 2007)	(NGUYEN, 2008)	(NGUYEN et al., 2009)	(HOUHAMDI, 2011b)	Nossa abordagem
Noção de agentes racionais de Russell e Norvig (2004)						●
Utilização de casos de teste gerados de acordo com os objetivos		●	●	●	●	●
Medida de avaliação de desempenho do agente testado	●	●	●	●		●
Consideração dos planos que são necessários para que este agente alcance estes objetivos	●	●	●	●	●	●
Simulação da avaliação de desempenho das interações do agente com o ambiente						●
Utilização de estratégias de busca local multiobjetivo orientada por utilidade		●	●	●	●	●

Fonte: Elaborado pelo autor (2013)

3.5 Conclusão

A partir da análise dos trabalhos relacionados, foi constatado que nenhuma das abordagens analisadas pode atender a todos os critérios definidos no início deste capítulo. Portanto, a definição de uma abordagem capaz de realizar esses critérios é essencial, pois abrange a realização dos testes para uma variedade de agentes projetados conforme as arquiteturas de agentes racionais definidas por Russell e Norvig (2004) e é destacável a identificação dos objetivos que não estão sendo satisfeitos e das falhas apresentadas pelo

agente, conforme a evolução dos casos de testes guiados pela medida de avaliação e pelos objetivos do agente. Além do mais, a simulação da avaliação de desempenho das interações do agente testado com seu ambiente possibilita a identificação, pelo projetista, das falhas em cada ação escolhida pelo agente para executar em determinado estado do ambiente.

Devido ao alto nível de dificuldade em testar os sistemas com agentes, outro critério que merece destaque, embora não seja exclusivo da abordagem proposta, é a utilização das estratégias de busca local multiobjetivo orientada pela função de utilidade projetada a partir dos objetivos do agente. A utilização de metaheurísticas possibilita a seleção objetiva de casos de testes, com diferentes níveis de complexidade e dificuldade, a partir da utilidade associada a cada um destes, conforme indicado pela medida de avaliação de desempenho obtida.

4 ABORDAGEM PARA O TESTE DE AGENTES RACIONAIS

Este capítulo apresenta a abordagem proposta para o teste de agentes racionais. Esta abordagem considera que além do Projetista existem quatro programas agentes envolvidos, ou seja, o programa a ser testado *Agent* concebido pelo projetista, o programa ambiente de tarefa *Amb*, um programa agente de resolução de problemas de seleção de casos de testes *Thestes* e um programa agente monitorador *ProMon*. Inicialmente o capítulo apresenta a definição de alguns aspectos envolvidos e uma visão geral da abordagem proposta para o teste de agentes racionais. Em seguida, é apresentada uma descrição da estrutura do programa *Thestes* e um esboço do monitorador *ProMon*.

4.1 Aspectos Considerados

Considerando que os testes dos agentes racionais consistem em identificar ambientes específicos em que o agente não foi bem avaliado, este capítulo apresenta inicialmente alguns aspectos que influenciam nessa definição e que foram considerados na abordagem proposta.

4.1.1 Casos de Teste

Na abordagem proposta, um caso de teste consiste em uma representação do ambiente em que o agente deverá atuar para realizar seus objetivos. Diferentes representações do ambiente apresentam diferentes níveis de dificuldade em que o agente deverá operar. Por exemplo, considerando um agente aspirador de pó em um mundo simplificado, o ambiente consiste em uma área contendo *n* salas, que podem estar sujas ou limpas. A FIGURA 12 apresenta uma configuração do ambiente, ou seja, um caso de teste para o mundo aspirador de pó, onde “L” representa sala limpa e “S” representa sala suja. Outros casos de teste podem ser representados modificando-se a localização e a quantidade dos elementos no ambiente.

FIGURA 12 – Representação de um caso de teste para o mundo aspirador de pó

L	S	S	L	L
S	L	S	L	L
L	L	S	L	S
S	L	L	S	L
S	L	L	L	S

Fonte: Elaborado pelo autor (2013)

4.1.2 História de um Agente em um Ambiente de Tarefa

Seja um agente a ser testado, *Agent*: $P^* \rightarrow A$, em um ambiente, *Amb*: $PxA \rightarrow P(P)$, configurado de acordo com um caso de teste específico *i*, $Caso_i$, e $Percepção^0$ a representação do estado inicial do ambiente em Caso. Uma sequência da forma:

$$h(Caso_i): (Percepção^0, Ação^0) \rightarrow (Percepção^1, Ação^1) \rightarrow \ldots \rightarrow (Percepção^K, Ação^K) \rightarrow \ldots$$

representará uma história possível de *Agent* em *Amb* configurado em um caso de teste $Caso_i$ se, e somente se, as duas condições abaixo forem válidas:

1. $\forall k \in N$, $Ação^K = Agent((Percepão^0, Percepão^1, ..., Percepção^k))$
2. $\forall k \in N$ tal que $K>0$, $Ação^K \in Amb(Percepção^{(K-1)}, Ação^{(K-1)})$

Assim, a experiência do agente pode ser dividida em episódios atômicos, representados formalmente por meio de pares ordenados do tipo $Ep^K = (Percepção^K, Ação^K)$, cujos elementos representam a percepção e a ação do agente na *K*-ésima interação com o ambiente.

4.1.3 Avaliação de Desempenho

A TABELA 2 exemplifica uma medida de avaliação de desempenho para o caso de um programa agente aspirador de pó que deve limpar um ambiente e maximizar os níveis de limpeza do ambiente e de energia em sua bateria ao final da tarefa. A primeira coluna descreve uma parte das informações nas percepções do agente em cada episódio possível. A segunda coluna descreve as ações possíveis nesses episódios (Aspirar (Asp), Direita (Dir), Esquerda (Esq), Acima (Ac), Abaixo (Ab) e Não operar (N-op)). Na terceira e quarta colunas respectivamente, associadas aos objetivos de energia e de limpeza, duas funções escalares (av_E e av_L) para medir o desempenho do agente em cada episódio de sua história no ambiente.

TABELA 2 – Medida de Avaliação de Desempenho

$Ep^k = (Percepção^K,$	**$Ação^K)$**	**$av_E(Ep^k)$**	**$av_L(Ep^k)$**
..., L, ...	Asp	-1.0	0.0
..., L, ...	Dir, Esq, Ac, Ab	-2.0	1.0
..., L, ...	N-op	0.0	0.0
..., S, ...	Asp	-1.0	2.0
..., S, ...	Dir, Esq, Ac, Ab	-2.0	-1.0
..., S, ...	N-op	0.0	-1.0
...	...	...	...

Fonte: Elaborado pelo autor (2013)

A TABELA 3 ilustra seis episódios fictícios de uma história das interações de *Agent* em *Amb*, configurado em um caso de teste $Caso_i$ e as medidas de desempenho de *Agent* em termos de energia e limpeza em cada episódio.

TABELA 3 – História parcial de *Agent* em *Amb*

Ep^k	$Percepção^k$	$Ação^k$	$av_E(Ep^k)$	$av_L(Ep^k)$
1	..., L, ...	Ab	-2.0	1.0
2	..., L, ...	Dir	-2.0	1.0
3	..., L, ...	Ab	-2.0	1.0
4	..., S, ...	Asp	-1.0	2.0
5	..., L, ...	Esq	-2.0	1.0
6	..., L, ...	Ab	-2.0	1.0
	...	...	...	...

Fonte: Elaborado pelo autor (2013)

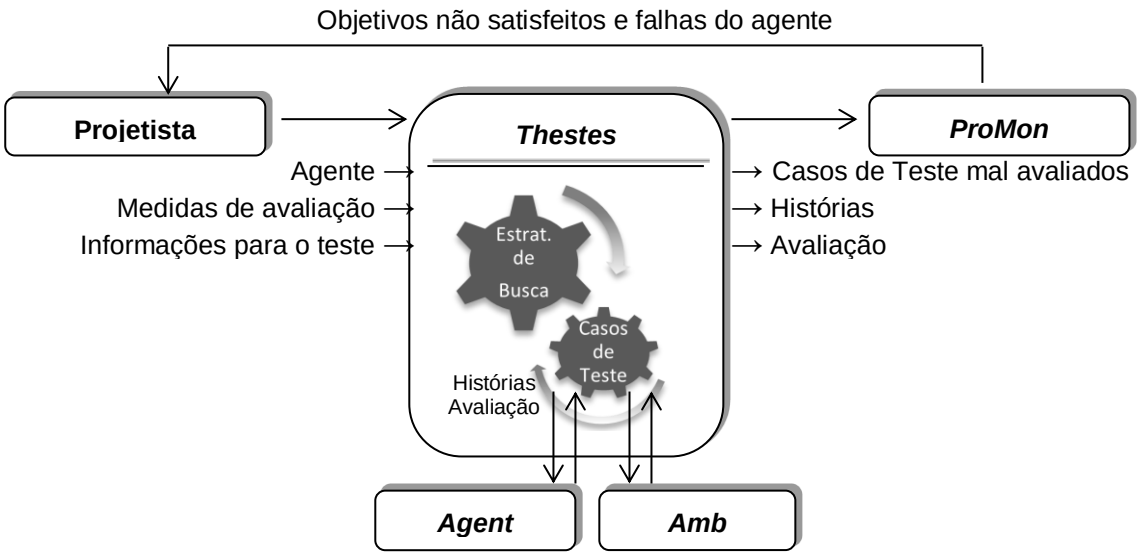
O projetista de um agente racional deve considerar esse tipo de medida de desempenho, as propriedades do ambiente de tarefa, as estruturas de agentes disponíveis e conceber o programa visando maximizar seu desempenho nas histórias do agente no ambiente.

4.2 Visão Geral da Abordagem

A abordagem proposta para o teste de agentes racionais está fundamentada, principalmente, na noção de agentes racionais, na utilização de casos de teste gerados de acordo com os objetivos na medida de avaliação de desempenho do agente testado, na simulação da avaliação de desempenho das interações do agente testado com seu ambiente (histórias) e em estratégias de busca local multiobjetivo orientada por utilidade para encontrar casos de teste e histórias correspondentes em que o agente não foi bem avaliado.

Mais especificamente, a abordagem considera que além do Projetista existem mais quatro programas agentes envolvidos, ou seja, o programa a ser testado *Agent* concebido pelo projetista, o programa ambiente de tarefa *Amb*, um programa agente para a seleção de casos de testes *Thestes* e um programa agente monitorador *ProMon*. A FIGURA 13 ilustra as interações entre estes agentes.

FIGURA 13 – Visão geral da abordagem



Fonte: Elaborado pelo autor (2013)

O Projetista é responsável por conceber o programa agente racional *Agent*, a medida de avaliação de desempenho e alimentar outras informações necessárias para o agente *Thestes* iniciar o processo de teste de *Agent* em *Amb*. O agente *Thestes* consiste em um agente de resolução de problemas de seleção de casos de teste que realiza busca local no espaço de estados de casos de teste orientado por utilidade. Este agente envia para o agente *ProMon* um conjunto de soluções eficientes determinada pela estratégia de busca multiobjetivo, ou seja, descrevendo casos de teste em que *Agent* possui o comportamento mais inadequado, um conjunto de histórias correspondentes e seus valores de utilidade. O agente *ProMon* recebe essas informações e identifica para o Projetista: os objetivos na medida de avaliação que não estão sendo satisfeitos adequadamente pelo *Agent*, os episódios nas histórias de *Agent* em *Amb* que são falhas, ou seja, que estão "distantes" do ideal, e quais são os episódios ideais correspondentes.

4.3 Agente *Thestes*

O agente *Thestes* foi concebido como um agente de resolução de problemas de seleção de caso de testes. O esqueleto principal deste agente fundamenta-se na estrutura do programa agente orientado por utilidade. O programa emprega uma estratégia de busca local, baseada em populações e orientada por uma função utilidade para encontrar conjuntos de casos de teste satisfatórios, ou seja, ambientes específicos em que as histórias associadas de *Agent* em *Amb* possuem baixo desempenho. O agente utiliza o protocolo de interação entre *Agent* em *Amb* para realizar simulações e anotar as histórias correspondentes durante o

processo de busca de um conjunto de casos de teste. As próximas subseções especificam o problema de seleção e a estrutura do programa agente.

4.3.1 Formulação do Problema de Seleção de Casos de Teste para Agentes Racionais

A realização dos objetivos implícitos na medida estabelecida pelo projetista e, consequentemente, a racionalidade do programa agente, dependerá da adequação das propriedades de seu ambiente interno, herdadas de um ou mais tipos de programas, e dos meios que o projetista empregar na instanciação e execução do programa agente (arquitetura), às propriedades de seu ambiente externo, onde será realizada a tarefa. Considerando este ponto de vista, o processo de teste desses sistemas deve avaliar o desempenho do programa em seu ambiente de tarefa, identificando os objetivos que não estão sendo satisfeitos (estados desejados) no ambiente externo e os componentes do ambiente interno do programa que estão restringindo a satisfação desses objetivos.

Em um ambiente de tarefa simples, em que o programa é bem avaliado, o projetista só perceberá o comportamento racional do programa. Em um ambiente mais complexo, em que o agente não foi bem avaliado, o comportamento do programa realizará parcialmente os objetivos na medida de avaliação de desempenho. Assim, é importante que o projetista selecione casos de teste relevantes, que representem adequadamente as condições iniciais do ambiente de tarefa real do agente, e que lhe permita realizar uma verificação dinâmica do comportamento do programa agente no ambiente através da simulação das interações agente-ambiente e da geração de informações úteis sobre o desempenho do programa e sobre aspectos de sua estrutura interna que interferem negativamente em seu desempenho.

Por um lado, as informações sobre desempenho permitem que o projetista avalie a racionalidade das ações e planos (soluções) que o agente selecionou ao longo das interações que manteve com o ambiente de tarefa. Essa avaliação deve dizer quanto cada um dos objetivos implícitos na medida de avaliação de desempenho foi satisfeito em cada interação e a utilidade final da história do agente no ambiente. As informações sobre a estrutura interna do agente permitem ao projetista identificar os subsistemas de processamento de informações (ver, próximo e ação) e as informações processadas (efeito de ação, evolução do mundo, regras condição-ação, metas e utilidade) que estão interferindo no desempenho do agente.

Diante deste contexto, pode-se dizer que a eficácia do processo de teste do agente dependerá dos casos de teste selecionados pelo projetista. Nem sempre os melhores casos

estão disponíveis a priori e, dependendo do ambiente de tarefa do agente, pode existir uma grande quantidade de casos a serem observados. Considerando este ponto de vista, a seleção de um conjunto de casos de teste é um problema de busca em um espaço de estados composto por uma grande família de conjuntos de casos possíveis. Um caso de teste ótimo é aquele em que o agente obtém o valor mínimo possível de desempenho. Assim, diante deste contexto, o problema de seleção de conjuntos de casos de teste para um agente racional foi formulado como um problema de programação multiobjetivo (PMO). Seja:

- **Agent**: um programa agente racional a ser testado;
- **Amb**: um programa ambiente capaz de interagir com *Agent* por meio de algum protocolo;
- **ProtocoloInteração**: uma descrição do protocolo de interação entre *Agent* e *Amb*;
- **Ω**: um conjunto de descrições de ambientes de tarefa factíveis de instanciar em *Amb* e testar *Agent*;
- **P(Ω)**: subconjuntos de ambientes possíveis de serem descritos em Ω;
- **CasosTEST ∈ P(Ω)**: um subconjunto de descrições de ambientes casos de teste específico no conjunto P(Ω), onde:
  - **Caso_i ∈ CasosTEST**: uma descrição específica de ambiente no subconjunto *CasosTEST*;
- **H(CasosTEST) ∈ P($(\text{PxA})^{\text{NInt}}$)**: conjunto de histórias de comprimento NInt de *Agent* em *Amb* considerando *ProtocoloInteração* e todos os casos em *CasosTEST* tal que $\forall i \in \{1, ..., \text{NCasos}\}$, $\forall k \in \{1, ..., N_{Int}\}$:
  - **h(Caso_i) ∈ $(\text{PxA})^{\text{NInt}}$**: história de comprimento N_{Int} de *Agent* em *Amb* correspondente ao $\text{Caso}_i \in$ *CasosTEST*;
  - **Ep^{K}(h(Caso_i)) ∈ PxA**: episódio na interação *K*, $K \leq N_{Int}$, da história de *Agent* em *Amb* correspondente ao caso $\text{Caso}_i \in$ *CasosTEST*;
- **f_{ad}(H(CasosTEST)) = (f_1(H(CasosTEST)), ..., f_m(H(CasosTEST))) ∈ $\mathbf{R}^{M}$**:

  um vetor de *m* funções objetivo (implícitas) na medida de avaliação de desempenho estabelecida pelo projetista ($m \geq 1$), mede a adequação de *Agent* a *Amb* considerando um conjunto de histórias H(CasosTEST), onde, $\forall m \in \{1, ..., M\}$:

$$f_m(\text{H(CasosTEST)}) = \frac{1}{\text{NCasos}} \sum_{i=1}^{\text{NCasos}} \text{Av}_m(\text{h}(\text{Caso}_i))$$

são as funções que o projetista busca maximizar, mede o nível de adequação de Agent a *Amb*, no que diz respeito à realização do *m*-ésimo objetivo na medida de avaliação de desempenho, considerando todas as histórias em H(CasosTEST); e $\forall i \in \{1, ..., NCasos\}$:

$$Av_m(h(Caso_i)) = \sum_{K=1}^{N_{int}} av_m(Ep^K(h(Caso_i)))$$

onde $av_m(Ep^k(h(Caso_i)))$ é o valor de recompensa/penalização no objetivo m atribuído pela medida de avaliação ao episódio k da história associada ao $Caso_i \in$ *CasosTEST*;

- $\mathbf{f_{inad}(H(CasosTEST)) = (-f_1(H(CasosTEST)), ..., -f_M(H(CasosTEST))) \in R^M}$:

  um vetor de M objetivos associado ao vetor $f_{ad}(H(CasosTEST))$, mede a inadequação de *Agent* a *Amb* considerando as histórias em H(CasosTEST).

Problema:

'maximizar' $f_{inad}(H(CasosTEST))$

s.a: $CasosTEST \in P(\Omega)$ e

$H(CasosTEST) \in P((PxA)^{NInt})$

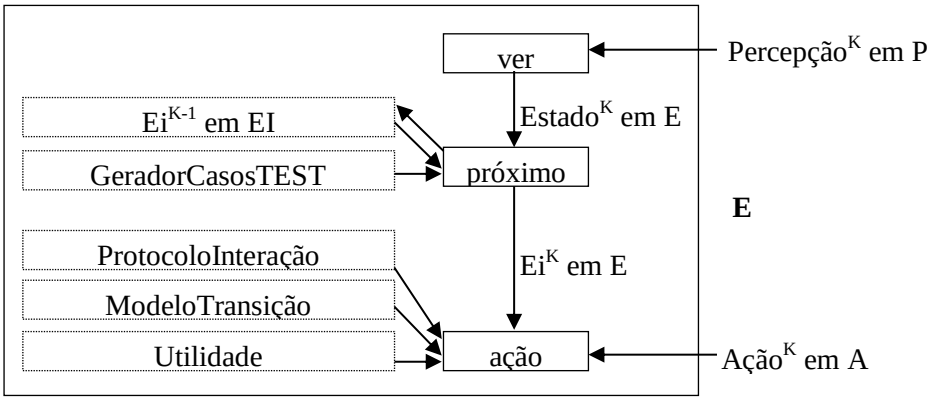
A formulação do problema de seleção de casos de testes considera que, se o programa agente for inadequado ao seu ambiente de tarefa então as funções objetivos de inadequação, ou seja, as funções objetivos na medida de avaliação modificadas pelo sinal de menos (–), serão maximizadas. São estes ambientes que uma abordagem para a seleção de casos de teste deve encontrar. Entretanto, dependendo dos objetivos em um PMO específico pode não existir um conjunto que seja ótimo em todas as funções. Neste caso, a tarefa consiste em encontrar um conjunto de casos satisfatório, ou seja, em que o desempenho do programa agente no ambiente é insatisfatório e, consequentemente, que permita ao projetista perceber as propriedades limitativas do programa.

4.3.2 Estrutura do Programa Agente *Thestes* para Seleção de Casos de Teste

O agente *Thestes* incorpora e processa as informações na formulação do problema de seleção e outras informações enviadas pelo projetista em $Percepção^K$, visando selecionar uma solução satisfatória para ser enviada em $Ação^K$ para o agente de monitoramento *ProMon*. Assim, conforme a FIGURA 13 indica, ao receber as informações oriundas do agente *ProMon*, o projetista toma as decisões que julgar necessárias para melhorar o desempenho de

Agent. Esse esquema de interação do projetista com *Thestes* deve ser continuado até que o projetista esteja satisfeito com relação ao comportamento racional de *Agent*. A FIGURA 14 ilustra a estrutura do agente de resolução de problemas *Thestes*.

FIGURA 14 – Estrutura do programa agente *Thestes*



Fonte: Elaborado pelo autor (2013)

Essa estrutura consiste em uma adaptação da estrutura de programas agentes orientados por utilidade, especificada por Russell e Norvig (2004), e da arquitetura abstrata do agente com estado interno, especificada por Wooldridge (2002). A FIGURA 15 apresenta o esqueleto do programa agente *Thestes*.

FIGURA 15 – Esqueleto de *Thestes*

função Thestes(Percepção) **retorna** uma ação

entradas: Percepção em $P_{Thestes}$

var: Estado em E, Ei em EInterno, Ação em $A_{Thestes}$, GeradorCasosTEST, ProtocoloInteração, ModeloTransição, Utilidade

$Estado^K \leftarrow ver(Percepção^K)$

$Ei^K \leftarrow próximo(Estado^K, Ei^{K-1}, GeradorCasosTEST)$

$Ação^K \leftarrow ação(Ei^K, ModeloTransição, ProtocoloInteração, Utilidade)$

retornar $Ação^K$

Fonte: Elaborado pelo autor (2013)

Mais especificamente, o subsistema de percepção (ver) mapeia as informações necessárias ao teste de *Agent* em uma representação computacional, $Estado^K$, adequada ao processamento dos outros dois subsistemas (próximo e ação): (*Agent*, *Amb*, *ParâmetrosBUSCA*, *ParâmetrosSimulação*). O subsistema de atualização de estado interno, próximo, armazena as informações em $Estado^K$, considera os parâmetros em

ParâmetrosSimulação e *GeradorCasosTEST* e gera um conjunto inicial *CasosTEST* de maneira aleatória ou, dependendo dos parâmetros, gera casos específicos, conforme o Projetista desejar testar certos aspectos da estrutura interna de *Agent* e, consequentemente, facilitar a próxima etapa de tomada de decisão (ação): (*CasosTEST*, *Agent*, *Amb*, *ParâmetrosBUSCA*, *ParâmetrosSimulação*).

Finalmente, considerando o estado interno atualizado, a função ação de *Thestes* inicia um processo de busca local visando encontrar uma ação satisfatória $Ação^K$. Esta função utiliza informações a respeito de um modelo de transição de estados, *ModeloTransição*, para gerar novos casos de teste a partir de *CasosTEST*, e o protocolo de interação, *ProtocoloInteração*, e uma função utilidade, *Utilidade*, para, respectivamente, obter as histórias correspondentes aos casos de teste no conjunto e avaliar o desempenho de *Agent* nestas histórias. Assim, enquanto o conjunto *CasosTEST* inicial e o modelo de transição permitem que ação gere o espaço de estados do problema de seleção, o protocolo e a função utilidade permitem que ação avalie os estados gerados (conjuntos de histórias) e, consequentemente, o comportamento de *Agent* nestes estados. A FIGURA 16 apresenta o esqueleto da função ação do programa agente *Thestes*.

FIGURA 16 – Esqueleto da função ação de *Thestes*

```
função ação(Ei, ModeloTransição, ProtocoloInteração, Utilidade) retorna uma solução
  entradas: Ei, ModeloTransição, ProtocoloInteração, Utilidade
  var: Pop, t, Histórias, Desempenhos

  t ← 0
  Pop^t ← CasosTEST[Ei]
  Histórias ← histórias(Pop^t, ProtocoloInteração)
  Desempenhos ← avaliar(Histórias, Utilidade)

  ...loop faça
    se Teste(T_max[Ei],U_max[Ei]) aplicado ao valor de t e aos valores em Desempenhos for V
      então retorne solução(Pop^t, Histórias, Desempenhos)
    Pop^t+1 ← modificar(Pop^t, Desempenhos, ModeloTransição, ParâmetrosBusca[Ei])
    Histórias ← histórias(Pop^t+1, ProtocoloInteração, Agent[Ei], Amb[Ei], ParâmetrosSimulação[Ei])
    Desempenhos ← avaliar(Histórias, Utilidade)
    t ← t+1
  fim
```

Fonte: Elaborado pelo autor (2013)

A função considera a existência de um contador de iterações (*t*) e de um teste (*Teste*) que serve como uma condição de parada a ser adotada pela estratégia de busca local. Esta condição é descrita por meio da conjunção e/ou disjunção de proposições envolvendo informações obtidas em tempo de execução da estratégia e outras informações no estado interno de *Thestes*, *ParâmetrosSimulação[Ei]*, como, por exemplo: o número máximo de iterações permitido para a estratégia de busca ($T_{máx}$) e um valor de utilidade satisfatória para os casos de teste ($U_{máx}$).

A função 'modificar' é invocada repetidamente para transformar um conjunto *CasosTEST* corrente em um novo conjunto, conforme os valores de utilidade dos casos de teste componentes do conjunto e o modelo de transição indicarem. O processo é encerrado quando o conjunto *CasosTEST* tiver as propriedades desejadas e a condição de parada em *Teste* for satisfeita. A função 'histórias' considera um conjunto de casos corrente e emprega o protocolo de interação para simular as interações de *Agent* com *Amb* e gerar o conjunto de histórias correspondentes. A função 'avaliar' considera o conjunto de histórias geradas e emprega a função utilidade para medir o desempenho de *Agent* em *Amb*. Ao final do processo, a função retorna uma solução, ou seja, o conjunto *CasosTEST* encontrado, os conjuntos *Histórias* e *Desempenhos* correspondentes até a condição de parada.

Vale ressaltar, a estratégia de busca geral que encapsula a ideia da busca local orientada por utilidade na função ação é uma adaptação da técnica de programação gerar-e-testar objetivando encontrar um conjunto *CasosTEST* satisfatório. Mais especificamente, neste caso, em que o objetivo é encontrar o melhor conjunto de acordo com uma função objetivo e que o conjunto que se deseja é a transformação de um conjunto inicial dado, pode-se dizer que a estratégia consiste em uma adaptação da técnica modificar-testar.

De posse das informações geradas pela função ação, *Thestes* envia para o agente *ProMon* três informações importantes em $Ação^{K}$: (1) o conjunto *CasosTEST* corrente como solução para o problema de seleção e os melhores casos encontrados até a solução, (2) as histórias correspondentes de *Agent* em *Amb* contidas no conjunto *Histórias*, (3) os valores de desempenho contidos em *Desempenhos*, considerando cada um dos objetivos na medida de avaliação de desempenho. As próximas seções descrevem mais especificamente alguns dos componentes mais importantes na estrutura de *Thestes*, relacionados aos subsistemas de processamento representados pelas funções próximo e ação.

4.4 Modelo de Transição

Conforme o esqueleto da função ação, na função que modifica os conjuntos de casos de teste, consideram-se os casos de teste em uma solução corrente, Pop^t = *CasosTEST*, um modelo de transição pré-definido *ModeloTransição* e os valores de desempenho correspondentes de *Agent* em *Amb*, medidos por uma função *Utilidade*, para gerar novos casos de teste candidatos à próxima solução, Pop^{t+1}. O modelo de transição genérico no esqueleto de *Thestes* considera os *NCasos* em um conjunto Pop^t corrente e escolhe: (a) os casos de teste que serão modificados, e (b) as mudanças que serão realizadas nestes casos.

Por exemplo, no caso específico do aspirador de pó, equivale a escolher: (a) os ambientes de tarefa a serem modificados, e (b.1) um subconjunto das n x n salas em cada ambiente escolhido em (a), e (b.2) a ação que deve ser realizada em cada uma das salas no subconjunto de salas em (b.1) de cada ambiente escolhido em (a) como, por exemplo, depositar ou retirar sujeira da sala, e colocar ou retirar o agente na sala.

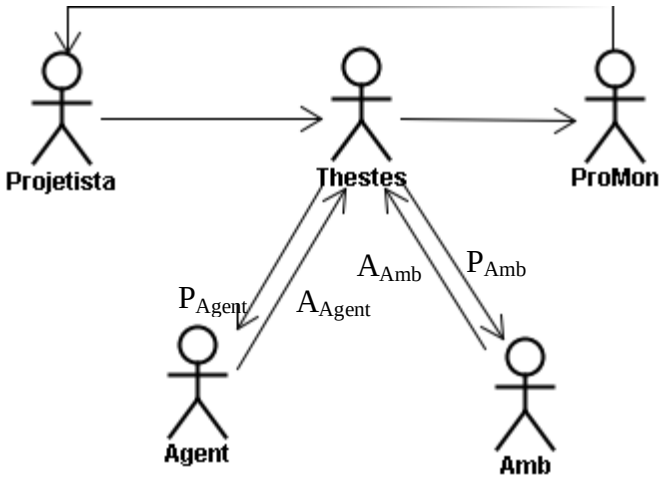
Diversos modelos de transição podem ser concretizados para realizar as escolhas acima. Esta primeira concretização do modelo aplicou algumas das principais noções presentes na metaheurística baseada em populações Algoritmo Genético (GA) (HOLLAND, 1975), ou seja, uma adaptação do processo de seleção de pares e dos operadores genéticos nesta metaheurística. Da mesma maneira, outras concretizações podem ser realizadas considerando ou não outras noções presentes em outras metaheurísticas, principalmente aquelas que são baseadas em populações.

4.5 Mecanismo de Simulação de Interações *Agent-Amb*

De acordo com a formulação do problema de seleção, *Thestes* conhece tanto *Agent* quanto *Amb*, bem como o protocolo *ProtocoloInteração* entre *Agent* e *Amb*. A função ação considera estas informações e as informações de entrada em *ParâmetrosSimulação* no processo de tomada de decisão em *Thestes*. Assim, foi concebido um mecanismo de interação que, de acordo com *ProtocoloInteração*, simula as interações entre *Agent* e *Amb*, quando *Amb* é previamente inicializado com informações a respeito de um ambiente de tarefa descrito em um caso de teste ($Caso_i \in CasosTEST$) e que anota os episódios componentes ($Ep^p(h(Caso_i)) \in PxA$) da história correspondente ao caso ($h(Caso_i) \in (PxA)^{NInt}$). A

FIGURA 17 ilustra o funcionamento deste mecanismo.

FIGURA 17 – Simulação das interações entre *Agent* e *Amb*



Fonte: Elaborado pelo autor (2013)

O mecanismo alimenta *Amb* com informações em P_{Amb} a respeito de um caso de teste específico pertencente ao conjunto solução corrente *CasosTEST*. Ao perceber essas informações, *Amb* armazena internamente informações a respeito do ambiente de tarefa correspondente, com as propriedades iniciais desejadas, e envia em A_{Amb} as informações a respeito de seu estado corrente. O mecanismo anota essas informações sobre estado e as repassa para *Agent*, que as percebe em P_{Agent}, processa-as e seleciona uma ação que é enviada em A_{Agent} para o mecanismo. Ao receber essas informações sobre ação, o mecanismo encerra a anotação de um episódio da história, envia em P_{Amb} as informações sobre ação para *Amb* e inicia outro ciclo de interação, o qual deve ser repetido até a anotação de uma história completa, conforme indicado em *ParâmetrosSimulação*.

4.6 Função Utilidade

Durante o processo de busca, na etapa de geração de novos casos de teste, a função *Utilidade* permite que o agente *Thestes* obtenha medidas de desempenho de *Agent* em *Amb* considerando os casos de teste modificados pelo *ModeloTransição*. Mais especificamente, estas medidas permitem que *Thestes* realize um julgamento a respeito dos casos de teste modificados e, consequentemente, selecione entre os casos modificados um subconjunto de NCasos que, preferencialmente, seja melhor que o anterior. Assim, considerando o caráter multiobjetivo do problema de seleção, os julgamentos de *Thestes*

foram modelados empregando-se a abordagem clássica da análise multicritério[1], conforme explicado abaixo.

No problema de seleção multiobjetivo que foi formulado na subseção 4.3.1 Formulação do Problema de Seleção de Casos de Teste para Agentes Racionais, a função inadequação a ser maximizada foi definida como $f_{inad}(H(CasosTEST)) = (-f_1(H(CasosTEST)), ..., -f_M(H(CasosTEST))) = У \in R^M$. Assim, vale notar que f_{inad}: $P(\Omega) \rightarrow У \subset R^M$ e que У é parcialmente ordenado de acordo com a relação de ordem '≤', ou seja: em geral, $\neg\exists CasosTEST^* \in P(\Omega)$ e $H(CasosTEST^*) \in P((PxA)^{NInt})$ ∋ ∀ $CasosTEST \in P(\Omega)$ e $H(CasosTEST) \in P((PxA)^{NInt})$, $f_{inad}(H(CasosTEST^*)) \geq. f_{inad}(H(CasosTEST))$. Assim, para decidir entre dois vetores comparáveis $y^1, y^2 \in У$, a estrutura de preferências de *Thestes* considera a relação característica clássica (**S**), ou seja, composta da união das relações binárias: (a) de indiferença **I**, que é transitiva, simétrica ($\forall y^1, y^2 \in У, y^1$ **I** $y^2 \rightarrow y^2$ **I** y^1) e reflexiva ($\forall y \in У$, y **I** y); e (b) de preferência estrita **P**, que é transitiva, assimétrica ($\forall y^1, y^2 \in У, y^1$ **P** $y^2 \rightarrow y^2 \neg$**P** y^1) e irreflexiva ($\forall y \in У$, y $\neg$**P** y).

Assim, considerando a relação característica **S**, $\forall y^1, y^2 \in У$, apenas uma das sentenças é verdadeira para *Thestes*: (1) y^1 **I** y^2, ou seja, *Thestes* é indiferente entre y^1 e y^2; (2) y^1 **P** y^2, ou seja, *Thestes* prefere y^1 a y^2; e (3) y^2 **P** y^1, ou seja, *Thestes* prefere y^2 a y^1. Assim, o problema de seleção formulado foi caracterizado em *Thestes* como: determinar $y^* \in У$ ∋ $\forall y \in У$, y^* **P** y. Para resolver o problema, os julgamentos de *Thestes* foram representados numericamente por meio de uma função utilidade *Utilidade*: $У \rightarrow R$, ou seja, que associa um real Utilidade(y) a cada $y \in У \subset R^M$ tal que: (1) y^1 **I** $y^2 \leftrightarrow U(y^1) = U(y^2)$; (2) y^1 **P** $y^2 \leftrightarrow U(y^1) > U(y^2)$; (3) y^1 **S** $y^2 \leftrightarrow U(y^1) \geq U(y^2)$.

Nesta primeira abordagem, supondo-se que a função *Utilidade* é preferencialmente independente, ou seja, o grau de utilidade de um objetivo independe dos valores assumidos pelos demais, foram incorporadas em *Thestes* duas formas especiais de função Utilidade, ou seja, uma função aditiva:

$$\text{Utilidade}(f_{inad}(H(CasosTEST))) = \sum_{m=1}^{M} u_m \left(-f_m\left(H(CasosTEST)\right)\right),$$

e outra no formato linear:

[1] A análise multicritério é a área da Tomada de Decisão Multicritério (TDMC) que lida com problemas em que o conjunto de soluções viáveis é discreto, pré-determinado e finito. Na área de Otimização Multiobjetivo (Multicritério ou Multiatributo), as soluções viáveis não são explicitamente conhecidas, mas, geralmente, representadas por funções de restrições.

$$\text{Utilidade}(f_{inad}(H(\text{CasosTEST}))) = -\sum_{m=1}^{M} w_m * f_m(H(\text{CasosTEST}))$$

onde $w_m \geq 0$, m = 1, ..., M.

Considerando estas duas possibilidades o problema de seleção de casos de teste foi reformulado como:

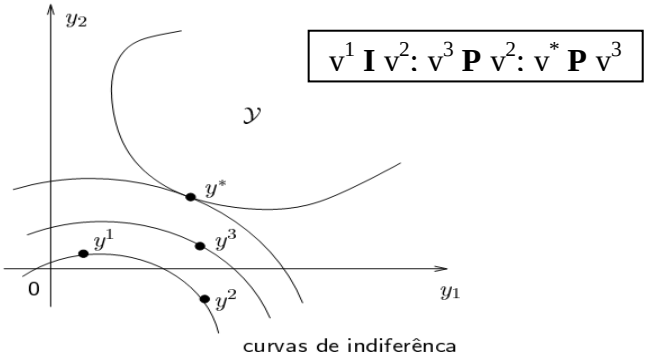
'maximizar' Utilidade(f_{inad}(H(CasosTEST)))
s.a: CasosTEST ϵ P(Ω) e
H(CasosTEST) ϵ P((PxA)^{NInt})

Ou seja, considerando que У = f_{inad}(P(Ω)) é a representação do mapeamento de P(Ω) no espaço dos objetivos: У = {y ϵ R^M| y = f_{inad}(H(CasosTEST))), *CasosTEST* ϵ P(Ω) e H (CasosTEST) ϵ P((PxA)$^+$)}; o problema pode ser estabelecido como:

'maximizar' Utilidade(y)
s.a: y $\in$ У

Essa reformulação permite que *Thestes* estruture suas preferências através de curvas de indiferença ou isopreferência, ou seja, curvas de nível da função utilidade. Mais especificamente, para um determinado nível de utilidade, uma curva de indiferença apresenta os valores de inadequação associados aos objetivos em que o agente é indiferente. São curvas que representam combinações de valores de inadequação nos objetivos que auferem o mesmo nível de utilidade. Assim, o agente não tem preferência entre uma combinação e a outra, já que cada uma provê um mesmo nível de utilidade (não muda). A FIGURA 18 ilustra a ideia de mapeamento de conjuntos solução *CasosTEST* ϵ P(Ω) para o espaço de dois objetivos (y_1 = $-f_1$(H(CasosTEST)) e y_2 = $-f_2$(H(CasosTEST)).

FIGURA 18 – Exemplo de curvas de indiferença considerando dois objetivos



Fonte: Elaborado pelo autor (2013)

Considerando a noção de curvas de indiferença, *Thestes* pode procurar por uma solução y^* no espaço dos objetivos, tal que $y^* \in У \ni \forall y \in У$, y^* **S** y, isto é, $U(y^*) \geq U(y)$, isto é, que pertença ao conjunto de soluções eficientes, $efi(У) = f_{inad}(efi(P(\Omega)))$. No entanto, caso *Thestes* não encontre estas soluções, ele pode procurar por soluções satisfatórias, mas, privilegiando aquelas soluções que estejam próximas à curva de eficiência e bem distribuídas ao longo da mesma. Este último requisito é importante, pois permitirá que o projetista analise situações problemas considerando as inadequações associadas a todos os objetivos envolvidos na medida de avaliação de desempenho.

4.7 Agente ProMon

Esta seção esboça as principais ideias no programa agente de monitoramento *ProMon*, em destaque na FIGURA 13. A seção foi dividida em duas subseções. A primeira subseção descreve resumidamente os aspectos dos agentes racionais que foram considerados pelo agente *ProMon* na composição tanto das manifestações (medidas) quanto do próprio diagnóstico das falhas. A segunda subseção destaca as principais funcionalidades que o agente emprega para realizar o diagnóstico.

4.7.1 Noção de Agentes Racionais em *ProMon*

A concepção do agente *ProMon* considera que, dependendo do ambiente, o projeto do agente racional pode ser idealizado considerando quatro tipos básicos de programas de agentes, ou seja: reativos simples, reativos baseados em modelos, baseados em objetivos e baseados em utilidade. Mais especificamente, a concepção considera uma síntese dos pontos de vista de Wooldridge (2002) a cerca das arquiteturas abstratas de agentes, e de Russell e Norvig (2004) a respeito das quatro estruturas de programas agentes mencionados.

Na etapa de escolha de um tipo de agente, é importante que as propriedades internas do programa sejam adequadas às propriedades de seu ambiente de tarefa. Os quatro tipos de programas agentes podem ser subdivididos em três subsistemas de processamento de informação principais. O primeiro, o subsistema de percepção, mapeia uma informação sobre percepção em uma representação abstrata útil para o agente, **ver**: P→Estado. O segundo, o subsistema de atualização de estado interno, mapeia a representação da percepção atual e a informação sobre o estado interno mantido pelo agente em um novo estado interno, **próximo**: Estado x EI → EI. Por último, o subsistema de tomada de decisão, mapeia uma informação sobre estado interno em uma ação correspondente, **ação**: EI → A.

Em resumo, a função ação do programa agente reativo simples seleciona ações baseando-se na percepção atual, mapeada pela função ver, e em um conjunto de regras no formato condição-ação. A função próximo nos programas agentes reativos baseados em modelos mantém em memória uma descrição de estado do ambiente do agente. A função ação dos programas agentes orientados por objetivos seleciona suas ações utilizando as informações processadas pela função próximo e uma informação sobre os objetivos que descrevem situações desejadas no ambiente. A função ação dos agentes orientados por utilidade emprega uma função utilidade para mapear descrições de estado do ambiente em um grau de felicidade associado.

Em ambientes desconhecidos, ou seja, em que o agente não conhece os estados possíveis, nem os efeitos das suas ações, a concepção de um agente racional pode requisitar que um tipo básico de programa seja convertido em um agente com capacidade de aprendizagem. Este novo tipo de agente tem a capacidade de melhorar seu desempenho à medida que explora o ambiente. A conversão de um programa básico em um agente com aprendizagem envolve a integração de três outros módulos de processamento de informação ao programa básico (elemento de desempenho), ou seja: um crítico, um elemento de aprendizagem e um gerador de problemas.

4.7.2 Funcionalidades do Agente

O agente *ProMon* recebe de *Thestes*: (1) um conjunto *CasosTEST* contendo os casos de teste onde *Agent* teve um comportamento inadequado, (2) as histórias de *Agent* em *Amb* e os valores de avaliação de desempenho associados, episódio por episódio. Considerando estas informações, o agente *ProMon* deve enviar para o projetista: (1) os episódios em todas as histórias em que *Agent* falhou e os episódios ideais correspondentes, e (2) uma identificação do tipo de falha, indicando o subsistema de processamento de informação de *Agent* que ele presume esteja causando a falha, ou seja: (a) subsistema de percepção ver, (b) subsistema de atualização de estado interno próximo, e (c) subsistema de tomada de decisão ação.

O agente *ProMon* foi concebido como um agente reativo baseado em modelos. Mais especificamente o processo de monitoramento e diagnóstico de falha realizado pelo agente *ProMon* foi proposto para ser realizado em duas etapas, correspondentes ao processamento de um subsistema do tipo próximo e outro do tipo ação. Na primeira etapa, a

função próximo do agente recebe as histórias associadas aos casos em *CasosTEST* H(CasosTEST) e identifica todos os episódios contendo falhas nessas histórias.

Assim, como no caso do agente *Thestes*, esta função considera o protocolo *ProtocoloInteração*, o programa ambiente *Amb* e uma versão totalmente observável do agente testado (onipresente), denotada por *Agent** para identificar se um episódio em uma interação *K*, em uma história associada a um caso *i* em *CasosTEST*, e gerar dois conjuntos de episódios: o conjunto de episódios ideais na interação, Ep_{ideais}^{K}, e o conjunto de episódios com falhas nas histórias associadas aos casos em *CasosTEST*, Ep_{falhas}.

Assim, de acordo com a notação empregada na formulação do problema de seleção de casos de teste do agente *Thestes*, seja:

- **Agent**: um programa agente racional a ser testado;
- **Agent***: uma versão onipresente do programa a ser testado
- **Amb**: um programa ambiente capaz de interagir com *Agent* por meio de *ProtocoloInteração*;
- $\mathbf{h(Caso_i) \in (PxA)^{NInt}}$ uma história de comprimento N_{Int} de *Agent* em *Amb* correspondente ao $Caso_i \in$ *CasosTEST*;
- $\mathbf{h^{*}(Caso_i) \in (PxA)^{NInt}}$ uma história de comprimento N_{Int} de *Agent** em *Amb* correspondente ao $Caso_i \in$ *CasosTEST*
- $\mathbf{Ep^{K}(h(Caso_i)) \in PxA}$ ou, mais especificamente, $Ep((P^K, A^K))) \in PxA$ um episódio na interação k da história de *Agent* em *Amb* correspondente ao caso $Caso_i \in$ *CasosTEST*.
- $\mathbf{Ep^{K}(h^{*}(Caso_i)) \in PxA}$ ou, mais especificamente, $Ep((P^K, A^K*))) \in PxA$ um episódio na interação *K* da história de *Agent** em *Amb* correspondente ao caso $Caso_i \in$ *CasosTEST*.

O conjunto de episódios ideais em uma interação *K*, Ep_{ideais}^{K}, é formado por todos os episódios possíveis de serem produzidos por *Agent** na interação, $Ep((P^K, A^K*))$, que satisfazem pelo menos uma das duas condições abaixo:

(1) Para todo atributo m na medida de avaliação de desempenho:

$$av_m(Ep((P^K, A^K*))) \geq av_m(Ep((P^K, A^K)));$$

(2) A^K* é melhor que ou é equivalente a A^K considerando o ponto de vista do projetista.

E, consequentemente, episódios $Ep((P^K, A^K))$, produzidos pelo agente testado *Agent* que não pertencem ao conjunto $Ep_{ideais}{}^K$, devem compor o conjunto de episódios com falha Ep_{falhas}.

A condição (2) depende do ponto de vista do projetista. Ela foi introduzida visando identificar falhas que não são percebidas diretamente na especificação da medida de avaliação de desempenho. Por exemplo, considerando a medida de avaliação de desempenho para um programa aspirador de pó descrita na TABELA 2, é possível perceber que as linhas 1, 5 e 6 identificam episódios que são falhas, ou seja, o agente aspirar em uma sala limpa, e movimentar-se para uma sala vizinha ou não operar quando a sala corrente está suja. Entretanto, a Tabela não identifica episódios que são falhas em função do agente movimenta-se para salas vizinhas que não estão sujas ou retornar desnecessariamente às salas que já foram visitadas. Assim, diferentemente da condição (1), que considera a medida de avaliação de desempenho, a condição (2) deverá ser especificada de acordo com o domínio de aplicação do agente testado *Agent*.

Encerrando a etapa de identificação dos episódios que são falhas em todas as interações *k* nas histórias associadas aos casos em *CasosTEST*, realizada pela função próximo do agente *ProMon*, a segunda etapa do processo de monitoramento e diagnóstico do agente pode ser iniciada. Nesta etapa a função ação do agente utiliza o conjunto de episódios Ep_{falhas} e uma função ação baseada em regras condição-ação para identificar o tipo de falha associada ao episódio com falha em uma interação específica $Ep((P^K, A^K))$, considerando o próprio episódio, os valores de avaliação associados ao episódio em todos os atributos *m* na medida de avaliação $av_m(Ep((P^K, A^K)))$, e os episódios ideais $Ep((P^K, A^K*))$ em $Ep_{ideais}{}^K$, produzidos por $Agent^*$ na mesma interação. A FIGURA 19 apresenta as regras genéricas utilizadas por *ProMon* para o caso em que *Agent* é um agente reativo simples ou um agente reativo baseado em modelos.

FIGURA 19 – Regras condição-ação de agente *ProMon* para agentes reativos

(1) se Agent for reativo simples e a Condição (1)' for satisfeita então:
"Condição (1)' foi satisfeita."
"Falha na função ver **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^K*$, ou
falha na função ação **se**
$ver(P^K) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$, ou
falha nas funções ver e ação **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$."
(2) se Agent for reativo simples e a Condição (2)' for satisfeita então:
"Condição (2)' foi satisfeita."
"Falha na função ver **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^K*$, ou
falha na função ação **se**
$ver(P^K) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$ ou
falha nas funções ver e ação **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$."
(3) se Agente é baseado em modelos e Condição (1)' foi satisfeita então:
"Condição (1)' foi satisfeita."
"Falha na função próximo **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^K*$, ou
falha na função ação **se**
$próximo(ver(P^K)) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$, ou
falha nas funções próximo e ação **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$."
(4) se Agent é baseado em modelos e Condição (2)' foi satisfeita então:
"Condição (2)' foi satisfeita."
"Falha na função próximo **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^K*$, ou
falha na função ação **se**
$próximo(ver(P^K)) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$ ou
falha nas funções próximo e ação **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^K*$."

Fonte: Elaborado pelo autor (2013)

Os antecedentes nas regras acima consistem em duas condições associadas às condições (1) e (2) descritas anteriormente. Estas condições associadas indicam a razão pela qual um episódio com falhas $Ep((P^K, A^K))$ foi inserido no conjunto Ep_{falha}, ou seja:

(1)' existe pelo menos um atributo $\underline{x}$ em que $av_x(Ep((P^K, A^K*))) > av_x(Ep((P^K, A^K)))$, enquanto que no restante dos atributos $av_m(Ep((P^K, A^K*))) \geq av_m(Ep((P^K, A^K)))$;

(2)' A^K* é melhor que A^K.

Os consequentes nas regras são mensagens enviadas ao projetista de *Agent*. A primeira mensagem indica a condição que foi satisfeita, a razão para o episódio pertencer ao conjunto Ep_{falha}. A segunda mensagem consiste em uma disjunção envolvendo regras no formato "consequente **se** antecedente", ou seja, "Falha no Módulo X **se** Condição Y for satisfeita". São estas mensagens que são enviadas ao projetista. As regras são sugeridas ao projetista para que ele avalie as condições descritas em seus antecedentes e perceba quais módulos de processamento de informação em *Agent* possivelmente estão causando as falhas, conforme as condições (1)' e (2)'.

Assim, como o antecedente em cada uma das regras sugeridas é uma conjunção de proposições envolvendo as saídas dos módulos de processamento da informação do agente testado *Agent* e do agente com observabilidade total *Agent**, a abordagem com o agente *ProMon* pressupõe que o projetista tem o controle do agente testado e é capaz de comparar o processamento realizado pelos módulos de *Agent* com o processamento realizado pelos módulos de *Agent**. As duas primeiras regras são aplicadas aos casos em o agente testado *Agent* é um programa agente do tipo reativo simples. As duas últimas, quando *Agent* for um programa reativo baseado em modelos.

A primeira e segunda regras indicam que a falha está no subsistema de percepção de *Agent*, quando o subsistema de percepção de *Agent** produz uma informação $Estado^K$ diferente da produzida pelo subsistema de *Agent* e o projetista supõe que o subsistema de tomada de decisão de *Agent* seria capaz de selecionar uma ação ideal A^K*, ou seja, uma das ações selecionadas por *Agent** e presentes em $Ep_{ideais}{}^K$, caso tivesse em mãos a informação $Estado^K$. Caso não existe falha em ver, as regras indicam que a falha está no subsistema de tomada de decisão de *Agent*, ou seja, apesar de perceber como *Agent**, o projetista supõe que o agente testado *Agent* não conseguiria tomar decisões equivalentes. Além das duas possibilidades, as regras indicam também que falha pode estar presente nos dois subsistemas.

Semelhantemente, para o caso em que *Agent* possui um estado interno, a terceira e quarta regras indicam que a falha está na função próximo quando o estado interno de *Agent* for diferente da informação produzida pelo subsistema de percepção de *Agent** e o projetista

supor que o subsistema de tomada de decisão de *Agent* seria capaz de selecionar uma ação ideal A^K* para $Estado^K$. Considerando que não existe a falha em próximo, essas regras indicam que a falha está no subsistema de tomada de decisão do agente reativo baseado em modelos *Agent*. Finalmente, as regras indicam que a falha pode estar presente tanto em próximo como em ação.

Vale ressaltar, as regras aplicadas ao agente reativo baseado em modelos não consideram a possibilidade de falha na função ver deste agente, significando que, especificamente neste caso, o projetista conhece as limitações do agente em termos de observabilidade do ambiente e, por isso, concebeu uma função próximo, ou seja, visando minimizar o baixo nível de confiabilidade da função ver. Assim, para ele, o projetista, é mais importante perceber se existem falhas na função próximo. Entretanto, caso desejado, pode-se incluir, no esquema adotado na

FIGURA 19, regras para o diagnóstico de falhas na função ver do agente reativo baseado em modelos. Apesar de não especificado, as regras adotadas para este agente podem ser adaptadas para os agentes orientados por objetivos e por utilidade, já que estes agentes também podem ser descritos em termos dos componentes: ver, próximo e ação.

5 AVALIAÇÃO EXPERIMENTAL DA ABORDAGEM

Este capítulo apresenta um estudo de caso conduzido para demonstrar o funcionamento e uma primeira avaliação de desempenho de uma concretização da abordagem para o teste de agentes racionais. Mais especificamente, o estudo considerou o esqueleto do programa agente *Thestes* resolvendo problemas de seleção de casos de teste para programas agentes aspiradores de pó em um mundo expandido. Foram testados dois tipos de programas agentes: um aspirador reativo simples e outro baseado em modelos. A análise dos resultados foi realizada considerando as funcionalidades do programa agente *ProMon*. O capítulo foi dividido em duas seções. A Seção 5.1 descreve o plano experimental, visando clarear os objetivos dos experimentos, definir melhor o domínio em que a abordagem foi avaliada e a metodologia dos experimentos. A Seção 5.2 descreve e analisa os resultados experimentais.

5.1 Plano Experimental

O eixo central dos experimentos visou validar as seguintes hipóteses: (H1) a abordagem contribui para o diagnóstico das falhas realizadas pelo projetista; e (H2) a abordagem, além de mérito, é relevante considerando os trabalhos no estado da arte na área. Para validar H1 e H2, os experimentos consideraram o teste de programas agentes reativos simples e baseados em modelos, regras condição-ação, concebidos para funcionarem como aspiradores de pó em um ambiente determinístico e estático formado por salas que podem conter sujeira. Quanto à observabilidade do ambiente, os agentes foram concebidos para atuar em ambientes parcialmente e totalmente observáveis. Além das versões originais desses agentes foram avaliadas versões contendo falhas em alguns dos subsistemas componentes destas versões. A medida de desempenho considerou critérios relacionados aos atributos de limpeza e energia.

Quanto à concretização do agente *Thestes*, o conjunto solução inicial *CasosTEST* foi gerado de maneira aleatória. A função modificar, componente da função ação do agente, foi implementada por meio de estratégias de geração e seleção de novos conjuntos solução semelhantes às estratégias empregadas na maioria dos algoritmos genéticos e, inclusive, semelhante à estratégia no algoritmo genético multiobjetivo denominado NSGA-II. Também componente do subsistema de tomada de decisão de *Thestes*, a função avaliar foi implementada considerando mais especificamente uma função Utilidade no formato linear,

descrita em termos dos atributos de energia e limpeza na medida de avaliação de desempenho dos agentes testados.

Vale ressaltar, a maneira como o plano experimental foi concebido ainda não é suficiente para comprovar as hipóteses levantadas. Ainda é necessário avaliar a abordagem (1) em outros domínios de resolução de problemas de agentes, e (2) considerar outros tipos de programas agentes, além dos reativos e baseados em modelos regas condição-ação. Entretanto, os resultados obtidos a partir da implementação deste primeiro plano contribuem fortemente para estas comprovações.

As três próximas subseções especificam o ambiente de tarefa, a medida de avaliação e os agentes testados. As duas últimas subseções, a concretização de *Thestes* e a descrição dos experimentos realizados para avaliar os resultados produzidos pela abordagem testando os agentes aspiradores de pó.

5.1.1 Ambiente de Tarefa

Mais especificamente, a tarefa de *Thestes* consiste em selecionar um conjunto de ambientes de tarefa formados por *n x n* salas (*Amb*) que sejam satisfatórios para testar programas agentes aspirador de pó com regras condição-ação (*Agent*) em sua função ação. Um ambiente difere de outro quanto à localização e à quantidade de salas sujas. A FIGURA 20 exemplifica um ambiente de tarefa específico composto de 25 (n = 5) salas.

FIGURA 20 – Ambiente de Tarefa/Caso de Teste

L	S	S	L	L
S	L	S	L	L
L	L	S	L	S
S	L	L	S	L
S	L	L	L	S

Fonte: Elaborado pelo autor (2013)

Para o estudo de caso foram consideradas duas situações: (a) ambiente parcialmente observável (Experimentos 1, 2 e 5), ou seja, em cada interação, pressupõe-se que o aspirador possui sensores que percebem (P) o ambiente, mas que a sua função ver consegue mapear apenas uma pequena representação desta informação (Estado), isto é, o estado, sujo ou limpo, na sala em que o agente está; (b) ambiente completamente observável (Experimentos 3 e 4), ou seja, em cada interação, pressupõe-se que a função ver do agente consegue mapear o estado de todas as salas presentes no ambiente.

Em cada interação com o ambiente, em qualquer sala, a função ação do agente pode escolher uma das seguintes Ações (A): aspirar (Asp), não operar (N-op), ou mover-se para outra sala vizinha (Ac, Esq, Dir, Ab). O ambiente é estático e cada caso de teste é instanciado em um programa ambiente (*Amb*) que obedece ao modelo determinístico representado na TABELA 4.

TABELA 4 – Modelo Determinístico do Ambiente

Percepção^{K}	**Ação^{K}**	**Percepção^{K+1}**
No, Oe, **L**, Le, Su	N-op	No, Oe, **L**, Le, Su
No, Oe, **L**, Le, Su	Asp	No, Oe, **L**, Le, Su
No, Oe, **L**, Le, Su	Ac	**No**, Oe, L, Le, Su
No, Oe, **L**, Le, Su	Esq	No, **Oe**, L, Le, Su
No, Oe, **L**, Le, Su	Dir	No, Oe, L, **Le**, Su
No, Oe, **L**, Le, Su	Ab	No, Oe, L, Le, **Su**
No, Oe, **S**, Le, Su	N-op	No, Oe, **S**, Le, Su
No, Oe, **S**, Le, Su	Asp	No, Oe, **L**, Le, Su
No, Oe, **S**, Le, Su	Ac	**No**, Oe, S, Le, Su
No, Oe, **S**, Le, Su	Esq	No, **Oe**, S, Le, Su
No, Oe, **S**, Le, Su	Dir	No, Oe, S, **Le**, Su
No, Oe, **S**, Le, Su	Ab	No, Oe, S, Le, **Su**

Fonte: Elaborado pelo autor (2013)

A tabela apresenta apenas uma informação parcial a respeito das leis que governam as mudanças de estado de qualquer ambiente em função das ações que são possíveis para o agente (Percepção^{K}, Ação^{K}, Percepção^{K+1}). Mais especificamente, as seis primeiras linhas descrevem o efeito das ações quando o agente está em uma sala limpa (Percepção^{K} = ..., L, ...,), enquanto que as seis últimas descrevem o efeito das ações quando o agente está em uma sala suja (Percepção^{K} = ..., S, ...,). A sujeira nas salas vizinhas à sala em que o aspirador está (salas: No, Oe, Le, Su) não é alterada com a execução de qualquer ação do agente (por exemplo: se Oe^{K} = L e Ação^{K} então Oe^{K+1} = L; ou se Oe^{K} = S e Ação^{K} então Oe^{K+1} = S), mas o estado global do ambiente (Percepção^{K+1}) pode ser alterado, já que o agente estará em uma nova sala em consequência da execução de uma das ações de movimento (Ac, Esq, Dir, Ab).

Na abordagem descrita, as leis que governam as mudanças de estado do ambiente de tarefa em função das ações possíveis para o aspirador, formam as regras condição-ação de um programa ambiente reativo com estado interno (*Amb*), que, em conjunto com as informações em *ProtocoloInteração*, permitem a realização de simulações das interações

entre o um programa aspirador (*Agent*) e o programa ambiente, quando inicializado com informações a respeito de um ambiente de tarefa descrito em um caso de teste específico.

5.1.2 Medida de Avaliação de Desempenho

A TABELA 5 apresenta a medida de avaliação de desempenho empregada nos experimentos, consiste em uma extensão da TABELA 2 apresentada na seção 4.1.3 Avaliação de Desempenho. Idealmente um programa agente aspirador de pó deve limpar um ambiente e maximizar os níveis de limpeza do ambiente e de energia em sua bateria ao final da tarefa. A primeira coluna descreve uma parte das informações nas percepções do agente em cada episódio possível. A segunda coluna descreve as ações possíveis nesses episódios. Na terceira e quarta colunas, respectivamente, associadas aos objetivos de energia e de limpeza, duas funções escalares (av_E e av_L) para medir o desempenho do agente em cada episódio de sua história no ambiente.

TABELA 5 – Medida de Avaliação de Desempenho

Ep^K = (Percepção^K,	**Ação^K)**	**$av_E(Ep^K)$**	**$av_L(Ep^K)$**	**Falha**
..., L, ...	Asp	-1.0	0.0	x
..., L, ...	Dir, Esq, Ac, Ab	-2.0	1.0	
..., L, ...	N-op	0.0	0.0	
..., S, ...	Asp	-1.0	2.0	
..., S, ...	Dir, Esq, Ac, Ab	-2.0	-1.0	x
..., S, ...	N-op	0.0	-1.0	x

Fonte: Elaborado pelo autor (2013)

Vale ressaltar, a medida de avaliação de desempenho na TABELA 5 não pontua implicitamente aspectos negativos da função *ver*, visto que o valor atribuído a cada episódio independe do estado das outras salas no ambiente, diferentes da sala em que o aspirador está. A quinta coluna destaca apenas os episódios que representam comportamento inadequado, provavelmente por uma falha na função ver e/ou no conjunto de regras condição-ação nas funções ação dos programas aspirador de pó.

5.1.3 Agentes Testados

O programa agente aspirador de pó reativo simples (*RS_Parcial*) foca na seleção das ações com base na percepção atual, ignorando o histórico de percepções obtido, em um ambiente parcialmente observável, ou seja, a função ver de *RS_Parcial* permite perceber apenas o estado, sujo ou limpo, da sala em que o agente está. Assim, este primeiro programa

foi concebido de maneira muito simples, mas útil para uma primeira ilustração e avaliação da abordagem. A FIGURA 21 apresenta as regras condição-ação de *RS_Parcial*.

FIGURA 21 – Regras condição-ação de *RS_Parcial*

se estado da sala é S **então faça** Asp
se estado da sala é L **então faça** movimento aleatório (Ac, Esq, Dir, Ab)

Fonte: Elaborado pelo autor (2013)

Considerando o ambiente de tarefa de *Agent* parcialmente observável, a condição que antecede cada regra considera apenas as informações percebidas e que são possíveis de serem mapeadas (Estado) pela função ver, enquanto que o consequente indica uma ação considerada ótima localmente pelo projetista, de acordo com a medida de avaliação de desempenho. Com o objetivo de verificar a sensibilidade da abordagem de testes quanto à introdução de falhas no subsistema tomada de decisão de *RS_Parcial*, foi realizada uma modificação nas regras condição-ação do agente conforme indicado na FIGURA 22 (*RS_Parcial_alterado*).

FIGURA 22 – Regras condição-ação de *RS_Parcial_alterado*

se estado da sala é S ou L **então faça** uma ação aleatória (Asp, Ac, Esq, Dir, Ab, N-Op)

Fonte: Elaborado pelo autor (2013)

O terceiro programa aspirador de pó também foi concebido de acordo com a estrutura reativa simples, entretanto com observabilidade total em relação ao ambiente, ou seja, a função ver de *RS_Total* permite perceber o estado de todas as salas do ambiente. Nas regras condição-ação, se a sala em que o agente está contém sujeira, a ação Asp será escolhida; caso contrário, o agente escolherá a ação de movimento que leva à sala suja mais próxima. A FIGURA 23 ilustra as regras condição-ação de *RS_Total*.

FIGURA 23 – Regras condição-ação de *RS_Total*

se estado da sala é S **então faça** a ação Asp
se estado da sala é L e PróximaSalaSuja(norte) **então faça** a ação Ac
se estado da sala é L e PróximaSalaSuja(sul) **então faça** a ação Ab
se estado da sala é L e PróximaSalaSuja(leste) **então faça** a ação Dir
se estado da sala é L e PróximaSalaSuja(oeste) **então faça** a ação Esq

Fonte: Elaborado pelo autor (2013)

O quarto programa agente testado foi concebido de acordo com a estrutura do agente reativo baseado em modelo considerando que o ambiente é parcialmente observável (*RM_Parcial*). Este agente possui um estado interno que armazena o histórico das percepções obtidas e uma ação é selecionada levando em considerando esta informação. A FIGURA 24 apresenta as regras condição-ação de *RM_Parcial*.

FIGURA 24 – Regras condição-ação de *RM_Parcial*

se estado da sala é S **então faça** a ação Asp
se estado da sala é L e NaoVisitou(norte) **então faça** a ação Ac
se estado da sala é L e NaoVisitou(sul) **então faça** a ação Ab
se estado da sala é L e NaoVisitou(leste) **então faça** a ação Dir
se estado da sala é L e NaoVisitou(oeste) **então faça** a ação Esq
se estado da sala é L e visitou todas **então faça** uma ação aleatória

Fonte: Elaborado pelo autor (2013)

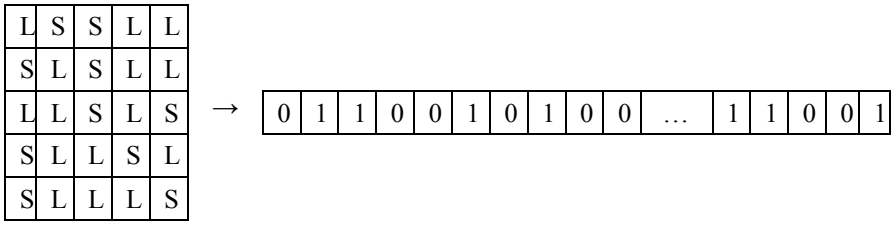
5.1.4 Concretização de *Thestes*

O esqueleto de *Thestes* pressupõe a existência de um conjunto *CasosTEST* corrente contendo *NCasos* de teste, que é uma solução inicial para um problema de seleção gerado pela função próximo. A função ação do agente considera quatro componentes principais, três deles essenciais no contexto das funções ‘modificar’ e ‘desempenho’, ou seja: (1) um modelo *ModeloTransição* de estados que opera sobre o conjunto *CasosTEST* corrente para gerar novos conjuntos, (2) uma função *Utilidade* para avaliar os conjuntos gerados, (3) uma estratégia para selecionar entre os conjuntos avaliados um novo conjunto corrente mais útil que o primeiro, e (4) um teste para o novo conjunto *CasosTEST* corrente, estabelecido em termos do valor de inadequação associado ao desempenho de *Agent* em *Amb*, instanciado com cada um dos NCasos no conjunto corrente, e do número de modificações realizadas até a iteração corrente para decidir se a função ação pode parar o processo.

Quanto ao componente (2), o estudo empregou uma função *Utilidade* no formato linear, conforme descrita no Capítulo 4, considerando valores de pesos iguais a 0,5 tanto para o atributo limpeza quanto para o atributo energia ($w_L = w_E = 0,5$), e acrescentou a esta função um ganho por caso de teste, que é igual ao número de salas sujas ao final das interações de *Agent* com *Amb*. Assim, considerando esta nova função, nos experimentos *Thestes* buscou selecionar casos de teste em que *Agent* teve um comportamento inadequado em termos de energia e limpeza, privilegiando aqueles casos em que o ambiente permaneceu com a maior quantidade de salas sujas ao final das interações.

Quanto aos componentes (1), (3) e (4), nos experimentos *Thestes* foi concretizado considerando noções presentes nos algoritmos genéticos (HOLLAND, 1975) e NSGA-II (DEB et al., 2000). Especificamente, quanto ao componente (1), no caso do algoritmo genético, a geração de novos casos de teste emprega o elitismo visando manter em uma geração corrente o melhor caso de teste previamente selecionado. A FIGURA 25 apresenta o ambiente de tarefa ilustrado na FIGURA 20 e um cromossomo equivalente expresso como uma cadeia de genes.

FIGURA 25 – Codificação do caso de teste



Fonte: Elaborado pelo autor (2013)

No contexto dos algoritmos genéticos aplicado ao problema de seleção proposto, um conjunto *CasosTEST* corrente, contendo as descrições de *m* ambientes de tarefa compostos de *n x n* salas, foi representado por uma população de cromossomos (indivíduos), ou seja, genes em cadeias de comprimento n^2, onde cada cadeia de genes na população codifica um ambiente e o valor de cada gene na cadeia codifica o estado da sala correspondente em termos de sujeira. Estas informações concretizam em *Thestes* os valores em *ParâmetrosSimulação*. A TABELA 6 apresenta os valores adotados em todos os experimentos, ou seja, cada população solução contém 10 casos (*NCasos*), cada caso formado por 25 salas. A avaliação de desempenho final considera que são realizadas 5 simulações (*Ns*) para cada caso, onde cada simulação dá origem a uma história contendo 25 episódios, ou seja, correspondentes às interações de *Agent* em *Amb* (N_{int}).

TABELA 6 – Informações *ParâmetrosSimulação*

NCasos	**n^2**	**N_{int}**	**Ns**
10	25	25	5

Fonte: Elaborado pelo autor (2013)

O *ModeloTransição* genérico proposto deve considerar os *NCasos* no conjunto *CasosTEST* corrente e indicar como escolher: (a) os casos do conjunto corrente que serão modificados e (b) as mudanças que serão realizadas nestes casos. Por exemplo, no caso dos experimentos com GA simples, empregou-se uma estratégia simples de seleção de pares para escolher os ambientes de tarefa em *CasosTEST* candidatos a sofrerem alterações, e operadores

genéticos simples de cruzamento e mutação, para escolher o subconjunto de salas em cada ambiente candidato e as mudanças a serem executada nas salas de cada subconjunto. Este tipo de estratégia sintetiza algumas das informações em *ParâmetrosBUSCA*. A TABELA 7 apresenta essas informações para os experimentos.

TABELA 7 – Informações em *ParâmetrosBUSCA* considerando GA e NSGA-II

Cruzamento		**Mutação**		**Seleção**
O_{Cros}	**P_{Cros}**	**O_{Mut}**	**P_{Mut}**	**O_{Sel}**
SinglePointCrossover	0,9	*Uniform*	0,6	Roleta

Fonte: Elaborado pelo autor (2013)

Essas informações descrevem a probabilidade e o operador de cruzamento (P_{Cros} e O_{Cros}) entre os casos de teste, a probabilidade e o operador de mutação (P_{Mut} e O_{Mut}) e o método para seleção dos melhores casos de teste (O_{Sel}). Mais especificamente, o operador de cruzamento *SinglePointCrossover* indica que apenas um ponto de cruzamento é selecionado, e 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. Encerrando as informações em *ParâmetrosBUSCA*, o valor de utilidade máxima ($U_{máx}$) é alto, significando que a condição de parada no mecanismo de teste da estratégia de busca foi definido considerando apenas um número ciclos de execuções da função ação ($K_{máx}$), ou seja, equivalente a 30 gerações no caso do GA e do NSGA-II.

5.1.5 Concretização do Agente *ProMon*

Conforme descrito no capítulo anterior, o agente *ProMon* considera que o conjunto de episódios ideais em uma interação *K*, $Ep_{ideais}{}^{K}$, é formado por todos os episódios possíveis de serem produzidos por *Agent** na interação, $Ep((P^K, A^K*))$, que satisfazem pelo menos uma das duas condições abaixo em relação ao episódio correspondente produzido por *Agent*, $Ep((P^K, A^K))$, na mesma interação:

(1) Para todo atributo *m* na medida de avaliação de desempenho:

$av_m(Ep((P^K, A^K*))) \geq av_m(Ep((P^K, A^K)))$;

(2) A^K* é melhor que ou é equivalente a A^K considerando o ponto de vista do projetista.

E, tão importante quanto o conjunto de episódios ideais, os episódios $Ep((P^K, A^K))$, produzidos pelo agente testado *Agent*, que não pertencem ao conjunto $Ep_{ideais}{}^{K}$ compõem o conjunto de episódios com falha Ep_{falhas}, ou seja, foram inseridos neste conjunto por satisfazerem uma das condições associadas a seguir:

(1)' existe pelo menos um atributo $\underline{x}$ em que $av_x(Ep((P^K, A^K*))) > av_x(Ep((P^K, A^K)))$, enquanto que no restante dos atributos $av_m(Ep((P^K, A^K*))) \geq av_m(Ep((P^K, A^K)))$;

(2)' A^K* é melhor que A^K.

Estas são as condições no antecedente das regras condição-ação do agente ProMon, que, quando satisfeitas, sugerem para o projetista o tipo de análise que ele deve realizar de maneira a identificar melhor o módulo de processamento de informação de *Agent* que está causando a falha.

No caso do mundo do agente aspirador de pó, três tipos de episódios com falha $Ep((P^K, A^K))$ podem ocorrer satisfazendo a Condição (1)', correspondentes às linhas 1, 5 e 6 da TABELA 5, que especifica a medida de avaliação de desempenho de *Agent*, conforme apresentado na TABELA 8.

TABELA 8 – Episódios com falhas para o agente aspirador de pó

Ep^K = (Percepção^K,	**Ação^K)**	**$av_E(Ep^K)$**	**$av_L(Ep^K)$**	**Falha**
..., L, ...	Asp	-1.0	0.0	Linha1
..., S, ...	Dir, Esq, Ac, Ab	-2.0	-1.0	Linha5
..., S, ...	N-op	0.0	-1.0	Linha6

Fonte: Elaborado pelo autor (2013)

Quanto aos episódios com falha que concorrem para a satisfação da Condição (2)', dois tipos podem ocorrer, considerando-se duas situações possíveis identificadas pelo projetista, mas que não estão explicitamente definidas na TABELA 5:

(a). *Agent* movimentou-se para uma sala vizinha diferente de uma sala vizinha que continha sujeira;

(b). *Agent* movimentou-se desnecessariamente para uma sala vizinha visitada anteriormente.

Assim, considerando estas cinco possibilidades de episódios faltosos e as quatro regras condição-ação genéricas descritas na

FIGURA 19, no capítulo anterior, a TABELA 9 ilustra apenas vinte falhas possíveis e suas justificativas, possíveis de serem detectadas pelo projetista a partir das mensagens enviadas pelo agente *ProMon* a respeito das falhas associadas aos episódios.

TABELA 9 – Falhas possíveis de serem detectadas pelo Projetista

Falha	Subsistema	Justificativa
1	ver	Linha 1 Tabela 5 / R1
2	ver	Linha 5 Tabela 5 / R1
3	ver	Linha 6 Tabela 5 / R1
4	ver	Sala vizinha suja / R2
5	ver	Sala visitada / R2
6	ação	Linha 1 Tabela 5 / R1
7	ação	Linha 5 Tabela 5 / R1
8	ação	Linha 6 Tabela 5 / R1
9	ação	Sala vizinha suja / R2
10	ação	Sala visitada / R2
11	próximo	Linha 1 Tabela 5 / R3
12	próximo	Linha 5 Tabela 5 / R3
13	próximo	Linha 6 Tabela 5 / R3
14	próximo	Sala vizinha suja / R4
15	próximo	Sala visitada / R4
16	ação	Linha 1 Tabela 5 / R3
17	ação	Linha 5 Tabela 5 / R3
18	ação	Linha 6 Tabela 5 / R3
19	ação	Sala vizinha suja / R4
20	ação	Sala visitada / R4

Fonte: Elaborado pelo autor (2013)

Vale ressaltar, na coluna justificativas, além das informações sobre as linhas da TABELA 5 e sobre os outros dois tipos de falhas considerados, a identificação do subsistema que está causando a falha é providenciada pela interpretação do projetista, a posteriori, das mensagens enviadas por *ProMon* considerando uma das regras condição-ação genéricas descritas na Seção 4.7. As falhas que consideram a presença de problemas simultaneamente em dois módulos de processamento da informação foram omitidas, considerando-se que o projetista supõe que o subsistema de tomada de decisão dos agentes testados será capaz de produzir os mesmos resultados que o agente ideal $Agent^*$ caso venham a possuir subsistemas de percepção e/ou de atualização de estado interno equivalente ao de $Agent^*$.

5.1.6 Experimentos

Os experimentos realizados para demonstrar o funcionamento da abordagem para o teste de agentes racionais, utilizando o agente *Thestes* para resolver problemas de seleção de casos de teste, consideraram os quatro programas agentes esboçados na Subseção 5.1.3 e as informações em *ParâmetrosSimulação* e *ParâmetrosBUSCA* descritas na Subseção 1.4. A TABELA 10 descreve os 6 (seis) experimentos realizados.

TABELA 10 – Experimentos

Exp	Agent-Amb	Busca
1	RS_Parcial	GA Simples
2	RS_Parcial_alterado	GA Simples
3	RS_Total	GA Simples
4	RM_Parcial	GA Simples
5	RS_Parcial_NSGA_II	NSGA-II
6	RM_Parcial_NSGA_II	NSGA-II

Fonte: Elaborado pelo autor (2013)

Nos Experimentos 1-4 os principais componentes de *Thestes* foram concretizados considerando o algoritmo genético simples (GA Simples). O Experimento 1 considerou o problema de seleção de ação do agente reativo simples regras condição-ação denominado *RS_Parcial* em um ambiente parcialmente observável. O Experimento 2, a alteração deste agente, ou seja, *RS_alterado*. O Experimento 3 considerou o terceiro programa agente em um ambiente com observabilidade total, ou seja, *RS_Total*. O Experimento 4, avalia o programa agente reativo baseado em modelo *RM_Parcial* em um ambiente parcialmente observável. Nos Experimentos 5 e 6 os componentes de *Thestes* foram concebidos considerando o NSGA-II. O Experimento 5 considerou o problema de seleção de ação do agente reativo simples regras condição-ação denominado *RS_Parcial_NSGA_II* em um ambiente parcialmente observável. O Experimento 6 avalia o programa agente reativo baseado em modelo *RM_Parcial_NSGA_II* em um ambiente parcialmente observável.

5.2 Apresentação e Análise dos Resultados

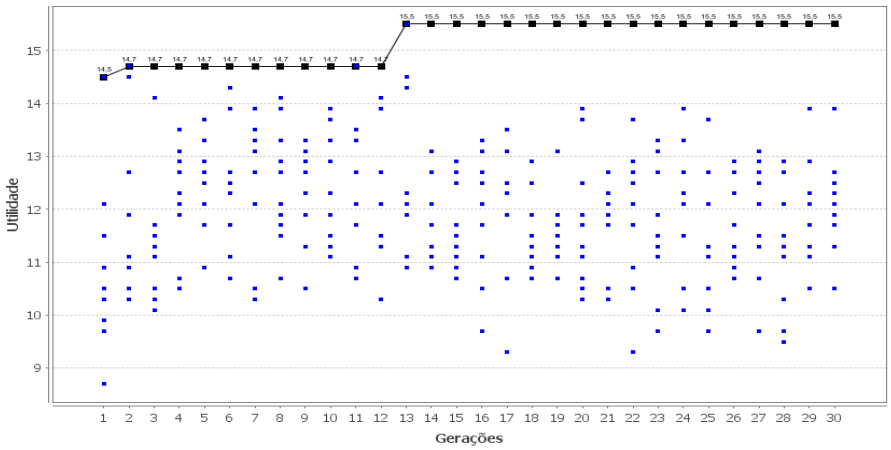
Nesta seção são apresentados e discutidos os resultados obtidos na avaliação de cada experimento definido. A Seção 5.2.1 apresenta e discute os resultados dos experimentos 1, 3 e 4, experimentos cujos agentes possuem regras condição-ação definidas para atender a arquitetura do agente, considerando a análise de resultados do agente *Thestes*. A Seção 5.2.2

apresenta a análise dos experimentos 1, 3 e 4 considerando o monitoramento e o diagnóstico das falhas realizado pelo agente *ProMon*. Enquanto a Seção 5.2.3 apresenta e discute os resultados do experimento 2, cujas regras condição-ação do agente são definidas aleatoriamente sob a análise do agente *Thestes* e a Seção 5.2.4 apresenta os resultados do experimento 2 a partir das funcionalidades do agente *ProMon.*

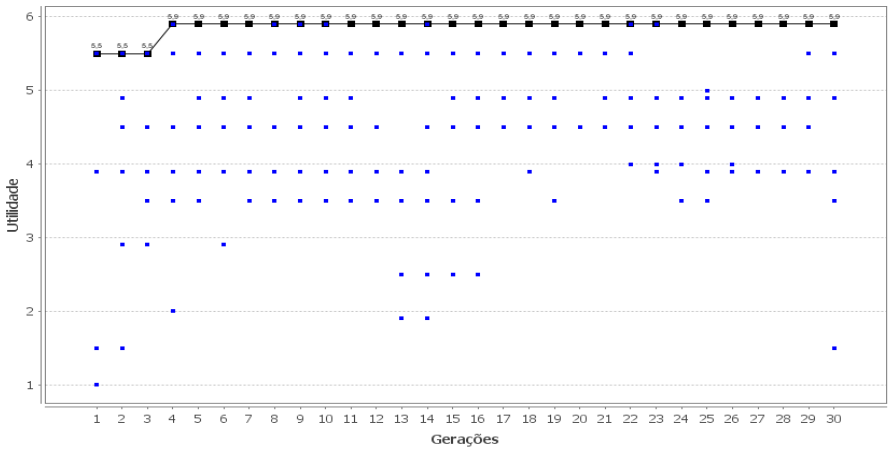
5.2.1 Experimentos 1, 3 e 4 – Resultados *Thestes*

A FIGURA 26 apresenta os resultados produzidos por *Thestes* empregando a função *Utilidade* no formato linear, descrita no Capítulo 4 , Seção 4.6, com valores de pesos iguais para os dois objetivos (energia e limpeza) na medida de avaliação, ou seja, $w_L = w_E = 0,5$, ao longo de 30 gerações. Em (a) são apresentados os resultados do Experimento 1; em (b) os resultados para o Experimento 3; e em (c) os resultados para o Experimento 4.

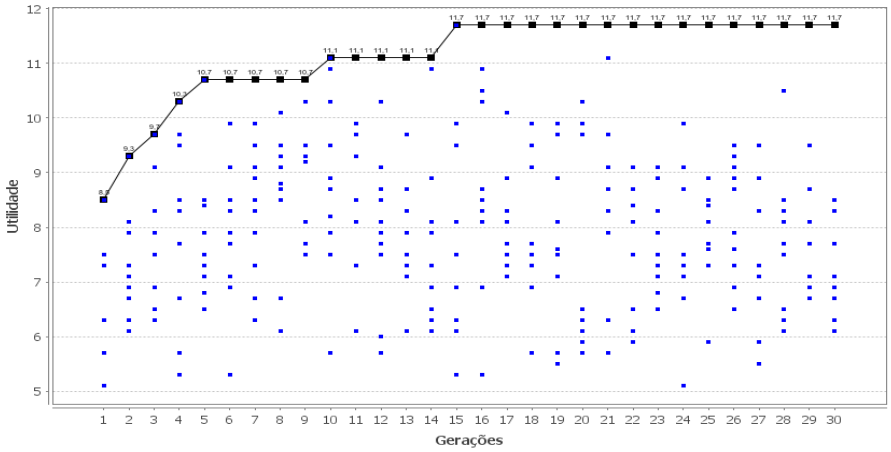
FIGURA 26 – Valores de utilidade de *CasosTEST* em 30 gerações (Experimentos 1, 3 e 4)



(a)



(b)



(c)

Fonte: Elaborado pelo autor (2013)

Os pontos não lineares representam os 10 casos de teste em *CasosTEST* em cada geração. Os pontos marcados linearmente identificam o melhor caso de teste em *CasosTEST* a cada geração. A utilização do elitismo previne a perda do melhor caso de teste encontrado em uma geração anterior. A TABELA 11 destaca a geração e o melhor valor de utilidade encontrado para os três agentes.

TABELA 11 – Geração e Utilidade do melhor caso em *CasosTEST*

Exp	Agent-Amb	Geração	Utilidade
1	RS_Parcial	14	15,5
3	RS_Total	4	5,9
4	RM_Parcial	10	11,7

Fonte: Elaborado pelo autor (2013)

No caso do agente reativo simples com observabilidade parcial (FIGURA 26 em (a)), o melhor caso de teste foi obtido na décima quarta geração, com valor de utilidade igual a 15,5. Este caso serviu como referencial para os casos em *CasosTEST* nas gerações posteriores com valores de utilidade menores que o melhor. No caso do agente reativo simples com observabilidade total (FIGURA 26 em (b)), o melhor caso de teste em *CasosTEST* foi obtido na quarta geração com valor de utilidade igual a 5,9. No caso do agente reativo baseado em modelo (FIGURA 26 em (c)), o melhor caso de teste foi obtido na décima quinta geração, com valor de utilidade igual a 11,7.

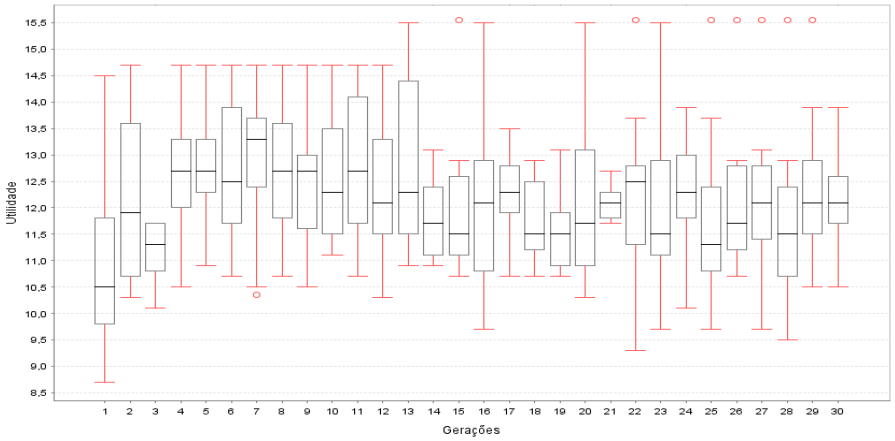
Conforme esperado, considerando os valores de utilidade dos melhores casos de teste, o agente *Thestes* detectou que o agente mais inadequado para o ambiente parcialmente observável é o reativo simples, quando comparado com o agente baseado em modelos. Neste

caso, a anotação das salas que foram visitadas pelo subsistema de atualização de estado interno (próximo) do agente baseado em modelos possibilitou a concepção de um subsistema de tomada de decisão (ação) mais refinado para o agente baseado em modelos. O conjunto de regras componentes deste subsistema evitou que o agente retornasse desnecessariamente às salas que já foram visitadas em interações anteriores com o ambiente, o que não foi possível para o agente reativo simples.

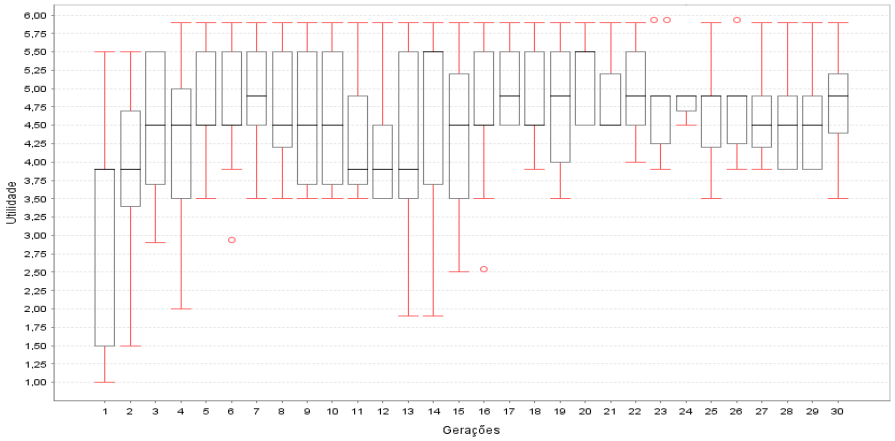
Por outro lado, também esperado, o valor de inadequação do agente reativo simples em um ambiente totalmente observável é bem menor que os valores dos outros dois agentes em um ambiente parcialmente observável. Comparando com o agente baseado em modelos, dificilmente o projetista conseguirá conceber um subsistema de atualização de estado interno que prevaleça em termos de desempenho a um subsistema de percepção (ver) capaz de perceber todo o ambiente e que, com isso, permite a concepção de um subsistema de tomada de decisão capaz de selecionar as ações que sejam realmente racionais em cada interação com o ambiente.

A FIGURA 27 ilustra a variação do valor de utilidade dos 10 casos em *CasosTEST* em cada geração. O box-plot é formato pelos quartis inferior (25%), mediana (50%) e superior (75%), assim como os valores máximo e mínimo entre os casos. Em (a) é apresentado o box-plot do Experimento 1, em (b) do Experimento 3 e em (c) do Experimento 4.

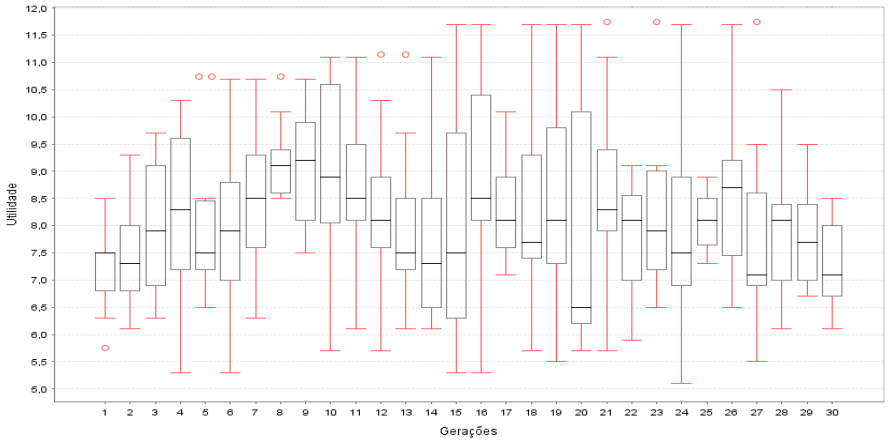
FIGURA 27 – Box-plot dos valores de utilidade de 10 casos *CasosTEST* em 30 gerações



(a)



(b)



(c)

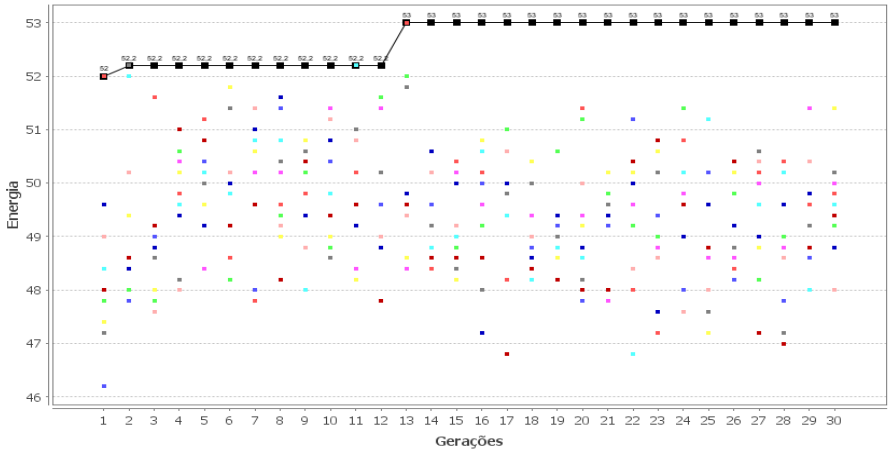
Fonte: Elaborado pelo autor (2013)

Nos experimentos, *Thestes* buscou selecionar um conjunto *CasosTEST* satisfatório. Com a adoção do elitismo, em cada geração *Thestes* manteve no conjunto o caso de teste em que o agente teve o comportamento mais inadequado em todas as gerações prévias, ou seja, cujo valor de utilidade é igual ao valor máximo entre os 10 casos. Casos cujos valores de utilidade são inferiores ao valor mínimo não são importantes, visto que quanto mais próximo do valor mínimo, mais adequado é o comportamento e melhor o desempenho do agente no caso de teste.

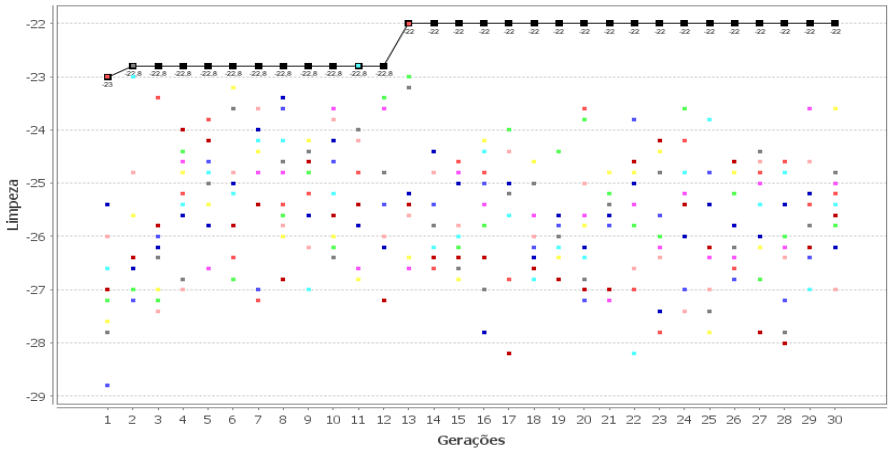
Enquanto que o projetista visou conceber agentes capazes de limpar o número de salas gastando o mínimo de energia, inversamente, a função ação de buscou selecionar os casos em cada geração em que o agente limpou o menor número e gastou o máximo de energia. A

FIGURA 28 destaca os casos em cada geração e os valores de inadequação de energia em (1) e os valores de inadequação de limpeza em (2) para o Experimento 1 em (a), Experimento 3 em (b) e Experimento 4 em (c).

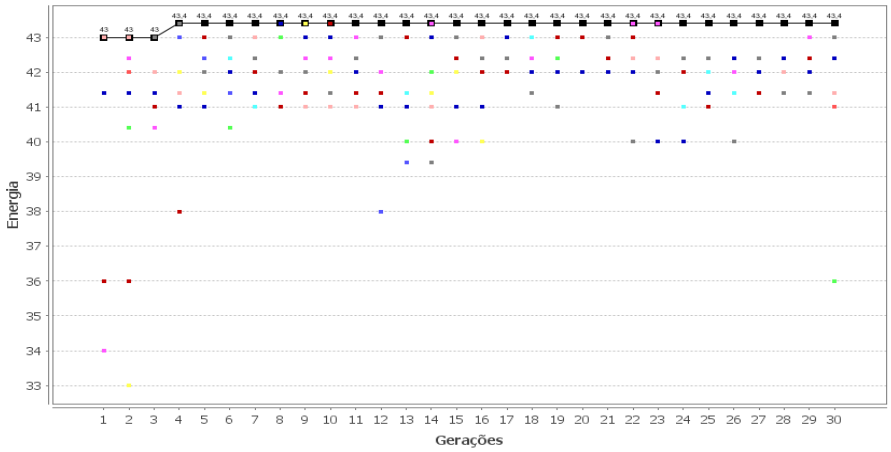
FIGURA 28 – Valores de inadequação de energia (1) e limpeza (2) (Experimentos 1, 3 e 4)



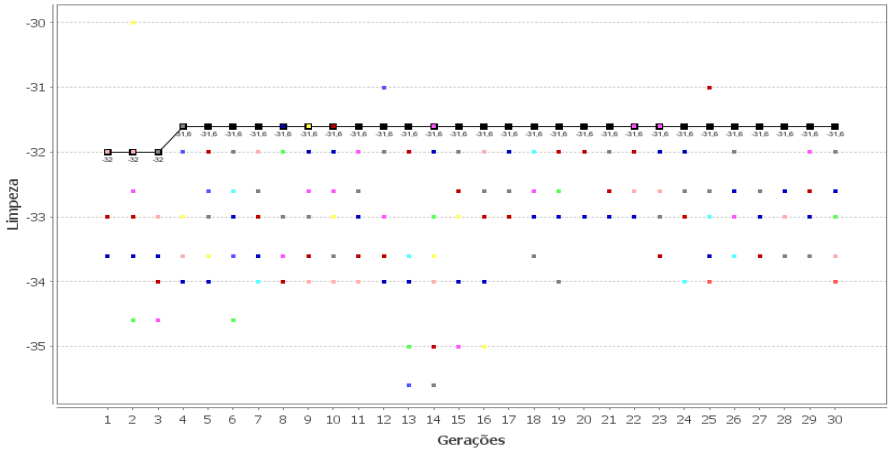
(a1)



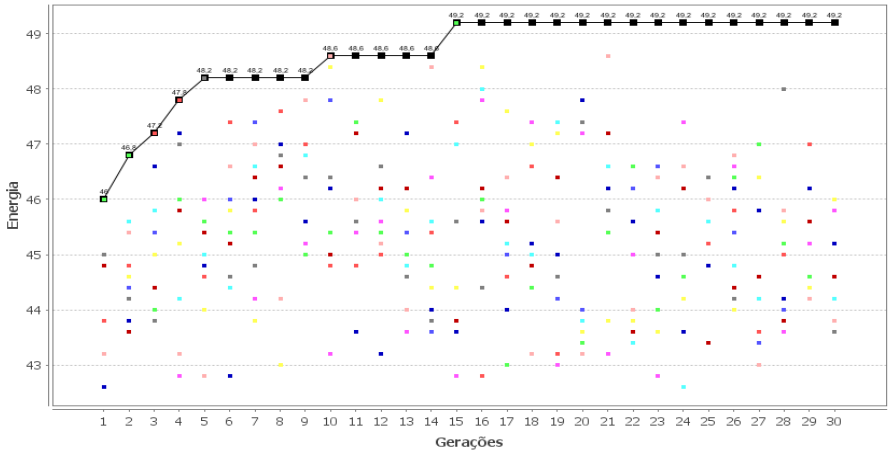
(a2)



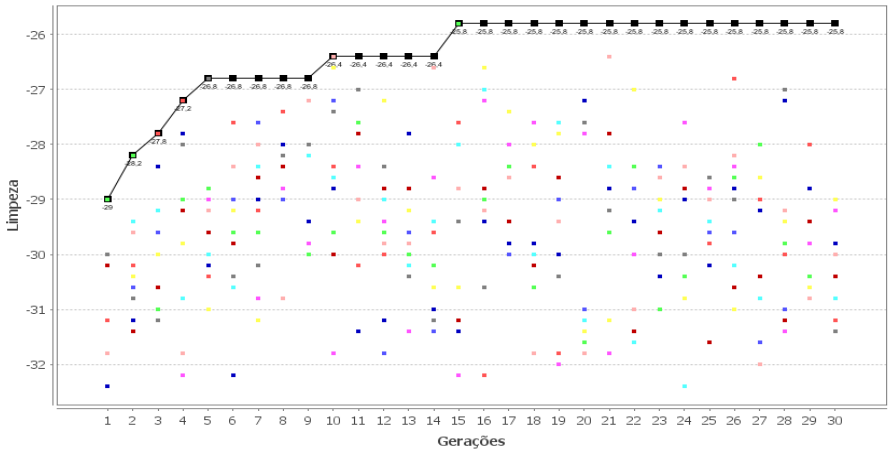
(b1)



(b2)



(c1)



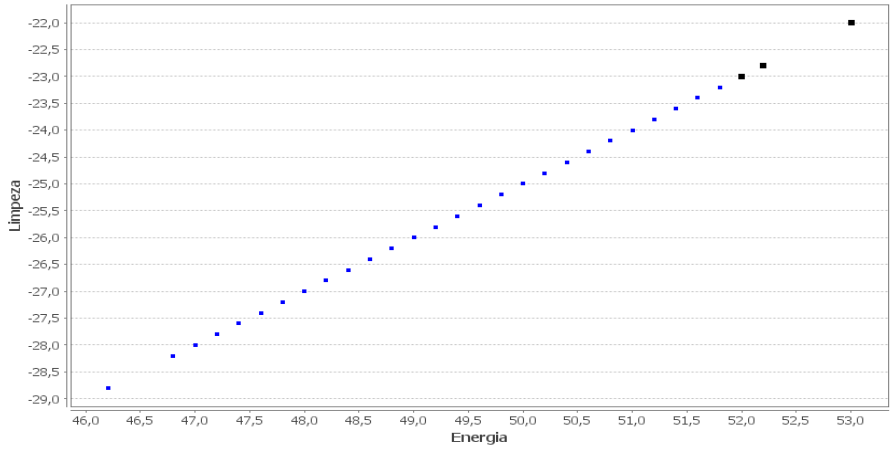
(c2)

Fonte: Elaborado pelo autor (2013)

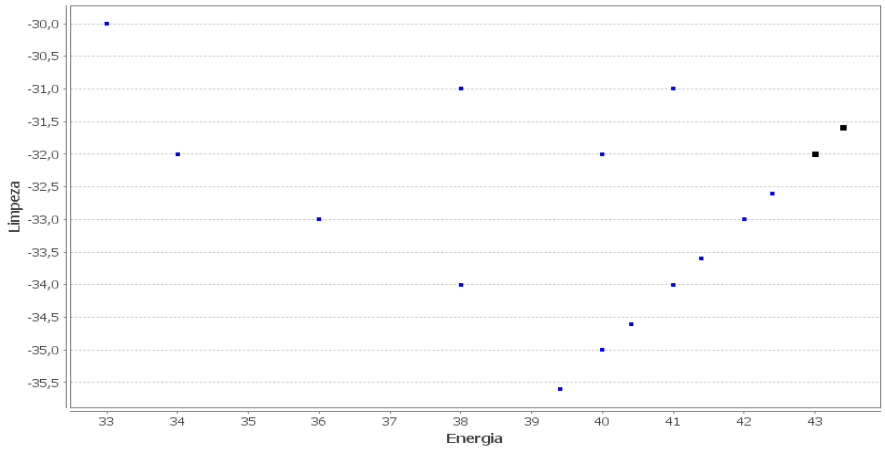
A

FIGURA 29 apresenta os valores das funções de inadequação de energia e limpeza associados aos 10 casos de teste em *CasosTEST* nas 30 gerações para o Experimento 1 em (a), Experimento 3 em (b) e Experimento 4 em (c).

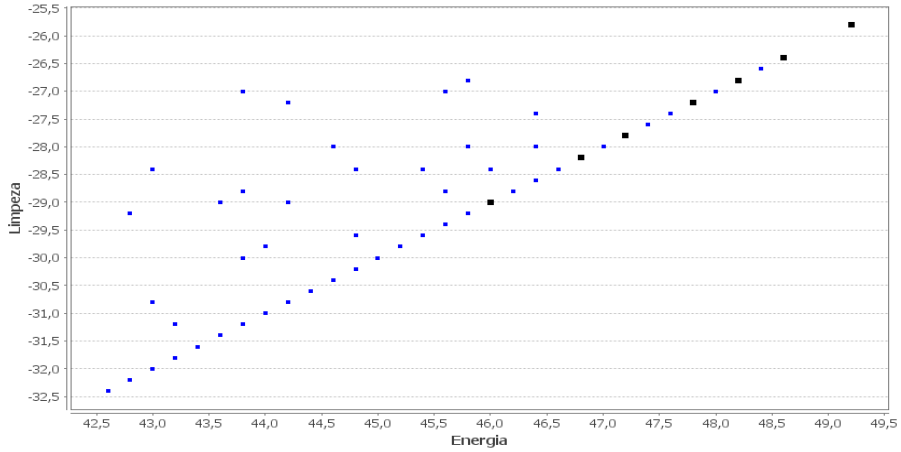
FIGURA 29 – Valores de inadequação de energia e limpeza (Experimentos 1, 3 e 4)



(a)



(b)



(c)

Fonte: Elaborado pelo autor (2013)

A TABELA 12 destaca os valores de energia e limpeza alcançados pelos agentes nos melhores casos de teste nos três experimentos.

TABELA 12 – Atributos energia e limpeza nos casos selecionados

Exp	Agent-Amb	Energia	Limpeza
1	RS_Parcial	52,0	-23,0
		52,2	-22,8
		53,0	-22,0
3	RS_Total	43,0	-32,0
		43,4	-31,6
4	RM_Parcial	46,0	-29,0
		46,8	-28,2
		47,2	-27.8
		47.8	-27,2
		48.2	-26.8
		48.6	-26.4
		49.2	-25.8

Fonte: Elaborado pelo autor (2013)

Comparando os resultados entre os agentes, considerando o caso de teste em *CasosTEST* com maior valor de utilidade conforme FIGURA 26, é possível observar que para ambos os atributos, energia e limpeza, apresentados na

FIGURA 28, o agente reativo simples com observabilidade parcial possui um comportamento mais inadequado, com $-f_E = 53,0$ e $-f_L = -22,0$, enquanto o agente reativo simples com observabilidade total possui um comportamento menos inadequado, com $-f_E = 43,4$ e $-f_L = -31,6$ e o agente reativo baseado em modelo possui valor de inadequação entre os demais agentes, com $-f_E = 49,2$ e $-f_L = -25,8$. Isto pode ser justificado devido às propriedades internas de cada um dos agentes.

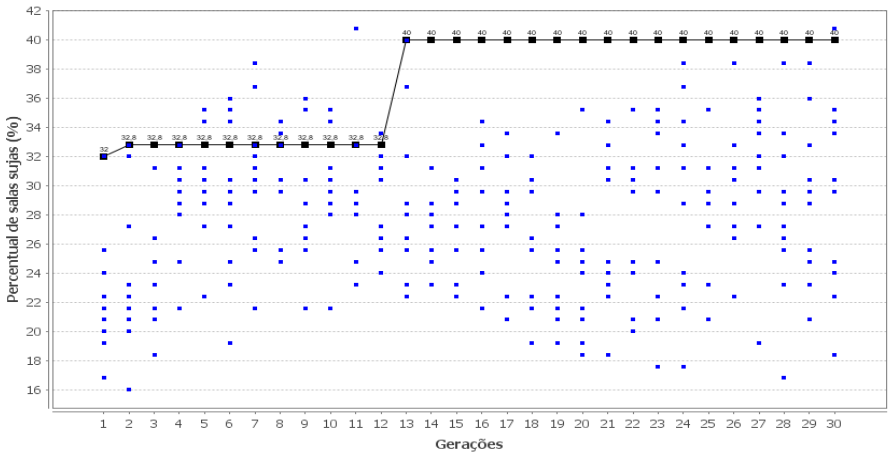
O agente reativo simples com observabilidade parcial possui visão limitada do ambiente, as ações que ele seleciona nem sempre são racionais. Por exemplo, como as ações de movimento são escolhidas aleatoriamente, nem sempre essas ações levam a sala suja mais próxima, fazendo com que o agente gaste mais energia do que o necessário e pontue menos em limpeza. Em contrapartida, como o agente reativo simples com observabilidade total possui uma completa percepção do ambiente, as ações selecionadas visam maximizar o desempenho dos atributos energia e limpeza, considerando que o agente sempre escolhe a melhor ação a cada percepção. Quanto ao agente reativo baseado em modelo, seu estado interno com o histórico de percepções obtidas até o momento permite que o agente evite o retorno às salas antes visitadas e, consequentemente, selecione ações melhores que as do agente reativo simples com observabilidade parcial.

A princípio, como os dois objetivos na medida de avaliação de desempenho tem o mesmo valor de importância ($w_L = w_E = 0,5$) na função utilidade, a abordagem privilegia os

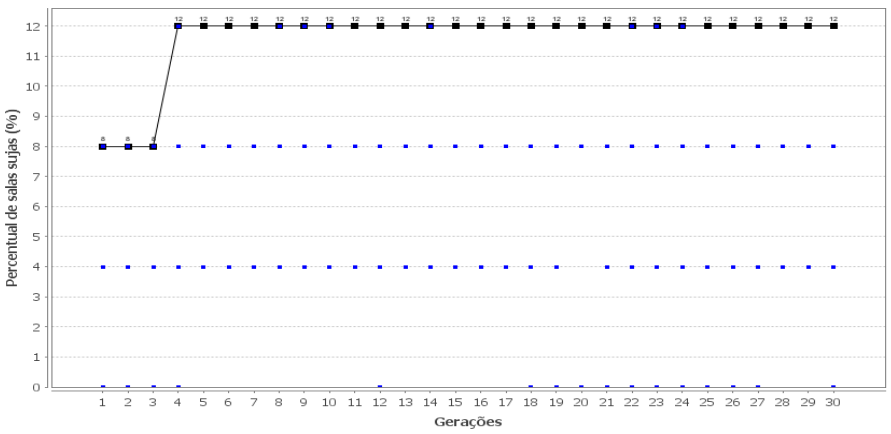
casos em que *Agent* tem um comportamento mais inadequado em termos do consumo de energia que em termos de limpeza, quase que maximizando este nível de inadequação conforme indicado em todas as gerações. Isto se justifica, pois a medida de avaliação de desempenho pontua negativamente em termos de energia todos os episódios possíveis para *Agent*, devido o agente sempre gastar energia ao executar suas ações, e positivamente em termos de limpeza os episódios.

Entretanto, o acréscimo de um ganho por caso de teste ao valor da função utilidade, igual ao número de salas sujas ao final das interações de *Agent* com *Amb*, minimiza o efeito mencionado acima e privilegia aqueles casos em que o ambiente permaneceu com a maior quantidade de salas sujas ao final das interações. A FIGURA 30 apresenta o percentual de salas sujas deixadas pelo agente após realizar 25 interações com o ambiente. Em (a) o percentual de salas sujas no Experimento 1, em (b) no Experimento 3 e em (c) o Experimento 4.

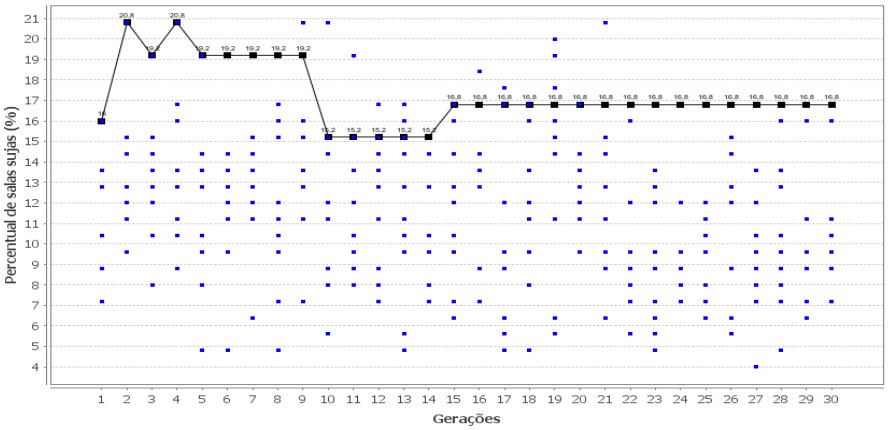
FIGURA 30 – Percentual de salas sujas nos *CasosTEST* (Experimentos 1, 3 e 4)



(a)



(b)



(c)

Fonte: Elaborado pelo autor (2013)

A partir do exposto na FIGURA 30 é possível observar que, no geral, *Thestes* escolheu os casos de teste que apresentam maior quantidade de salas sujas ao final dos testes, sendo estes considerados os casos de teste com pior desempenho.

A TABELA 13 destaca o percentual de salas sujas deixada pelo *Agent* em *Amb* nos casos de teste marcados linearmente nos três experimentos.

TABELA 13 – Percentual de salas sujas nos casos selecionados

Exp	Agent-Amb	Percentual de salas sujas
1	RS_Parcial	32,0
		32,8
		40,0
3	RS_Total	8,0
		12,0
4	RM_Parcial	16,0
		20,8
		19,2
		20,8
		19,2
		15,2
		16,8

Fonte: Elaborado pelo autor (2013)

Comparando os resultados entre os agentes, considerando o caso de teste em *CasosTEST* com maior percentual de salas sujas conforme TABELA 13, é possível observar que o agente reativo simples com observabilidade parcial possui o maior percentual de salas sujas, 40,0%, enquanto o agente reativo simples com observabilidade total possui o menor percentual de salas sujas, 12,0%, e o agente reativo baseado em modelo possui o percentual de 16,8% de salas sujas, cujo valor está entre os demais agentes. Isto pode ser justificado devido às propriedades de cada um dos agentes.

5.2.2 Experimentos 1, 3 e 4 – Funcionalidades *ProMon*

As TABELA 14 a TABELA 21 ilustram os episódios 1-9 da história de *Agent* em *Amb*, inicializado com um dos casos de teste que obteve o melhor valor médio de utilidade nas simulações para o Experimento 1, ou seja, quando $-f_E = 49{,}0$ e $-f_L = -26{,}0$ e Falhas = 11. As tabelas apresentam também os episódios ideais associados gerados por *Agent**, seus valores de avaliação de desempenho e as falhas correspondentes identificadas pelo agente *ProMon*. A TABELA 14 apresenta os quatro primeiros episódios realizados pelo agente *RS_Parcial*, enquanto que a TABELA 15 apresenta os episódios ideais correspondentes produzidos por *Agent**. A TABELA 16, o significado de cada símbolo empregado na representação dos estados do ambiente.

TABELA 14 – Quatro primeiros episódios na história de RS_Parcial

História Realizada – Ep^1, Ep^2, Ep^3, Ep^4							
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	2.0	-1.0	2.0	-1.0	1.0	-2.0
Falha		**Falha**		**Falha**		**Falha**	
Não		Não		Não		Não	
[**0**, 0, 0, 0, 1]		[**0**, 0, 0, 0, 1]		[**0**, 0, 0, 0, 1]		[**0**, 0, 0, 0, 1]	
[**0**, 0, 1, 1, 0]		[**0**, **0**, 1, 1, 0]		[**0**, **0**, 1, 1, 0]		[**0**, **0**, 1, 1, 0]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, **1**, 1, 1, 1]		[0, **0**, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 15 – Conjunto de episódios ideais produzidos por *Agent**

Histórias Ideais – Ep^K (K = 1, ..., 4)							
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	**-1.0**	**2.0**	**-1.0**	**2.0**	**-1.0**	**1.0**	**-2.0**
[**0**, **0**, 0, 0, 1]		[**0**, **0**, **0**, 0, 1]		[**0**, **0**, **0**, 0, 1]		[**0**, **0**, **0**, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]		[0, 0, **1**, 1, 0]		[0, 0, **0**, 1, 0]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[**0**, **0**, 0, 0, 1]		[**0**, **0**, 0, 0, 1]		[**0**, **0**, 0, 0, 1]		[**0**, **0**, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, **0**, 1, 1, 0]		[0, **0**, **1**, 1, 0]		[0, **0**, **0**, 1, 0]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[**0**, **0**, 0, 0, 1]		[**0**, **0**, 0, 0, 1]		[**0**, **0**, 0, 0, 1]		[**0**, **0**, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, **0**, 1, 1, 0]		[0, **0**, 1, 1, 0]		[0, **0**, 1, 1, 0]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, **1**, 1, 1, 1]		[0, **0**, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	

[**0**, 0, 0, 0, 1] [**0**, 0, 1, 1, 0] [0, 1, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]	[**0**, 0, 0, 0, 1] [**0**, **0**, 1, 1, 0] [0, 1, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]	[**0**, 0, 0, 0, 1] [**0**, **0**, 1, 1, 0] [0, **1**, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]	[**0**, 0, 0, 0, 1] [**0**, **0**, 1, 1, 0] [0, **0**, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]
[**0**, 0, 0, 0, 1] [**0**, 0, 1, 1, 0] [0, 1, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]	[**0**, 0, 0, 0, 1] [**0**, 0, 1, 1, 0] [**0**, 1, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]	[**0**, 0, 0, 0, 1] [**0**, 0, 1, 1, 0] [**0**, **1**, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]	[**0**, 0, 0, 0, 1] [**0**, 0, 1, 1, 0] [**0**, **0**, 1, 1, 1] [0, 0, 0, 1, 1] [0, 0, 1, 1, 1]

Fonte: Elaborado pelo autor (2013)

TABELA 16 – Significado dos símbolos

Símbolo	Significado
0	sala limpa
1	sala suja
0	sala limpa visitada
1	sala suja visitada
0	agente está em sala limpa
1	agente está em sala suja
0	agente estava em sala limpa
1	agente estava em sala suja
0	agente está em sala limpa visitada
1	agente está em sala suja visitada
0,	agente está em sala limpa e última ação foi na sala
1	agente está em sala suja e última ação foi na sala

Fonte: Elaborado pelo autor (2013)

O programa agente *RS_Parcial* não cometeu falhas nas quatro primeiras interações que manteve com *Amb*. A moldura na terceira linha da TABELA 15 identifica que os quatro episódios produzidos por *RS_Parcial* pertencem aos conjuntos de episódios ideais Ep_{ideais}^{K} gerados nas quatro interações (K = 1, ..., 4) de *Agent** com *Amb* (cada coluna na TABELA 15 corresponde a um conjunto de episódios ideais em uma interação). As TABELA 17 e TABELA 18 ilustram os dois episódios seguintes em que *RS_Parcial* cometeu uma Falha 4. A TABELA 17 ilustra o quinto episódio $Ep((P^5, A^5)) = Ep((...L..., Esq))$ de *RS_Parcial* em *Amb* e o episódio ideal $Ep((P^5, A^5*)) = Ep((...L..., Dir))$ produzido por *Agent**. A TABELA 18 ilustra o sexto episódio $Ep((P^6, A^6)) = Ep((...L..., Ab))$ de *RS_Parcial* em *Amb* e o episódio ideal $Ep((P^6, A^6*)) = Ep((...L..., Dir))$ produzido por *Agent**.

TABELA 17 – Primeiro episódio com falha na história de *RS_Parcial*

Histórias – Ep[5]			
Realizada		**Ideal**	
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**
2.0	-1.0	2.0	-1.0
Falha			
4			
[0, 0, 0, 0, 1]		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 18 – Segundo episódio com falha na história de *RS_Parcial*

Histórias – Ep[6]			
Realizada		**Ideal**	
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**
2.0	-1.0	2.0	-1.0
Falha			
4			
[0, 0, 0, 0, 1],		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0],		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1],		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1],		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

O agente *RS_Parcial* cometeu uma Falha 4 no episódio realizado na TABELA 17, ou seja, existe uma ação melhor que 'Esq' no quinto episódio, isto é, 'Dir' que leva o agente para uma sala vizinha suja. Neste caso, a Falha 4 indica que quem está causando a falha é a função ver do agente, visto que a função ação de *RS_Parcial* poderia ser capaz de escolher a ação 'Dir' se soubesse que a sala à sua direita estava suja. Neste caso, apesar dos valores de avaliação de *RS_Parcial* e *Agent** serem iguais, o agente evita ganhar pontos ao deixar de se movimentar para uma sala suja. A TABELA 19 apresenta os dois próximos episódio produzido *RS_Parcial* e a TABELA 20 apresenta os episódios ideais produzidos por *Agent**.

TABELA 19 – Episódios na história de *RS_Parcial*

História Realizada – Ep[7] e Ep[8]			
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**
2.0	-1.0	2.0	-1.0
Falha		**Falha**	
Não		Não	
[0, 0, 0, 0, 1]		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	

[0, 0, 0, 1, 1]	[0, 0, 0, 1, 1]
[0, 0, 1, 1, 1]	[0, 0, 1, 1, 1]

Fonte: Elaborado pelo autor (2013)

TABELA 20 – Episódios ideais produzidos por *Agent**

Histórias Ideais – Ep[7] e Ep[8]			
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**
2.0	-1.0	2.0	-1.0
[0, 0, 0, 0, 1]		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 0, 1]		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 0, 1]		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 0, 1]		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

A moldura na primeira linha da TABELA 20 indica que os episódios produzidos por *RS_Parcial* nas interações 7 e 8 pertencem aos conjuntos de episódios ideais nestas interações (cada coluna um conjunto de episódios), o agente testado movimentou-se para uma sala vizinha não visitada próxima a uma sala que contém sujeira. A TABELA 21 apresenta o nono episódio gerado tanto por *RS_Parcial* quanto por *Agent**.

TABELA 21 – Nono episódio da história de *RS_Parcial*

Histórias – Ep[9]			
Realizada		**Ideal**	
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**
2.0	-1.0	2.0	-1.0
Falha			
4 e 5			
[0, 0, 0, 0, 1]		[0, 0, 0, 0, 1]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	

[0, 0, 1, 1, 1]	[0, 0, 1, 1, 1]

Fonte: Elaborado pelo autor (2013)

Percebe-se na tabela que o episódio de *RS_Parcial* é diferente do episódio ideal produzido por *Agent**, ou seja: $Ep((P^9, A^9)) = Ep((...L..., Esq)) \neq Ep((P^9, A^9*)) = Ep((...L..., Dir))$. Semelhantemente ao que aconteceu no quinto episódio, o agente deixou de se movimentar para uma sala vizinha suja, assim, o projetista identificou a Falha 4. Entretanto, diferentemente do que ocorreu antes, *RS_Parcial* movimentou-se para uma sala anteriormente visitada e que estava limpa (0), logo, o projetista identificou a Falha 5. Vale ressaltar, este tipo de falha no mundo do programa agente aspirador de pó *RS_Parcial*, pode-se considerar que a falha na função ver do agente pode ser controlada com a introdução de um módulo de atualização de estado interno, que está ausente no agente reativo simples com observabilidade parcial.

As TABELA 22 a TABELA 25 ilustram os seis episódios inicias e finais da história de *Agent* em *Amb*, inicializado com um dos casos de teste que obteve o melhor valor médio de utilidade nas simulações para o Experimento 3, ou seja, quando $-f_E = 42{,}0$ e $-f_L = -33{,}0$ e Falhas = 0.0. As tabelas apresentam também os episódios ideais associados gerados por *Agent** e seus valores de avaliação de desempenho. A TABELA 22 apresenta os seis primeiros episódios realizados pelo agente *RS_Total*, enquanto que a TABELA 23 apresenta os episódios ideais correspondentes produzidos por *Agent**. Na TABELA 24 e TABELA 25 são apresentados, respectivamente, os seis últimos episódios realizados pelo agente *RS_Total* e os episódios ideais produzidos por *Agent**.

TABELA 22 – Seis primeiros episódios na história de *RS_Total*

História Realizada – Ep^1, Ep^2, Ep^3, Ep^4, Ep^5, Ep^6											
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	1.0	-2.0	2.0	-1.0	1.0	-2.0	2.0	-1.0	2.0	-1.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**		**Falha**	
Não		Não		Não		Não		Não		Não	
[0, 0, 1, 0, 0]		[0, 0, 1, 0, 0]		[0, 0, 1, 0, 0]		[0, 0, 1, 0, 0]		[0, 0, 1, 0, 0]		[0, 0, 1, 0, 0]	
[1, 1, 0, 1, 0]		[0, 1, 0, 1, 0]		[0, 1, 0, 1, 0]		[0, 0, 0, 1, 0]		[0, 0, 0, 1, 0]		[0, 0, 0, 1, 0]	
[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0],		[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]		[0, 0, 1, 1, 0]	
[1, 0, 0, 1, 1]		[1, 0, 0, 1, 1]		[1, 0, 0, 1, 1]		[1, 0, 0, 1, 1]		[1, 0, 0, 1, 1]		[1, 0, 0, 1, 1]	
[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]	

Fonte: Elaborado pelo autor (2013)

TABELA 23 – Conjunto de episódios ideais produzidos por *Agent** para os seis episódios iniciais de *RS_Total*

História Ideais – Ep^K (K = 1, ..., 6)											
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	1.0	-2.0	2.0	-1.0	1.0	-2.0	2.0	-1.0	2.0	-1.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**		**Falha**	
Não		Não		Não		Não		Não		Não	

[**0**, 0, 1, 0, 0] [**1**, 1, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, 1, 0, 1, 0] [0, 0, 1, 1, 0], [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **1**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, **0**, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, **0**, **1**, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]
[**0**, 0, 1, 0, 0] [**1**, 1, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, 1, 0, 1, 0] [0, 0, 1, 1, 0], [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **1**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, **0**, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, **1**, 0, 0] [**0**, **0**, **0**, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]
[**0**, 0, 1, 0, 0] [**1**, 1, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, 1, 0, 1, 0] [0, 0, 1, 1, 0], [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **1**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, **0**, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, **0**, **1**, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]
[**0**, 0, 1, 0, 0] [**1**, 1, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, 1, 0, 1, 0] [0, 0, 1, 1, 0], [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **1**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, **0**, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, **0**, 1, 0] [0, 0, **1**, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]
[**0**, 0, 1, 0, 0] [**1**, 1, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, 1, 0, 1, 0] [0, 0, 1, 1, 0], [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **1**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, **0**, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, **0**, 1, 0] [0, 0, **1**, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]
[**0**, 0, 1, 0, 0] [**1**, 1, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, 1, 0, 1, 0] [0, 0, 1, 1, 0], [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **1**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, 0, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, **0**, 1, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]	[**0**, 0, 1, 0, 0] [**0**, **0**, 0, 1, 0] [0, **0**, **1**, 1, 0] [1, 0, 0, 1, 1] [1, 1, 0, 0, 0]

Fonte: Elaborado pelo autor (2013)

TABELA 24 – Seis últimos episódios na história de *RS_Total*

História Realizada – Ep^{20}, Ep^{21}, Ep^{22}, Ep^{23}, Ep^{24}, Ep^{25}											
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	1.0	-2.0	2.0	-1.0	2.0	-1.0	2.0	-1.0	2.0	-1.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**		**Falha**	
Não		Não		Não		Não		Não		Não	
[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]	
[**0**, **0**, 0, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]	
[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]	
[1, 0, 0, **0**, **0**]		[1, 0, 0, **0**, **0**]		[1, 0, 0, **0**, **0**]		[1, 0, **0**, **0**, **0**]		[1, 0, **0**, **0**, **0**]		[1, 0, **0**, **0**, **0**]	
[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, **0**, 0, 0]		[1, **1**, **0**, 0, 0]	

Fonte: Elaborado pelo autor (2013)

TABELA 25 – Conjunto de episódios ideais produzidos por *Agent** para os seis últimos episódios de *RS_Total*

História Ideais – Ep^K (K = 1, ..., 6)											
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	1.0	-2.0	2.0	-1.0	2.0	-1.0	2.0	-1.0	2.0	-1.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**		**Falha**	
Não		Não		Não		Não		Não		Não	
[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]		[**0**, 0, **0**, **0**, 0]	
[**0**, **0**, 0, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]		[**0**, **0**, **0**, **0**, **0**]	
[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]	

Fonte: Elaborado pelo autor (2013)

[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0]

[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[1, 0, 0, 0, 0] [1, 1, 0, 0, 0]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [0, 0, 0, 0, 0] [1, 0, 0, 0, 0] [1, 1, 0, 0, 0]

Fonte: Elaborado pelo autor (2013)

Considerando as propriedades internas do agente de observabilidade total do ambiente e a história do *Agent* em H(CasosTEST), o programa agente *RS_Total* não cometeu nenhuma falhas nas interações que manteve com *Amb*. Isto pode ser justificado devido a observabilidade total que o *Agent* possui do *Amb*. Mais especificamente, todos os episódios de *Agent* pertencem ao conjunto de episódios ideais produzidos por *Agent**. A moldura na TABELA 24 e TABELA 25 indica que os episódios produzidos por *RS_Total* pertencem ao conjunto de episódios ideais produzidos por *Agent**.

As TABELAS 26 a 27 apresentam os episódios 1-6 da história de *Agent* em *Amb*, inicializado com um dos casos de teste que obteve o melhor valor médio de utilidade nas simulações para o Experimento 4, ou seja, quando $-f_E = 44.0$ e $-f_L = -31,0$ e Falhas = 10. As tabelas apresentam também os episódios ideais produzidos por *Agent**, seus valores de avaliação de desempenho e as falhas correspondentes identificadas por *ProMon*. A TABELA 26 apresenta os seis primeiros episódios realizados pelo agente *RM_Parcial*, enquanto a TABELA 27 ilustra os episódios ideais gerados por *Agent**.

TABELA 26 – Seis primeiros episódios na história de *RM_Parcial*

História Realizada – Ep^1, Ep^2, Ep^3, Ep^4, Ep^5, Ep^6											
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
1.0	-2.0	1.0	-2.0	2.0	-1.0	2.0	-1.0	2.0	-1.0	2.0	-1.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**		**Falha**	
Não		Não		Não		Não		Não		Não	
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]		[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]		[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]		[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]		[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]		[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 27 – Conjunto de episódios ideais gerados por *Agent** para os seis primeiros episódios de *RM_Parcial*

História Realizada – Ep^1, Ep^2, Ep^3, Ep^4, Ep^5, Ep^6											
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
1.0	-2.0	1.0	-2.0	2.0	-1.0	2.0	-1.0	2.0	-1.0	2.0	-1.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**		**Falha**	
Não		Não		Não		Não		Não		Não	

[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]
[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]	[0, 0, 0, 0, 0] [0, 0, 0, 0, 1] [0, 0, 0, 1, 0] [0, 0, 1, 0, 0] [0, 1, 1, 0, 1]

Fonte: Elaborado pelo autor (2013)

O programa agente *RM_Parcial* não cometeu falhas nas seis primeiras interações que manteve com *Amb*. A moldura na primeira linha da TABELA 27 identifica que os seis primeiros episódios gerados por *RM_Parcial* pertencem ao conjunto de episódios ideais produzidos nas seis interações (K = 1, ..., 6) de *Agent**.

A TABELA 28 ilustra os episódios 17-22 da história de *Agent* em *Amb* em que *RM_Parcial* cometeu Falha 4 nos episódios Ep^{18}, Ep^{19} e Ep^{20}. A TABELA 29 ilustra o

episódio Ep((P^{18}, A^{18})) = EP((…L…, Esq)) de *RM_Parcial* em *Amb* e o episódio ideal Ep((P^{18}, A^{18}*)) = EP((…L…, Dir) produzido por *Agent**. A TABELA 30 ilustra o episódio Ep((P^{19}, A^{19})) = Ep((…L…, Ac)) de *RM_Parcial* e o episódio ideal Ep((P^{19}, A^{19}*)) = Ep((…L…, Dir)) produzido por *Agent**. A TABELA 31 apresenta o episódio Ep((P^{20}, A^{20})) = Ep((…L…, Ac)) e o conjunto de episódios ideais gerado por *Agent** na interação 20 para *RM_Parcial*, Ep((P^{20}, A^{20}*)) = Ep((…L…, Dir), (…L…, Ab)).

TABELA 28 – Seis últimos episódios na história de *RM_Parcial*

História Realizada – Ep^{17}, Ep^{18}, Ep^{19}, Ep^{20}, Ep^{21}, Ep^{22}											
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**	**-av$_E$**	**-av$_L$**	**-av$_E$**	**-av$_L$**	**-av$_E$**	**-av$_L$**	**-av$_E$**	**-av$_L$**
1.0	-2.0	2.0	-1.0	2.0	-1.0	1.0	-2.0	2.0	-1.0	2.0	-1.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**		**Falha**	
Não		4		4		4		Não		Não	
[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]	
[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]	
[0, 0, **0**, **0**, **0**]		[0, 0, **0**, **0**, **0**]		[0, 0, **0**, **0**, **0**]		[**0**, 0, **0**, **0**, **0**]		[***0***, **0**, **0**, **0**, **0**]		[**0**, ***0***, **0**, **0**, **0**]	
[0, 0, **0**, 0, 0]		[0, 0, **0**, 0, 0]		[**0**, 0, **0**, 0, 0]		[***0***, 0, **0**, 0, 0]		[**0**, 0, **0**, 0, 0]		[**0**, **0**, **0**, 0, 0]	
[0, ***0***, **0**, 0, 1]		[**0**, ***0***, **0**, 0, 1]		[***0***, **0**, **0**, 0, 1]		[**0**, **0**, **0**, 0, 1]		[**0**, **0**, **0**, 0, 1]		[**0**, **0**, **0**, 0, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 29 – Ep^{18} com falha na história de *RM_Parcial*

Histórias – Ep^{18}			
Realizada		**Ideal**	
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**
2.0	-1.0	2.0	-1.0
Falha			
4			
[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]	
[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]	
[0, 0, **0**, **0**, **0**]		[0, 0, **0**, **0**, **0**]	
[0, 0, **0**, 0, 0]		[0, 0, **0**, 0, 0]	
[**0**, ***0***, **0**, 0, 1]		[0, ***0***, **0**, 0, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 30 – Ep^{19} com falha na história de *RM_Parcial*

Histórias – Ep^{19}			
Realizada		**Ideal**	
-av$_E$	**-av$_L$**	**-av$_E$**	**-av$_L$**
2.0	-1.0	2.0	-1.0
Falha			
4			
[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]	
[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]	
[0, 0, **0**, **0**, **0**]		[0, 0, **0**, **0**, **0**]	
[**0**, 0, **0**, 0, 0]		[0, 0, **0**, 0, 0]	
[***0***, **0**, **0**, 0, 1]		[***0***, **0**, **0**, 0, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 31 – Ep^{20} com falha na história de *RM_Parcial*

Histórias – Ep^{20}					
Realizada		**Ideal**			
-av_E	**-av_L**	**-av_E**	**-av_L**	**-av_E**	**-av_L**
2.0	-1.0	2.0	-1.0	2.0	-1.0
Falha					
4					
[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]		[**0**, **0**, 0, 0, 0]	
[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]		[0, **0**, **0**, **0**, **0**]	
[**0**, 0, **0**, **0**, **0**]		[0, 0, **0**, **0**, **0**]		[0, 0, **0**, **0**, **0**]	
[**0**, 0, **0**, 0, 0]		[**0**, **0**, **0**, 0, 0]		[**0**, 0, **0**, 0, 0]	
[**0**, **0**, **0**, 0, 1]		[**0**, **0**, **0**, 0, 1]		[**0**, **0**, **0**, 0, 1]	

Fonte: Elaborado pelo autor (2013)

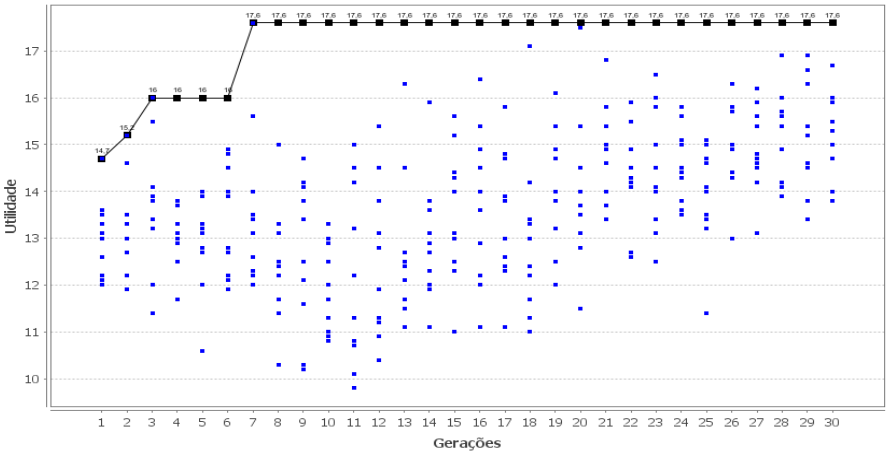
O agente *RM_Parcial* cometeu uma Falha 4 no episódio realizado na TABELA 29, ou seja, existe uma ação melhor que 'Esq' no Ep^{18}, isto é, 'Dir' que leva o agente para uma sala vizinha suja. Neste caso, a Falha 4 indica que quem está causando a falha é a função ver do agente, visto que a função ação de *RM_Parcial* poderia ser capaz de escolher a ação 'Dir' se soubesse que a sala à sua direita estava suja. Neste caso, apesar dos valores de avaliação de *RM_Parcial* e *Agent** serem iguais, o agente evita ganhar pontos ao deixar de se movimentar para uma sala suja. A TABELA 30 e TABELA 31 apresentam os dois próximos episódios produzidos por *RM_Parcial* e os episódios ideais correspondentes gerados por *Agent**. Percebe-se nessas tabelas que o episódio de *RM_Parcial* é diferente do episódio ideal produzido por *Agent**, ou seja: $Ep((P^{19}, A^{19})) = Ep((\ldots L\ldots, Ac) \neq Ep((P^{19}, A^{19}*)) = Ep((\ldots L\ldots, Dir))$. Semelhantemente ao que aconteceu com a interação 20, o agente deixou de se movimentar para uma sala vizinha suja, assim, o projetista identificou a Falha 4.

Assim, conforme esperado, o agente aspirador de pó com arquitetura / característica agente reativo simples com observabilidade parcial apresenta o pior desempenho na avaliação, visto que ao realizar uma análise breve nas regras condição-ação do agente se confirmará que, por limitação da arquitetura, as mesmas não consideram nas condições em seus antecedentes aspectos relacionados aos critérios de energia e de limpeza. Visto que o aspirador foi concebido como um programa reativo simples, pouco pode ser feito para melhorar seu desempenho, sendo necessário realizar uma extensão em sua estrutura visando ampliar a observabilidade do ambiente, permitindo executar ações melhores, ou acomodar um estado interno, o que permitirá ao agente economizar energia ao perceber em seu histórico de percepções que uma determinada sala já foi visitada.

5.2.3 Experimentos 2 – Resultados *Thestes*

A FIGURA 31 ilustra os resultados do Experimento 2 obtidos por *Thestes* considerando a função *Utilidade* no formato linear, conforme descrito no Capítulo 4, Seção 4.6, com valores de pesos $w_L = w_E = 0,5$ para os objetivos energia e limpeza ao longo de 30 gerações.

FIGURA 31 – Valores de utilidade de *CasosTEST* em 30 gerações (Experimento 2)



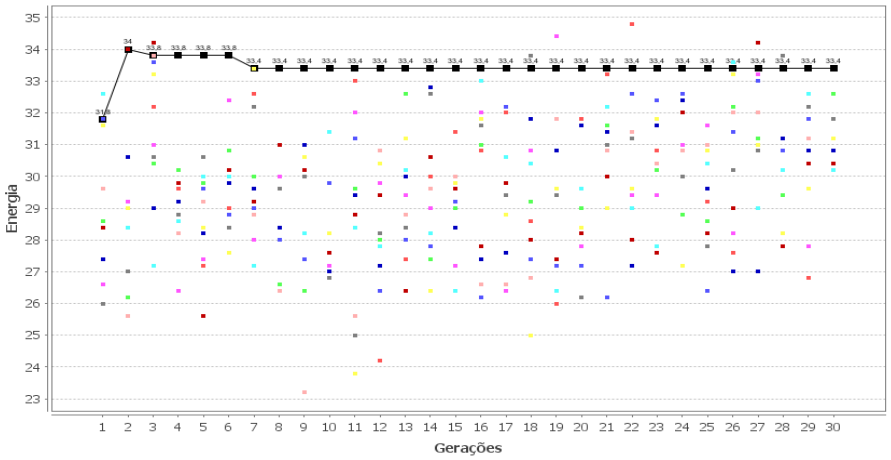
Fonte: Elaborado pelo autor (2013)

No Experimento 2, o melhor valor da função Utilidade é obtido na sétima geração com Utilidade = 17,6. Nas próximas gerações, esse valor é tomado apenas como referência e não é selecionado nenhum caso de teste com valor de utilidade superior. Conforme esperando, considerando os melhores casos de teste, *Thestes* detectou que o programa agente *RS_Parcial_alterado* é mais inadequado que o agente *RS_Parcial*. Isso pode ser justificado devido as regras condição-ação do agente *RS_Parcial_alterado* serem definidas aleatoriamente, possibilitando que o subsistema de tomada de decisão (ação) seja capaz de selecionar ações que podem produzir episódios com falhas, tornando o valor de *Utilidade* mais inadequado. Dificilmente o projetista conseguirá conceber um subsistema de tomada de decisão (ação) capaz de selecionar as ações que sejam realmente racionais em cada interação com o ambiente quando as regras condição-ação são definidas aleatoriamente.

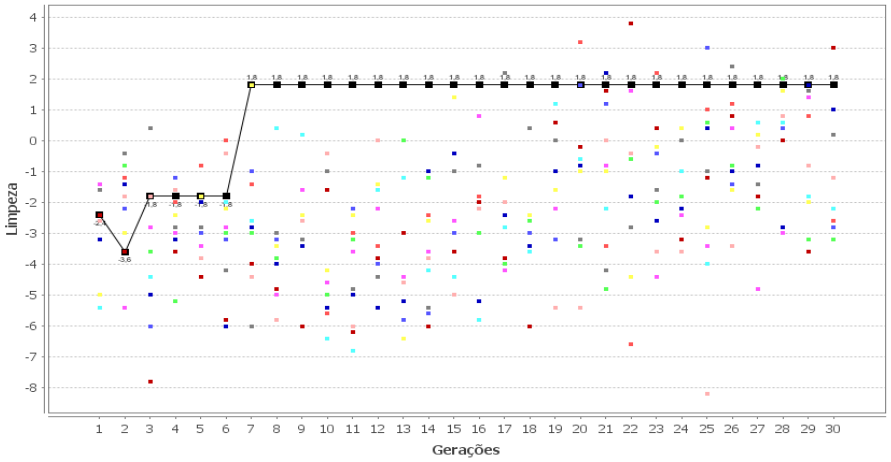
No decorrer das gerações, *Thestes* buscou selecionar um conjunto de *CasosTEST* com valores inadequados de energia e de limpeza. Enquanto o projetista tem como objetivo projetar agentes capazes de limpar o maior número de salas com o menor consumo de energia possível, *Thestes* objetiva identificar os cenários em que *Agent* tem o comportamento mais inadequado em termos de energia e limpeza. A FIGURA 32 ilustra os casos de teste em cada

geração e os valores de inadequação de energia (em (1)) e os valores de inadequação de limpeza (em (2)) para o Experimento 2.

FIGURA 32 – Valores de inadequação de energia (1) e limpeza (2) (Experimento 2)



(a1)



(a2)

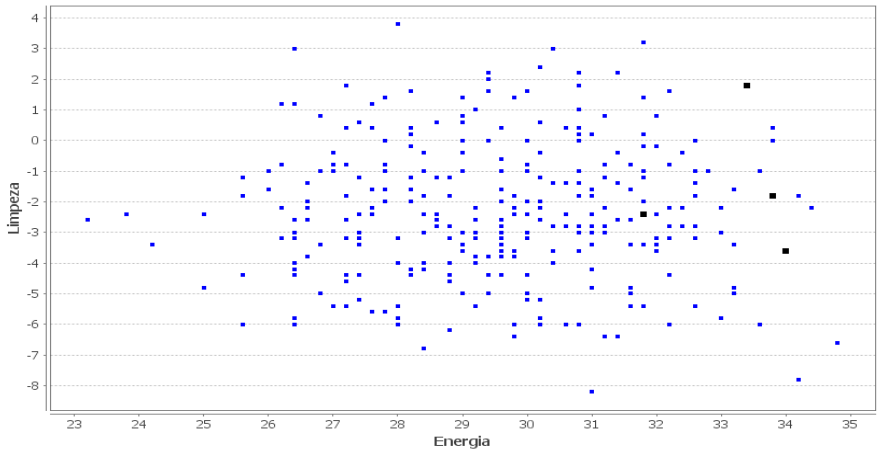
Fonte: Elaborado pelo autor (2013)

Na avaliação de desempenho do *Agent*, os objetivos energia e limpeza possuem o mesmo valor de importância ($w_L = w_E = 0,5$), obtendo para o caso de teste mais inadequado em *CasosTEST* os seguintes valores para as funções escalares $-f_E = 33,4$ e $-f_L = 1,8$. A medida de avaliação de desempenho do *Agent* foi elaborada considerando os valores apresentados na TABELA 5. De acordo com o valor de inadequação de energia e limpeza, durante as gerações, *Thestes* privilegiou e selecionou os casos de teste em que *Agent* apresenta o comportamento mais inadequado relacionado ao maior valor de energia e menor valor de limpeza, ou seja, gastou mais energia ao final da tarefa e limpou menos o ambiente.

Considerando os atributos de energia e limpeza também avaliados no decorrer dos episódios, a FIGURA 33 apresenta os valores associados as funções de inadequação de

energia e limpeza associados aos 10 casos de teste em *CasosTEST* nas 30 gerações para o Experimento 2 .

FIGURA 33 – Valores de inadequação de Energia e Limpeza (Experimento 2)



Fonte: Elaborado pelo autor (2013)

A TABELA 32 destaca os valores de energia e de limpeza alcançados pelo *RS_Parcial_alterado* nos melhores casos de teste nos três experimentos.

TABELA 32 – Energia e limpeza nos casos selecionados de *RS_Parcial_alterado*

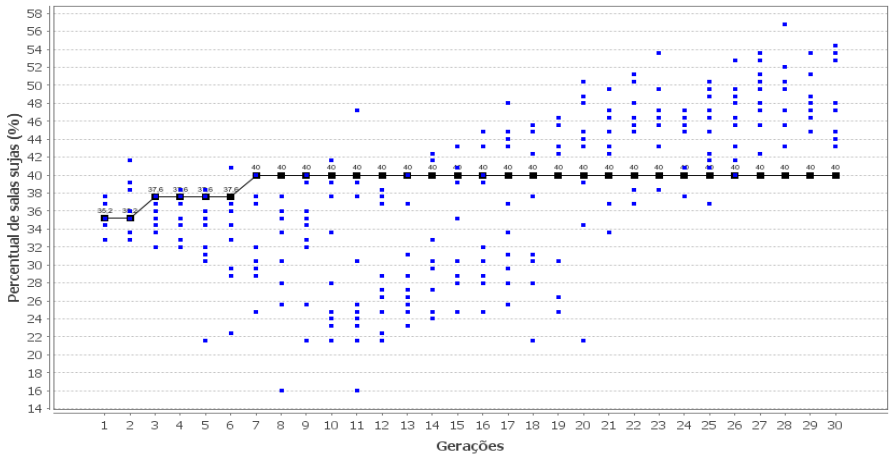
Exp	Agent-Amb	Energia	Limpeza
2	RS_Parcial_alterado	31,8	-2,4
		34,0	-3,6
		33,8	-1,8
		33,4	1,8

Fonte: Elaborado pelo autor (2013)

Como *RS_Parcial_alterado* é um agente reativo simples com observabilidade parcial possui visão limitada do ambiente e regras condição-ação aleatórias, as ações que ele seleciona nem sempre são racionais. Mais especificamente, como as ações são escolhidas aleatoriamente, nem sempre as ações escolhidas levam o *Agent* a uma sala suja mais próxima, fazendo com que o agente gaste mais energia e pontue menos em limpeza.

A FIGURA 34 destaca o percentual de salas sujas resultantes após 25 interações entre *Agent* e *Amb* no Experimento 2.

FIGURA 34 – Percentual de salas sujas nos *CasosTEST* em 30 gerações (Experimento 2)



Fonte: Elaborado pelo autor (2013)

Levando em consideração que o principal objetivo do agente aspirador de pó é aspirar o ambiente, os casos de teste em que *Agent* possui uma maior quantidade de salas sujas ao final da simulação com o *Amb* representa, além dos demais critérios avaliados, que o agente não conseguiu obter um bom desempenho. Na maioria dos casos, *Thestes* selecionou os casos de teste em *CasosTEST* cujas histórias do *Agent* apresenta um maior percentual de salas sujas ao final dos testes.

A TABELA 33 apresenta o percentual de salas sujas deixadas ao final das interações entre *Agent* e *Amb* para os casos de teste com maior valor de utilidade para o Experimento 2.

TABELA 33 – Percentual de salas sujas nos casos selecionados do *RS_Parcial_alterado*

Exp	Agent-Amb	Percentual de salas sujas
2	RS_Parcial_alterado	36,2
		36,2
		37,6
		40,0

Fonte: Elaborado pelo autor (2013)

O caso de teste selecionado em *CasosTEST* apresentou um percentual de 40,0% de salas sujas ao final das interações entre *Agent* e *Amb*, ou seja, devido as ações aleatórias, o *Agent* movimenta-se e retorna para as mesmas salas, independentemente se as salas estão sujas ou limpas, levando o *Agent* a perder pontos em energia (devido o consumo nos movimentos) e em limpeza.

5.2.4 Experimentos 2 – Funcionalidades *ProMon*

As TABELAS 34 a 38 a ilustram os episódios 1-7 da história de *Agent* em *Amb*, inicializado com um dos casos de teste que obteve o melhor valor médio de utilidade nas simulações para o Experimento 2, ou seja, quando $-f_E$ = 27,0 e $-f_L$ = -5,0 e Falhas = 21. As tabelas destacam também os episódios ideais associados produzidos por *Agent**, seus valores de avaliação e as falhas correspondentes identificadas por *ProMon*. A TABELA 34 apresenta os cinco primeiros episódios realizados pelo agente *RS_Parcial_alterado*. A TABELA 35 ilustra o segundo episódio $Ep((P^2, A^2)) = Ep(...S..., N\text{-}op)$ de *RS_Parcial_alterado* em *Amb* e o episódio ideal $Ep((P^2, A^2*)) = Ep((...S..., Asp)$ produzido por *Agent**. A o episódio ideal para Ep^6 correspondentes gerados por *Agent**. A TABELA 36 ilustra o quarto episódio $Ep((P^4, A^4)) = Ep((...L..., N\text{-}op))$ e o conjunto de episódios ideais $Ep_{ideais}{}^4$ gerados na interação 4 de *Agent** com *Amb*, $Ep((P^4, A^4*)) = Ep((...L..., Dir), (...L..., Ab))$.

TABELA 34 – Quatro primeiros episódios na história de *RS_Parcial_alterado*

História Realizada – Ep^1, Ep^2, Ep^3, Ep^4, Ep^5									
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	0.0	-1.0	1.0	-2.0	1.0	0.0	1.0	-2.0
Falha		**Falha**		**Falha**		**Falha**		**Falha**	
Não		8		Não		6		Não	
[0, 1, 1, 0, 0]		[0, 1, 1, 0, 0]		[0, 0, 1, 0, 0]		[0, 0, 1, 0, 0]		[0, 0, 1, 0, 0]	
[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]	
[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 35 – Primeiro episódio com falha na história de *RS_Parcial_alterado*

Histórias – Ep^2			
Realizada		**Ideal**	
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**
0.0	-1.0	1.0	-2.0
Falha			
8			
[0, 1, 1, 0, 0]		[0, 0, 1, 0, 0]	
[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]	
[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 36 – Segundo episódio com falha na história de *RS_Parcial_alterado*

Histórias – Ep^4					
Realizada		**Ideal**			
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**

1.0	0.0	2.0	-1.0	2.0	-1.0
Falha					
6					
[**0**, **0**, 1, 0, 0]		[**0**, **0**, **1**, 0, 0]		[**0**, **0**, 1, 0, 0]	
[1, 1, 0, 0, 0]		[1, 1, 0, 0, 0]		[1, **1**, 0, 0, 0]	
[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]	

Fonte: Elaborado pelo autor (2013)

Devido as regras condição-ação do *RS_Parcial_alterado* um número significado de falhas é identificado no decorrer da história do *Agent*. Conforme mostrado pela TABELA 34, o primeiro, o terceiro e o quinto episódio não contém falhas, ou seja, esses episódios pertencem aos conjuntos de episódios ideais Ep_{ideais}^{K} gerados nas interações (K = 1, 3 e 5) de *Agent** com *Amb*. A TABELA 35 ilustra o episódio 2 em que *RS_Parcial_alterado* cometeu uma Falha 8, ou seja, existe uma ação melhor que 'N-op' no segundo episódio, isto é, 'Asp' que leva o agente a aspirar a sala suja. Neste caso, a Falha 8 indica que quem está causando a falha é a função ação do agente, visto que o *Agent* poderia ser capaz de escolher a ação 'Asp'. A TABELA 36 ilustra o episódio 4 em que *RS_Parcial_alterado* cometeu uma Falha 6, ou seja, existe ações melhor que 'N-op' no quarto episódio, isto é, 'Dir' ou 'Ab' que leva o agente para uma sala vizinha suja. Neste caso, a Falha 4 também é causada pela função ação, visto que a função ação do agente poderia ser capaz de escolher as ações 'Dir' ou 'Ab'.

As TABELAS 37 e 38 apresentam os dois próximos episódios produzidos por *RS_Parcial_alterado* com os episódios ideais correspondentes produzidos por *Agent**.

TABELA 37 – Ep^6 com falha na história de *RS_Parcial_alterado*

Histórias – Ep^6			
Realizada		**Ideal**	
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**
0.0	-1.0	1.0	-2.0
Falha			
8			
[**0**, **0**, 1, 0, 0]		[**0**, **0**, 1, 0, 0]	
[1, **1**, 0, 0, 0]		[1, **0**, 0, 0, 0]	
[1, 0, 0, 1, 0]		[1, 0, 0, 1, 0]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 1, 0, 0, 1]		[1, 1, 0, 0, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 38 – Ep^7 com falha na história de *RS_Parcial_alterado*

Histórias – Ep^7			
Realizada		**Ideal**	
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	1.0	2.0	-1.0

Falha	
7	
[0, 0, 1, 0, 0] [1, 1, 0, 0, 0] [1, 0, 0, 1, 0] [0, 0, 0, 1, 1] [1, 1, 0, 0, 1]	[0, 0, 1, 0, 0] [1, 0, 0, 0, 0] [1, 0, 0, 1, 0] [0, 0, 0, 1, 1] [1, 1, 0, 0, 1]

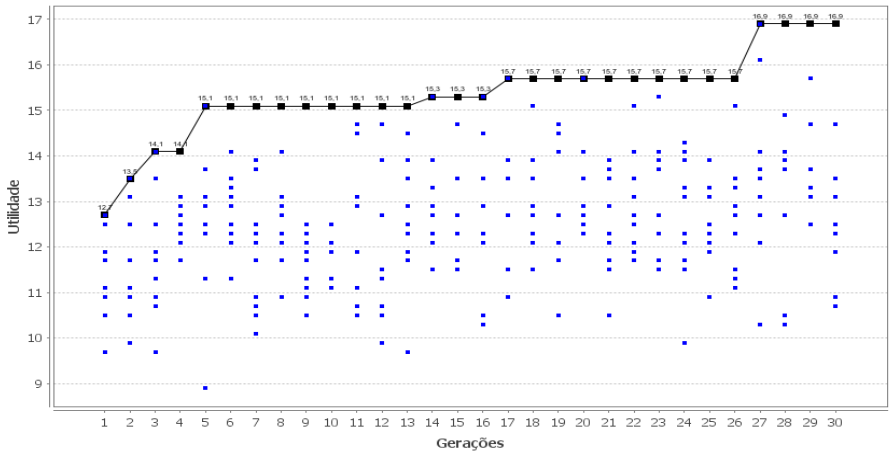
Fonte: Elaborado pelo autor (2013)

Percebe-se nas tabelas que os episódios 6 e 7 de *RS_Parcial_alterado* são diferentes dos episódios ideais produzidos por *Agent**, ou seja: $Ep((P^6, A^6)) = Ep((...S..., N\text{-}op) \neq Ep((P^6, A^6*)) = Ep((...S..., Asp))$. Semelhantemente ao que aconteceu no sétimo episódio, $Ep((P^7, A^7)) = Ep((...S..., Ac)) \neq Ep((P^7, A^7*)) = Ep((...S..., Asp)$, o agente deixou de aspirar uma sala suja. No Ep^6, cuja P^6 = ...S..., o Agent escolheu A^6 = N-op, logo o projetista identificou a Falha 8, pois o agente não opera em uma sala suja. No Ep^7, cuja P^7 = ...S..., o *Agent* escolheu A^7 = Ac, logo o projetista identificou a Falha 7, pois o agente escolheu movimentar-se a aspirar a sala. Vale ressaltar, este tipo de falha no mundo do programa agente aspirador de pó *RS_Parcial_alterado*, pode-se considerar que a falha na função ação do agente pode ser controlada com a retirada da aleatoriedade das regras condição-ação.

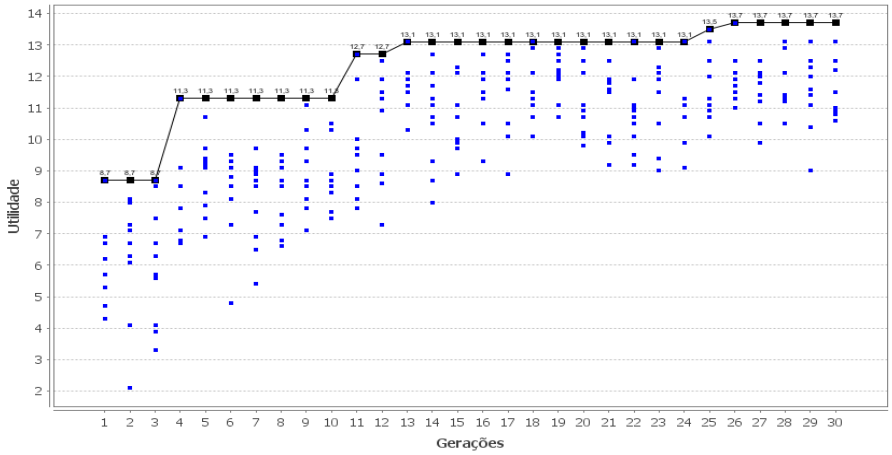
5.2.5 Experimentos 5 e 6 – Resultados *Thestes*

Analisando os experimentos em que os componentes de *Thestes* foram concebidos considerando o NSGA-II, a FIGURA 35 apresenta os resultados da medida de avaliação de desempenho, mais especificamente o valor da função utilidade, cujos valores de pesos dos dois objetivos (energia e limpeza) são iguais. Em (a) é ilustrado o Experimento 6 e em (b) o Experimento 7.

FIGURA 35 – Valores de utilidade de *CasosTEST* em 30 gerações (Experimentos 6 e 7)



(a)



(b)

Fonte: Elaborado pelo autor (2013)

A TABELA 39 destaca a geração e o melhor valor utilidade encontrado para os agentes dos Experimentos 5 e 6 que utilizam a estratégia de busca local NSGA-II.

TABELA 39 – Geração e Utilidade do melhor caso em *CasosTEST* (Experimentos 5 e 6)

Exp	Agent-Amb	Geração	Utilidade
5	RS_Parcial_NSGA_II	27	16,9
6	RM_Parcial_NSGA_II	25	13,7

Fonte: Elaborado pelo autor (2013)

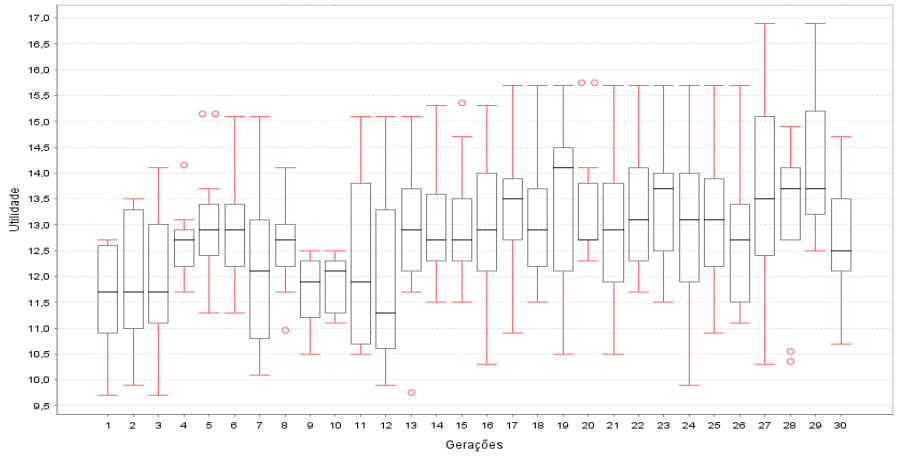
Considerando a medida de avaliação de desempenho obtida pelo agente reativo simples com observabilidade parcial (*RS_Parcial_NSGA_II*) com a seleção de casos de teste realizada por meio da estratégia de busca NSGA-II (FIGURA 35 em (a)), existe uma progressão de identificação de melhores casos de teste à medida que as gerações procedem e o melhor caso de teste é obtido na vigésima sétima geração, com U = 16,9. Comparando com o Experimento 1 (*RS_Parcial*), experimento que representa a mesma estrutura do agente e diferente no *ModeloTransição* pela utilização do algoritmo genético na seleção de novos casos de teste, é possível constatar que o valor identificado é inferior, U = 15,5. Considerando que o obtivo de *Thestes* é identificar situações em que *Agent* possui o pior valor de avaliação, apesar de valores próximos, a seleção de casos de testes por meio da estratégia NSGA-II atingiu o melhor valor neste caso.

Avaliando o agente reativo baseado em modelo com utilização do NSGA-II (*RM_Parcial_NSGA_II*) para seleção de casos de teste (FIGURA 35 em (b)), o melhor caso de teste foi identificado na vigésima quinta geração, com U = 13,7. Ao comparar com o Experimento 4 (*RM_Parcial*), U = 11,7, é possível afirmar que a estratégia NSGA-II

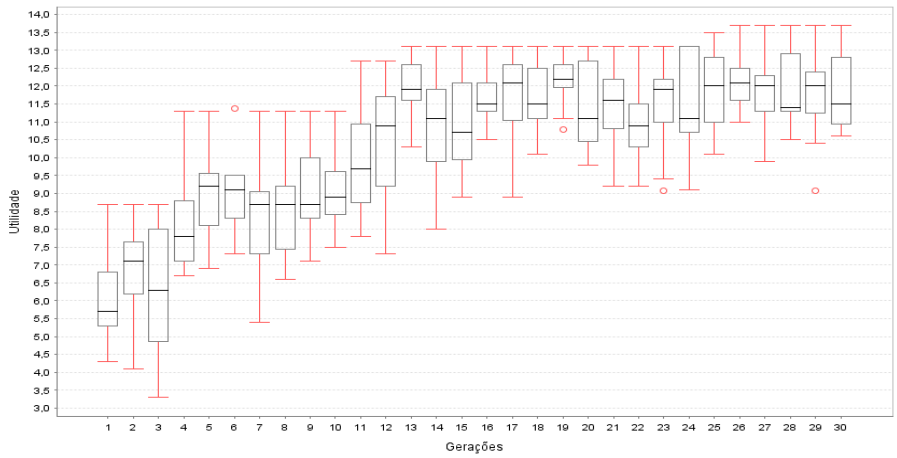
identificou situações de casos de teste em que o *Agent* possui um maior valor da função Utilidade, ou seja, foi pior avaliado.

A FIGURA 36 ilustra a variação do valor de utilidade nos 10 casos em *CasosTEST* em cada geração. O box-plot é formato pelos quartis inferior (25%), mediana (50%) e superior (75%), assim como os valores máximo e mínimo entre os casos. Em (a) é apresentado o box-plot do Experimento 5 e em (b) do Experimento 6.

FIGURA 36 – Box-plot dos valores de utilidade dos Experimentos 5 e 6 em 30 gerações



(a)



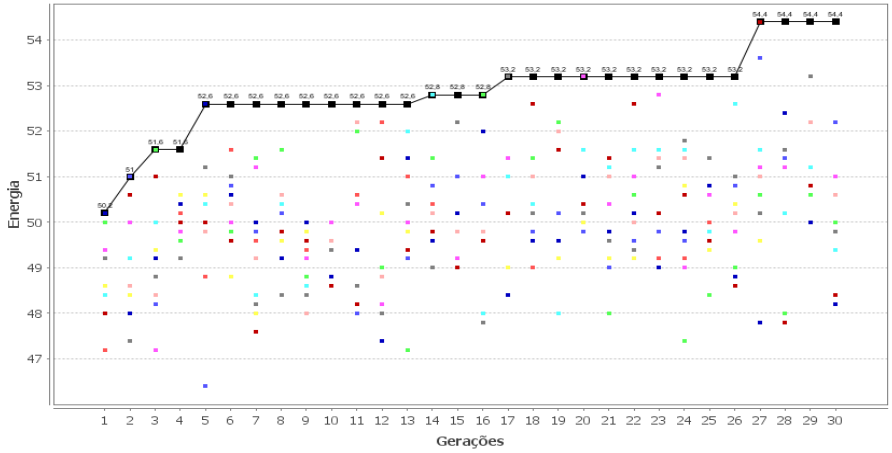
(b)

Fonte: Elaborado pelo autor (2013)

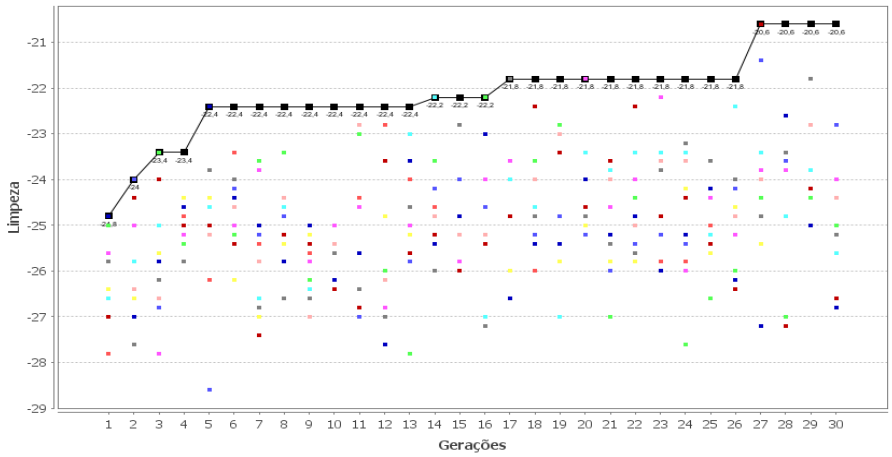
Em ambos os experimentos, *Thestes* buscou selecionar um conjunto *CasosTEST* satisfatório. Em cada geração *Thestes* obteve em *CasosTEST* o caso de teste em que o agente teve o comportamento mais inadequado, ou seja, o valor de utilidade é igual ao valor máximo entre os 10 casos de teste. Os casos de teste cujo valor é inferior ao valor máximo, não são importantes, visto que quanto mais próximo do valor mínimo, mais adequado é o comportamento do agente e melhor é o desempenho do agente no caso de teste.

Considerando os atributos energia e limpeza, a FIGURA 37 ilustra os valores das funções de inadequação de energia e limpeza individuais associado a *CasosTEST* nas 30 gerações para o Experimento 5 em (a) e Experimento 6 em (b). O atributo energia é identificado em (1), enquanto limpeza é identificado em (2).

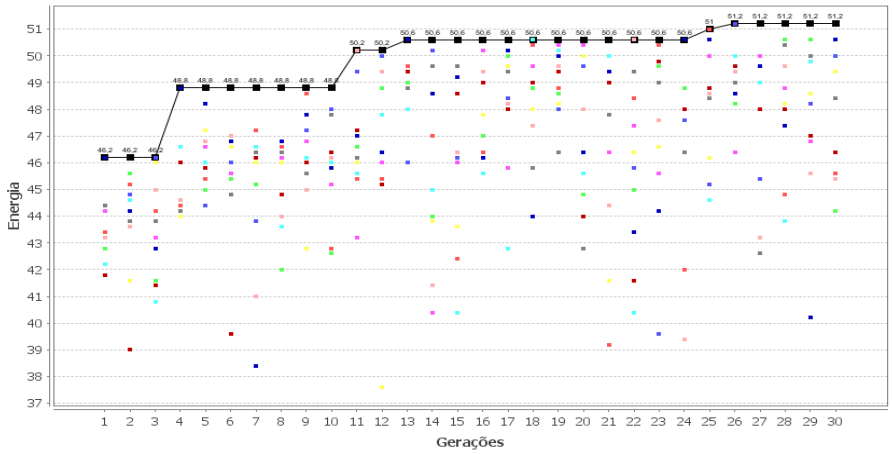
FIGURA 37 – Valores de inadequação de energia (1) e limpeza (2) (Experimentos 5 e 6)



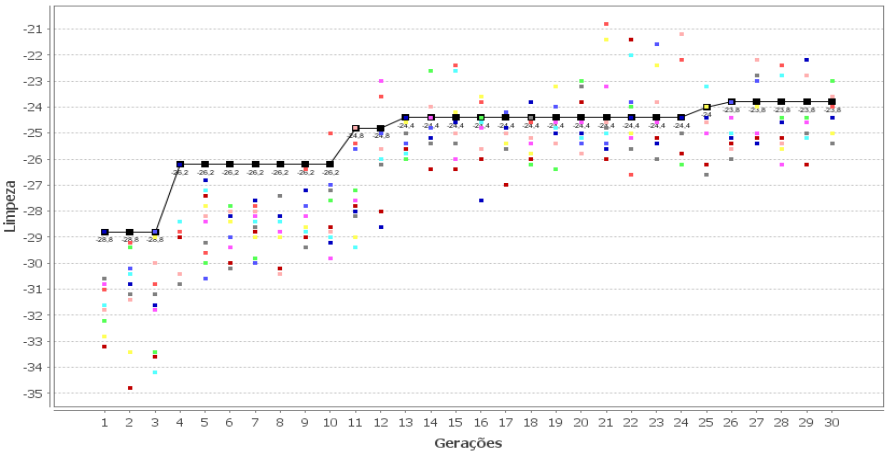
(a1)



(a2)



(b1)



(b2)

Fonte: Elaborado pelo autor (2013)

A TABELA 40 destaca os valores de energia e limpeza alcançados pelos agentes nos melhores casos de teste, ou seja, os casos de testes selecionados em cada geração, em cada experimento.

TABELA 40 – Energia e limpeza nos casos selecionados dos Experimentos 5 e 6

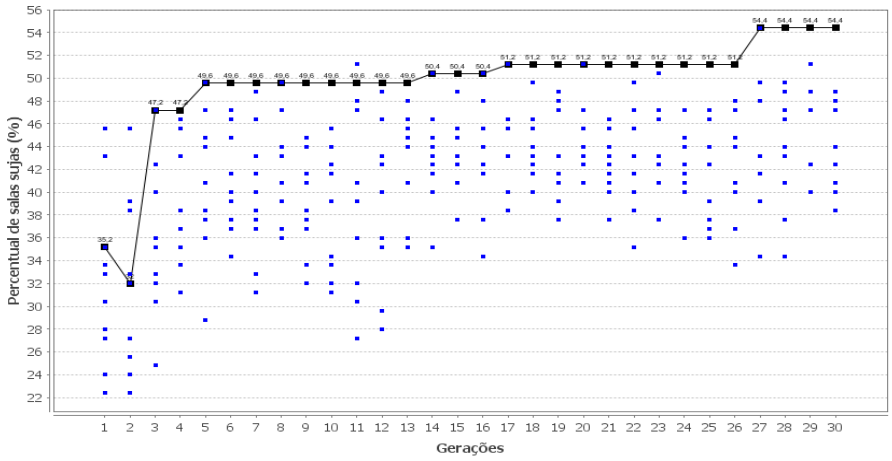
Exp	Agent-Amb	Energia	Limpeza
5	RS_Parcial_NSGA_II	50,2	-24,8
		51,0	-24,0
		51,6	-23,4
		52,6	-22,4
		52,8	-22,2
		53,2	-21,8
		54,4	-20,6
6	RM_Parcial_NSGA_II	46,2	-28,8
		48,8	-26,2
		50,2	-24,8
		50,6	-24,4
		51,0	-24,0
		51,2	-23,8

Fonte: Elaborado pelo autor (2013)

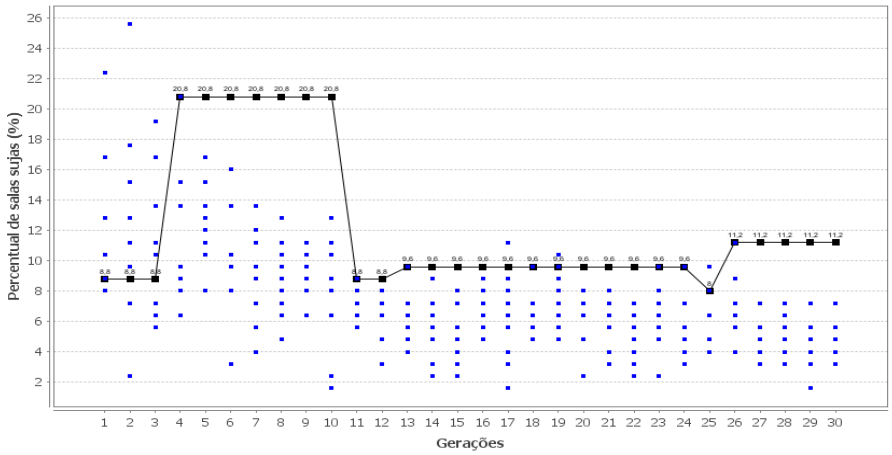
No geral, *Thestes* privilegiou os casos de teste com um valor mais inadequado nos termos de energia e em termos de limpeza, quase maximizando este nível de inadequação em ambos os experimentos conforme apresentado em todas as gerações. Comparando os resultados entre os agentes, no Experimento 5, os valores de $-f_E$ = 54,4 e $-f_L$ = -20,6. No Experimento 6, os valores de $-f_E$ = 51,2 e $-f_L$ = -23,8. Isto é, *Thestes* identifica um valor mais inadequado para os atributos energia e limpeza para o agente reativo simples com observabilidade parcial.

A FIGURA 38 mostra o percentual de salas sujas deixadas pelo *Agent* ao finalizar as interações entre *Agent* e *Amb* para o Experimento 5 em (a) e do Experimento 6 em (b).

FIGURA 38 – Percentual de salas sujas nos *CasosTEST*(Experimentos 6 e 7)



(a)



(b)

Fonte: Elaborado pelo autor (2013)

Considerando o objetivo principal do agente aspirador de pó que é manter o ambiente limpo, o percentual de salas sujas restantes ao final dos testes é um indicativo do mal desempenho de *Agent* ao buscar atingir seus objetivos. No geral, o agente reativo simples com observabilidade parcial resulta em um maior número de salas sujas se comparado ao agente reativo baseado em modelo.

A TABELA 41 ilustra o percentual de salas sujas deixadas pelo *Agent* em *Amb* nos casos de teste selecionados nas gerações para os Experimentos 5 e 6.

TABELA 41 – Percentual de salas sujas nos casos selecionados dos Experimentos 5 e 6

Exp	Agent-Amb	Percentual de salas sujas
5	RS_Parcial_NSGA_II	35,2
		32,0

		47,2
		49,6
		50,4
		51,2
		54,4
6	RM_Parcial_NSGA_II	8,8
		20,8
		8,8
		9,6
		8,0
		11,2

Fonte: Elaborado pelo autor (2013)

Considerando o caso de teste com maior percentual de salas sujas, conforme a TABELA 41, é possível observar que o agente reativo simples com observabilidade parcial deixou o maior percentual de salas sujas em *Amb*, 54,4%, enquanto o agente reativo baseado em modelo possui o percentual de 11,2%. Com isso pode-se identificar que *RS_Parcial_NSGA_II* possui o comportamento mais inadequado em relação a *RM_Parcial_NSGA_II*.

5.2.6 Experimentos 5 e 6 – Funcionalidades *ProMon*

As TABELAS 42 a 45 ilustram os episódios 1-4 da história de *Agent* em *Amb*, inicializado com um dos casos de teste que obteve o melhor valor médio de utilidade no Experimento 5, ou seja, quando $-f_E = 54,4$, $-f_L = -20,6$ e Falhas = 13. As tabelas também apresentam os episódios ideais gerados por *Agent**, seus valores de avaliação e as falhas correspondentes identificadas pelo agente *ProMon*. A TABELA 42 apresenta os quatro primeiros episódios realizados pelo agente *RS_Parcial_NSGA_II*.

TABELA 42 – Quatro primeiros episódios na história de *RS_Parcial_NSGA_II*

História Realizada – Ep^1, Ep^2, Ep^3, Ep^4							
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	2.0	-1.0	2.0	-1.0	1.0	-2.0
Falha		**Falha**		**Falha**		**Falha**	
4		4		Não		4	
[*0*, 0, 1, 1, 1]		[*0*, 0, 1, 1, 1]		[*0*, *0*, 1, 1, 1]		[*0*, *0*, 1, 1, 1]	
[*0*, 0, 1, 1, 1]		[*0*, 0, 1, 1, 1]		[*0*, 0, 1, 1, 1]		[*0*, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

O programa agente *RS_Parcial_NSGA_II* cometeu três falhas nos quatro primeiros episódios. A TABELA 43 ilustra o primeiro episódio $Ep((P^1, A^1)) = Ep((...L...,$

Ab)) de *RS_Parcial_NSGA_II* em *Amb* e o episódio ideal Ep((P^1, A^1*)) = Ep((...L..., Dir)) produzido por *Agent**. A TABELA 44 apresenta o segundo episódio Ep((P^2, A^2)) = Ep((...L..., Ac)) e o episódio ideal gerado por *Agent** Ep((P^2, A^2*)) = Ep((...L..., Dir)). A TABELA 45 apresenta o episódio quatro Ep((P^4, A^4)) = Ep((...L..., Esq)) e o episódio ideal correspondente gerado por *Agent** Ep((P^4, A^4*)) = Ep((...L..., Dir)).

TABELA 43 – Primeiro episódio com falha na história de *RS_Parcial_NSGA_II*

Histórias – Ep^1			
Realizada		**Ideal**	
-av_E	**-av_L**	**-av_E**	**-av_L**
2.0	-1.0	2.0	-1.0
Falha			
4			
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 44 – Segundo episódio com falha na história de *RS_Parcial_NSGA_II*

Histórias – Ep^2			
Realizada		**Ideal**	
-av_E	**-av_L**	**-av_E**	**-av_L**
2.0	-1.0	2.0	-1.0
Falha			
4			
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 45 – Terceiro episódio com falha na história de *RS_Parcial_NSGA_II*

Histórias – Ep^4			
Realizada		**Ideal**	
-av_E	**-av_L**	**-av_E**	**-av_L**
2.0	-1.0	2.0	-1.0
Falha			
4			
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 1, 1, 1]		[0, 0, 1, 1, 1]	
[0, 0, 0, 1, 1]		[0, 0, 0, 1, 1]	
[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]	
[0, 1, 1, 1, 1]		[0, 1, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

O agente *RS_Parcial_NSGA_II* cometeu uma Falha 4 nas interações 1, 2 e 4, ou seja, para cada uma dessas interações existe uma ação melhor que leva o agente para uma sala vizinha suja. Percebe-se nessas tabelas que o episódio de *Agent* é diferente do episódio ideal produzido por *Agent**, ou seja: $Ep((P^1, A^1)) = Ep((...L..., Ab)) \neq Ep((P^1, A^1*)) = Ep((...L..., Dir))$, $Ep((P^2, A^2)) = Ep((...L..., Ac)) \neq Ep((P^2, A^2*)) = Ep((...L..., Dir))$ e $Ep((P^4, A^4)) = Ep((...L..., Esq)) \neq Ep((P^4, A^4*)) = Ep((...L..., Dir))$. Neste caso, a Falha 4 indica que quem está causando a falha é a função ver do agente, visto que a função ação de *RS_Parcial_NSGA_II* poderia ser capaz de escolher a ação ideal, conforme a ação produzida por *Agent**, se soubesse que a sala à sua direita estava suja. Vale ressaltar que, apesar dos valores de avaliação do *Agent* e do *Agent** serem iguais, o agente evita ganhar pontos ao deixar ao deixar de se movimentar para uma sala suja.

As TABELAS 46 a 48 apresentam os quatro episódios iniciais da história de *Agent* em *Amb* no Experimento 6 inicializado com um dos casos de teste que obteve o melhor valor médio de utilidade, ou seja, quando $-f_E = 51{,}2$, $-f_L = -23{,}8$ e Falhas = 10. As tabelas apresentam também os episódios ideais associados gerados por *Agent** e seus valores de avaliação de desempenho.

TABELA 46 – Quatro primeiros episódios na história de *RM_Parcial_NSGA_II*

História Realizada – Ep^1, Ep^2, Ep^3, Ep^4							
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	2.0	-1.0	2.0	-1.0	1.0	-2.0
Falha		**Falha**		**Falha**		**Falha**	
4		Não		4		Não	
[***0***, **0**, 0, 0, 0]		[**0**, ***0***, **0**, 0, 0]		[**0**, **0**, ***0***, 0, 0]		[**0**, ***0***, **0**, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]	
[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 47 – Ep^1 com falha na história de *RM_Parcial_NSGA_II*

Histórias – Ep^1			
Realizada		**Ideal**	
$-av_E$	**$-av_L$**	**$-av_E$**	**$-av_L$**
2.0	-1.0	2.0	-1.0
Falha			
4			
[***0***, **0**, 0, 0, 0]		[***0***, 0, 0, 0, 0]	
[0, 0, 0, 0, 0]		[**0**, 0, 0, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]	
[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

TABELA 48 – Ep^3 com falha na história de *RM_Parcial_NSGA_II*

Histórias – Ep^3			
Realizada		**Ideal**	
-av_E	**-av_L**	**-av_E**	**-av_L**
2.0	-1.0	2.0	-1.0
Falha			
4			
[**0**, **0**, ***0***, 0, 0]		[**0**, **0**, ***0***, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, **0**, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]	
[0, 0, 0, 0, 0]		[0, 0, 0, 0, 0]	
[1, 0, 1, 1, 1]		[1, 0, 1, 1, 1]	

Fonte: Elaborado pelo autor (2013)

A TABELA 47 ilustra o episódio $Ep((P^1, A^1)) = Ep((...L..., Dir))$ de *RM_Parcial_NSGA_II* e o episódio ideal $Ep((P^1, A^1*)) = Ep((...L..., Ab))$ produzido por *Agent**. O *Agent* cometeu uma Falha 4, ou seja, existe uma ação melhor que 'Dir' no Ep^1, isto é, 'Ab' leva o agente para uma sala vizinha suja. A TABELA 48 ilustra o episódio $Ep((P^3, A^3)) = Ep((...L..., Esq))$ de *RM_Parcial_NSGA_II* e o episódio ideal $Ep((P^3, A^3*)) = Ep((...L..., Ab))$ produzido por *Agent**. Semelhantemente ao que aconteceu com a interação 1, o agente deixou de se movimentar para uma sala vizinha suja, assim, o projetista identificou a Falha 4.

6 CONCLUSÃO

Neste capítulo serão feitas as considerações finais deste trabalho, apresentando as principais contribuições e limitações da pesquisa realizada, bem como delineando oportunidades para trabalhos futuros.

6.1 Contribuições e Limitações da Pesquisa

Considerando que o agente racional é uma entidade autônoma, capaz de realizar seus objetivos de maneira independente, testes adequados devem ser desenvolvidos para avaliar o desempenho do agente na execução das ações e dos planos necessários para atingir seus objetivos em seu ambiente de tarefa. A motivação desta pesquisa se deve à lacuna existente em termos de técnicas de testes aplicadas especificamente para agentes autônomos, de modo que possam avaliar o comportamento e a confiança dos sistemas baseado em agentes.

Este trabalho propôs uma formulação para o problema de seleção de casos de teste e uma abordagem solução para o problema, que levam em consideração a medida de avaliação de desempenho do agente, descrita em termos de vários objetivos que podem ser conflitantes entre si. Foi apresentando o esboço de um agente de resolução de problemas de seleção de casos de teste para programas agentes racionais (*Thestes*). O esboço de *Thestes* fundamenta-se na arquitetura de agentes orientados por utilidade e emprega uma estratégia de busca local, baseada em população e orientada pela função utilidade, para gerar e selecionar objetivamente casos de teste a partir da utilidade associada a cada um destes. Considerando que o processo de teste do agente dependerá dos casos de teste selecionados, *Thestes* objetiva selecionar um conjunto de casos de teste ótimo, ou seja, aquele que, na simulação da avaliação de desempenho das interações do agente testado com seu ambiente (histórias), o agente não foi bem avaliado, isto é, obteve um baixo desempenho.

As informações geradas pela abordagem indicaram uma medida de utilidade média associada ao desempenho do agente testado e os objetivos na medida de avaliação que não estão sendo satisfeitos. Considerando o melhor conjunto de histórias do agente em seu ambiente, associado ao conjunto de casos de teste selecionados pela abordagem ao final do processo de busca, o projetista e/ou outros sistemas automáticos auxiliares podem identificar aqueles episódios problemáticos, e quais subsistemas de processamento da informação e módulos de informação associados estão causando o desempenho insatisfatório no agente.

Acredita-se que os objetivos delineados no início deste trabalho foram alcançados ao longo de seu desenvolvimento. No entanto, apresentam-se como principal limitação dessa pesquisa: avaliação em diferentes concretizações da função ação do agente *Thestes* que selecione conjuntos de casos de teste para outros tipos de programas agentes racionais em diferentes ambientes de tarefa.

6.2 Trabalhos Futuros

O aperfeiçoamento e a continuidade deste trabalho, além das próprias limitações apontadas anteriormente, constituem-se em oportunidades para trabalhos futuros que incluem:

(i) Concretização da função ação de *Thestes* para selecionar casos de teste que avaliem outros tipos de programas agentes racionais em diferentes ambientes de tarefa, tais como agentes baseados em objetivos e agentes baseados em utilidade;

(ii) Concretização da função ver de *Thestes* para perceber ambientes com diferentes categorias, tais como: sequencial, estocástico e dinâmico;

(iii) Extensão de *Thestes* para contemplar os testes da simulação da avaliação de desempenho das interações entre os agentes presentes em sistemas multi-agentes e o ambiente de tarefa;

(iv) Extensão de *Thestes* para contemplar a avaliação e, consequentemente, a identificação de quais subsistemas de processamento da informação e módulos da informação associados ao agente estão causando o desempenho insatisfatório do agente;

(v) Incorporação da dificuldade das propriedades do ambiente como um critério para a avaliação do desempenho do agente;

(vi) Investigação de outros aspectos que possam ser incluídos na medida de avaliação de desempenho, ou um critério mais apropriado, que contribuam na identificação de episódios problemáticos do agente;

(vii) Aplicação de outras metaheurísticas multiobjetivas baseadas em população para avaliação dos resultados obtidos.

6.3 Conclusão

O benefício obtido por meio da formulação para o problema de seleção de casos de teste e uma abordagem solução para o problema representa uma contribuição tanto na área

de Inteligência Artificial (IA) como de Engenharia de Software (ES). Na área de ES, o presente trabalho contribui especificamente com a modelagem e a com as orientações para geração de casos de teste, através de estratégias evolucionárias, para o processo de teste.

Do ponto de vista de IA, a contribuição deste trabalho está voltada para a possibilidade de testar os agentes racionais propostos por Russell e Norvig (2004). Neste contexto, o presente trabalho apresenta uma abordagem que considera, no caso dos agentes racionais, uma medida de avaliação definida pelo projetista que, em geral, envolve mais de um objetivo e estes podem ser conflitantes entre si. O resultado dos testes pode indicar o desempenho médio do agente e, principalmente, os objetivos que não estão sendo satisfeitos. Além de informações sobre as interações entre o agente e seu ambiente de tarefa (histórias) que sejam úteis para o projetista identificar, a partir dos episódios problemáticos, os comportamentos insatisfatórios do agente e realizar mudanças que são necessários para que o agente melhore seu desempenho.

Atrelando o contexto de ES e IA, pressupondo que uma boa avaliação do agente depende dos casos de teste selecionados e que nem sempre os melhores casos estão disponíveis inicialmente, o presente trabalho apresenta a geração de casos de teste a partir de metaheurísticas baseadas em população, possibilitando gerar casos de teste com níveis de dificuldades crescentes. Com isso, como diferentes representações do ambiente apresentam diferentes níveis de dificuldade em que o agente deverá operar, estamos possibilitando que o agente seja observado e avaliado em uma grande quantidade de casos de teste com diferentes níveis de dificuldade e complexidade.

6.4 Publicações Resultantes

Diversos artigos foram publicados a partir dos resultados obtidos, gerando algumas publicações, e esta dissertação é o resultado final deste trabalho. A seguir são listadas em ordem cronológica essas publicações:

1. “Agentes Racionais para o Teste de Agentes Racionais” escrito com Gustavo Augusto Lima de Campos e Mariela Inés Cortés. Apresentado no V Encontro Unificado de Computação em Parnaíba em 2012. Prémio de 3º melhor trabalho (SILVEIRA *et al.* 2012).
2. “Rational Agents for the Test of Rational Agents” escrito com Gustavo Augusto Lima de Campos e Mariela Inés Cortés. Publicado na IEEE Latin America Transactions em 2013 (SILVEIRA *et al.* 2013a).

3. “Um Agente de Resolução de Problemas de Seleção de Casos de Teste de Agentes Racionais” escrito com Gustavo Augusto Lima de Campos e Mariela Inés Cortés. Apresentado no 4th Workshop on Autonomous Software Systems (AutoSoft) em 2013. Prémio *Best Paper.* (SILVEIRA *et al.* 2013b)
4. Devido ao prémio *Best Paper* no AutoSoft 2013, uma versão ampliada será enviada para a Revista de Informática Teórica e Aplicada (RITA) em 2013.

REFERÊNCIAS

ANTUNES, J. F. *Tolerância a Falhas como um Modelo de Agentes*. Dissertação, Pontífica Universidade Católica do Rio Grande do Sul, Porto Alegre, 2009.

BACK, T.; FOGEL, D.; MICHALEWICZ, Z. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

BOURQUE, P.; DUPUIS, R. Guide to the Software Engineering Body of Knowledge. *IEEE Computer Society*, 2004.

BRESCIANI, P.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*. p. 203-236, 2004.

DEB, K. *Introduction to Evolutionary Multiobjective*. Berlin: Springer-Verlag, 2008.

DEB, K.; AGARWAL, S.; PRATAP, A.; MEYARIVAN, T. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*. p. 849-858. London, 2000.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*. p. 34-41, 1978.

FACCIOLI, R. A. *Implementação de um Framework de Computação Evolutiva Multi-Objetivo para Predição Ab Initio da Estrutura Terciária de Proteínas*. Tese, Universidade de São Paulo, São Paulo, 2012.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers and Operation Research*. v. 13, p. 533–549, 1986.

GONÇALVES, E. J. T. *Modelagem de arquiteturas internas de agentes de software utilizando a linguagem MAS-ML 2.0*. Dissertação, Universidade Estadual do Ceará, Fortaleza, 2009.

HAMLET, R. G. Testing programs with the aid of a compiler. *IEEE Transactions on Software Engineering*. p. 279-290, 1977.

HARMAN, M.; JONES, B. F. Search-Based Software Engineering. *Information and Software Technology*. p. 833-839, 2001.

HENDERSON-SELLERS, B.; GIORGINI, P. Agent-Oriented Methodologies. *Idea Group Inc*, 2005.

HOLLAND, J. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

HOUHAMDI, Z. Multi-Agent System Testing: A Survey. *International Journal of Advanced Computer Science and Applications*. v. 2, 2011.

HOUHAMDI, Z. Test Suite Generation Process for Agent Testing. *Indian Journal of Computer Science and Engineering (IJCSE)*. v. 2. 2011.

MCMINN, P.; HOLCOMBE, M. The state problem for evoltionary testing. *Proc. of the Genetic and Evolutionary Computation Conference*. p. 2488-2498. 2003.

MICHALEWICZ, Z. *Genetic algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1996.

MYLOPOULOS, J.; CASTRO, J. Tropos: A Framework for Requirements-Driven Software Development. *Information Systems Engineering: State of the Art and Research Themes, Lecture Notes in Computer Science, Springer*. 2000.

NGUYEN, C. D. *Testing Techniques for Software Agents*. Dissertação PhD, International Doctorate School in Information & Communication Technologies. DIT - University of Trento, Trento, 2008.

NGUYEN, C. D.; PERINI, A.; TONELLA, P. Automated Continuous Testing of Multi-Agent Systems. *5th European Workshop on Multi-Agent Systems*. Hammamet, 2007.

NGUYEN, C. D.; PERINI, A.; TONELLA, P.; MILES, S.; HARMAN, M.; LUCK, M. Evoluctionary Testing of Autonomous Software Agents. *8th Int. Conf. on Autonomous Agents and Multiagent Systems*. Budapest, Hungary, 2009.

NGUYEN, C.; PERINI, A.; BERNON, C.; PAVÓN, J.; THANGARAJAH, J. Testing in multi-agent systems. *Springer*. v. 6038, p. 180-190, 2011.

PARGAS, R.; HARROLD, M. J.; PECK, R. Test-data generation using genetic algorithms. Journal of Software Testing, Verifications, and Reliability. p. 263-282. 1999.

PRESSMAN, R. S. *Engenharia de Software*. 6. ed. São Paulo: McGraw-Hill, 2006.

ROUFF, C. A Test Agent for Testing Agents and their Communities. *IEEE Aerospace Conference Proceedings*. v. 5. p. 2633-2638. 2002.

RUSSELL, S.; NORVIG, P. *Inteligência Artificial*: uma abordagem moderna. 2. ed. São Paulo: Novatec, 2004.

SCHARDONG, A. *Aplicação de Algoritmos Evolucionários à Gestão Integrada de Sistemas de Recursos Hídricos*. Tese, Escola Politécnica da Universidade de São Paulo, São Paulo. 2011.

SILVEIRA, F. R. V.; CAMPOS, G. A. L.; CORTÉS, M. I. Agentes Racionais para o Teste de Agentes Racionais. *V Encontro Unificado de Computação em Parnaíba (ENUCOMP)*. Parnaíba, 2012.

SILVEIRA, F. R. V.; CAMPOS, G. A. L.; CORTÉS, M. I. Rational Agents for the Test of Rational Agents. *IEEE Latin America Transactions*. v. 11. n. 1. Feb, 2013.

SILVEIRA, F. R. V.; CAMPOS, G. A. L.; CORTÉS, M. I. Um Agente de Resolução de Problemas de Seleção de Casos de Teste de Agentes Racionais. *4th Workshop on Autonomous Software Systems (AutoSoft) 2013*. Brasília, 2013.

SOUSA, A. S. *Desenvolvimento de Modelos e Algoritmos Sequenciais e Paralelos para o Planejamento da Expansão de Sistemas de Transmissão de Energia Elétrica*. Tese, Escola de Engenharia de São Carlos, São Paulo. 2012.

SRINIVAS, N.; DEB, K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*. v. 2. n. 3. p. 221-248. 1994.

TALBI, E. G. *Metaheuristics: From Design to Implementation*. Canada: John Wiley & Sons, 2009.

TSUI, F.; KARAM, O. *Fundamentos de Engenharia de Software*. 2 ed. Rio de Janeiro: LTC, 2013.

VIANA, G. V. R. *Meta-Heurísticas e Programação Paralela em Otimização Combinatória*. Fortaleza: Edições UFC, 1998.

WEGENER, J. *Stochastic Algorithms*: Foundations and Applications. Berlin: Springer / Heidelberg: 2005.

WEISS, G. *Multiagent Systems*: A Modern Approach to Distributed Artificial Intelligence. Massachusetts: MIT Press, 1999.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. England: John Wiley and Sons Ltd, 2002.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*. Cambridge University Press. v. 10. 1995.