Incorporando Filtros de Percepção para Aumentar o Desempenho de Agentes Jason

Márcio F. Stabile Jr.[1]*, **Jaime S. Sichman**[1,2]†

[1]Programa de Pós-Graduação em Ciência da Computação
Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

[2]Escola Politécnica – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

mstabile@ime.usp.br, jaime.sichman@poli.usp.br

Abstract. *The use of simulation systems is gradually increasing due to its ability to reproduce the real world. In particular, we are interested in those systems where simulated humans interact with an environment. To better exploit the potential of these systems, we can make use of intelligent autonomous agents to simulate human behavior. The use of the Jason interpreter in the creation and implementation of intelligent agents can then reduce the effort required for its development. As agents become more sophisticated, also increases the processing time required for the agent to perceive its environment, to reason about their actions, and to decide the best action to take. Such time interval can be bigger than the limit required by the simulator for an agent to act on the environment. Hence, the objective of this research is to modify the Jason interpreter in order to filter his perceptions of the environment, so as to reduce the time taken to process them and consequently reduce the agent total running time. Finally, an empirical analysis is made in order to verify the impact of perception filters on the reduction in agent's time response and performance.*

Resumo. *Sistemas de simulação são cada vez mais utilizados devido à sua capacidade de reproduzir o mundo real. Em particular, estamos interessados em sistemas que simulam a interação entre humanos e o ambiente. Para melhor utilizar o potencial desses sistemas, podemos fazer uso de agentes autônomos inteligentes para simular o comportamento humano. Utilizar o interpretador Jason no processo de criação e execução dos agentes inteligentes pode então reduzir o esforço necessário para seu desenvolvimento. Conforme os agentes se tornam mais sofisticados, cresce também o tempo de processamento necessário para que o agente possa perceber o ambiente, raciocinar sobre suas ações, e decidir qual deve ser aplicada. Tal tempo pode ser maior do que o disponibilizado pelo simulador para que o agente efetue uma ação no ambiente. O objetivo desta pesquisa é então modificar o interpretador Jason para que este seja capaz de filtrar as percepções vindas do ambiente, de modo a reduzir o tempo levado para processá-las e consequentemente reduzir o tempo total de execução do agente. Finalmente, é feita uma análise empírica para verificar o*

*Márcio F. Stabile Jr. é financiado pelo CNPq.
†Jaime Simao Sichman é parcialmente financiado pelo CNPq, proc. 303950/2013-7.

impacto dos filtros de percepção na redução do tempo de processamento e do desempenho do agente.

1. Introdução

Os sistemas de simulação se fazem cada vez mais importantes pela sua capacidade de imitar a realidade. Em particular, estamos interessados em sistemas que simulam a interação entre humanos e o ambiente. Por permitir a avaliação de diferentes cenários, esses sistemas são uma ferramenta poderosa para avaliar os resultados de ações sobre um ambiente, já que eles permitem aos usuários compreender as características de um ambiente ou uma situação complexa sem a necessidade de reproduzi-la no mundo real, podendo identificar potenciais áreas problemáticas, e assim realizar modificações apropriadas no projeto final.

Os sistemas de simulação como o da RoboCup[1] e do MAPC[2] se tornam mais complexos a cada ano, fazendo com que seja necessária uma maior capacidade de tomada de decisão. Uma importante característica destes sistemas de simulação é que a execução do simulador é realizada em passos. No início de cada passo o simulador envia as informações do ambiente para os agentes. Os agentes devem então decidir suas ações e as enviar de volta ao servidor em um determinado tempo pré-fixado.

Modelar o comportamento humano para esse tipo de situação é extremamente difícil, pois além de modelar o raciocínio individual, é necessário também modelar a tomada de decisão em grupo. Uma das maneiras utilizadas é a criação de agentes inteligentes que imitam as características gerais dos seres humanos. Integrar tecnologias baseadas em agentes em mundos criados por sistemas de simulação permite aumentar o grau de realismo na representação dos comportamentos.

Jason[3] [Bordini et al. 2007b] é um interpretador para uma versão estendida de AgentSpeak [Rao 1996] (linguagem de programação lógica com eventos e ações) que pode ser utilizado para o desenvolvimento de Agentes inteligentes. Jason implementa a semântica operacional dessa linguagem e proporciona uma plataforma para o desenvolvimento de sistemas multi-agentes. Jason tem seu código aberto e é construído na linguagem Java.

Dado que o ciclo de raciocínio do Jason é independente do simulador, realizar a integração dos dois é uma tarefa difícil, uma vez que aspectos como o tempo de resposta dos agentes pode influenciar diretamente no seu desempenho. Com isso temos que por muitas vezes o tempo levado para encontrar a melhor ação é maior que o tempo fixado pelo simulador, o que acarreta na perda da ação e inércia do agente durante todo o passo. Além disso, quanto mais complexos ficam os agentes, maior é o tempo utilizado para se processar, agravando mais o problema.

Uma possível solução para o problema seria a utilização de *filtros de percepção* de forma a reduzir o número de percepções recebidas e assim possivelmente reduzir o tempo de processamento do agente.

O objetivo desta pesquisa é então construir uma modificação do interpretador Jason que seja capaz de filtrar as percepções vindas do ambiente para que seja possível

[1]RoboCup: `http://www.robocup.org/`
[2]MAPC: `https://www.multiagentcontest.org/`
[3]Jason: `http://jason.sourceforge.net/wp/`

responder a perguntas como: (i) Dado um certo limite de tempo de resposta do simulador, é possível para o agente processar todas as informações provenientes das percepções? (ii) Ao utilizar um filtro de percepções, é possível melhorar o tempo de resposta? (iii) Caso melhore, como se identificaria o filtro adequado?

2. Estado da Arte

Em [van Oijen and Dignum 2011] os autores tem como objetivo a integração de agentes criados em plataformas multi-agentes como 2APL [Dastani et al. 2007], Jadex [Braubach et al. 2003] e Jason [Bordini et al. 2007a] com ambientes virtuais em tempo real, mais especificamente jogos de computador. Nesse artigo foi proposto um *middleware* para fazer a integração entre os dois sistemas. Esse *middleware* é responsável por receber sinais do ambiente virtual e fazer todo o processamento para entregar ao agente somente o conjunto de percepções que sejam de seu interesse. Isso acontece pois, como descrito no artigo, um dos problemas do paradigma BDI quando se une os agentes a ambientes virtuais é a falta de controle sobre as percepções. Se não houver alguma forma de percepção direcionada ao objetivo, o agente pode ser inundado com informações sensoriais que podem resultar no raciocínio sobre uma enorme quantidade de informações irrelevantes. Além disso, a falta de controle também não é realista quando olhamos para a fisiologia da percepção humana. Como a atenção é considerada um recurso limitado, não se pode atender a todos os aspectos do ambiente. Durante a execução de uma tarefa, os seres humanos, portanto, tendem a direcionar sua atenção para informações selecionadas do ambiente que podem apoiá-los na realização da tarefa. Isto sugere que uma abordagem similar deve ser utilizada também para agentes BDI.

Para atacar esse problema foi criado um Gerenciador de Assinaturas de Interesse que contém assinaturas para informações específicas do ambiente. Quando um agente adota um objetivo, o sistema cria automaticamente um conjunto de inscrições de informações sobre o ambiente usando o Gerenciador de Assinaturas de Interesse. Assim, quando um objetivo é alcançado ou descartado, o sistema automaticamente cancela a inscrição da informação correspondente. O sistema é capaz de fazer isso com base em uma tabela criada off-line que mapeia os objetivos do agente para conjuntos de assinaturas. Uma exemplo de tabela pode ser visto na figura 1. Nela é possível ver que existem condições específicas de quando e com que frequência se deve perceber determinados objetos como pessoas e carros.

O objetivo de [Bordeux et al. 1999] é criar uma extensão para uma arquitetura de software para a gestão das acções chamada AGENTlib, criada em [Boulic et al. 1997], com um mecanismo de percepção. Esse mecanismo de percepção conta com um filtro de percepções que tem como objetivo filtrar eficientemente informações valiosas da cena onde se encontra o agente. No artigo, o filtro de percepção representa a entidade básica do mecanismo de percepção. Esse filtro recebe entidades perceptíveis a partir da cena como entrada, extrai informações específicas sobre elas, e, finalmente, decide se envia a percepção referente a essa entidade ao agente ou não. Para diferentes finalidades, foram criados alguns tipos de filtro. Entre eles os que se destacam são:

- **Filtro de intervalo:** Este filtro seleciona somente os objetos que estão fisicamente ao redor do agente em um determinado intervalo de espaço.
- **Filtro de campo de visão:** Este filtro simula um campo de visão de um agente com uma determinada abertura angular.

Table I
OBJECT ATTRIBUTE SUBSCRIPTION EXAMPLES

OBJECT CLASS	PROPERTY	CONDITION	SOURCE	UPDATE POLICY	DESCRIPTION
PHYSICALOBJECT	LOCATION	-	-	FREQUENCY-1	LOCATION ALL OBJECTS AT 1 HZ.
HUMAN	LOCATION	-	-	FREQUENCY-5	LOCATION OF ALL HUMANS AT 5 HZ.
CAR	LOCATION	[COLOR=RED]	-	FREQUENCY-2	LOCATION OF RED CARS AT 2 HZ.
CAR	MODEL	-		ONE-TIME	MODEL OF A CAR
DOOR	STATUS	-		VALUE-CHANGE	DOORS OPENING/CLOSING
HUMAN	GENDER	-	HUMAN21	ONE-TIME	GENDER OF ONE SPECIFIC PERSON

Table II
EVENT SUBSCRIPTION EXAMPLES

EVENT CLASS	PARAMETER	CONDITION	SOURCE	DESCRIPTION
COMMUNICATION	-	-	-	ALL COMMUNICATION EVENTS
COMMUNCIATION	-	-	HUMAN21	COMMUNICATIONS FROM ONE SPECIFIC PERSON
COMMUNICATION	RECEIVER	HUMAN22	HUMAN21	COMMUNICATIONS BETWEEN TWO SPECIFIC SPEECH PARTNERS
PHONERINGING	-	-	CELLPHONE1	RINGING OF A SPECIFIC PHONE

Figura 1: Tabela de inscrições utilizada em [van Oijen and Dignum 2011].

- **Filtro de colisão:** Este filtro detecta os objetos sujeitos a potenciais impactos com o agente de acordo com sua trajetória.
- **Filtro detector de tipo:** Este filtro procura ao redor do agente por objetos com tipos e propriedades específicas.

Também é possível combinar diferentes filtros para atingir o objetivo desejado. Com isso, foi possível alcançar o objetivo inicial e permitir aos agentes perceber os objetos de interesse no ambiente.

Nossa abordagem é baseada nestes dois trabalhos. Da abordagem de [van Oijen and Dignum 2011], utilizamos a idéia de especificar documentos indicando as percepções a serem descartadas. Entretanto, em nosso caso não é adequado a construção de um middleware específico para esta tarefa, já que desejamos incorporar tais filtros ao Jason. Assim, nos inspiramos na abordagem de [Bordeux et al. 1999] para tal, além de também incorporarmos a idéia de tipos de filtros distintos. O detalhamento destas idéias é apresentado na seção 4.

3. Ciclo de Percepção e Raciocínio em Jason

Jason é um interpretador open-source implementado na linguagem Java para uma versão estendida da linguagem AgentSpeak, implementando a semântica operacional da linguagem e fornecendo uma plataforma para o desenvolvimento de sistemas multi agentes.

Em Jason, os agentes são executados por meio de um ciclo de raciocínio que pode ser dividido em 10 passos principais. Durante cada ciclo de raciocínio o agente recebe as percepções do ambiente, as mensagens de outros agentes e executa uma ação. [Bordini et al. 2007b]

O processo interno é ilustrado pela figura 2 se dá da seguinte forma:

1. **Percepção do ambiente:** O primeiro passo na execução do agente é perceber o ambiente. As percepções são um conjunto de literais onde cada literal representa uma propriedade no atual estado do ambiente.
2. **Atualização da Base de Crenças:** Uma vez em posse das percepções, a base de crenças é atualizada para refletir as mudanças do ambiente. Isso é feito através de

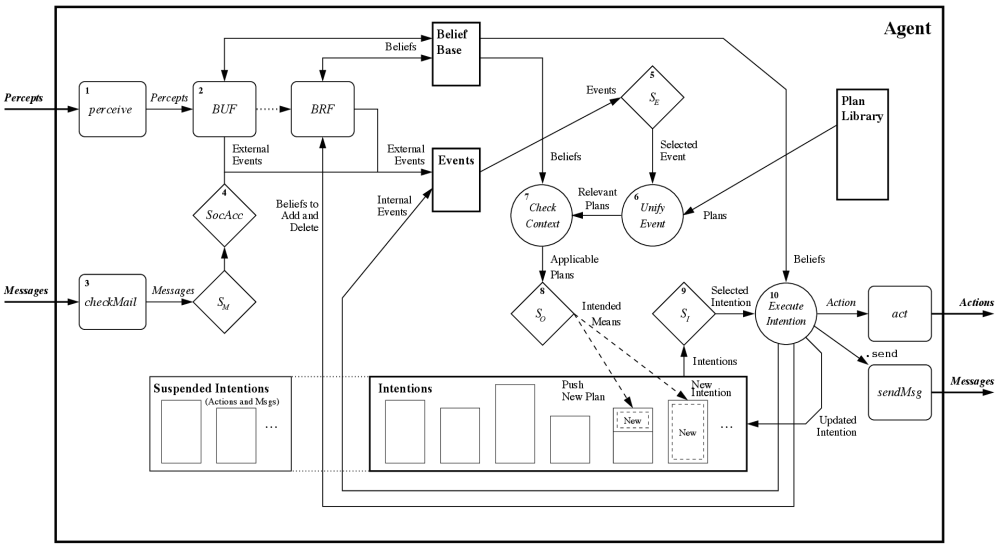


Figura 2: Ciclo de raciocínio de um agente Jason.

uma função de atualização personalizável. Na implementação padrão do Jason, assume-se que tudo o que pode ser percebido pelo agente está listado no conjunto de percepções. Assim, o método padrão percorre a base de crenças e a lista de percepções para incluir os literais novos, remover os literais que não foram mais percebidos e manter aqueles que já eram conhecidos. Cada inclusão ou exclusão de literais gera um evento que é adicionado à uma lista que será utilizada posteriormente.

3. **Recebendo comunicação de outros agentes:** Neste estágio, o interpretador verifica se o agente recebeu alguma mensagem de outro agente. Caso afirmativo, é selecionada uma delas para ser processada.
4. **Selecionando mensagens "Socialmente Aceitáveis":** Antes de processada, a mensagem passa por uma verificação para determinar se ela deve ou não ser aceita pelo agente, verificando o remetente e o conteúdo através de uma função personalizável.
5. **Selecionando um Evento:** Da lista de eventos criada na fase de atualização das crenças, é selecionado um evento para ser executado. Caso não haja alteração da função de seleção, o primeiro evento da lista é escolhido.
6. **Recuperando os planos relevantes:** Selecionado um evento, é necessário encontrar um plano que permita ao agente agir de forma a lidar com o evento. O primeiro passo é percorrer a lista de planos e encontrar aqueles cujo evento seja igual ao selecionado.
7. **Determinando os planos aplicáveis:** Com a lista de planos relevantes, é preciso verificar quais deles podem realmente ser executados, ou seja, cujas pré-condições são válidas.
8. **Selecionando um dos planos aplicáveis:** Neste ponto, já foram determinados todos os planos que podem ser utilizados para lidar com o evento selecionado. Assim, teoricamente, qualquer um dos planos pode ser utilizado para esse fim. Para decidir então qual deles executar é utilizada mais uma função de seleção personalizável que o adiciona à lista existente de intenções.
9. **Selecionando uma intenção para posterior execução:** Durante a execução, ti-

picamente um agente possui mais de uma intenção para executar, cada uma representando um foco de atenção diferente. No entanto, somente uma intenção pode ser executada por vez. Assim, é necessário escolher uma única intenção para ser executada.

10. **Executando um passo da intenção:** Neste último estágio, o agente já atualizou suas informações sobre o ambiente, lidou com um dos eventos gerados e deve agora, efetuar uma ação. Da intenção escolhida, é selecionada a primeira ação que ainda não foi executada e esta é efetuada no ambiente.

Dado que o interpretador Jason é implementado na linguagem Java, é possível utilizar conjuntamente com o AgentSpeak métodos desenvolvidos na linguagem Java para tarefas como por exemplo para cálculos matemáticos ou leituras de arquivos. Utilizando então estes mecanismos, o Jason é capaz de executar os agentes de forma que interajam com o ambiente quando necessário e realizem as tarefas especificadas.

Durante a análise do código-fonte, notamos que um dos pontos críticos de processamento é o segundo passo, onde ocorre a atualização das crenças. Por ter que verificar quais percepções são novas, remover as que não foram mais percebidas e manter aquelas que já eram conhecidas, no pior caso é realizada a comparação de todas as percepções recebidas com todas as que já estavam na base de crenças. Esse processo se torna muito custoso conforme se aumenta a quantidade de percepções. Além disso, cada inclusão ou exclusão na base de crenças gera um evento a ser analisado, fazendo então com que a quantidade de percepções recebidas do ambiente seja um fator ainda mais determinante para o tempo de resposta do agente, já que serão gastos mais ciclos para avaliar eventos que não irão gerar intenções antes de se obter uma resposta para o simulador.

Tendo esses pontos em vista, um filtro de percepções pode afetar diretamente o desempenho dos agentes e reduzir o tempo de processamento de diversas partes do ciclo de raciocínio.

4. Implementando Filtros de Percepção em Jason

Como dito anteriormente, na implementação padrão do Jason, assume-se que tudo o que pode ser percebido no ambiente está listado no conjunto de percepções recebido. Como nem sempre é viável para os agentes perceber todo o ambiente e ainda manter um alto nível de desempenho, é importante reconhecer as mudanças *relevantes* que ocorrem no ambiente.

Os filtros de percepção são então uma solução para esse problema, fazendo com que seja possível selecionar previamente e enviar ao agente somente aquilo que é interessante para ele naquele determinado momento.

Com base nos trabalhos de [van Oijen and Dignum 2011] e [Bordeux et al. 1999] e no funcionamento do Jason, foi idealizado um modo de incluir alguns dos tipos de filtro no mecanismo de percepção do agente Jason.

Um agente Jason recebe as percepções na forma de uma lista de literais em linguagem lógica no passo 1 do seu ciclo de raciocínio. Além disso, o Jason permite a adição de anotações aos literais, de forma que é possível adicionar informações extras às percepções. Por padrão, é adicionada a cada percepção a anotação *source* para que seja

possível saber se uma crença na base de crenças do agente surgiu de uma percepção, do raciocínio do próprio agente ou se foi uma informação vinda de outro agente. As anotações são, assim como as percepções, compostas por um predicado e parâmetros. Assim, um exemplo de percepção seria a seguinte lista:

```
posicao(inseto1,10,10)[source(self)]
posicao(inseto2,5,15)[source(percept)]
posicao(inseto3,18,4)[source(agente2)]
```

Desta forma, possuímos todas essas informações que podem ser utilizadas na hora de decidir se uma percepção deve ou não ser avaliada. De forma semelhante ao realizado com as tabelas em [van Oijen and Dignum 2011], para cada agente são criados documentos de restrição, que informam ao interpretador quais percepções devem ser descartadas. Estes documentos de restrição são construídos em formato xml, onde o exemplo a seguir mostra o documento de restrição que foi utilizado nos experimentos deste artigo:

```
<?xml version="1.0"?>
<PerceptionFilter>
  <filter>
      <predicate>posicao</predicate>
      <parameter operator="EQ" id="0">inseto.*</parameter>
      <parameter operator="GT" id="2">7</parameter>
      <anotation>
        <name>source</name>
        <limit operator="EQ" id="0">percept</limit>
      </anotation>
  </filter>
</PerceptionFilter>
```

Cada arquivo desse pode conter inúmeras tags *filter* para filtrar os diferentes predicados utilizados. Para que uma percepção seja descartada ela deve obedecer a todos os critérios especificados no filtro. Em cada filtro existem três critérios: o predicado, os parâmetros e as anotações, sendo que o único critério obrigatório é o predicado, dado que não é possível que exista um literal sem um predicado associado.

A restrição para o predicado é direta, é verificado se o predicado associado ao literal é o mesmo predicado especificado no filtro. Já para os parâmetros, existem duas informações necessárias, a primeira é o *id* que informa qual dos parâmetros deve ser comparado e o *operator*. Este último indica como deve ser feita a comparação do valor dentro da tag com o valor no literal vindo da percepção. *Operator* pode assumir cinco valores:

- **EQ:** O valor do parâmetro na percepção deve ser igual ao valor dentro da tag. Este último pode ser uma expressão regular para a linguagem Java.
- **GT:** O valor do parâmetro na percepção deve ser maior do que o valor dentro da tag (inteiro ou de ponto flutuante).
- **GE:** O valor do parâmetro na percepção deve ser maior ou igual ao valor dentro da tag (inteiro ou de ponto flutuante).

- **LT:** O valor do parâmetro na percepção deve ser menor do que o valor dentro da tag (inteiro ou de ponto flutuante).
- **LE:** O valor do parâmetro na percepção deve ser menor ou igual ao valor dentro da tag (inteiro ou de ponto flutuante).
- **NE:** O valor do parâmetro na percepção deve ser diferente valor dentro da tag (inteiro ou de ponto flutuante).

Dado que as anotações também são literais em linguagem lógica, é necessário que sejam comparados seu predicado e seus parâmetros. Assim, o valor dentro da tag *name* é comparado ao predicado da anotação para ver se são iguais. A tag *limit* funciona da mesma forma que a tag *parameter*, comparando os parâmetros contidos na anotação.

Mesmo com a possibilidade de se selecionar exatamente as informações que se deseja restringir, pode ser que dependendo da situação e do objetivo atual do agente, seja interessante que os filtros sejam alterados. Com essa finalidade, foi incluída uma ação interna *change filter* que pode ser incluída como parte de um plano para que o filtro atual seja alterado.

Em qualquer plano pode ser incluída a ação interna:

```
+!carry_to(R)
   <- !take(object,R);
        .change_filter(busca);
        -object(r1);
        !!search(slots).
```

Assim, depois que o agente executou o objetivo de pegar o objeto, o filtro é alterado para que haja o controle das percepções nos ciclos de raciocínio futuros. A palavra "busca"utilizada como parâmetro se refere ao nome do arquivo XML onde se encontra o filtro desejado. Assim, é possível ter diversos filtros para diversas situações.

Enquanto um filtro previamente selecionado não for alterado, sempre que o agente receber uma lista de percepções cada uma delas será analisada e eventualmente apagada caso corresponda aos critérios do filtro ativo. Sabendo qual o filtro a ser utilizado, foi inserido entre os passos 1 e 2 do ciclo de raciocínio o mecanismo de filtragem. Dessa forma, é possível verificar quais percepções recebidas pelo agente no passo 1 devem ser enviadas à função de atualização de crenças no passo 2.

5. Estudo de caso

Para tentar encontrar a resposta para as perguntas propostas na seção 1, foi implementado um ambiente onde um agente Jason deve perseguir, capturar insetos que se movem e levá-los de volta para sua base em um grid, como mostrado na Figura 3.

A execução do simulador é feita em passos: (i) o simulador envia os dados da percepção atual ao agente e espera a ação que ele irá executar no ambiente. (ii) ao receber essa ação ele a executa, atualiza a posição dos insetos (movimentando-os aleatoriamente para os lados, para cima ou ficando parados) e gera um novo conjunto de percepções para enviar ao agente no ciclo seguinte.

Para que houvesse impacto das percepções no funcionamento do agente, foi definido que o agente deve guardar a posição dos insetos no passo anterior para comparar

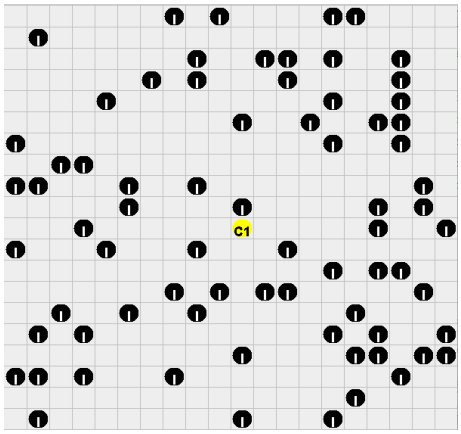


Figura 3: Ambiente utilizado para os testes de desempenho.

com a posição no passo atual. Assim ele se move para o quadrante onde houver a maior quantidade de insetos que se moveu em sua direção. Além disso, foi construído um filtro que ignora todos os insetos que se encontram a uma distância maior que um valor limite pré-definido.

Definimos como n a dimensão do grid, que representa a sua largura e altura (o grid contém n^2 espaços). Os experimentos aqui mostrados apresentam o valor de $n = 20$, permitindo uma variação significativa na quantidade de insetos.

Para os diferentes experimentos, diferentes parâmetros foram utilizados. São eles: (i) O alcance do filtro f, que representa a distância euclidiana máxima que o agente consegue perceber os insetos (ii) O número de insetos i, sendo que $i \leq n^2$.

Experimento	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
i	80	80	80	80	200	200	200	200	280	280	280	280
f	n	7	5	2	n	7	5	2	n	7	5	2

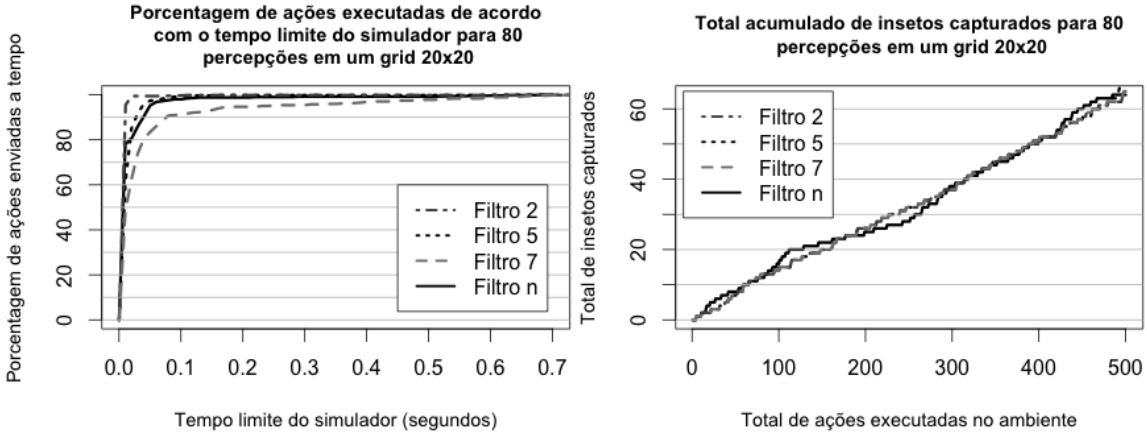
Tabela 1: Experimentos realizados

Foram realizados um total de 12 experimentos, onde se variou os valores desses parâmetros, conforme mostra a Tabela 1. Basicamente, variou-se o número de insetos num grid 20x20, ocupando 20% (80 insetos, experimentos 1 a 4), 50% (200 insetos, experimentos 5 a 8) e 70% (280 insetos, experimentos 9 a 12) do grid. Para cada um destes casos, simulou-se a ação dos agentes sem filtro de percepção (“n”) e com filtros correspondentes a distâncias de 2, 5 e 7 células.

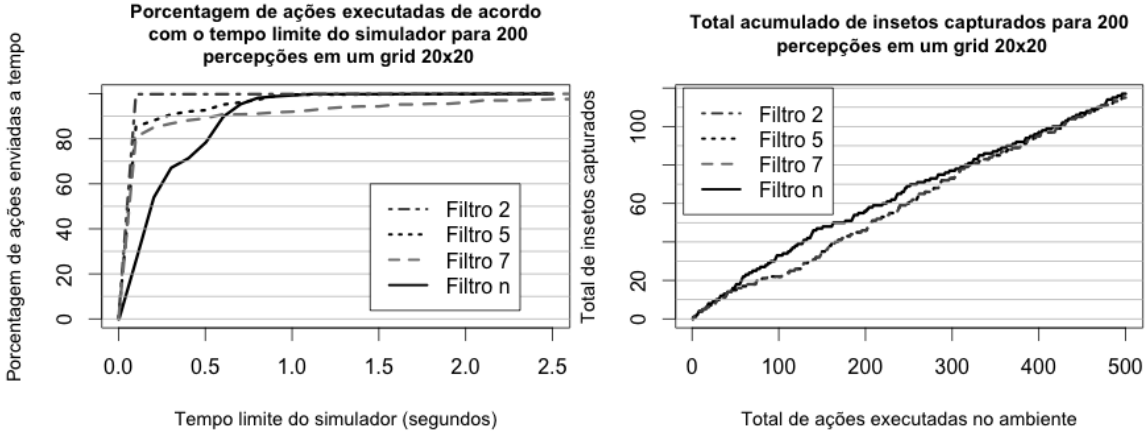
6. Resultados obtidos

Nos gráficos aqui apresentados, foi medida a capacidade do agente responder em um determinado tempo limite. O eixo horizontal marca o tempo limite do simulador em segundos. O eixo vertical mostra quantos por cento das ações foram executadas em tempo menor ou igual ao tempo limite.

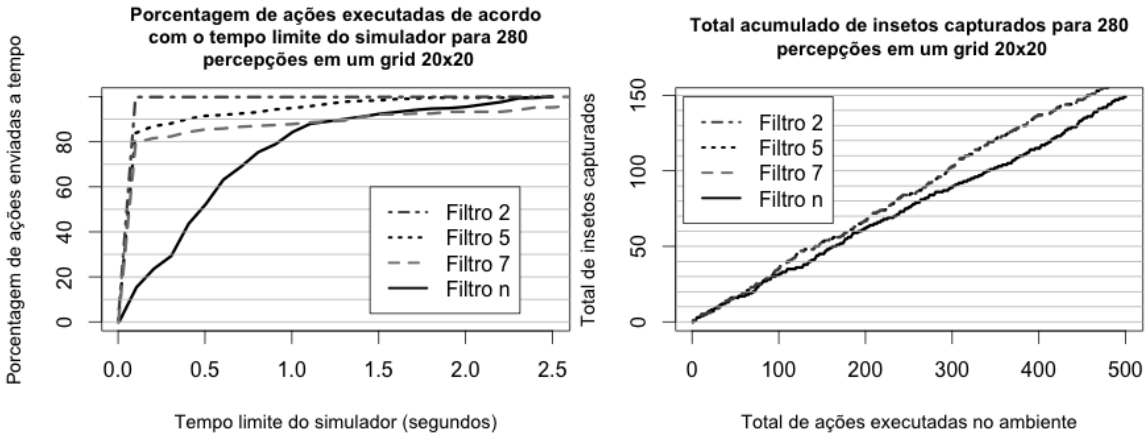
Na figura 4, são apresentados os resultados obtidos nos diversos experimentos. Nas figuras 4a e 4b, mostramos respectivamente o tempo de resposta para deliberar uma



(a) Tempo de resposta - 80 percepções (b) Métrica de desempenho - 80 percepções



(c) Tempo de resposta - 200 percepções (d) Métrica de desempenho - 200 percepções



(e) Tempo de resposta - 280 percepções (f) Métrica de desempenho - 280 percepções

Figura 4: Resultados dos experimentos. Os gráficos à esquerda mostram o desempenho dos agentes em função da quantidade de percepções, os da direita mostram o total de insetos capturados ao longo da simulação.

ação e a métrica de desempenho (número de insetos capturados) para um cenário com 80 insetos, reunindo os experimentos 1, 2, 3 e 4. Analogamente, as figuras 4c e 4d representam o cenário com 200 insetos com os experimentos 5, 6, 7 e 8 e as figuras 4e e 4f os cenários com 280 insetos reunindo os experimentos 9, 10, 11 e 12.

É possível verificar pelos gráficos que conforme se aumenta o número de percepções, a execução sem filtro é a que apresenta a maior perda de desempenho, necessitando de muito mais tempo que as outras para atingir porcentagens mais altas. Vale ressaltar que a escala horizontal do gráfico 4a se encontra reduzida pela metade, para que seja possível visualizar melhor a diferença entre os filtros.

Alguns outros fatos interessantes surgem ao analisar esses gráficos. Conforme esperado, o filtro mais restritivo ($f = 2$) é aquele que sempre necessita de menos tempo para execução e é também o primeiro que atinge a marca de 100%, ou seja, é garantido que todas as ações do agente serão enviadas ao simulador dentro do tempo limite especificado. Já o filtro com $f = 5$ é sempre o segundo a atingir a marca de 100%, mas talvez não seja sempre interessante seu uso, pois em um caso como no gráfico 4a, se o limite de tempo do simulador fosse de 0,01 segundos, o desempenho deste filtro é o mesmo daquele obtido sem filtro, e ainda levando ao agente menos informação. Mais interessante que os casos anteriores, observa-se que o filtro menos restritivo ($f = 7$) em muitos casos tem maior tempo de processamento do que o experimento sem filtro. Isso demonstra que não são todos os casos onde a utilização de um filtro diminui o tempo de processamento. Esse comportamento possivelmente se deve ao fato de que o tempo de processamento no ciclo de raciocínio do agente ganho com a exclusão destas percepções (possivelmente em número não suficiente) não compense o tempo de processamento para realizar esta filtragem.

Ao se analisar os três últimos gráficos, é possível notar que o desempenho do agente (quantidade de capturas) quando se utiliza os filtros é bastante semelhante a quando um filtro não é utilizado. O que demonstra que o agente ainda recebe informações suficientes para fazer boas decisões.

7. Conclusões e Trabalhos Futuros

Retomando as questões levantadas na seção 1, temos agora condições de respondê-las em vista dos experimentos realizados:

(i) Dado um certo limite de tempo de resposta do simulador, é possível para o agente processar todas as informações provenientes das percepções? A partir dos resultados obtidos em nossos experimentos, pode-se ver que dependendo do limite de tempo de resposta do simulador, é possível se prever aproximadamente se o agente terá ou não tempo para processar todas as percepções e executar suas ações. Para tal, basta entrar com o tempo nas abscissas dos gráficos da Figura 4 e verificar se a porcentagem de ações enviadas corresponde a 100%.

(ii) Ao utilizar um filtro de percepções, é possível melhorar o tempo de resposta? Os experimentos mostraram que ao utilizar os filtros de percepção, os agentes obtêm um controle maior sobre o tempo de resposta, reduzindo-o até onde for necessário.

(iii) Caso melhore, como se identificaria o filtro adequado? Os experimentos também tornaram possível identificar, para um dado tempo limite de resposta, quais dos

filtros testados garantem que a porcentagem de ações enviadas corresponda a 100%. Dentre estes, poderia-se escolher aquele menos restritivo, ou seja, que oferece maior informação ao agente.

Deste modo, nossos experimentos permitiram concluir que a quantidade de percepções afeta o tempo de resposta de um agente Jason. Além disto, mostramos que o uso de filtros de percepção contribui para a redução e o controle do tempo de raciocínio de agentes Jason.

Como próximos passos, pretendemos: (i) Medir com maior precisão em quais casos o custo extra de processamento usado para filtrar as percepções é justificado pela redução do tempo usado para a função de atualização de crenças; (ii) Realizar um teste estatístico de significância que permita atestar o efeito exercido sobre a quantidade de percepções no tempo de resposta de um agente Jason; (iii) Mapear o tempo relativo levado pela função de atualização de crenças e pelo filtro comparado com o restante das operações (seleção de mensagens, seleção de eventos, seleção de planos, etc.). (iv) Efetuar testes com outros métodos de filtragem, como por exemplo eliminar percepções de elementos que não gerem eventos para os quais hajam planos ou propriedades do ambiente que não estejam listadas em nenhum plano do agente; (v) Realizar testes em diferentes cenários.

Referências

Bordeux, C., Boulic, R., and Thalmann, D. (1999). An Efficient and Flexible Perception Pipeline for Autonomous Agents. *Computer Graphics Forum*, 18(3):23–30.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007a). *Programming Multi-Agent Systems in AgentSpeak using Jason*. JohnWiley & Sons Ltd.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007b). *Programming multi-agent systems in {AgentSpeak} using {Jason}*. Wiley Series in Agent Technology. John Wiley & Sons Ltd, Chichester, UK.

Boulic, R., Bécheiraz, P., Emering, L., and Thalmann, D. (1997). Integration of Motion Control Techniques for Virtual Human and Avatar Real-Time Animation. *Proceedings of ACM VRST*, pages 111–118.

Braubach, L., Lamersdorf, W., and Pokahr, a. (2003). Jadex : Implementing a BDI-Infrastructure for JADE. *EXP in search of innovation*, 3(September):76–85.

Dastani, M., Mol, C., Tinnemeier, N. a. M., and Meyer, J. J. C. (2007). 2APL: A practical agent programming language. *Belgian/Netherlands Artificial Intelligence Conference*, pages 427–428.

Rao, A. S. (1996). {AgentSpeak(L)}: {BDI} agents speak out in a logical computable language. In de Velde, W. V. and Perram, J. W., editors, *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world (MAAMAW'96)*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 42–55, Secaucus, USA. Springer-Verlag.

van Oijen, J. and Dignum, F. (2011). Scalable Perception for BDI-Agents Embodied in Virtual Environments. *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pages 46–53.