CENTRO UNIVERSITÁRIO FEI

LEONARDO ANJOLETTO FERREIRA

PROGRAMAÇÃO EM LÓGICA NÃO-MONOTÔNICA APLICADA À REDUÇÃO DO ESPAÇO DE PLANOS EM PROCESSOS DE DECISÃO DE MARKOV

São Bernardo do Campo

2016

LEONARDO ANJOLETTO FERREIRA

PROGRAMAÇÃO EM LÓGICA NÃO-MONOTÔNICA APLICADA À REDUÇÃO DO ESPAÇO DE PLANOS EM PROCESSOS DE DECISÃO DE MARKOV

Tese de doutorado apresentada ao Centro Universitário FEI para obtenção do título de doutor em engenharia elétrica. Orientado pelo Prof. Dr. Paulo Eduardo Santos.

São Bernardo do Campo

2016

Ferreira, Leonardo Anjoletto.
Programação em Lógica Não-Monotônica Aplicada à Redução do Espaço de Planos em Processos de Decisão de Markov / Leonardo Anjoletto Ferreira. São Bernardo do Campo, 2016.
123 f. : il.

Tese - Centro Universitário FEI.
Orientador: Prof. Dr. Paulo Eduardo Santos.
Coorientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi.

1. Processos Markovianos de Decisão. 2. Programação Não-Monotônica. 3. Domínio Não-Estacionário. I. Santos, Paulo Eduardo, orient. II. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

centro universitário FEI

APRESENTAÇÃO DE TESE
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Doutorado

PGE-10

Aluno: Leonardo Anjoletto Ferreira **Matrícula:** 512203-1

Título do Trabalho: Programação em lógica não-monotônica aplicada à redução do espaço de planos em processos de decisão de markov.

Área de Concentração: Inteligência Artificial Aplicada à Automação

Orientador: Prof. Dr. Paulo Eduardo Santos

Data da realização da defesa: 29/08/2016

ORIGINAL ASSINADA

Avaliação da Banca Examinadora

São Bernardo do Campo, 29 / 08 / 2016.

MEMBROS DA BANCA EXAMINADORA	
Prof. Dr. Paulo Eduardo Santos	Ass.: _____
Prof. Dr. Agenor de Toledo Fleury	Ass.: _____
Prof. Dr. Flavio Tonidandel	Ass.: _____
Prof. Dr. Flávio Soares Correa da Silva	Ass.: _____
Prof. Dr. José Reinaldo Silva	Ass.: _____

A Banca Examinadora acima-assinada atribuiu ao aluno o seguinte:

APROVADO ☒ REPROVADO ☐

VERSÃO FINAL DA TESE

ENDOSSO DO ORIENTADOR APÓS A INCLUSÃO DAS RECOMENDAÇÕES DA BANCA EXAMINADORA

_____

Aprovação do Coordenador do Programa de Pós-graduação

_____

Prof. Dr. Carlos Eduardo Thomaz

“I could be bounded in a nutshell, and count myself a king of infinite space, were it not that I have bad dreams.”
(SHAKESPEARE, 1904)

RESUMO

Um desafio presente em problemas de tomada de decisão sequencial é o fato de que, ao longo do tempo, um domínio pode sofrer alterações não previstas. Enquanto que descrever apenas o domínio atual faz com que a chance de falhas na tomada de decisão aumente conforme o domínio sofre mudanças, descrever todas as possibilidades deste domínio com a finalidade de garantir que não haverá falhas quando o domínio sofrer alterações pode ser uma solução com alto custo de armazenamento e longo tempo de busca pela solução ótima. Para resolver este problema, este trabalho propõe o ASP(MDP) que utiliza *Answer Set Programming* para a descrição de um processo Markoviano de decisão em que avaliação de política de Monte Carlo ou Aprendizado por Reforço podem ser utilizados para realizar a interação com o ambiente e encontrar a solução ótima do problema. Enquanto a utilização de *Answer Set Programming* permite que a descrição do domínio seja revista conforme as alterações ocorrem, Aprendizado por Reforço ou avaliação de política de Monte Carlo permitem que as interações com o ambiente forneçam as informações restantes necessárias para que a solução ótima seja encontrada. Para avaliar o ASP(MDP) foram propostos quatro experimentos que demonstraram que a utilização de *Answer Set Programming* para descrever o processo Markoviano de decisão é capaz de reduzir o espaço de busca pela solução ótima, além de permitir que esta solução do problema seja encontrada sem a necessidade de reiniciar o processo de busca pela solução quando o domínio sofre alterações. No primeiro experimento deseja-se obter a melhor alocação de aplicações em servidores, sendo considerados o tempo necessário para o processamento de cada aplicação e a probabilidade de falha dos servidores. A utilização do ASP(MDP) permitiu verificar que o espaço de busca foi reduzido e a solução ótima obtida é a mesma com ou sem a utilização do *Answer Set Programming* para descrever o problema. Para o segundo experimento foi utilizado o sistema de controle por reações de um ônibus espacial, em que se deseja realizar uma manobra no espaço. Os resultados mostram que a utilização do *Answer Set Programming* permitiu não somente a redução no espaço de busca, mas também uma redução no tempo necessário para a obtenção da solução ótima e a própria descrição do processo Markoviano de decisão. O terceiro experimento se passa em um mundo de grade determinístico em que são comparados algoritmos de Aprendizado por Reforço com ASP(MDP). Neste experimento nota-se que é possível utilizar o ASP(MDP) em problemas que os conjuntos de estados e ações sofrem alterações ao longo do tempo, ao contrário do RL, e que a utilização de conhecimento adquirido antes das mudanças no ambiente fazem com que o aprendizado no novo mapa seja mais rápida do que quando é utilizado somente RL. O último experimento se passa na versão não-determinística do experi-

mento anterior e mostra que ASP(MDP) pode ser utilizado em domínios não-determinísticos e não-estacionários, permitindo a redução do espaço de busca e do tempo necessário para encontrar a solução ótima. Portanto, este trabalho apresenta um método tolerante à elaboração que permite a busca por soluções ótimas em processos Markovianos de decisão não-estacionários e não-determinísticos de forma que a solução seja encontrada mais rapidamente pela exploração de um espaço de busca menor.
Palavras-chave: Processos Markovianos de Decisão. Programação Não-Monotônica. Domínio Não-Estacionário.

ABSTRACT

In sequential decision making problems, dealing with unforeseen changes in the environment is still an open problem. While describing only the current domain increases the chances of failing when making a decision in an environment that has already changed, describing every possibility of the domain in order to avoid errors may lead to a description that has a high memory cost and also needs long periods in the search for the optimal solution. In order to solve this problem, this work proposes the ASP(MDP) which uses Answer Set Programming to describe a Markov Decision Process in which Monte Carlo Policy Evaluation and Reinforcement Learning can be used to interact with the environment and find the optimal solution. While Answer Set Programming allows for the revision of the domain's description, Reinforcement Learning and Monte Carlo Policy Evaluation interact with the environment and discovers the missing information so that optimal solution can be found. For ASP(MDP)'s evaluation, four experiments were proposed and has been shown that using Answer Set Programming to describe the Markov Decision Process can reduce the search space and also allows the search to proceed when the environment changes without the need to restart the searching process. In the first experiment the best sequence for allocating processes in servers is wanted, considering the processing power and failure rate of the servers. For this situation, not only ASP(MDP) provided a reduction in the search space but the optimal solution found is the same as the one present by a method that did not use Answer Set Programming. For the second experiment, the reaction control system of the space shuttle was used and the objective was to perform a manoeuvre in the space. Results show that ASP(MDP) was capable of greatly reduce the search space and time needed to find the optimal solution and also was able to find the description of the Markov decision process that was being used. The third experiment takes place in a non-stationary deterministic grid world in which ASP(MDP) is used with Reinforcement Learning algorithms. In this case, ASP(MDP) was capable of dealing with changes in the sets of states and actions, which is Reinforcement Learning methods are not capable of dealing with. Furthermore, the use of the knowledge gained in the previous map allows the learning process to be faster than that of Reinforcement Learning alone. The last experiment is presented as the non-deterministic version of the previous one and shows that ASP(MDP) can be used in non-deterministic non-stationary problems, which may also allows for reduction in the search space and acceleration in the learning process. Thus, this work presents a method that is elaboration tolerant and can be used to search for the optimal solution in non-stationary non-deterministic Markov Decision Processes, such that the optimal solution can be found faster by exploring a

reduced search space.
Keywords: Markovian Decision Processes. Non-Monotonic Programming. Non-Stationary Domains.

LISTA DE ILUSTRAÇÕES

Ilustração 1 – Relação entre MDP estacionário e não-estacionário 24
Ilustração 2 – Interação do agente com o ambiente para o RL 38
Ilustração 3 – exemplo de programa lógico 43
Ilustração 4 – resultado do *grounding* do exemplo da figura 3 43
Ilustração 5 – resultado de *solving* do exemplo da figura 4 utilizando *Clasp* 43
Ilustração 6 – Descrição do mundo de grades em $BC+$ 48
Ilustração 7 – Resultado fornecido pelo *CPLUS2ASP* para o programa apresentado na figura 6 49
Ilustração 8 – Código utilizado para a distribuição de programas em servidores . . . 61
Ilustração 9 – Código utilizado para a instância do domínio 61
Ilustração 10 –Movimentos que podem ser realizados pelo ônibus espacial 63
Ilustração 11 –Código que descreve o estado inicial dos RCS e a manobra que se deseja realizar 64
Ilustração 12 –Configurações do mundo de grade utilizado no experimento e seus respectivos *answer sets* 69
Ilustração 13 –Probabilidades de execução de cada ação no mundo de grades não-determinístico 72
Ilustração 14 –Saída exibida pelo programa *Clasp* (algumas linhas foram omitidas) . 75
Ilustração 15 –Diferença quadrática em cada episódio para os quatro algoritmos testados 80
Ilustração 16 –Retorno obtido para os quatro obtidos testados 84
Ilustração 17 –Número de passos em cada episódio para os quatro algoritmos testados 85
Ilustração 18 –MDP sem política que chega ao estado final 87
Ilustração 19 –MDP com apenas uma política que chega ao estado final 88
Ilustração 20 –MDP com apenas um estado e uma ação 90
Ilustração 21 –Domínio do precipício apresentado por Sutton e Barto (2015) 91
Ilustração 22 –Descrição em $BC+$ das paredes do mundo de grades conforme o mapa da figura 12a 105
Ilustração 23 –Parte da descrição em $BC+$ das paredes do mundo de grades conforme o mapa da figura 12c. O resto do código foi omitido por conta de sua extensão 106
Ilustração 24 –Descrição em $BC+$ das paredes do mundo de grades conforme o mapa da figura 12d 106

Ilustração 25 –Parte do código em ASP traduzido do programa em $BC+$ da figura 6 que apresenta a *query* e as descrições independente do domínio. A tradução foi realizada pelo programa *CPLUS2ASP* 107
Ilustração 26 –Código que descreve as constantes utilizadas pela tradução em ASP . . 108
Ilustração 27 –Descrição das regras e subconjuntos na tradução do código ASP 109
Ilustração 28 –Descrição de ações e fluentes na tradução em ASP 110
Ilustração 29 –Configuração de exibição da saída da execução do código em ASP . . 111
Ilustração 30 –Descrição do ambiente para a variável `X` e resultados das ações nesta coordenada 112
Ilustração 31 –Descrição do ambiente para a variável `Y`, resultados das ações nesta coordenada e parâmetros para execução 113

LISTA DE TABELAS

Tabela 1 – Descrição de variáveis do sistema de controle por reação 65
Tabela 2 – Valores de $Q(s,a)$ aproximados pelo MCASE para o problema do Cliente-Servidor 76
Tabela 3 – Quantidade de cada variável utilizada por cada processo de otimização para o problema do RCS do ônibus espacial 77
Tabela 4 – Tempo necessário (em segundos) para os algoritmos aproximarem a função valor-ação até chegar a uma diferença quadrática de $DQ \leq 10^{-5}$ . . . 81
Tabela 5 – Tempos para encontrar a solução ótima para cada um dos casos apresentados 90

LISTA DE ALGORITMOS

Algoritmo 1 – Avaliação de Política de Monte Carlo 38
Algoritmo 2 – Algoritmo Q-Learning 40
Algoritmo 3 – Algoritmo SARSA 40
Algoritmo 4 – Algoritmo ASP(MDP) 57
Algoritmo 5 – Algoritmo Monte Carlo Answer Set Evaluation (MCASE) 57
Algoritmo 6 – Algoritmo ASP(RL) 57

LISTA DE ABREVIATURAS

ASP	*Answer Set Programming*.
DP	Programação Dinâmica (*Dynamic Programming*).
DQ	Diferença Quadrática.
DQN	Rede Q Profunda (*Deep Q-Network*).
MC	Métodos de Monte Carlo.
MCASE	Avaliação de Answer Set de Monte Carlo (*Monte Carlo Answer Set Evaluation*).
MCPE	Avaliação de Política de Monte Carlo (*Monte Carlo Policy Evaluation*).
MDP	Processo Markovianos de Decisão (*Markov Decision Process*).
POMDP	Processo Markovianos de Decisão Parcialmente Observáveis (*Partially Observable Markov Decision Process*).
RL	Aprendizado por Reforço (*Reinforcement Learning*).
SPC-MDP	Processo Markovianos de Decisão Saturado com Restrição de Caminho (*Saturated Path-Constrained Markov Decision Process*).
TD	Diferenças Temporais (*Temporal Differences*).

LISTA DE SÍMBOLOS

a_i	Ação no instante i.
$\mathcal{A}$	Conjunto de ações.
A	Conjunto de Átomos.
$\mathcal{I}$	Conjunto dos Estados Iniciais.
$\mathcal{G}$	Conjunto dos Estados Objetivos.
$\mathcal{S}$	Conjunto de estados.
$\mathbb{R}$	Conjunto dos números reais.
$\mathcal{T}$	Conjunto dos instantes de tempo.
$\mathcal{H}$	Conjunto de trajetórias.
$\varnothing$	Conjunto vazio.
$\mathcal{D}$	Descrição.
γ	Desconto.
s_i	Estado no instante i.
Q	Função Valor-Ação.
V	Função Valor.
$\mathcal{R}$	Função de recompensa.
$\mathcal{P}$	Função de transição.
t	Instante de tempo.
L	Literal.
`not`	Negação forte (negação clássica).
$-$	Negação fraca (negação por falha).
M	Processo Markoviano de Decisão.
Π	Programa lógico.
π	Política.
p	Átomo.
r	Recompensa.
PF	Sequência de fórmulas proposicionais.
α	Taxa de aprendizado.
h	Trajetória.
p	Transição.

SUMÁRIO

1 INTRODUÇÃO 23
1.1 OBJETIVO 29
1.2 CONTRIBUIÇÕES DESTE TRABALHO 29
1.3 ORGANIZAÇÃO DO TEXTO 29
2 TRABALHOS CORRELATOS 31
3 CONCEITOS FUNDAMENTAIS 35
3.1 TOMADA DE DECISÃO SEQUENCIAL 35
3.2 ANSWER SET PROGRAMMING 41
3.3 A LINGUAGEM DE AÇÕES $BC+$ 44
4 ASP(MDP) 51
4.1 MDP M DESCRITO EM UM PROGRAMA LÓGICO $\Pi(\mathcal{S},\mathcal{A})$ 51
4.2 CONJUNTO DE ESTADOS CONTIDOS EM TRAJETÓRIAS 51
4.3 DESCRIÇÃO DE UM NOVO MDP $\dot{M}$ 52
4.4 SOLUÇÃO DO MDP $\dot{M}$ E OPTIMALIDADE 53
4.5 MUDANÇAS NÃO PREVISTAS NO DOMÍNIO AO LONGO DO TEMPO . . . 54
4.6 ALGORITMOS PARA ASP(MDP) 55
5 EXPERIMENTOS 59
5.1 DOMÍNIO CLIENTE-SERVIDOR 59
5.1.1 Descrição do Domínio 59
5.1.2 Resultados Esperados 60
5.2 DOMÍNIO DO SISTEMA DE CONTROLE POR REAÇÃO DE UM ÔNIBUS ESPACIAL 62
5.2.1 Descrição do Domínio 63
5.2.2 Resultados esperados 66
5.3 DOMÍNIO DO MUNDO DE GRADES NÃO-ESTACIONÁRIO 67
5.3.1 Descrição do Domínio 67
5.3.2 Resultados Esperados 70
5.4 DOMÍNIO DO MUNDO DE GRADES NÃO-DETERMINÍSTICO 71
5.4.1 Descrição do Domínio 71
5.4.2 Resultados Esperados 72

6 RESULTADOS 75
6.1 DOMÍNIO DO CLIENTE-SERVIDOR 75
6.2 DOMÍNIO DO SISTEMA DE CONTROLE POR REAÇÃO DE UM ÔNIBUS ESPACIAL 77
6.3 DOMÍNIO DO MUNDO DE GRADES NÃO-ESTACIONÁRIO 79
6.4 DOMÍNIO DO MUNDO DE GRADES NÃO-DETERMINÍSTICO 83
7 DISCUSSÃO 87
7.1 CASOS ESPECÍFICOS DA UTILIZAÇÃO DO ASP(MDP) 87
7.1.1 Não há solução para o MDP 87
7.1.2 Só há uma solução para o MDP 88
7.1.3 Não ocorre redução no espaço de busca 89
7.2 RESULTADO DE MÉTODOS *ON-POLICY* E *OFF-POLICY* APÓS UTILIZAÇÃO DO ASP(MDP) 91
7.3 POLÍTICA ÓTIMA E *ANSWER SET* ÓTIMO 92
7.4 RELAÇÕES ENTRE MDP ESTACIONÁRIOS E NÃO-ESTACIONÁRIOS . . . 93
7.5 HORIZONTE E CONVERGÊNCIA 94
8 CONCLUSÃO 95
8.1 TRABALHOS FUTUROS 96
REFERÊNCIAS 98
ANEXO A – CÓDIGOS UTILIZADOS NO TERCEIRO EXPERIMENTO . 104
ANEXO B – TRABALHOS PUBLICADOS 114

1 INTRODUÇÃO

Processos Markovianos de Decisão (MDP) permitem a representação de problemas de tomada de decisão sequencial em que o objetivo é encontrar a sequência de ações que permite alcançar um determinado estado final a partir de um estado inicial de forma a maximizar (ou minimizar) um certo critério de avaliação.

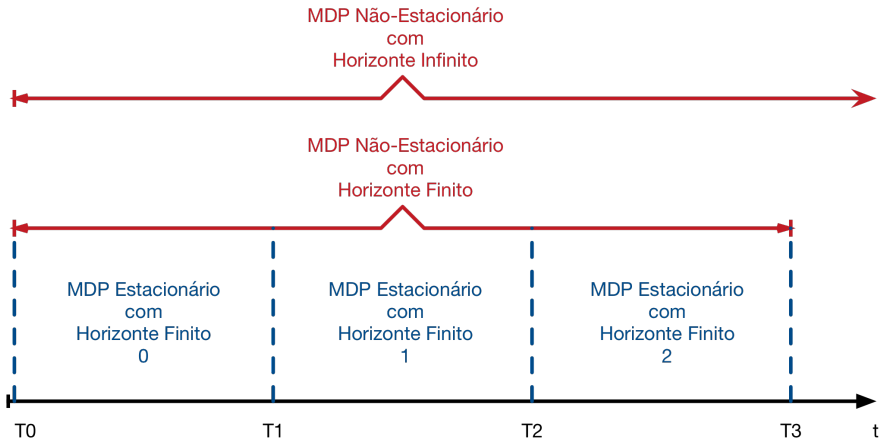
Métodos como o Aprendizado por Reforço (RL) e Avaliação de Política de Monte Carlo (MCPE) (SUTTON; BARTO, 1998; SUTTON; BARTO, 2015) que podem ser utilizados para solucionar MDPs permitem que a busca pela solução do problema seja realizada apesar de faltar conhecimento sobre parte do ambiente ou mesmo que certa parte do domínio seja alterada (como por exemplo, a função que se deseja maximizar). Entretanto, mesmo estes métodos não são capazes de lidar com alterações que podem ocorrer simultaneamente em dois dos conjuntos que formalizam um MDP: o conjunto de estados e o de ações. Entretanto, diversos problemas reais dependem de possíveis modificações que podem ocorrer nestes conjuntos ao longo do tempo.

Um exemplo de um domínio real é a estrutura viária de uma cidade com suas ruas e avenidas (estados). Neste domínio tem-se como objetivo sair de um determinado lugar desta cidade (estado inicial) e chegar a outro (estado final) atravessando parte da cidade (política). É possível descrever uma cidade como um MDP que possui como função que se deseja maximizar a velocidade que se deseja utilizar para ir de um ponto a outro. Neste MDP, caso a velocidade máxima das ruas seja alterada (função que se deseja maximizar), é possível encontrar a solução que maximiza a velocidade a partir da interação com o ambiente. Entretanto, caso as ruas sejam alteradas (estados) ou a direção em que elas podem ser percorridas (ações) sofra alterações, este MDP não representará esta cidade, mas uma versão anterior dela. Da mesma forma, é possível que por causa de eventos naturais (como por exemplo chuva e neve), a forma como é possível transitar pela cidade seja alterada sem conhecimento prévio.

Problemas como este podem ser representados como um MDP não-estacionário, em que a descrição do domínio necessita ser alterada com o tempo, uma vez que o domínio com o qual se interage também sofre alterações. Este trabalho estuda uma forma de descrever este tipo de domínio de forma que seja possível utilizar métodos que solucionam MDPs estacionários para encontrar a solução enquanto não ocorre mudança no domínio e a sequência destes MDPs estacionários que sofrem mudanças ao longo do tempo, descreve o MDP não-estacionário, como apresentado na figura 1. Nesta figura, MDPs estacionários com Horizonte Finito descrevem o ambiente por um certo período de tempo, de forma que o MDP “0” descreve o ambiente

entre $T0$ e $T1$, o MDP "1" descreve entre $T1$ e $T2$ e o MDP "2" no período de $T2$ à $T3$. Um MDP não-estacionário com Horizonte Finito é definido como a concatenação dos MDPs estacionários em que a interação com o ambiente é iniciada no MDP "0" e se encerra no MDP "2". A representação do não-estacionário com Horizonte Infinito é definido, neste caso, como a união das descrições estacionárias com Horizonte Infinito, porém após o término do MDP "2", a interação continua com o MDP estacionário que descreve o próximo período.

Figura 1 – Relação entre MDP estacionário e não-estacionário



Fonte: Autor

Dado o problema de encontrar a solução ótima de um MDP não-estacionário, primeiro é necessário apresentar a descrição estacionária e formas de encontrar sua solução.

O termo processo Markoviano de decisão foi utilizado pela primeira vez por Bellman (1957), porém estudos em tomada de decisão sequencial iniciaram com os trabalhos de Wald (WALD, 1947; WALD, 1949) em problemas de decisão sequencial estocástica. O livro de Bellman e Dreyfus (1971) apresenta o problema de tomada de decisão sequencial, o método de programação dinâmica (DP) (também chamado de *backward induction* (PUTERMAN, 2005), a definição de MDP e o princípio da optimalidade, definidos em seus trabalhos anteriores (BELLMAN, 1954; BELLMAN, 1957).

Um MDP é constituido por um conjunto $\mathcal{S}$ de estados, um conjunto $\mathcal{A}$ de ações, uma função de transição $\mathcal{P}$ (que indica a probabilidade de alcançar um certo estado futuro a partir da execução de uma ação no estado atual) e uma função de reforço $\mathcal{R}$ que deseja-se maximizar (ou minimizar) a partir de um critério de avaliação. A solução de um MDP é uma sequência de decisões (chamada de política) que otimiza $\mathcal{R}$ segundo um determinado critério, sendo esta solução chamada de política ótima (PUTERMAN, 2005; MAUSAM; KOLOBOV, 2012; SUTTON; BARTO, 2015). A busca por esta política ótima se dá pela atualização de uma fun-

ção valor $V(s)$ e/ou valor-ação $Q(s,a)$ que descreve o MDP. Para realizar a atualização destas funções, alguns métodos foram propostos.

Além do método de programação dinâmica que pode ser utilizado para encontrar a solução de MDPs (MAUSAM; KOLOBOV, 2012; PUTERMAN, 2005), outros métodos foram desenvolvidos, como a avaliação de política de Monte Carlo (MCPE) e o Aprendizado por Reforço (RL) (SUTTON; BARTO, 2015). A diferença entre estes três métodos é que, enquanto DP necessita de todas as informações do MDP, MCPE e RL são capazes de realizar a busca sem o conhecimento prévio da função de transição e/ou de reforço. Esta diferença no método de busca se dá pelo fato do MCPE e do RL utilizarem a interação com o domínio como forma de busca pela solução.

O MCPE segue os mesmos conceitos do método de Monte Carlo (METROPOLIS et al., 1953) para realizar a aproximação de uma função a partir de amostragem. No MCPE, ações são executadas seguindo uma política aleatória nos estados de forma a amostrar os valores de reforços que são obtidos ao longo do tempo. Dado um número suficiente de amostras, é possível obter a sequência de ações que formam uma política ótima (SUTTON; BARTO, 2015).

O Aprendizado por Reforço (RL) (SUTTON, 1988) permite de certa forma uma mistura entre DP e MCPE utilizando o método das diferenças temporais. No instante de selecionar a ação que será executada no ambiente, RL permite que uma ação seja escolhida aleatoriamente (como no MCPE) ou que seja executada a ação que apresenta o maior valor atualmente (como DP). Da mesma forma, a função valor-ação é atualizada a partir de cada reforço recebido (ao contrário de MCPE, que depende do episódio terminar) e é possível realizar a busca pela solução ótima apesar de mudanças na função de reforço ou de transição (ao contrário do que é feito em DP) (SUTTON; BARTO, 2015).

Dentre os algoritmos de RL existem duas formas de realizar a atualização de uma função valor-ação $Q(s,a)$ que é utilizada para encontrar a solução ótima de um MDP. O métodos *on-policy* utilizam o valor de $Q(s,a)$ da política que está sendo seguida (amostrada) para realizar a atualização desta função Q, sendo um dos métodos o SARSA (RUMMERY; NIRANJAN, 1994; SUTTON, 1996). O nome SARSA vem da sequência $s, a, r, s_{t+1}, a_{t+1}$ que denota respectivamente, o estado atual, ação atual, reforço recebido, estado futuro e ação futura determinada pela política que está sendo seguida. Uma outra forma de atualizar a função Q é realizada por métodos *off-policy*, como o Q-Learning (WATKINS, 1989), em que independente da ação que será executada no próximo estado, a atualização é realizada com a ação que possui o maior valor de $Q(s_{t+1},a)$.

Diversos trabalhos foram desenvolvidos sobre estes métodos de busca de solução de um MDP, entretanto estes são geralmente voltados à melhoria do método de busca pela solução (por exemplo, inicialização de valores (SUTTON; BARTO, 2015), realização de um processo mais rápido de busca pela solução ótima (SUTTON, 1991) e mistura de métodos que permitam aproximar a função necessária para diferentes domínios (MNIH et al., 2015) e não na mudança na descrição do MDP (por exemplo, descrever um MDP específico para um certo tipo de problema, mas com conhecimento prévio de todo o conjunto de estados e ações do MDP (PUTERMAN, 2005). Os trabalhos que tratam da descrição do MDP tendem a estudar problemas que podem ser descritos utilizando MDPs ou casos específicos destes que podem ter sua descrição feita de uma forma específica, como por exemplo os trabalhos que são discutidos no capítulo 2.

Uma apresentação aprofundada dos conceitos de MDP que são utilizados neste trabalho encontra-se na seção 3.1.

Entretanto, para solucionar o problema de MDPs não-estacionários, a proposta deste trabalho também depende de *Answer Set Programming* (ASP) (GELFOND, 2008).

ASP é uma abordagem para resolução de problemas utilizando uma linguagem declarativa em que não é descrito no programa como o problema deve ser resolvido, porém o programa descreve qual é o problema que deve ser resolvido, deixando que a solução do problema seja encontrada por um computador (GEBSER; KAMINSKI; KAUFMANN, 2013).

Uma possível solução para um programa lógico Π descrito utilizando ASP é chamada de *answer set*, que são modelos mínimos encontrados para um programa ASP. Estes modelos seguem a semântica de modelos estáveis proposta por Gelfond e Lifschitz (1988), sendo este um modelo que faz com que a interpretação do programa descrito seja verdadeira e não possui nenhuma negação por falha, porém um *answer set* permite a existência de dois tipos de negação: forte (ou clássica, representada pela notação "$-$") e fraca (representando a negação por falha que utiliza a notação "`not`").

Um aspecto importante de ASP é o princípio de racionalidade que diz que não é necessário acreditar em algo que não se é obrigado a acreditar (GELFOND; LIFSCHITZ, 1988), permitindo que além dos valores de verdade "verdadeiro" e "falso", ASP também admita o valor de "desconhecido". Por exemplo, seja P um *answer set* para um programa lógico Π e p um átomo de Π, se $p \in P$, então diz-se que p é verdadeiro. Se $-p \in P$, então diz-se que p é falso. Se nenhuma destas condições é verdadeira, então p é desconhecido.

Um outro aspecto de ASP é a tolerância à elaboração (MCCARTHY, 1998) descrita como "a habilidade de aceitar alterações na representação de fatos sobre um assunto sem a necessidade de reiniciar o processo". Um formalismo que é tolerante à elaboração é capaz

de modificar seu conjunto de fatos a fim de considerar fatos que não estão na sua base de conhecimento atual e mudança nos fatos que já possui.

Além destas características, ASP é capaz de representar problemas de busca em *NP* e problemas também em NP^{NP} (GEBSER; KAMINSKI; KAUFMANN, 2013).

A implementação de um sistema capaz de encontrar *answer sets* de um programa lógico descrito em ASP em geral faz o uso de um *grounder* (que gera uma representação finita de proposições a partir do programa lógico) e um *solver* (que encontra os modelos estáveis ou *answer sets* dependendo do programa utilizado). Dada a complexidade da implementação destes programas, a *ASP Competition* (IANNI; KRENNWALLNER; CALIMERI, 2013; CALIMERI et al., 2014) foi proposta para que diferentes times realizassem a implementação e comparação de suas propostas de *grounders* e *solvers*. Além da implementação permitir diversas abordagens, um programa lógico descrito segundo a semântica de ASP também pode ser chamado de A-Prolog (BALDUCCINI; GELFOND; NOGUEIRA, 2006; BALDUCCINI et al., 2001; BALDUCCINI et al., 2002; NOGUEIRA et al., 2001) por conta de sua relação com Prolog.

Por conta da forma como um programa lógico é escrito, ASP permite que certas construções em domínios de planejamento sejam mais facilmente representadas (LIFSCHITZ, 1999), permitindo que um problema de planejamento seja descrito em ASP e os planos que resolvem o problema tomem a forma de *answer sets*, como utilizado por Subrahmanian e Zaniolo (1995), Eiter et al. (2002), Eiter et al. (2003), Son e Pontelli (2003), Khandelwal et al. (2014), Yang et al. (2014), Zhang, Sridharan e Wyatt (2015) e Sridharan et al. (2015).

O ASP possui uma linguagem padronizada (que é apresentada em detalhes na seção 3.2), porém, além desta sintaxe, uma outra forma de desenvolver programas que fazem uso de *grounders* e *solvers* é a partir de linguagens de ações e de um tradutor. Estas linguagens de ações permitem que a descrição de um problema de planejamento seja realizada em nível mais alto do que em ASP, a partir da descrição dos estados, ações e a consequência da execução destas ações nos estados, sendo o tradutor utilizado para gerar um programa lógico escrito em linguagem que um *grounder* de ASP consiga interpretar. Por exemplo, Khandelwal et al. (2014) utiliza uma linguagem de ação para a descrição de um problema de planejamento enquanto Yang et al. (2014) utiliza a linguagem padronizada para ASP para um problema do mesmo tipo. Para tornar possível a utilização de *grounders* e *solver* com linguagens de ação, o programa *CPLUS2ASP* foi desenvolvido (BABB; LEE, 2013).

Apesar de diversas linguagens de ações poderem ser traduzidas para ASP, a linguagem $BC+$ proposta por Babb e Lee (2015b, 2015a) é uma das mais recentes e permite que algumas das construções de ASP sejam utilizadas diretamente em $BC+$, sendo também possível a utili-

zação do programa *CPLUS2ASP* para realizar a tradução. A linguagem $BC+$ será apresentada em detalhes na seção 3.3.

A partir dos fundamentos apresentados, para encontrar a solução de um MDP não-estacionário (como o da figura 1), a presente tese propõe o ASP(MDP), um método que utiliza um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ em ASP que descreve um MDP M.

A partir dos *answer sets* encontrados para $\Pi(\mathcal{S},\mathcal{A})$, um segundo MDP $\dot{M}$, que possui espaço de busca menor ou igual a M é definido. Utilizando $\dot{M}$, é realizada a busca pela política ótima a partir de um método capaz de encontrar a solução de um MDP estacionário como MCPE ou RL. Estes dois métodos foram nomeados de MCASE e ASP(RL) respectivamente.

A utilização do ASP(MDP) permite que as alterações que ocorrem no domínio possam ser alteradas no programa lógico, de forma que o processo de busca pela solução ótima possa ser continuado independente destas alterações (propriedade da tolerância a elaboração).

Para permitir a análise do ASP(MDP) quatro experimentos foram realizados. O primeiro descreve um ambiente não-determinístico em que processos são submetidos para um servidor com o objetivo de obter a melhor sequência de servidores em que os processos devem ser alocados. Este experimento permitiu o estudo da utilização de ASP para descrever um problema não-determinístico.

O segundo experimento utiliza o sistema de controle por reação que foi utilizado no ônibus espacial. Neste experimento o objetivo é encontrar a sequência de ações que permitem realizar uma determinada manobra no espaço. O experimento realizado com este sistema de controle permitiu verificar que é possível utilizar o ASP(MDP) para reduzir o espaço de busca.

O terceiro experimento realizado ocorre em um mundo de grades não-estacionário que sofre alterações não descritas anteriormente no MDP. A realização deste experimento demonstrou que é possível utilizar o ASP(MDP) para realizar a busca da política ótima em MDPs não-estacionários, além de demonstrar que um método *off-policy* (ASP(SARSA) encontra a mesma solução ótima que um método *on-policy* (ASP(Q-Learning) e também necessita do mesmo tempo.

O quarto experimento se passa em um mundo de grade não-determinístico que permite realizar a comparação do desempenho do ASP(MDP) com algoritmos que não utilizam o ASP, como Q-Learning e SARSA.

Portanto, o ASP(MDP) proposto neste trabalho é capaz de encontrar a solução de MDPs não-estacionários e não-determinísticos, além de permitir a diminuição do espaço de busca de um MDP estacionário.

1.1 OBJETIVO

Formalizar um Processo Markoviano de Decisão não-estacionário de forma que seja possível obter a política ótima deste nos períodos em que o domínio não está sofrendo alterações e sem a necessidade de reiniciar o processo de aproximação da função valor-ação ao se verificar que o domínio sofreu mudanças.

1.2 CONTRIBUIÇÕES DESTE TRABALHO

Este trabalho contribui nos seguintes tópicos:

a) obtenção da descrição de um Processo Markoviano de Decisão estacionário a partir de *answer sets*;

b) redução do espaço de busca com a utilização desta descrição;

c) busca pela política ótima de um Processo Markoviano de Decisão não-estacionário utilizando métodos desenvolvidos para MDPs estacionários;

d) o algoritmo ASP(MDP) que pode ser utilizado tanto para MDPs estacionários quanto para não-estacionários, com horizonte finito ou infinito, não-determinísticos ou determinísticos, e duas variações: MCASE e ASP(RL).

1.3 ORGANIZAÇÃO DO TEXTO

Uma vez formalizado o problema que este trabalho procura solucionar, o próximo capítulo apresenta trabalhos relacionados ao problema estudado neste. A fim de facilitar a compreensão da proposta deste trabalho, a descrição dos métodos utilizados (capítulo 3) apresenta os fundamentos de tomada de decisão sequencial (seção 3.1), *Answer Set Programming* (seção 3.2) e a linguagem de ações $BC+$ (seção 3.3). A união destes três temas permitiu o desenvolvimento do ASP(MDP) (capítulo 4), um algoritmo capaz de tratar MDPs não-estacionários.

Para verificar o funcionamento desta proposta, foram realizados quatro experimentos (capítulo 5) que permitiram analisar o algoritmo proposto em cada um dos experimentos de forma independente (capítulo 6), além de características do próprio algoritmo que são independentes do domínio tratado (capítulo 7).

2 TRABALHOS CORRELATOS

O método ASP(MDP) descrito neste trabalho não é a primeira proposta de reunir *Answer Set Programming* com métodos de busca pela solução ótima de um MDP. Alguns métodos propõem a combinação de ASP diretamente com MDPs e RL (NICKLES, 2012; KHANDELWAL et al., 2014; YANG et al., 2014) enquanto outros métodos realizam alterações na descrição do programa lógico permitindo a busca pela solução ótima segundo algum critério (ZHANG; SRIDHARAN; WYATT, 2015; SRIDHARAN et al., 2015).

Uma combinação relacionada com a proposta deste trabalho foi feita por Otterlo (2008) com a combinação de lógica de primeira ordem com RL. A proposta de Otterlo (2008) utiliza lógica de primeira ordem para descrever os conjuntos de estados e ações e realizar a expansão dos átomos utilizados para descrever estes conjuntos para obter todas as possibilidades do domínio, permitindo que a otimização seja realizada através da interação com o ambiente utilizando um método de RL que aproxima a função $Q(s,a)$.

Desta forma, é possível obter generalizações que permitam encontrar a solução de variações do mesmo problema sem a necessidade de descrever todo o domínio novamente. A proposta de Otterlo (2008) realiza a descrição do MDP em programação lógica, mas não utiliza as soluções para descrever um novo MDP e também não realiza a revisão do conjunto de estados e ações e reaproveita a função Q aprendida no MDP anterior, ao contrário do que este trabalho propõe com o ASP(MDP).

A utilização de ASP e Q-Learning especificamente para encontrar a solução de um problema de tomada de decisão sequencial também foi proposta por Nickles (2012) no algoritmo chamado QASP (união de Q-Learning com ASP), em que um agente interage com o ambiente e, a partir do estado atual, gera *answer sets* que descrevem a política que pode ser seguida. No momento da seleção da ação que será executada no estado atual, ao agente só é permitido escolher uma das ações que aparecem em um dos *answer sets*. Desta forma, as ações que o agente executam estão no conjunto de ações que possivelmente levarão o agente ao estado final, de maneira próxima à utilização de ASP no ASP(MDP). A diferença entre QASP e ASP(MDP) é que, enquanto o primeiro gera os *answer sets* após cada ação executada, o segundo gera todos os *answer sets* possíveis antes do processo de aproximação da função valor-ação, permitindo também que mudanças na descrição do domínio sejam realizadas.

Seguindo a mesma linha do QASP, Khandelwal et al. (2014) e Yang et al. (2014) propõem a combinação de ASP (ou a linguagem de ações $\mathcal{BC}$) com a utilização de custo de realização de ações em cada estado como uma forma de encontrar a política ótima. Da mesma forma

que o QASP, *answer sets* são encontrados após a realização de cada ação e estes *answer sets* são percorridos para que o agente chegue ao estado final. Entretanto, a atualização dos custos de cada ação não é realizada como no Aprendizado por Reforço, mas de uma maneira proposta por Khandelwal et al. (2014) e Yang et al. (2014). Portanto, estes trabalhos se relacionam ao ASP(MDP) uma vez que utilizam ASP ou uma linguagem de ação junto com a escolha de ações a partir do custo da sua execução. Porém, ASP(MDP) permite a utilização de um método de aproximação da função valor-ação, enquanto que estes dois métodos possuem o custo de ação atualizado conforme um método diferente do utilizado por RL e MC e que foi proposto pelos autores para os respectivos trabalhos.

Uma outra forma de combinar ASP com RL é o ASP+POMDP (ZHANG; SRIDHARAN; WYATT, 2015; SRIDHARAN et al., 2015) em que ASP é utilizado para descrever o domínio gerando apenas um *answer set* na forma de política capaz de resolver o problema. Cada átomo deste *answer set* encontrado é descrito por um POMDP hierárquico que deve ter sua função valor-ação aproximada através da interação com o ambiente utilizando um método de RL. O ASP+POMDP se parece com o ASP(MDP) proposto neste trabalho por conta da combinação de ASP para descrever o domínio e RL como método para encontrar a solução ótima. Entretanto as propostas se diferenciam por conta da forma como é feita a combinação das duas técnicas. Enquanto ASP+POMDP utiliza POMDPs como descrição das ações, ASP(MDP) utiliza ASP como a própria descrição do MDP. Por causa destas diferenças, é possível realizar a combinação de ASP+POMDP com ASP(MDP), fazendo com que cada ação do ASP(MDP) seja um POMDP hierárquico como no ASP+POMDP.

Relacionado à restrição em MDPs que permite a diminuição do espaço de busca, tem-se o MDP saturado com restrições de caminhos (*saturated path-constrained* MDP – SPC-MDP) proposto por Sprauel, Teichteil-Königsbuch e Kolobov (2014). Este tipo de MDP tem como objetivo utilizar as restrições do problema para encontrar os caminhos permitidos para o MDP, fazendo com que apenas certas sequências de ações possam ser executadas. Utilizando SPC-MDP é possível descrever domínios em que uma certa ação deva ser obrigatoriamente executada antes de uma outra ação ou um certo estado visitado antes de outro. O SPC-MDP se relaciona com o ASP(MDP) pela utilização de restrições na descrição do domínio. Porém, o SPC-MDP é proposto com a utilização de programação dinâmica, não permitindo o aprendizado a partir da interação do ambiente, enquanto o ASP(MDP) utiliza a descrição do ambiente não somente para aproximar a função $Q(s,a)$, mas também para verificar se ocorreram modificações no domínio.

Um trabalho que também utilizou o domínio do ônibus espacial (que é descrito em detalhes na seção 5.2 e é utilizado para um dos experimentos) para testes é o P-Log que propõe

a descrição das probabilidades de falhas nos componentes utilizados por ônibus espaciais. O P-Log (GELFOND; RUSHTON, 2010; BARAL; GELFOND; RUSHTON, 2008) descreve a transição dos estados como uma rede Bayesiana permitindo que, ao seguir uma determinada política, a probabilidade de falhas dos componentes do ônibus espacial possa ser calculada. Desta forma, pode-se seguir o plano que tem a maior probabilidade de não falhar. O que diferencia o P-Log do ASP(MDP) é que, enquanto o P-Log utiliza apenas as probabilidades para encontrar um plano com baixa probabilidade de falha, ASP(MDP) utiliza a função de recompensa de um MDP para encontrar a solução ótima, considerando probabilidade de transição e custo de ação.

Dois exemplos que possuem uma proposta oposta à deste trabalho são as recentes implementações de métodos baseados no Aprendizado por Reforço realizada pelos pesquisadores do DeepMind (MNIH et al., 2015; SILVER et al., 2016). O *Deep Q-Network* (DQN) utilizado para os jogos de Atari permitiu a representação de um espaço de busca grande em que o aprendizado foi realizado a partir da interação com os jogos. Da mesma forma, a implementação utilizada para o AlphaGo permitiu o aprendizado em um dos maiores espaços de busca disponíveis (o espaço de busca do Go com tabuleiro de 19x19 é próximo à 250^{150}(SILVER et al., 2016), sendo o resultado do AlphaGo o suficiente para chegar ao nível de jogadores profissionais e inclusive vencer quatro das cinco partidas contra um dos melhores jogadores do mundo[1]. Apesar destes métodos considerarem o trabalho com grande espaços de busca, enquanto o ASP(MDP) procura encontrar uma descrição com um espaço menor de busca, o DQN e AlphaGo não conseguem resolver problemas de MDPs não-estacionários, ao contrário do ASP(MDP).

Esta seção apresentou trabalhos relacionados ao ASP(MDP) proposto neste trabalho. O próximo capítulo apresenta de forma mais detalhada os fundamentos deste trabalho.

[1]Os vídeos comentados dos jogos podem ser vistos no canal do DeepMind no YouTube (<https://www.youtube.com/channel/UCP7jMXSY2xbc3KCAE0MHQ-A>)

3 CONCEITOS FUNDAMENTAIS

Este capítulo apresenta os fundamentos necessários para o desenvolvimento do ASP(MDP), que utiliza métodos de tomada de decisão sequencial (como Monte Carlo e Aprendizado por Reforço) junto com *Answer Set Programming* de forma que um agente possa interagir com um ambiente. Ambiente neste trabalho é definido como o conjunto de estados de um problema de decisão.

3.1 TOMADA DE DECISÃO SEQUENCIAL

Problemas de tomada de decisão sequencial podem ser encontrados em diferentes áreas, como economia (WALD, 1947; WALD, 1949), pesquisa operacional e controle ótimo (BELLMAN, 1954; BELLMAN; DREYFUS, 1971) e planejamento (MAUSAM; KOLOBOV, 2012).

Um dos primeiros trabalhos nesta área é o de Wald (1947) sobre decisão sequencial estocástica em que um observador deve escolher qual é o melhor instante para realizar a amostragem de um problema uma vez que existe um risco associado à realização de cada nova observação.

O método de programação dinâmica (DP) proposto por Bellman (1954) apresenta uma forma de solucionar um problema de tomada de decisão sequencial que segue o mesmo princípio proposto em Wald (1947). Em DP, uma solução para o problema, chamada de política π, é constituída por uma sequência de transformações no ambiente (BELLMAN, 1954; BELLMAN; DREYFUS, 1971), podendo ter a forma de decisões sobre qual ação tomar (PUTERMAN, 2005) ou simplesmente uma sequência de ações (SUTTON; BARTO, 1998; SUTTON; BARTO, 2015).

O objetivo de solucionar um problema de decisão sequencial é encontrar a política ótima π^* que leva de um estado inicial a um estado objetivo. O Princípio da Optimalidade de Bellman que descreve uma política ótima é definido por Bellman e Dreyfus (1971) como:

Definição 1. Princípio da Optimalidade de Bellman (BELLMAN; DREYFUS, 1971): uma política ótima tem a propriedade de que, independente do estado inicial e da decisão inicial, as decisões seguintes constituem uma política ótima no que diz respeito ao estado resultante da decisão inicial.

Entretanto, a definição do conceito de ótimo para uma política pode ser definido de diversas formas, como por exemplo o retorno acumulado ao final da política ou a média de reforços recebidos durante sua execução (PUTERMAN, 2005).

A partir dos problemas de programação dinâmica, Bellman (1957) propõe uma formalização de problemas de tomada de decisão sequencial com estados que respeitem a propriedade de Markov de 1ª ordem. A propriedade de Markov de 1ª ordem afirma que é possível saber qual é o estado atual apenas com as informações do estado anterior e a ação executada (PUTERMAN, 2005; BOCHMAN, 2013). Este processo de decisão com estados Markovianos foi denominado como processo Markoviano de decisão:

Definição 2. Processo Markoviano de Decisão (MDP) (PUTERMAN, 2005; MAUSAM; KOLOBOV, 2012): Um MDP é formalizado por uma tupla $M = \langle \mathcal{T}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, em que:

a) $\mathcal{T} \mapsto [0,N]$: conjunto de valores finitos ou infinitos ($N = \infty$) representando os instantes de tempo (épocas) em que são realizadas observações e ações são executadas;

b) $\mathcal{S}$ é conjunto de estados que podem ser observados a cada instante de tempo $t \in \mathcal{T}$;

c) $\mathcal{A}$ é conjunto de ações $a \in \mathcal{A}$ que podem ser executadas em cada um dos estados $s \in \mathcal{S}$ em cada um dos instantes de tempo $t \in \mathcal{T}$;

d) $\mathcal{P} : \mathcal{T} \times \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0,1]$ é a função de transição que mapeia a probabilidade $p_t(s_t, a_t, s_{t+1})$de alcançar um estado futuro $s_{t+1} \in \mathcal{S}$ após a execução de uma ação $a_t \in \mathcal{A}$ no estado atual $s_t \in \mathcal{S}$;

e) $\mathcal{R} : \mathcal{T} \times \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ é a função de reforço que fornece um valor real $r_t(s_t,a,s_{t+1})$ ao alcançar o estado futuro $s_{t+1} \in \mathcal{S}$ como consequência da execução da ação $a_t \in \mathcal{A}$ no estado atual $s_t \in \mathcal{S}$ no instante de tempo $t \in \mathcal{T}$.

Apesar do conjunto de instantes de tempo $\mathcal{T}$ serem utilizados na definição de um MDP, alguns autores, como Sutton e Barto (2015) (assim como neste trabalho) omitem este conjunto sem nenhuma consequência. Uma segunda diferença entre alguns autores se encontra no conjunto de ações (BOUTILIER; DEAN; HANKS, 1999), em que é possível definir conjuntos independentes para cada estado ($\mathcal{A}_s \in \mathcal{A}$), como feito por Puterman (2005) ou apenas um conjunto de ações que podem ser utilizadas em todos os estados, como utilizado por Sutton e Barto (2015).

Inicialmente proposto por Bellman (1957), um MDP pode ser solucionado utilizando DP. Dada uma função valor $V(s)$, que associa um valor a cada estado $s \in \mathcal{S}$, ou uma função valor-ação $Q(s,a)$, que associa um valor para cada par estado-ação com estados $s \in \mathcal{S}$ e ações $a \in \mathcal{A}$, encontrar a solução de um MDP é realizar a atualização de $V(s)$ e/ou $Q(s,a)$ utilizando os valores de reforços obtidos a partir da função $\mathcal{R}$ em cada instante de interação

com o ambiente, procurando aproximar estas funções daquela que descreve a solução ótima do MDP. Entretanto, a forma como esta atualização é feita, qual função entre $V(s)$ e $Q(s,a)$ (ou até mesmo as duas em conjunto) será utilizada e qual o critério que será utilizado para determinar qual solução (política) é ótima, depende do método utilizado e do problema que se deseja resolver.

Para encontrar a solução de um MDP M utilizando DP é necessário o conhecimento de cada um dos componentes da tupla $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, de forma que nenhuma amostragem seja realizada a partir da interação com o ambiente. Apesar de DP ser um método largamente utilizado, caso ocorram alterações nas funções de transição ou de reforço, a função valor (valor-ação) aproximada não estará mais de acordo com o domínio. Porém, esta possível mudança no domínio não afeta os processos de Avaliação de Política de Monte Carlo ou o Aprendizado por Reforço.

O método de Monte Carlo (MC) foi inicialmente proposto por Metropolis et al. (1953) para a aproximação de funções conhecidas a partir da amostragem do ambiente. Segundo Sutton e Barto (2015) atualmente o nome de Monte Carlo é dado à algoritmos que apresentam algum componente aleatório em sua descrição.

Para a solução de um MDP utilizando MC é realizada a geração de políticas π com ações $a \in \mathcal{A}$ que são executadas no domínio até o final de um episódio. Para cada ação $a \in \pi$, o domínio fornece um reforço $r(s_t,a_t,s_{t+1})$ que é utilizado para a atualização da função valor (valor-ação) que deseja-se aproximar. Dada uma quantidade suficiente de amostragens, tem-se V (e/ou Q) que aproxima a função que descreve a solução ótima.

O algoritmo de Avaliação de Política de Monte Carlo (algoritmo 1 segue estes passos para realizar a aproximação de $V(s)$ ou $Q(s,a)$ utilizando políticas geradas aleatoriamente sem a necessidade de algum critério para escolher qual ação deve ser executada em cada estado. Porém, outros Métodos de Monte Carlo para MDPs (como o algoritmo de controle de Monte Carlo (SUTTON; BARTO, 2015) podem utilizar outras formas de selecionar uma ação para gerar a política.

Por conta da natureza do MC, a amostragem realizada no ambiente permite que, caso a função de transição e/ou de reforço sejam modificadas, a aproximação da função valor (valor-ação) continue refletindo o MDP pois não é necessário o conhecimento de $\mathcal{P}$ ou $\mathcal{R}$ para realizar a aproximação. Esta forma de interação direta com o ambiente para realizar a aproximação de V ou Q também é utilizada no RL.

O Aprendizado por Reforço (RL) proposto por Sutton (1988) utiliza o conceito de diferenças temporais (TD) para realizar a atualização da função valor (valor-ação). Desta forma,

Algoritmo 1 – Avaliação de Política de Monte Carlo

Inicialize:
$\pi \leftarrow$ uma política a ser avaliada
$Q \leftarrow$ uma função valor-ação arbitrária
$Retorno(s,a) \leftarrow$ uma lista vazia, $\forall s \in \mathcal{S}$ e $\forall a \in \mathcal{A}$

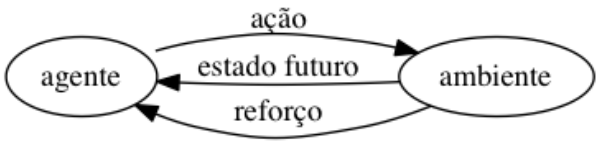
Repita para sempre:
  Gere um episódio utilizando π
  Para cada *par estado-ação que estiver no episódio* **faça:**
    $G \leftarrow$ retorne após a primeira ocorrência de s,a
    Acrescente G à $Retorno(s,a)$
    $Q(s,a) \leftarrow$ média de $(Retorno(s,a))$
  Fim
Fim

Fonte: Sutton e Barto, 2015

assim como MC, TD consegue realizar a aproximação de V ou Q sem o conhecimento prévio de todo o MDP e assim como DP, a atualização desta função utiliza valores já estimados, sem a necessidade de esperar até o final do episódio para realizar a atualização de V ou Q.

Um agente que aprende por reforço interage diretamente com o ambiente de forma a escolher e executar uma ação $a_t \in \mathcal{A}$ no estado atual $s_t \in \mathcal{S}$ em que se encontra. Após a realização desta ação, o agente observa o novo estado (futuro) $s_{t+1} \in \mathcal{S}$ e o reforço $r(s_t,a_t,s_{t+1})$ fornecido pelo ambiente, como mostra a figura 2

Figura 2 – Interação do agente com o ambiente para o RL



Fonte: Autor

A atualização da função valor-ação de um MDP utilizando TD se dá a partir do valor de reforço r recebido pela interação e a utilização do valor do próximo estado (ou par estado-ação) encontrado como retorno da interação com o ambiente. Desta forma, pode-se ter duas maneiras de atualizar a função conforme a escolha do valor futuro que será utilizado: *on-policy* e *off-policy*. Caso a aproximação seja dada por uma função valor $V(s)$, não há a separação entre *on-policy* e *off-policy* uma vez que as ações não são utilizadas para descrever a função que se deseja aproximar, conforme definido a seguir.

Em um método *on-policy*, como o SARSA (algoritmo 3, o agente que se encontra no estado s_t executa uma ação a_t no ambiente e recebe, além do reforço r, um estado futuro s_{t+1}. Para este estado futuro o agente escolhe uma nova ação a_{t+1} e a atualização do valor de $Q(s_t,a_t)$ utiliza os valores do reforço r e de $Q(s_{t+1},a_{t+1})$. Ou seja, em um método *on-policy* a atualização da função valor-ação depende da ação futura que se encontra na política que o agente está seguindo.

Para um método *off-policy*, como o Q-Learning (algoritmo 2, a atualização da função valor-ação também se dá pelo reforço r fornecido pelo ambiente, porém a ação a_{t+1} escolhida para o estado futuro s_{t+1} recebido é a ação que fornece o maior valor de $Q(s_{t+1},a_{t+1})$. Portanto, em um método *off-policy* são utilizados os valores r do reforço e $\max_{a_{t+1}} Q(s_{t+1},a_{t+1})$ para realizar a atualização da função valor-ação que deseja-se aproximar, ou seja, a atualização não depende da política que o agente está seguindo para interagir com o ambiente.

Esta diferença entre *on-policy* e *off-policy* pode ser observada entre as linhas 12 do algoritmo 3 e 10 do algoritmo 2.

O processo de aproximação da função valor-ação utilizando SARSA se dá por episódios que possuem início em um estado inicial e terminam em um estado final (não necessariamente um estado objetivo). Para cada instante de interação com o ambiente, o agente escolhe uma ação $a_t \in \mathcal{A}$ que pode ser executada no estado $s_t \in \mathcal{S}$. Com a execução de a_t no estado atual, o agente observa o reforço r e o estado s_{t+1} que é a consequência da execução de a_t em s_t. Para este novo estado s_{t+1} o agente escolhe uma nova ação s_{t+1} que será executada e a função $Q(s_t,a_t)$ é atualizada utilizando:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right], \tag{1}$$

em que α é a taxa de aprendizado e γ o desconto utilizado para a atualização. Em seguida, atualiza-se os valores dos estado e ação atual s_t e a_t respectivamente com o estado s_{t+1} observado e a ação a_{t+1} escolhida para ser executada. Este processo de seleção de ação, execução e atualização de Q é realizado até o fim de um episódio.

O Q-Learning realiza as mesmas etapas do SARSA, porém com as diferenças necessárias para ser um método *off-policy*. Para cada episódio, um agente que utiliza Q-Learning escolhe uma ação a ser executada. Com a execução desta ação, o agente observa o estado futuro s_{t+1} e o reforço r. Ao contrário do SARSA, no Q-Learning não é necessário realizar a escolha de uma ação para este novo estado s_{t+1} e a atualização da função valor-ação é realizada

Algoritmo 2 – Algoritmo Q-Learning

Entrada: $Q(s,a)$ a ser aproximada
Saída: $Q(s,a)$ aproximada
Inicialize: *Uma função* $Q(s,a)$ *de forma arbitrária*
Repita para cada episódio:
  Inicialize: s_t
  Repita para cada passo no episódio:
    Escolha a_t para s_t utilizando uma política derivada de Q
    Execute a ação a_t
    Observe o estado futuro s_{t+1} e o reforço r
    Atualize a função $Q(s,a)$ utilizando:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right]$$

    $s_t \leftarrow s_{t+1}$
  até s *ser terminal*

Fim

Fonte: Sutton e Barto, 2015

Algoritmo 3 – Algoritmo SARSA

Entrada: $Q(s,a)$ a ser aproximada
Saída: $Q(s,a)$ aproximada
Inicialize: *Uma função* $Q(s,a)$ *de forma arbitrária*
Repita para cada episódio:
  Inicialize: s_t
  Escolha a_t para s_t utilizando uma política derivada de Q
  Repita para cada passo no episódio:
    Execute a ação a_t
    Observe o estado futuro s_{t+1} e o reforço r
    Escolha a_{t+1} para s_{t+1} utilizando uma política derivada de Q
    Atualize a função $Q(s,a)$ utilizando:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right]$$

    $s_t \leftarrow s_{t+1}$
    $a_t \leftarrow a_{t+1}$
  até s *ser terminal*
Fim

Fonte: Sutton e Barto, 2015

segundo:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right], \tag{2}$$

com α e γ sendo a taxa de aprendizado e o desconto, respectivamente. Após a atualização, o estado atual s_t é atualizado com o valor do estado futuro s_{t+1} fornecido pelo ambiente e o processo é repetido até o fim do episódio.

A escolha da ação a ser executada durante um processo de RL pode ser realizada de diversas maneiras. O método escolhido para os experimentos deste trabalho é o ϵ-*greedy*, em que tem-se a probabilidade ϵ de se escolher uma das ações disponíveis de forma aleatória e uma probabilidade $1 - \epsilon$ de escolher a ação que possui o maior valor para aquele estado. Ou seja, em um determinado estado s_t e com um número gerado de forma aleatória ρ, a seleção de uma ação por ϵ-*greedy* é feita segundo a equação:

$$a_t = \begin{cases} a_{aleatória}, \text{ se } \rho \leq \epsilon; \\ \max_a Q(s_t,a), \text{ caso contrário} \end{cases} \tag{3}$$

Apesar de métodos como MC e RL permitirem que a aproximação da função valoração seja feita apesar de possíveis alterações nas funções de $\mathcal{P}$ e $\mathcal{R}$, estes mesmos métodos não permitem que os conjuntos de estados e ações sejam alterados ao longo do tempo.*Answer Set Programming* provê um método que permite que as definições de estados e ações sejam alteradas ao longo do tempo.

3.2 ANSWER SET PROGRAMMING

Answer Set Programming (ASP) é uma forma de programação declarativa utilizada para a representação de conhecimento e raciocínio capaz de resolver problemas do tipo NP-hard (GELFOND, 2008; LIFSCHITZ, 2008). Um programa escrito na linguagem ASP utiliza semântica de modelos estáveis ou de *answer sets*.

Um átomo em linguagem ASP é definido como $\mathtt{p}$ e sua negação como $-\mathtt{p}$, sendo um átomo ou sua negação um literal. Uma regra em ASP tem a forma:

$$H \leftarrow B, \tag{4}$$

em que H é a cabeça da regra e é formada por literais $L_0 \, or \, L_1 \, or \, \ldots \, or \, L_k$ e B é o corpo da regra, formado por literais $L_{k+1}, \ldots, L_m, not \, L_{m+1}, \ldots, not \, L_n$. Portanto, a formula 4 pode ser

escrita como:

$$L_0 \, or \, L_1 \, or \, \ldots \, or \, L_k \leftarrow L_{k+1}, \ldots, L_m, not \, L_{m+1}, \ldots, not \, L_n. \quad (5)$$

Neste caso, o conectivo or indica disjunção, a vírgula representa a conjunção e not a negação por falha (GELFOND, 2008). Uma regra que possui somente a cabeça H é chamada de fato e uma regra que não possui cabeça, mas somente o corpo B, é chamada de restrição.

Dada a natureza da descrição em ASP, a linguagem aceita dois tipos de negação: a negação forte (ou clássica) e a negação fraca (negação por falha). Para a utilização em um programa lógico Π em ASP, a negação forte é descrita utilizando "$-$" (símbolo de menos), enquanto que a negação fraca utiliza o "`not`".

Um exemplo dado por John McCarthy e apresentado por Lifschitz (2008) para descrever a diferença entre estes tipos de negação versa sobre um ônibus que tem que atravessar uma linha de trem quando nenhum trem está se aproximando. A regra (utilizando a linguagem de ASP) `atravessar :- not trem` utiliza a negação fraca, dizendo que é permitido atravessar quando é desconhecida qualquer informação sobre o trem (negação por falha). Entretanto, a regra `atravessar :- -trem` utiliza a negação forte e indica que é permitido atravessar quando não há trem.

Um programa lógico Π é um conjunto de regras na forma da fórmula 4 e uma interpretação para este programa é um conjunto de átomos que faz com que as regras de Π sejam verdadeiras. A partir de um conjunto de átomos A de Π, seja um programa lógico Π_A obtido a partir da:

a) remoção de todas as regras que possuem negação em seu corpo B;

b) remoção de todos os literais com negação por falha nas fórmulas restantes.

Se A é um dos modelo minimais (uma interpretação que não contém outras interpretações) de Π_A, então A é um modelo estável de Π. Quando este modelo estável aceita a utilização da negação forte em seus átomos, ele é chamado de *answer set*. Segundo Gelfond e Lifschitz (1988), modelos estáveis podem ser descritos como "possíveis conjuntos de crenças em que um agente pode acreditar, dada as premissas de Π".

Encontrar os *answer sets* ou modelos estáveis de um programa lógico Π é realizar os processos de *grounding* e *solving* neste programa lógico, obtendo modelos minimais deste programa lógico Π.

A transformação de Π a partir da substituição de suas variáveis por átomos de forma a gerar um programa que não possui variáveis ou funções aritméticas é chamada de *grounding* e o

programa lógico resultante desta transformação é $gr(\Pi)$. A busca pelos *answer sets* de Π é feita após encontrar $gr(\Pi)$ através da busca pelas soluções de $gr(\Pi)$, sendo este segundo processo chamado de *solving*.

Por exemplo, dado o programa lógico apresentado na figura 3, o processo de *grounding* transforma este programa no da figura 4. A partir do resultado apresentado na figura 4, pode ser realizado o *solving*, tendo como resposta a saída apresentada na figura 5. O resultado do *solving* que se encontra na figura 5 é a saída apresentada pelo programa *Clasp* (GEBSER; KAMINSKI; KAUFMANN, 2013).

Figura 3 – exemplo de programa lógico

```
p(a). p(b). p(c).
q(x) :- p(x).
2 {r(x) : p(x)}.
```

Fonte: Lifschitz, 2008

Figura 4 – resultado do *grounding* do exemplo da figura 3

```
p(a). p(b). p(c).
q(a) :- p(a).
q(b) :- p(b).
q(c) :- p(c).
2 {r(a), r(b), r(c)}.
```

Fonte: Lifschitz, 2008

Figura 5 – resultado de *solving* do exemplo da figura 4 utilizando *Clasp*

```
solving...
answer: 1
p(a) p(b) p(c) q(a) q(b) q(c) r(b) r(c)
answer: 2
p(a) p(b) p(c) q(a) q(b) q(c) r(a) r(b) r(c)
answer: 3
p(a) p(b) p(c) q(a) q(b) q(c) r(a) r(c)
answer: 4
p(a) p(b) p(c) q(a) q(b) q(c) r(a) r(b)
satisfiable
```

Fonte: Autor

Um resolvedor ASP ou ASP *engine* é um programa (ou a união de dois programas) que realiza os processos de *grounding* e *solving*, tendo como entrada Π e saída os *answer sets*. Para um resolvedor ASP que são fornecidos como dois programas separados (um *grounder* e um *solver*), é possível trocar partes da resolvedor ASP por outros programas que realizam o mesmo processo. Um exemplo de resolvedor ASP que permanece sendo desenvolvido por Gebser et

al. (2011) são os programas *Gringo* e *Clasp* que são *grounder* e *solver* respectivamente e a união destes em apenas um programa, o *Clingo*. Um outro resolvedor ASP é a combinação dos programas *lparse* e *smodels*[1].

É importante ressaltar que a linguagem aceita por um resolvedor ASP não era padronizada inicialmente, permitindo que diferentes desenvolvedores implementassem seus *grounders* e *solvers* com sintaxes diferentes. A padronização desta linguagem ainda está sendo discutida pelo *ASP Standardization Working Group*[2] e já apresenta um documento com uma versão da linguagem que devem aceitar[3].

ASP também permite a descrição de problemas de planejamento (LIFSCHITZ, 1999; EITER et al., 2002; YANG et al., 2014) e pode ser traduzida diretamente de linguagens de ações, como por exemplo a linguagem $BC+$ (BABB; LEE, 2015b; BABB; LEE, 2015a).

3.3 A LINGUAGEM DE AÇÕES $BC+$

Uma linguagem de ações é um modelo formal de partes da linguagem natural que pode ser utilizado para descrever efeitos de ações (GELFOND; LIFSCHITZ, 1998). Estas linguagens apresentam uma sintaxe que define um sistema de transição (GELFOND; LIFSCHITZ, 1998; BABB; LEE, 2015a), permitindo que, em algumas linguagens, este sistema de transição seja traduzido diretamente para ASP. A vantagem da utilização de uma linguagem de ações em relação ao ASP puro na descrição de um sistema de transição é que linguagens de ações possuem uma sintaxe de mais alto nível, permitindo que o problema seja descrito de forma mais próxima à linguagem natural (GELFOND; LIFSCHITZ, 1998; BABB; LEE, 2015a).

Uma tentativa de solucionar certas limitações de linguagens de ações foi proposta por Babb e Lee (2015b, 2015a) na forma da linguagem $BC+$, que utiliza notações de alto nível para fórmulas proposicionais definidas sobre a semântica de modelos estáveis, da mesma forma que a linguagem $\mathcal{C}+$.

Em $\mathcal{C}+$ (e, portanto, $BC+$), uma fórmula é descrita da mesma maneira que uma fórmula proposicional, porém mais genérica, em que seus átomos podem ser igualdades assim como as igualdades encontradas em problemas de satisfação de restrição (GIUNCHIGLIA et al., 2004). Uma assinatura em $\mathcal{C}+$ é um conjunto de símbolos σ chamados de constantes. Um domínio para

[1] Os programas *lparse* e *smodels* podem ser obtidos em <http://www.tcs.hut.fi/Software/smodels/>.

[2] A descrição do projeto pode ser encontrada em <http://www.kr.tuwien.ac.at/research/projects/WASP/index.html>

[3] O documento que apresenta esta padronização pode ser obtido em <https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03c.pdf>.

constantes $c \in \sigma$ é chamado de $Dom(c)$, sendo que para cada átomo c na assinatura σ existe uma expressão $c = v$ em que $v \in Dom(c)$. Portanto, uma fórmula de σ é uma combinação proposicional de átomos (GIUNCHIGLIA et al., 2004) e uma regra é uma expressão na forma $\mathrm{F} \Leftarrow \mathrm{G}$ para fórmulas F e G de σ.

Uma interpretação I de σ satisfaz átomos $c = v$ mapeando para todos os elementos $c \in \sigma$ um valor $v \in Dom(c)$. Assim como em ASP, um modelo para um conjunto de fórmulas Π é uma interpretação que satisfaz Π. Se Π possui um modelo, então este conjunto de fórmulas é dito consistente ou satisfazível.

Em $\mathcal{C}+$ e $BC+$, uma constante pode assumir dois tipos: fluente (constantes que são parâmetros que definem um estado) ou ações (constantes que caracterizam um evento que leva de um estado para outro). Uma constante de fluente pode ser definida de duas formas: constante regular (que possui seu valor determinado por resultados de ações executadas em outros estados) e constante determinada estaticamente (o valor do fluente é determinado por outros fluentes no mesmo estado) (GIUNCHIGLIA et al., 2004; BABB; LEE, 2015c).

Da mesma maneira, uma fórmula é definida sobre as contantes que a constitui, sendo que uma fórmula de fluente possui apenas constantes de fluente (descrição de estado) e uma fórmula de ação não possui constante de fluente e, no mínimo, uma constante de ação.

Dadas as fórmulas fluentes F e G, uma regra estática pode ser utilizada para descrever dependências entre fluentes no mesmo estado e possui a forma:

$$\text{causado F se G.} \tag{6}$$

Caso F seja uma fórmula de ação (que possui pelo menos uma constante de ação e nenhuma constante de fluente) e G uma fórmula, então a fórmula 6 é uma regra dinâmica de ação. A forma

$$\text{causado F se G depois de H} \tag{7}$$

é uma regra dinâmica de fluente caso F e G sejam fórmulas de fluente e H seja uma fórmula.

Uma descrição de ações em $BC+$ é um conjunto de regras causais sendo estas definidas como regras estáticas, regras dinâmicas de ações ou regras dinâmicas de fluente.

A partir de uma descrição de ações $\mathcal{D}$ de uma assinatura σ, um sistema de transição em $\mathcal{D}$ é descrito por um modelo estável de uma sequência de fórmulas proposicionais $PF_0(\mathcal{D})$, $PF_1(\mathcal{D})$, ..., de forma que uma assinatura σ_t de $PF_t(\mathcal{D})$ consiste de átomos na forma $i : c = v$ em que:

a) para cada constante de fluente c de $\mathcal{D}$, $i \in (0, \ldots, t)$ e $v \in Dom(c)$ (em que $Dom(c)$ representa o domínio com todos os valores possíveis de c);

b) para cada constante de ação c de $\mathcal{D}$, $i \in (0, \ldots, t-1)$ e $v \in Dom(c)$.

Dada uma constante (ou conjunto) de fórmula F, o resultado de inserir i para cada ocorrência de F resulta em $i : \mathrm{F}$. Desta forma, $PF_t(\mathcal{D})$ é a conjunção de:

a) para todo $i \in \{0, \ldots, t\}$ com cada regra estática em $\mathcal{D}$ e para cada regra dinâmica de ação em $\mathcal{D}$ com $i \in \{0, \ldots, t\}$,

$$i : \mathrm{F} \leftarrow i : \mathrm{G};$$

b) para cada regra dinâmica de fluente em $\mathcal{D}$ com $i \in \{0, \ldots, t\}$,

$$i + 1 : \mathrm{F} \leftarrow (i + 1 : \mathrm{G}) \wedge (i : \mathrm{H});$$

c) para cada constante regular de fluente c e todo $v \in Dom(c)$,

$$\{0 : c = v\};$$

d) dado $Dom(c)$ como $\{v_1, \ldots, v_t\}$, o agregado de contagem

$$\bot \leftarrow -(1 \leq \{i : c = v_1, \ldots, i : c = v_t\} \leq 1).$$

Considerando os subconjuntos σ^{fl} e σ^{ac} de σ que são constituídos, respectivamente, pelos átomos de constante de fluente e átomos de constante de ações, uma interpretação de σ_t com os átomos que possuem valor verdadeiro e representam um caminho na forma

$$(0 : s_0) \cup (0 : a_0) \cup (1 : s_1) \cup (1 : a_1) \cup \ldots \cup (t : s_t), \tag{8}$$

em que $s_0, s_1, \ldots, s_t$ são interpretações de σ^{fl} e $a_0, a_1, \ldots, a_{t-1}$ são interpretações de σ^{ac}.

Dada uma descrição $\mathcal{D}$, estados e transições podem ser definidos a partir de modelos estáveis de $PF(\mathcal{D})$. Um estado s é uma interpretação de σ^{fl}, tornando a forma $0 : s$ um modelo estável de $PF_0(\mathcal{D})$. Uma transição é descrita por uma tripla $\langle s_{t-1}, a_{t-1}, s_t \rangle$ em que s_{t-1} e s_t são interpretações de σ^{fl} e a_{t-1} uma interpretação de σ^{ac}, fazendo com que $0 : s_0 \cup 0 : a_0 \cup s_1$ seja um modelo estável de $PF_1(\mathcal{D})$.

A partir destas definições, Babb e Lee (2015a) provam que para todo $t \geq 1$, X_t é um modelo estável de $PF_t(\mathcal{D})$ se e somente se $X^0, \ldots, X^1$ são transições de $\mathcal{D}$, sendo X_m um conjunto de elementos da assinatura σ_m e X_m^i com $i < t$ definido pela tripla:

a) um conjunto de átomos A que contém constantes de fluente de forma que $i : A \in X_m$;

b) um conjunto de átomos A de constantes de ações com $i : A \in X_m$;

c) um conjunto de átomos A de constantes de fluente de forma que $(i + 1) : A \in X_m$.

Um programa escrito utilizando $BC+$ pode ser traduzido para diretamente para ASP utilizando o *CPLUS2ASP* (BABB; LEE, 2013), permitindo que as soluções para um problema descrito em $BC+$ sejam encontradas utilizando *grounders* e *solvers* para ASP.

O programa apresentado na figura 6 contém um exemplo de implementação de um mundo de grades determinístico. As linhas logo após `:- macros`, definem as variáveis utilizadas para o ambiente (neste caso, as dimensões do mundo de grades, a posição do objetivo e o número máximo de passos que devem ser utilizados para resolver o problema). Em seguida são definidas as constantes do mundo de grade (linha `:- constants`), que contém as variáveis que descrevem um estado – constantes de fluente – (`x` e `y`) na grade, junto com suas dimensões, e as quatro ações permitidas – constantes de ação – (`up`, `down`, `left` e `right`). As linhas seguintes definem as posições possíveis no mundo de grades utilizando as variáveis `X` e `Y`. A seguir, a transição determinística (regra dinâmica de fluente) das ações é descrita para cada uma das 4 ações permitidas, seguido pelo resultado da execução destas ações ao chegar a um limite da grade. A linha `noconcurrency` indica ao $BC+$ que apenas uma ação deve ser realizada por vez. A seguir, é descrito o que se deseja encontrar em `:- query`. A primeira linha após `:- query` apresenta o nome que será passado ao *CPLUS2ASP* para realizar a busca. A linha `maxstep :: 0..MAX_STEP` indica que deseja-se políticas que possuem no mínimo 0 passos e no máximo 1000 passos (como a macro `MAX_STEP` foi definida). Em `0: x=MIN_X, y=MIN_Y` é definido o estado inicial (no passo 0) e `maxstep: x=GOAL_X, y=GOAL_Y` define o estado objetivo que deve ser alcançado entre 0 e 1000 passos. As últimas quatro linhas estão comentadas e definem os arquivos que são incluídos (`:- include`) para obter as outras três configurações do mundo de grades que serão utilizadas no terceiro experimento. Os códigos para estas três linhas que estão comentadas encontram-se no anexo A.

A solução encontrada para o programa da figura 6 pelo *CPLUS2ASP* encontra-se na figura 7 (linhas foram omitidas por conta da grande quantidade de soluções). Pode-se notar que a saída do *CPLUS2ASP* para um programa em $BC+$ é dada pelo estado atual seguida da ação executada, até que o estado objetivo seja encontrado. As últimas linhas apresentam, que

Figura 6 – Descrição do mundo de grades em $BC+$

```
% gridworld variables
% world size and objective
:- macros
 MAX_X -> 10;
 MAX_Y -> 10;
 MIN_X -> 1;
 MIN_Y -> 1;
 GOAL_X -> MAX_X;
 GOAL_Y -> MAX_Y;
 MAX_STEP -> 1000.

:- constants
x :: inertialFluent(MIN_X..MAX_X);
y :: inertialFluent(MIN_Y..MAX_Y);
up, down, left, right :: exogenousAction.

:- variables
 X :: MIN_X..MAX_X-1;
 Y :: MIN_Y..MAX_Y-1.

% action transitions
up causes y=Y+1 if y=Y.
down causes y=Y-1 if y=Y.
right causes x=X+1 if x=X.
left causes x=X-1 if x=X.

% world limits
nonexecutable up if y=MAX_Y.
nonexecutable down if y=MIN_Y.
nonexecutable right if x=MAX_X.
nonexecutable left if x=MIN_X.

% only one action at a time
noconcurrency.

:- query
 label :: policies;
 maxstep :: 0..MAX_STEP;
 0: x=MIN_X, y=MIN_Y;
 maxstep: x=GOAL_X, y=GOAL_Y.

% what should be learned
%:- include walls1.
%:- include walls2.
%:- include walls3.
```

Fonte: Autor

são exibidas após as soluções, apresentam se o programa possui solução (`SATISFIABLE`), a quantidade de soluções encontradas para o programa, o número de passos necessários para as soluções encontradas e os tempos de execução, preparação e pré-processamento (tradução de $BC+$ para ASP).

Este capítulo apresentou os fundamentos necessários para o desenvolvimento do ASP(MDP), que é descrito no próximo capítulo.

Figura 7 – Resultado fornecido pelo *CPLUS2ASP* para o programa apresentado na figura 6

```
% Pragma: Running Query 'Query 'planning''.
% Pragma: Minimum Step = '0', Maximum Step = '100'.
% Pragma: Searching for all solutions.
Solution: 1
      0: x=1 y=1
      ACTIONS: right
      1: x=2 y=1
      ACTIONS: right
      2: x=3 y=1
      ACTIONS: right
      3: x=4 y=1
      ACTIONS: right
      4: x=5 y=1
      ACTIONS: up
      5: x=5 y=2
      ACTIONS: up
      6: x=5 y=3
      ACTIONS: right
      7: x=6 y=3
      ACTIONS: right
      8: x=7 y=3
      ACTIONS: up
      9: x=7 y=4
      ACTIONS: right
      10: x=8 y=4
      ACTIONS: right
      11: x=9 y=4
      ACTIONS: up
      12: x=9 y=5
      ACTIONS: up
      13: x=9 y=6
      ACTIONS: up
      14: x=9 y=7
      ACTIONS: up
      15: x=9 y=8
      ACTIONS: right
      16: x=10 y=8
      ACTIONS: up
      17: x=10 y=9
      ACTIONS: up
      18: x=10 y=10

...

SATISFIABLE
Models : 48620
Total Steps : 18
Time : 4.810
Prepare : 0.040
Prepro. : 0.000
```

Fonte: Autor

4 ASP(MDP)

Após a apresentação dos conceitos fundamentais (capítulo 3), este capítulo apresenta o algoritmo ASP(MDP), proposto neste trabalho.

4.1 MDP M DESCRITO EM UM PROGRAMA LÓGICO $\Pi(\mathcal{S},\mathcal{A})$

Dado um MDP $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ em que a função $\mathcal{R}$ de reforço é desconhecida pelo agente e será fornecida pelo ambiente, o agente tem o conhecimento dos estados em $\mathcal{S}$, das ações em $\mathcal{A}$ e da função de transição $\mathcal{P}$. Portanto, se torna possível uma descrição do ambiente utilizando uma linguagem de ações como a $BC+$, porém sem a possibilidade de encontrar a política ótima por conta da falta de conhecimento da função de reforço $\mathcal{R}$.

Como descrito por Babb e Lee (2015a), um estado pode ser descrito em $BC+$ como um conjunto de constante de fluentes possibilitando que o conjunto de estados $\mathcal{S}$ possa ser diretamente traduzido para a linguagem. Desta forma, todos os estados contidos em $\mathcal{S}$ também fazem parte do programa lógico $\Pi(\mathcal{S},\mathcal{A})$, permitindo que a busca por *answer sets* percorra todo o espaço de estados.

Da mesma forma, as ações $a \in \mathcal{A}$ são descritas como constantes de ação em $BC+$, permitindo que estas representem as ações com as mesmas consequências que ocorreria no ambiente utilizando a função de transição $\mathcal{P}$. Sendo que a função $\mathcal{P}$ de transição é representada em $BC+$ pelas regras no sistema de transição $\mathcal{D}$.

Portanto, tendo os mesmos estados e ações e produzindo a mesma função de transição, $\Pi(\mathcal{S},\mathcal{A})$ consegue representar parte de M, porém, uma vez que a função de recompensa $\mathcal{R}$ não é descrita em $BC+$, não é possível realizar as atualizações na função valor-ação para obter a política ótima. Desta forma, ao encontrar as soluções para o programa lógico $\Pi(\mathcal{S},\mathcal{A})$ tem-se as políticas possíveis que resolvem o MDP descrito por M.

4.2 CONJUNTO DE ESTADOS CONTIDOS EM TRAJETÓRIAS

Dados uma descrição de ações $\mathcal{D}$ (como descrito na seção 3.3), um conjunto dos possíveis estados iniciais $\mathcal{I}$ e um conjunto de estados objetivos $\mathcal{G}$, um problema descrito por $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ em que o objetivo é encontrar a sequência de ações que levam dos possíveis estados iniciais $s_0 \in \mathcal{I}$ aos estados objetivos $s_g \in \mathcal{G}$, respeitando as transições em $\mathcal{D}$, tem como resposta uma

trajetória T que possui a forma:

$$T = \langle\langle s_0, a_0, s_1\rangle, \langle s_1, a_1, s_2\rangle, \ldots, \langle s_n, a_n, s_g\rangle\rangle, \tag{9}$$

em que s_t representa o estado $s \in \mathcal{S}$ observado no instante t e a_t representa a ação $a \in \mathcal{A}$ executada neste estado neste mesmo instante.

Sendo $\Pi(\mathcal{S},\mathcal{A})$ um programa lógico que descreve $\langle\mathcal{D},\mathcal{I},\mathcal{G}\rangle$, um *answer set* para $\Pi(\mathcal{S},\mathcal{A})$ é uma trajetória T para $\langle\mathcal{D},\mathcal{I},\mathcal{G}\rangle$. A união de todos as trajetórias T obtidas para $\Pi(\mathcal{S},\mathcal{A})$ constitui o conjunto $\mathcal{H}$ que possui os estados $s \in \mathcal{S}$ que o agente pode visitar para chegar ao objetivo e as ações $a \in \mathcal{A}$ que podem ser realizadas em cada um destes estados.

4.3 DESCRIÇÃO DE UM NOVO MDP $\dot{M}$

Considerando o conjunto $\mathcal{H}$ que contém todas as histórias h obtidas pela resolução de $\langle\mathcal{D},\mathcal{I},\mathcal{G}\rangle$, uma história h descreve um subconjunto de estados de $\mathcal{S}$ e de ações de $\mathcal{A}$ e das transições em $\mathcal{P}$ do MDP $M = \langle\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}\rangle$. Portanto, a partir de $\mathcal{H}$ é possível obter um MDP $\dot{M} = \langle\dot{\mathcal{S}},\dot{\mathcal{A}},\dot{\mathcal{P}},\mathcal{R}\rangle$ de forma que $\dot{M} \subseteq M$

Lemma 1. *O conjunto $\mathcal{H}$ de todas trajetórias restritas pelo programa $\Pi(\mathcal{S},\mathcal{A})$ define um MDP $\dot{M} = \langle\dot{\mathcal{S}},\dot{\mathcal{A}},\dot{\mathcal{P}},\mathcal{R}\rangle$ de forma que $\dot{M} \subseteq M$, com $M = \langle\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}\rangle$.*

Demonstração. Dado um MDP $M = \langle\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}\rangle$ descrito em um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ que possui histórias $h \in \mathcal{H}$ que são as soluções de M, as restrições em $\Pi(\mathcal{S},\mathcal{A})$ definem um conjunto $\overline{\mathcal{S}}$ de estados que o agente não pode visitar e um conjunto $\overline{\mathcal{A}}$ de ações que o agente não pode executar.

Por conta destas restrições tem-se um conjunto de estados $\dot{\mathcal{S}} \subseteq \mathcal{S} \,|\, \dot{\mathcal{S}} = \mathcal{S} - \overline{\mathcal{S}}$ formado pelos estados que podem ser visitados. Da mesma forma, tem-se um conjunto de ações $\dot{\mathcal{A}} \subseteq \mathcal{A} \,|\, \dot{\mathcal{A}} = \mathcal{A} - \overline{\mathcal{A}}$ que são possíveis de serem executadas. Desta forma, a remoção de estados e ações não afetam as soluções do MDP M uma vez que estas não levam à solução desejada.

Uma história $h \in \mathcal{H}$ para M percorre uma sequência de estados e ações de forma que o estado inicial de h é $s_0 \in \mathcal{S}$ e o estado final é $s_g \in \mathcal{S}$. Dado que $\dot{\mathcal{S}} \subseteq \mathcal{S}$ e $\dot{\mathcal{A}} \subseteq \mathcal{A}$, tem-se que $s_0 \in \dot{\mathcal{S}}$ e $s_g \in \dot{\mathcal{S}}$. Da mesma forma, $\forall s \in h,\ s \in \dot{\mathcal{S}}$ e $\forall a \in h,\ a \in \dot{\mathcal{A}}$ para cada história $h \in \mathcal{H}$. Portanto, é possível extrair $\dot{\mathcal{S}}$ e $\dot{\mathcal{A}}$ a partir do conjunto de trajetórias $\mathcal{H}$ do MDP M.

Dado o conjunto de trajetórias $\mathcal{H}$, também é possível extrair a função de transição $\dot{\mathcal{P}}(\dot{\mathcal{S}},\dot{\mathcal{A}})$ a partir de cada trajetória $h \in \mathcal{H}$ de forma que, a partir das restrições contidas em $\Pi(\mathcal{S},\mathcal{A})$, tem-se:

a) um agente não pode executar ações em um estado que não pode visitar: $\overline{\mathcal{S}} \times \mathcal{A}$;

b) um agente não pode executar ações restritas em $\Pi(\mathcal{S},\mathcal{A})$ nos estados que pode visitar: $\mathcal{S} \times \overline{\mathcal{A}}$;

c) não é permitido a execução de uma ação proibida em um estado que o agente não pode visitar: $\overline{\mathcal{S}} \times \overline{\mathcal{A}}$;

d) não é possível visitar um estado que não possua transição para ele: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto 0$;

e) não é permitido executar certas ações em certos estados: $\overline{\mathcal{A}_{\mathcal{S}}}$.

Consequentemente, a nova função de transição $\dot{\mathcal{P}}$, dada as restrições em $\Pi(\mathcal{S},\mathcal{A})$ é:

$$\dot{\mathcal{P}}(\dot{\mathcal{S}},\dot{\mathcal{A}}) = \Big((\mathcal{S} \times \mathcal{A}) - (\overline{\mathcal{S}} \times \mathcal{A}) - (\mathcal{S} \times \overline{\mathcal{A}}) - (\overline{\mathcal{S}} \times \overline{\mathcal{A}}) - \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto 0) - \overline{\mathcal{A}_{\mathcal{S}}}\Big) \mapsto \dot{\mathcal{S}}. \tag{10}$$

Portanto, a partir do conjunto de histórias $\mathcal{H}$ do MDP M é possível extrair um conjunto de estados $\dot{\mathcal{S}} \subseteq \mathcal{S}$, um conjunto de ações $\dot{\mathcal{A}} \subseteq \mathcal{A}$ e a função de transição $\dot{\mathcal{P}} \subseteq \mathcal{P}$. Utilizando estes três conjuntos e a função de recompensa $\mathcal{R}$ definida para M, tem-se $\dot{M} = \langle \dot{\mathcal{S}}, \dot{\mathcal{A}}, \dot{\mathcal{P}}, \mathcal{R} \rangle$ de forma que $\dot{M} \subseteq M$ dada as restrições em $\Pi(\mathcal{S},\mathcal{A})$. □

Uma vez que o MDP $\dot{M} = \langle \dot{\mathcal{S}}, \dot{\mathcal{A}}, \dot{\mathcal{P}}, \mathcal{R} \rangle$ é conhecido, é necessário garantir que a política ótima π^*_M de $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ seja a mesma política ótima $\pi^*_{\dot{M}}$ de $\dot{M}$.

4.4 SOLUÇÃO DO MDP $\dot{M}$ E OPTIMALIDADE

A partir dos lemas anteriores, sabe-se que o MDP $\dot{M}$ possui as mesmas políticas possíveis de M, sendo que a função de reforço $\mathcal{R}$ não sofre alterações entre M e $\dot{M}$. Entretanto, ainda é necessário garantir que a política ótima $\pi^*_{\dot{M}}$ de $\dot{M}$ seja a mesma política ótima π^*_M de M.

Teorema 1. A política ótima π^*_M do MDP M é equivalente à política ótima $\pi^*_{\dot{M}}$ do MDP $\dot{M} \subseteq M$ descrito a partir das histórias $h \in \mathcal{H}$ que são soluções de M restritas pelo programa lógico $\Pi(\mathcal{S},\mathcal{A})$.

Demonstração. Dada as políticas ótimas $\pi^*_{\dot{M}}$ para o MDP $\dot{M}$, π^*_M para o MDP M, caso $\Pi(\mathcal{S},\mathcal{A})$ seja um programa lógico que não possui restrições, se $\pi^*_{\dot{M}} \neq \pi^*_M$, $\dot{M}$ e M são obrigatoriamente diferentes e, portanto, não descrevem o mesmo problema.

Caso existam restrições em $\Pi(\mathcal{S},\mathcal{A})$, se $\pi^*_{\dot{M}} \neq \pi^*_M$, as restrições existentes em $\Pi(\mathcal{S},\mathcal{A})$ removem estados e ações que fazem parte da solução ótima e, portanto, $\dot{M} \neq M$. □

Portanto, é possível obter um MDP $\dot{M} \subseteq M$ que possui a mesma política ótima que M, de forma que encontrar a solução ótima de $\dot{M}$ fornecerá a solução de M sem que seja perdida nenhuma propriedade do domínio ou das políticas.

4.5 MUDANÇAS NÃO PREVISTAS NO DOMÍNIO AO LONGO DO TEMPO

Um MDP é capaz de representar um domínio em um certo período de tempo, sendo possível que algumas das funções que formalizam o MDP não precisem ser conhecidas ou possam sofrer alterações ao longo do tempo. Por exemplo, Métodos de Monte Carlo (MC) ou Aprendizado por Reforço (RL) não necessitam do conhecimento das funções de transição $\mathcal{P}$ e de recompensa $\mathcal{R}$ (SUTTON; BARTO, 2015) para realizar a aproximação de $Q(s,a)$. Como RL apresenta esta capacidade de aproximar a função valor-ação mesmo com alterações nestas funções ao longo do tempo, é possível dizer que RL apresenta um comportamento adaptativo (OTTERLO, 2008).

Entretanto, os algoritmos de RL e MC podem lidar apenas com as alterações das funções de transição e reforço, não sendo possível a adição ou remoção de estados em $\mathcal{S}$ ou ações em $\mathcal{A}$ de forma formal. Portanto, caso o ambiente sofra modificações, é necessário formalizar um novo MDP a partir dos novos conjuntos $\mathcal{S}$ e $\mathcal{A}$.

Uma outra maneira de tratar as mudanças que ocorrem no domínio é a formalização de todas as possibilidades das variáveis que descrevem um estado. Desta forma tem-se a garantia de que independente da transição que ocorrer no domínio, o estado futuro observado pelo agente está contido no conjunto de estados $\mathcal{S}$ do MDP. Entretanto, descrever todas estas possibilidades pode gerar um espaço de estados muito grande (com possíveis pares estado-ação que não serão utilizados) por conta da Maldição da Dimensionalidade (BELLMAN; DREYFUS, 1971), em que o número de pares estado-ação aumenta conforme a quantidade de variáveis utilizadas para descrever um estado do problema.

Para realizar a modificação do MDP formalizado para o domínio, este trabalho utiliza *Answer Set Programming* (ASP), que permite que revisão (inferência não-monotônica) dos conjuntos de estados e ações seja realizada na descrição do ambiente, alterando a formalização do MDP utilizado para descrever o domínio (como descrito no lema 1). Esta revisão dos conjuntos de estados e ações de um MDP é realizada quando percebe-se que o domínio sofreu mudanças, fazendo com que o processo de aproximação da função $Q(s,a)$ pare, e os *answer sets* de um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ que descreve o novo MDP sejam encontrados, permitindo a utilização de uma nova descrição do MDP a partir destes novos *answer sets*.

4.6 ALGORITMOS PARA ASP(MDP)

A partir dos lemas e do teorema descritos nas seções anteriores, foi possível o desenvolvimento de um algoritmo que utiliza *Answer Set Programming* com processos Markovianos de decisão de forma que um método que aproxime a função valor-ação de um MDP estacionário possa ser utilizado para encontrar a solução ótima de um MDP não-estacionário.

Neste trabalho se faz uso de Métodos de Monte Carlo ou Aprendizado por Reforço, uma vez que estes não necessitam do conhecimento da função de recompensa por parte do agente para realizar o processo de aproximação da função $Q(s,a)$. Entretanto, isto não significa que outros métodos não possam ser utilizados.

O pseudo-código para o algoritmo ASP(MDP), que realiza os processos de encontrar $\dot{M}$ a partir de um MDP M e realiza o processo de busca pela solução ótima, encontra-se descrito no algoritmo 4, com as mudanças necessárias para a utilização de Métodos de Monte Carlo ou Aprendizado por Reforço apresentadas nos algoritmos 5 e 6 respectivamente.

Os algoritmos de ASP(MDP) recebem como parâmetro um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ que descreve um MDP e, caso já seja conhecida, a função $Q(s,a)$ (linha 2 do algoritmo 4). O primeiro passo é encontrar todos os *answer set* para o programa lógico $\Pi(\mathcal{S},\mathcal{A})$ de forma que cada *answer sets* descreva uma história do MDP M (linha 4). A partir dos *answer sets* encontrados e, com base no lema 1, o MDP $\dot{M}$ é formalizado e a função $Q(s,a)$ é alterada (caso esta tenha sido passada como parâmetro) ou obtida (caso esta seja previamente desconhecida), conforme apresentado na linha 5 do pseudo-código.

O programa lógico $\Pi(\mathcal{S},\mathcal{A})$ que é recebido pelo algoritmo contêm a descrição do ambiente no instante em que o algoritmo é utilizado. Portanto, na primeira execução do algoritmo, tem-se a descrição do ambiente em que mudanças passadas não são conhecidas (e mesmo que conhecidas, não são utilizadas pelo algoritmo). Porém, nas execuções seguintes, o programa lógico utilizado contêm a última descrição do ambiente, após a realização das alterações a partir das observações sobre o ambiente, ou seja, é utilizado o programa lógico retornado pelo próprio algoritmo na execução anterior.

De posse da função valor-ação que se deseja aproximar, é iniciada a interação com o ambiente seguindo o MDP $\dot{M}$ formalizado e utilizando um algoritmo que permita aproximar a função $Q(s,a)$ obtida (linhas 6 à 8). Esta interação ocorre enquanto o ambiente não sofrer alterações, pois quando isto ocorrer, é necessário realizar uma nova formalização do MDP que o descreve. Quando o ambiente sofrer alterações, o programa lógico $\Pi(\mathcal{S},\mathcal{A})$ que descreve o

MDP é modificado de forma a representar estas alterações no ambiente (linha 9) e ambos são retornados (linha 10).

Para tratar das alterações que ocorrem no domínio (MDP não-estacionário), o algoritmo 4 é chamado quantas vezes forem necessárias sendo utilizado como parâmetros de entrada o programa lógico $\Pi(\mathcal{S},\mathcal{A})$ e a função $Q(s,a)$ obtidas no término da chamada anterior. Para a primeira execução, é possível utilizar apenas $\Pi(\mathcal{S},\mathcal{A})$, obtendo $Q(s,a)$. A modificação da função valor-ação pode ser feita pela adição de novos pares estado-ação ou pela adição dos novos e remoção dos pares que não aparecem nos *answer sets* encontrados para $\dot{M}$. Esta escolha entre somente adição ou adição e remoção de estados depende do conhecimento prévio do problema.

Os algoritmos 5 e 6 apresentam a modificação feita no instante da interação com o ambiente para que esta se dê utilizando Avaliação de Política de Monte Carlo ou Aprendizado por reforço, respectivamente. A partir destes algoritmos, os *answer sets* de um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ são utilizados para descrever o MDP cuja busca da solução é realizada por MC ou RL. Como para estes dois métodos não é necessário o conhecimento prévio da função de recompensa $\mathcal{R}$, a busca pela política ótima é feita através da interação com o ambiente (no caso do RL) ou a partir de amostragem (no caso de MC).

No algoritmo 5 é utilizada a Avaliação de Política de Monte Carlo (algoritmo 1 (SUTTON; BARTO, 2015) para realizar a amostragem do ambiente, porém ao invés de gerar políticas aleatórias, os próprios *answer sets* obtidos no passo anterior são utilizados como políticas, sendo que cada *answer set* é amostrado de forma a garantir que todos os pares estado-ação tenham sido utilizados.

Para o algoritmo ASP(RL) que utiliza Aprendizado por Reforço (algoritmo 6), pode-se utilizar qualquer método de RL que aproxime a função valor-ação obtida, não dependendo das características do método escolhido (por exemplo, *on-policy* ou *off-policy*).

Sobre a complexidade da utilização do ASP no método proposto, ASP é utilizado para obter as trajetórias de um MDP e, neste caso, é utilizado para verificar a existência de *answer sets*, que são soluções deste MDP. Desta forma, a complexidade da utilização do ASP encontra-se em co-NP, como demonstrado por Eiter et al. (2004), Eiter et al. (2007).

Este capítulo apresentou o método proposto para encontrar a solução de um MDP não-estacionário. O próximo capítulo apresenta os quatro experimentos propostos para avaliar esta proposta.

Algoritmo 4 – Algoritmo ASP(MDP)

Algoritmo: ASP(MDP)
Entrada: Um MDP descrito como um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ e uma função $Q(s,a)$ a ser aproximada (opcional).
Saída: A função $Q(s,a)$ aproximada e o novo programa lógico $\Pi(\mathcal{S},\mathcal{A})$.
Encontre todos os *answer sets* $\mathcal{H}$ para $\Pi(\mathcal{S},\mathcal{A})$.
Atualize a função $Q(s,a)$ usando $\dot{\mathcal{S}}$ e $\dot{\mathcal{A}}$ encontradas em $\mathcal{H}$.
enquanto *o ambiente não sofrer alterações* **faça**
    Aproxime a função $Q(s,a)$ utilizando um método de solução para MDPs.
fim
Inclua as alterações do ambiente em $\Pi(\mathcal{S},\mathcal{A})$
Execute ASP(MDP) com $\Pi(\mathcal{S},\mathcal{A})$ e a função $Q(s,a)$ aproximada.

Fonte: Autor

Algoritmo 5 – Algoritmo Monte Carlo Answer Set Evaluation (MCASE)

Algoritmo: MCASE
Entrada: Um MDP descrito como um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ e uma função $Q(s,a)$ a ser aproximada (opcional).
Saída: A função $Q(s,a)$ aproximada e o novo programa lógico $\Pi(\mathcal{S},\mathcal{A})$.
Encontre os *answer sets* $\mathcal{H}$ para $\Pi(\mathcal{S},\mathcal{A})$.
Atualize a função $Q(s,a)$ usando $\dot{\mathcal{S}}$ e $\dot{\mathcal{A}}$ encontradas em $\mathcal{H}$.
enquanto *o ambiente não sofrer alterações* **faça**
    para cada answer set *em* $\mathcal{H}$ **faça**
        Execute o algoritmo Avaliação de Política de Monte Carlo (Algoritmo 1) utilizando o *answer set* a ser avaliado.
    fim
fim
Inclua as alterações do ambiente em $\Pi(\mathcal{S},\mathcal{A})$
Execute MCASE com $\Pi(\mathcal{S},\mathcal{A})$ e a função $Q(s,a)$ aproximada.

Fonte: Autor

Algoritmo 6 – Algoritmo ASP(RL)

Algoritmo: ASP(RL)
Entrada: Um MDP descrito como um programa lógico $\Pi(\mathcal{S},\mathcal{A})$ e uma função $Q(s,a)$ a ser aproximada (opcional).
Saída: A função $Q(s,a)$ aproximada e o novo programa lógico $\Pi(\mathcal{S},\mathcal{A})$.
Encontre os *answer sets* $\mathcal{H}$ para $\Pi(\mathcal{S},\mathcal{A})$.
Atualize a função $Q(s,a)$ usando $\dot{\mathcal{S}}$ e $\dot{\mathcal{A}}$ encontradas em $\mathcal{H}$.
enquanto *o ambiente não sofrer alterações* **faça**
    Aproxime $Q(s,a)$ utilizando um método de RL
fim
Inclua as alterações do ambiente em $\Pi(\mathcal{S},\mathcal{A})$
Execute ASP(RL) com $\Pi(\mathcal{S},\mathcal{A})$ e a função $Q(s,a)$ aproximada.

Fonte: Autor

5 EXPERIMENTOS

Com o objetivo de avaliar a proposta apresentada no capítulo anterior foram realizados quatro experimentos.

O primeiro experimento (seção 5.1) propõe um problema não-determinístico sobre a chance de um programa ter sua execução realizada sem falhas em um servidor. O segundo experimento (seção 5.2) trata da diminuição do espaço de estados do MDP utilizando como domínio a realização de uma determinada manobra em um ônibus espacial. Neste experimento é feita uma comparação entre utilizar somente as políticas que levam ao objetivo e o domínio completo. O terceiro experimento (seção 5.3) tem como objetivo verificar a possibilidade de tratar modificações que ocorrem no domínio ao longo do tempo, assumindo um ambiente não-estacionário em que um mundo de grades é alterado sem conhecimento prévio. O quarto experimento (seção 5.4) analisa a utilização do ASP(MDP) em um mundo de grades não-determinístico, permitindo verificar a influência das probabilidades de transição no processo de busca pela solução ótima.

5.1 DOMÍNIO CLIENTE-SERVIDOR

O domínio do cliente-servidor é um domínio simples e não-determinístico que é utilizado aqui com o objetivo de verificar se é possível utilizar ASP para descrever um domínio não-determinístico e encontrar a política ótima deste domínio.

5.1.1 Descrição do Domínio

Neste domínio tem-se cinco servidores que podem executar programas submetidos e cinco programas que podem ser submetidos para qualquer um destes servidores para que sejam executados. O agente funciona como um *Broker* e deve gerenciar os programas fazendo a submissão destes para os servidores.

Cada programa possui um tempo esperado de execução de forma que, se utilizado o mesmo servidor para executá-los, seriam necessários tempos diferentes para cada programa. Os tempos esperados de execução (em unidades de tempo) para os cinco programas utilizados neste experimento são $(100; 100; 100; 150; 150)$. No programa lógico os programas receberam os números de `1` a `5`.

Da mesma forma, cada servidor requer um certo tempo para executar a mesma quantidade de instruções, além de possuir uma probabilidade de falha na computação. Portanto, o mesmo programa sendo executado em servidores diferentes possui diferentes tempos de execução. A execução de um processo pode ter terminado ou não, dependendo da probabilidade do servidor de devolver a resposta correta. Os tempos necessários para computação para cada servidor (também em unidades de tempo) utilizados neste experimento são $(50; 75; 100; 125; 150)$ e as probabilidades de cada servidor realizar o processamento corretamente são $(0{,}5; 1{,}0; 1{,}0; 0{,}9; 0{,}5)$. Para o programa lógico, foram utilizadas as letras de `a` ao `e` para referenciar os servidores.

Um detalhe importante deste experimento é que deseja-se encontrar a política ótima que distribua os programas pelos servidores, ou seja, a política não pode alocar mais do que um programa por vez em um servidor.

Para este experimento, o MDP formalizado consiste em:

a) $\mathcal{S}$: a fila de programas que ainda faltam ser executados;

b) $\mathcal{A}$: a submissão de um programa para um servidor;

c) $\mathcal{P}$: cada submissão pode ir para um estado em que o programa atual foi executado corretamente ($s_{t+1} \neq s$), removendo este da fila, ou retorna ao mesmo estado por conta da falha na execução do programa ($s_{t+1} = s$);

d) $\mathcal{R}$: o valor da recompensa é dado a partir do tempo necessário para execução T_n e do tempo esperado de execução T_e, de forma que

$$r(s,a,s_{t+1}) = \begin{cases} r^+ = T_e * \dfrac{T_n}{100}, & \text{caso o programa seja executado corretamente;} \\ r^- = -T_e * \dfrac{100}{T_n}, & \text{caso a execução do programa falhe.} \end{cases} \tag{11}$$

5.1.2 Resultados Esperados

O código em programação lógica que foi utilizado neste experimento para fazer a distribuição dos programas nos servidores encontra-se na figura 8, enquanto que a instância utilizada (configuração dos servidores e dos programas) encontra-se na figura 9.

Para este experimento foram utilizados *Gringo* e *Clasp* (GEBSER; KAMINSKI; KAUFMANN, 2013) para encontrar os *answer sets* e o algoritmo MCASE (algoritmo 5) para aproximar a função valor-ação obtida a partir dos *answer sets* encontrados para os códigos apre-

Figura 8 – Código utilizado para a distribuição de programas em servidores

```
server(S) :- server(S,TP,PP).
proc(P) :- proc(P,TP).

1 <= { run_in(P,S) : server(S) } <= 1 :- proc(P).

:- run_in(P1,S1), run_in(P2,S2), S1 == S2,
  P2 = P1 + P, P = 1 .. (nserv - 1).

allocate(P,S) :- run_in(P,S).

#show allocate/2.
```

Fonte: Autor

Figura 9 – Código utilizado para a instância do domínio

```
server(a,50,50).
server(b,75,100).
server(c,100,100).
server(d,125,90).
server(e,150,50).
#const nserv=5.

proc(1..3,100).
proc(4..5,150).
```

Fonte: Autor

sentados nas figuras 8 e 9. No programa lógico da figura 9, `server(X,Y,Z)` descreve o servidor de nome `X` que possui tempo de execução `Y` e probabilidade de execução sem falha `Z` e `proc(X,Y)` descreve que o processo `X` necessita de tempo `Y` para ser executado.

No caso deste experimento, o domínio é não-determinístico por conta da probabilidade de falha na execução do programa. ASP foi utilizado para a descrição do domínio e esta linguagem não contém métodos para tratar probabilidades de forma quantitativa. Portanto, a descrição deste domínio se torna determinística (nenhum servidor possui chance de falhar na execução dos programas), mas a distribuição descrita pela linha

```
1 <= { run_in(P,S): server(S) } <= 1 :- proc(P).
```

permite que a possibilidade de falha seja tratada através da distribuição de processos `P` em servidores `S` independente da probabilidade real de falha no processamento e a interação com o ambiente seja realizada de forma não-determinística.

Neste experimento existe apenas uma ação: a de alocar um programa em um servidor. Portanto, esta ação não pode ser removida uma vez que resultaria em $\dot{\mathcal{A}} = \varnothing$. Para o conjunto de

estados $\dot{\mathcal{S}}$, todos os estados de $\mathcal{S}$ estarão contidos em $\dot{\mathcal{S}}$, uma vez que, neste caso, é necessário visitar todos os estados para encontrar a solução ótima, portanto $\mathcal{S} = \dot{\mathcal{S}} \neq \varnothing$. O que será alterado quando for utilizado ASP para descrever o MDP é a função $\dot{\mathcal{P}}$, que possui menos transições que $\mathcal{P}$ por conta das restrições inclusas em $\Pi(\mathcal{S},\mathcal{A})$.

Portanto, este experimento foi utilizado para verificar se a descrição de um MDP em ASP é realmente capaz de fornecer um novo MDP que pode ser otimizado mesmo em casos em que o domínio é não-determinístico. Neste domínio, não há uma grande redução no espaço de busca, porém a verificação da função $Q(s,a)$ aproximada é de fácil verificação.

Os resultados obtidos experimentalmente para este domínio encontram-se na seção 6.1 no próximo capítulo.

5.2 DOMÍNIO DO SISTEMA DE CONTROLE POR REAÇÃO DE UM ÔNIBUS ESPACIAL

Uma vez verificado que é possível utilizar *answer sets* para realizar a formulação de um MDP, o segundo experimento utiliza a realização de manobras utilizando o sistema de controle por reação (RCS)[1] de um ônibus espacial como a formalização de um MDP determinístico. A descrição em ASP para o RCS foi desenvolvida através de uma parceria entre a *United Space Alliance* (USA), a Universidade do Texas em El Paso e a Universidade Tecnológica do Texas. A descrição do RCS teve a supervisão de pessoas relacionadas às missões do ônibus espacial e que eram responsáveis por descrever a sequência de ações que os astronautas deveriam executar para realizar uma determinada manobra (BALDUCCINI; GELFOND; NOGUEIRA, 2006; BALDUCCINI et al., 2001; BALDUCCINI et al., 2002; NOGUEIRA et al., 2001). Ao programa lógico final deu-se o nome de *USA-Adivisor* e, por motivos de compatibilidade com a linguagem, é necessária a utilização dos programas *lparse* e *smodels* para obter os *answer sets*.

O objetivo deste experimento foi verificar a possibilidade de redução do espaço de busca, uma vez que a aproximação da função valor-ação considerando o MDP que descreve o RCS por completo se torna inviável por conta do grande espaço de busca. Esta redução permitiria que fosse realizada a busca pela solução ótima em um espaço de busca menor.

[1]Este programa e uma descrição do seu funcionamento encontram-se em <http://www.depts.ttu.edu/cs/research/krlab/software.php>

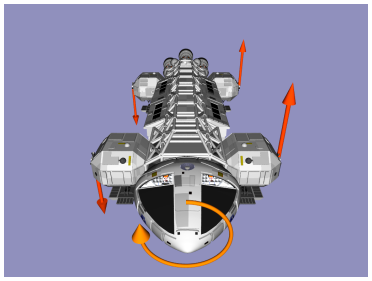
5.2.1 Descrição do Domínio

O RCS de um ônibus espacial é constituído de válvulas que podem ser abertas ou fechadas, jatos que precisam ser ligados para realizar a manobra, tanques de combustível, canos que ligam os tanques às válvulas e jatos, e um circuito que permite que a manobra seja realizada através do computador de um ônibus espacial. Uma manobra em um RCS consiste em direcionar os jatos para a direção correta e encontrar um caminho sem vazamento que ligue um tanque de combustível até um jato. Este caminho é descrito pelas válvulas que estão abertas ou fechadas, sendo que o controle das válvulas pode ser realizado manualmente (o astronauta abre ou fecha a válvula) ou através do computador interno (o circuito que constitui o RCS realiza a mudança na posição da válvula).

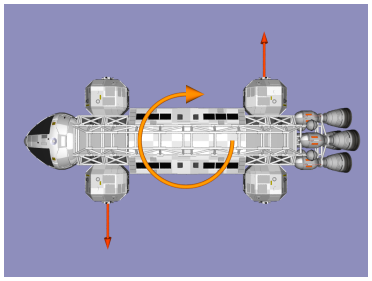
Um ônibus espacial possui 3 RCS: da esquerda (`left_rcs`), da direita (`right_rcs`) e o para frente (`fw_rcs`), sendo que uma manobra pode necessitar de todos os RCS. As possíveis manobras que podem ser realizadas por um ônibus espacial são as rotações no próprio eixo (como, por exemplo, nas figuras 10a e 10b) e translação (como o exemplo da figura 10c).

Figura 10 – Movimentos que podem ser realizados pelo ônibus espacial

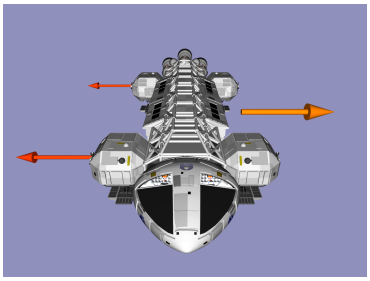
(a) Rotação no eixo X



(b) Rotação no eixo Y



(c) Translação



Fonte: Jean-Christophe, 2011

Uma manobra que se deseja realizar é descrita no programa lógico na forma

```
goal(T, X) := time(T), h(maneuver_of(Y,X), T)
```

em que `T` é o instante de tempo (passo) em que a ação foi realizada (a quantidade máxima de passos deve ser fornecida como um parâmetro na chamada do resolvedor ASP), `X` é o RCS que realizará a manobra e `Y` é a manobra que se deseja realizar, como por exemplo na manobra descrita no programa da figura 11.

A descrição inicial do RCS permite dizer se existe alguma válvula travada e que não poderá ter sua posição alterada. Estas restrições são descritas pela relação `stuck(X,Y)` que es-

pecifica que a válvula X está travada na posição Y. Por exemplo, na figura 11, stuck(rm1,open) descreve que a válvula rm1 está travada na posição aberta.

Figura 11 – Código que descreve o estado inicial dos RCS e a manobra que se deseja realizar

```
%%%%%%%%% Initial Situation %%%%%%%%%%
%%%%%%%%% FAULTS and OTHER EXCEPTIONS %%%%%%%%%%
stuck(lfdummy,open).
stuck(rha,open).
stuck(rm1,open).
%%%%%%%%%%%% GOALS %%%%%%%%%%%%%

% Maneuver to be performed: plus_x

goal(T,fwd_rcs) :- time(T),
               h(maneuver_of(plus_x,fwd_rcs),T).

goal(T,left_rcs) :- time(T),
               h(maneuver_of(plus_x,left_rcs),T).

goal(T,right_rcs) :- time(T),
               h(maneuver_of(plus_x,right_rcs),T).
```

Fonte: Autor

Para que este experimento fosse realizado, a resposta fornecida pelo resolvedor ASP foi alterada no código para que na saída do *solver* sejam fornecidos os estados e as ações que devem ser utilizadas para realizar a manobra desejada no ônibus espacial. Desta forma, uma variável do estado é representada por h(in_state(X,Y),T) descrevendo que no passo T a válvula X está na posição Y, com Y = {gpc, open, closed} (em que gpc indica que a válvula está sendo controlada pelo computador de um ônibus espacial). Por exemplo, h(in_state(lom3,closed),1) que significa que no passo 1 a válvula lom3 está na posição closed. Um estado é descrito pelas situações de cada válvula existente no ônibus espacial.

A relação utilizada para descrever que uma ação que foi realizada é occurs(X,T), em que foi realizado X no passo T. Neste caso a ação é descrita como flip(X,Y) representando que a válvula X foi alterada para o estado Y = {open, closed, gpc}. Por exemplo, occurs(flip(lhb,open),0) descreve que a válvula lhb foi aberta no instante 0. Além de occurs(flip(X,Y),T) é possível que apareça apenas occurs(X,T) sendo esta a consequência de uma ação realizada.

A descrição dos átomos e valores utilizados como exemplos para o sistema de controle por reação estão descritos na tabela 1. Não serão descritos todos os átomos existentes na codificação por conta de sua extensão, uma vez que o código fornecido descreve todas as válvulas existentes em um sistema por controle de reação real utilizado em ônibus espaciais.

Tabela 1 – Descrição de variáveis do sistema de controle por reação

Variável	Significado
`left_rcs`	RCS da esquerda
`right_rcs`	RCS da direita
`fwd_rcs`	RCS da frente
`lom3`	válvula
`lhb`	válvula
`lfdummy`	válvula
`rha`	válvula
`rm1`	válvula
`gpc`	*general purpose computer* usado no ônibus espacial
`open`	válvula aberta
`close`	válvula fechada
`time(T)`	tempo
`goal(T,X)`	objetivo de fazer `X` no instante de tempo `T`
`maneuver_of(Y,X)`	manobra `Y` que se deseja realizar no RCS `X`
`h(X,T)`	estado da variável ou função `X` no instante de tempo `T`
`in_state(X,T)`	estado da válvula `X` no instante de tempo `T`
`flip(X,Y)`	virar a válvula `X` para a posição `Y`
`occurs(X,T)`	`X` ocorre no instante `T`
`stuck(X, Y)`	válvula `X` esta travada no estado `Y`

Fonte: Autor

Os *answer sets* fornecidos para este domínio são uma sequência de variáveis descrevendo o estado em um determinado instante seguido pela ação que deve ser realizada naquele instante. Por conta da extensão dos *answer sets*, estes não serão apresentados.

Para a formalização deste MDP foram utilizados os estados, ações e transições fornecidos pelo programa lógico. A função de reforço escolhida considera que os recursos existentes no ônibus espacial não são renovados ao chegar no estado objetivo, portanto para cada ação realizada até chegar no estado final o reforço dado é de -1 (sendo este pelo fato de que recursos são gastos), ao chegar no estado final é fornecido o reforço de 0 e ao tentar mudar a posição de uma válvula que está travada, o reforço é -1000. O MDP foi formalizado da seguinte forma:

a) $\mathcal{S}$: o conjunto de variáveis que descreve a situação de cada válvula;

b) $\mathcal{A}$: a ação de abrir ou fechar uma válvula de forma manual ou através do computador (são consideradas ações diferentes da abertura de cada válvula em cada passo);

c) $\mathcal{P}$: a função de transição é descrita pelo programa lógico; apenas as transições que estão descritas nos *answer sets* são conhecidas. Como o domínio é determinístico, a ação de virar uma válvula só não muda a situação da válvula quando esta está travada;

d) $\mathcal{R}$: a função de recompensa definida como:

$$r(s,a,s_{t+1}) = \begin{cases} r_{s_g} = 0, & \text{ao chegar ao estado final;} \\ r^{-} = -1000, & \text{ao tentar mudar a posição de uma válvula travada;} \\ r_s = -1, & \text{para qualquer outra ação ou consequência da ação} \end{cases}$$

5.2.2 Resultados esperados

Para a realização dos experimentos foi escolhida apenas uma instância do problema dentre as já fornecidas em conjunto com o programa lógico. Cada uma destas instâncias descrevem o estado inicial do ônibus espacial (quais válvulas estão travadas e em quais posições) e também a manobra a ser realizada. A instância utilizada encontra-se na figura 11. O restante do código utilizado para o experimento não será adicionado por ser muito extenso, mas este pode ser obtido na página do projeto [2]. A quantidade máxima de passos que podem ser utilizados é fornecida na chamada do resolvedor ASP.

Neste experimento não foi realizada a reimplementação de nenhuma parte do *USA-Adivisor* que descreve o funcionamento de um ônibus espacial, apenas a execução do código fornecido com uma instância específica, junto com a modificação realizada apenas na exibição dos resultados. Portanto, a obtenção do MDP que é descrito pelos *answer sets* pode ser feito da mesma maneira que no experimento anterior (seção 5.1).

Porém para realizar a comparação e verificar se ocorreu redução no espaço de busca, é necessário obter o MDP sem as restrições. Para isto foi realizada a execução de todas as combinações possíveis das restrições existentes no código da figura 11, para garantir que seria gerada pelo menos uma política com cada combinação de restrições (por conta da grande quantidade de *answer sets* possíveis para cada instância, não foi viável a utilização de todas as respostas). Portanto o MDP M que possui todos os pares estado-ação é descrito pela união dos *answer sets* encontrados para as oito configurações possíveis da instância, enquanto o MDP $\dot{M}$ é descrito apenas pelos *answer sets* da instância da figura 11.

A busca pelos *answer sets* seguiu a mesma forma que foi utilizada pelos desenvolvedores do *USA-Adivisor*, de forma que o número mínimo de passos utilizado foi 3 e o número máximo foi 10. Por conta do grande número de *answer sets* que podem ser obtidos para cada instância e a necessidade de ter uma resposta rápida para o problema por conta da natureza deste, foram gerados apenas 50 *answer sets* para cada quantidade de passos. Portanto, para o MDP $\dot{M}$ que

[2] A implementação completa encontra-se em <http://www.depts.ttu.edu/cs/research/krlab/software.php>

utiliza apenas a instância da figura 11 tem-se $8 \times 50 \times 1 = 400$ políticas e para o MDP M com todas as variações desta instância pode-se ter até $8 \times 50 \times 8 = 3200$ políticas, caso não sejam obtidas políticas repetidas entre as instâncias. Como foram utilizadas apenas 50 políticas para cada instância, sabe-se que neste experimento não está sendo realizada a busca por todo espaço de estados.

Considerando esta configuração final, espera-se verificar que a realização do processo de otimização utilizando apenas o MDP $\dot{M}$ seja mais rápida e necessite de menos estados e ações. No melhor caso, em que as 400 políticas que resolvem $\dot{M}$ não aparecem em nenhuma outra instância utilizada para descrever M, pode-se ter uma redução no espaço de busca para aproximadamente 12,5% do espaço de busca completo. No pior dos casos, em que as 400 políticas são repetidas para todas as instâncias e as restrições não afetam de nenhuma forma a solução do problema, não haverá redução na quantidade de pares estado-ação e $M = \dot{M}$.

Por fim, espera-se verificar se os valores obtidos para cada par estado-ação nos dois MDPs M e $\dot{M}$ possuem os mesmos valores, ou seja, se a política ótima do MDP que foi descrito por todas as instâncias é a mesma da política ótima do MDP descrito apenas pela instância da figura 11, conforme teorema 1. Para encontrar estes valores, foi utilizado o MCASE (algoritmo 5) amostrando cada *answer set* 1.000.000 de vezes.

Os resultados obtidos para este experimento são apresentados na seção 6.2.

5.3 DOMÍNIO DO MUNDO DE GRADES NÃO-ESTACIONÁRIO

Uma vez confirmada a viabilidade de utilizar *answer sets* para formalizar um MDP e a possível diminuição no espaço de busca, o último experimento tem como objetivo verificar se a utilização de ASP para obter a descrição do MDP também permite que o MDP seja alterado ao longo do tempo, sem perder as informações que já foram obtidas e atualizando os valores conforme necessário. O experimento realizado se passa em um mundo de grades determinístico que pode sofrer alterações inesperadas ao longo do tempo.

5.3.1 Descrição do Domínio

Este experimento é executado em um mundo de grades de 10x10 que permite a realização de quatro ações: *para cima*, *para baixo*, *para esquerda* e *para direita*. A cada instante o agente pode executar apenas uma das ações e esta faz com que o agente se mova por apenas uma posição da atual, portanto, *para cima* faz com que o agente se mova uma posição acima no

mundo de grades, *para baixo* faz com que o agente se mova uma posição abaixo, *para esquerda* move o agente uma célula para a esquerda e *para direita* faz com que o agente se mova uma célula para a direita. As transições neste mundo de grades são determinísticas, fazendo com que a cada ação em um estado só possa levar a um novo estado conforme as descrições das ações utilizadas. Portanto, não há chance do agente realizar uma ação e estar no instante seguinte em um estado não esperado. Foi escolhido a utilização de um mundo de grades determinístico para este experimento pois a partir da análise dos resultados seria possível verificar de forma mais clara a influência que os *answer sets* possuem sobre o MDP que descreve o problema. O objetivo do agente que interage com este mundo de grades é sair de um estado inicial pré-definido e chegar a um estado objetivo também pré-definido.

Porém este mundo de grades pode sofrer alterações ao longo do tempo de forma que paredes e buracos apareçam ou desapareçam. Nos casos em que o agente executa uma ação que o direciona para a parede, no instante seguinte ele se encontra no mesmo estado. Quando o agente executa uma ação que o leva a cair em um buraco, o episódio atual é terminado e o agente inicia um novo episódio a partir do estado inicial.

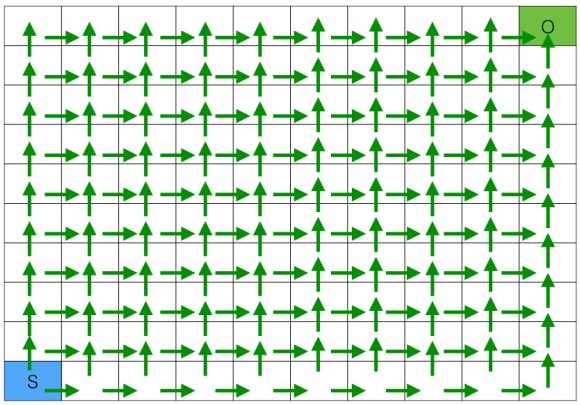
Para a realização deste experimento foram escolhidas quatro configurações do mundo de grades. Estas configurações estão descritas na figura 12. Nestas figuras o quadrado azul com a letra S indica o estado inicial e o quadrado verde com a letra O o estado objetivo do problema. As paredes são representadas pelos quadrados cinza com a letra W e os buracos pelos quadrados amarelos com a letra H. As setas indicam os *answer sets* que foram encontrados para este problema (estes *answer sets* serão discutidos na seção a seguir).

Este experimento foi utilizado para verificar dois fatores da proposta. O primeiro consiste em verificar se o agente que aprende utilizando ASP(MDP) é capaz de lidar com as mudanças no espaço de estados ao longo do tempo. Para tanto foram realizados três testes diferentes: o primeiro tem início da interação com o ambiente usando o mapa da figura 12a e após 100 episódios é realizada a troca para o mundo da figura 12b. Neste segundo mapa também são realizados 100 episódios. O segundo é semelhante ao anterior, porém a mudança é do mapa na figura 12a para a figura 12c. No último teste a mudança é do mapa da figura 12a para o mapa da figura 12d.

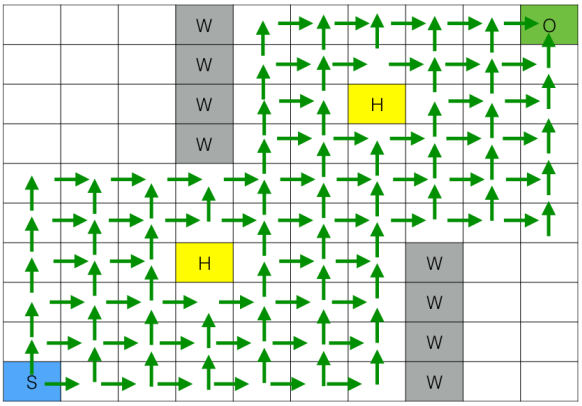
O segundo fator que desejamos observar é o resultado da aproximação da função valor-ação por um método *on-policy* e um método *off-policy*. Uma vez que o MDP obtido através dos *answer sets* não percorrem todos os pares estados-ação, como ocorre no MDP que descreve todo o mundo de grades, algumas das políticas que o método *on-policy* realizaria são excluídas, mudando os valores que serão utilizados para realizar a aproximação da função valor-ação,

Figura 12 – Configurações do mundo de grade utilizado no experimento e seus respectivos *answer sets*

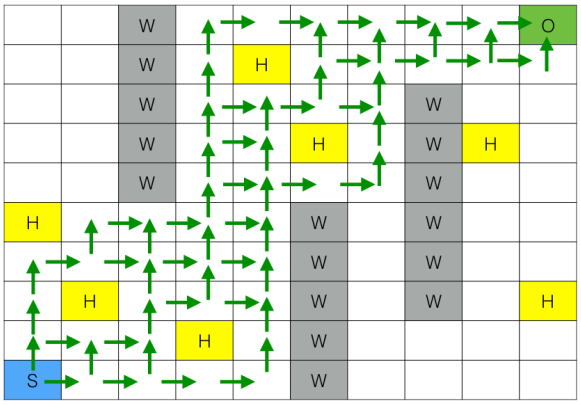
(a) Mapa 1



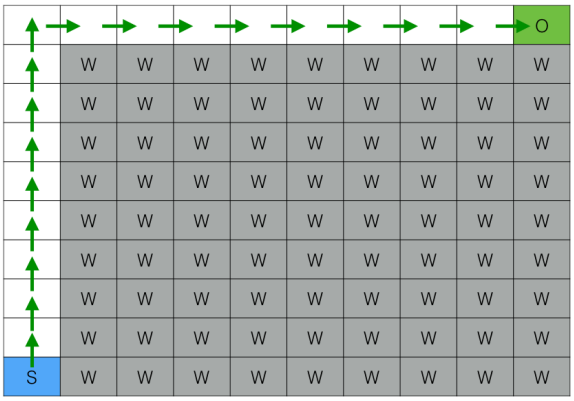
(b) Mapa 2



(c) Mapa 3



(d) Mapa 4



Fonte: Autor

fazendo com que o agente encontre uma política ótima diferente do que se o mesmo algoritmo fosse usado no MDP não descrito pelos *answer sets*.

Para este problema foram utilizados os algoritmos de ASP(RL) (algoritmo 6) com o Q-Learning e SARSA como métodos de aproximação da função valor-ação, estes algoritmos serão denominados como ASP(Q-Learning) e ASP(SARSA) respectivamente. O MDP utilizado para formalizar este domínio é o seguinte:

a) $\mathcal{S}$: cada posição do mundo de grades representa um estado diferente;

b) $\mathcal{A}$: as quatro ações $\mathcal{A} = \{para\ cima, para\ baixo, para\ esquerda, para\ direita\}$;

c) $\mathcal{P}$: como o mundo de grades é determinístico, cada ação muda a posição do agente para um estado na direção descrita pela ação. No caso de encontrar uma parede, o

agente permanece no mesmo estado. Quando o agente cai em um buraco, o episódio é reiniciado;

d) $\mathcal{R}$: a função de recompensa para todos os mapas é a mesma, independente da existência de paredes e buracos. Esta função foi definida da seguinte forma:

$$r(s,a,s_{t+1}) = \begin{cases} r^- = -100, & \text{quando o agente cair em um buraco;} \\ r_{s_g} = +100, & \text{quando o agente chega ao estado objetivo;} \\ r_s = -1, & \text{em qualquer outra transição.} \end{cases} \quad (12)$$

A representação deste domínio em um programa lógico foi feita na linguagem $BC+$ (BABB; LEE, 2015a) que pode ser traduzida para ASP utilizando o software *CPLUS2ASP* (BABB; LEE, 2013). O código que descreve o mundo de grades utilizando $BC+$ encontra-se na figura 6 na seção 3.3. Os códigos apresentados no anexo A nas figuras 22, 23 e 24 descrevem os buracos e as paredes conforme os mapas das figuras 12b, 12c e 12d respectivamente.

5.3.2 Resultados Esperados

Observando as figuras que descrevem o mundo de grades é possível perceber que as mudanças no ambiente não são todas iguais. O primeiro mapa (figura 12a) não possui nenhum obstáculo, portanto os pares estado-ação que levam à solução do problema são uma combinação das ações *para cima* e *para direita*. Portanto a utilização de ASP para descrever este MDP permite a redução do conjunto de ações para metade do que seria o mundo de grades por completo.

Ao realizar o primeiro teste, são realizados 100 episódios no mapa da figura 12a e em seguida o mapa é alterado para o da figura 12b. Neste segundo mapa existem menos estados que o agente pode visitar por conta das paredes e buracos que foram adicionados. Desta forma, espera-se que encontrar os *answer sets* para este mapa leve menos tempo do que o anterior, uma vez que existem menos combinações que podem levar o agente do estado inicial ao estado final. Da mesma forma, espera-se que o aprendizado para este mapa seja mais rápido do que o aprendizado para o primeiro mapa, uma vez que há menos pares estado-ação que podem ser visitados para alcançar o objetivo.

O segundo teste utiliza o mapa da figura 12c como a mudança no episódio 100 a partir do mapa 1 (figura 12a). Comparando com a figura 12b, percebe-se que este mapa possui mais paredes e buracos do que o anterior, possuindo menos *answer sets* que levam o agente do estado

inicial ao estado final. Portanto espera-se que o processo de encontrar os *answer sets* e realizar a busca pela política ótima leve menos tempo não somente do que em relação ao mapa da figura 12a, mas também em relação a figura 12b.

O terceiro teste utiliza o mapa da figura 12d como mudança no episódio 100 a partir do mapa 1 (figura 12a). Este mapa possui apenas uma política que leva o agente ao estado final sem que seja executada uma ação que leve o agente à alguma parede. Neste caso, a busca pelos *answer sets* deve ser ainda mais rápida do que os casos anteriores, assim como o processo de otimização. Entretanto, neste caso em que há apenas uma política, não seria necessário realizar o processo de otimização por conta de não existir outra política que resolva o problema.

Um fator importante no uso do $BC+$ é que os resultados obtidos descrevem apenas as políticas que resolvem o problema com o menor número de passos possíveis, o que também é um critério da função de reforço ao fornecer o valor de -1 a cada passo que não leve ao estado final. Entretanto, os *answer sets* obtidos não passam por alguns estados pois não há necessidade dado que o domínio é determinístico e passar por qualquer outro estado geraria uma política com número de passos maior que o mínimo encontrado. Um exemplo disto é o que ocorre no canto superior esquerdo da figura 12b em que o agente não executa ações que o leve a estados que estejam na mesma direção em que há uma parede, pois executar uma ação que o leve para este canto faz com que o número de passos para alcançar o estado objetivo seja maior do que o número de passos mínimo encontrado.

Os resultados obtidos para este experimento encontram-se na seção 6.3 do próximo capítulo.

5.4 DOMÍNIO DO MUNDO DE GRADES NÃO-DETERMINÍSTICO

A partir do experimento descrito na seção anterior, o último experimento realizado trata dos mesmos mundo de grades não-estacionários, porém em sua versão não-determinística.

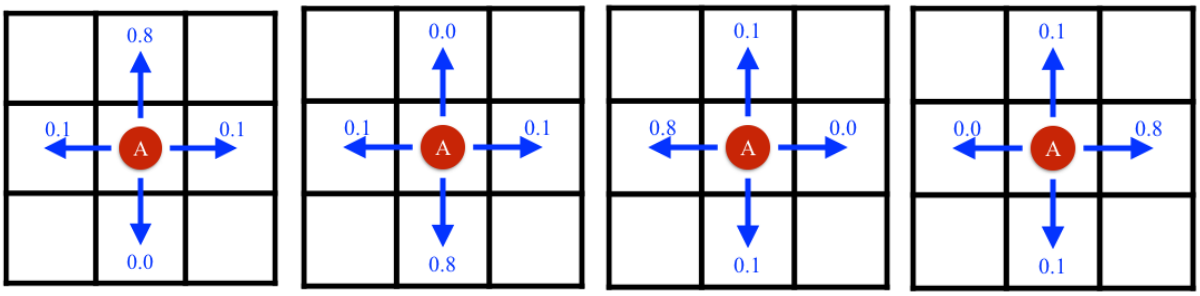
5.4.1 Descrição do Domínio

Os mapas utilizados para a realização deste experimento são os mesmos descritos para o experimento anterior e encontram-se na figura 12. Da mesma forma, as ações, os estados inicial e final e os valores de reforços também foram mantidos os mesmos. Portanto, $\mathcal{S}$, $\mathcal{A}$ e $\mathcal{R}$ não foram alterados.

A diferença entre este experimento e o da seção anterior encontra-se na função de transição $\mathcal{P}$ do MDP que o descreve. No mundos de grades utilizados neste experimento, o agente que interage com o ambiente possui probabilidade de 80% de realizar a transição da forma desejada (por exemplo, ao executar a ação *para cima* o agente vai para o estado acima do atual) e 0% de chance de ir para a direção oposta (no caso da ação *para cima*, o agente não consegue ir para o estado abaixo do atual) e 10% de chance de ir para cada uma das outras direções (para a esquerda ou para a direita, no caso da ação *para cima*). As probabilidades para as quatro ações utilizadas encontram-se na figura 13.

Figura 13 – Probabilidades de execução de cada ação no mundo de grades não-determinístico

(a) Ação *para cima* (b) Ação *para baixo* (c) Ação *para esquerda* (d) Ação *para direita*



Fonte: Autor

Neste experimento foram utilizados os mesmos algoritmos do anterior: Q-Learning, SARSA, ASP(Q-Learning) e ASP(SARSA). Desta forma é possível verificar a influência do ASP no processo de aproximação da função valor-ação, além da possível diferença entre os resultados obtidos para métodos *on-policy* (Q-Learning e ASP(Q-Learning)) e *off-policy* (SARSA e ASP(SARSA)) em um ambiente não-determinístico.

Para este experimento foram realizados 5000 episódios no primeiro mapa antes de realizar a transição para o próximo mapa. Após esta transição, a mesma quantidade de episódios foi realizada no novo domínio, sendo utilizadas as mesmas três situações do experimento anterior.

5.4.2 Resultados Esperados

Ao considerar a probabilidade de transição é necessário levar em conta que ações que eram removidas no mundo de grades determinístico devem agora ser consideradas. Por exemplo, caso o agente deseje ir do estado atual s_t para o estado s_{t+1} à sua direita, a ação *para direita* permite que esta transição seja realizada (como no ambiente determinístico), porém as

ações *para cima* e *para baixo* podem gerar esta mesma transição por conta do não-determinismo do mundo de grades. Portanto, para o caso em que no mundo determinístico era considerada apenas uma ação, para o não-determinístico deverão ser consideradas mais duas ações: as que permitem que a mesma transição entre s_t e s_{t+1} seja realizada.

Desta forma, a redução do espaço de busca ainda é realizada, porém esta é menor (ou aproximadamente nula) do que no caso do mundo determinístico. Portanto, espera-se que os resultados obtidos para este experimento sejam próximos aos do anterior, porém os resultados obtidos para o ASP(RL) são mais próximos dos resultados obtidos para os algoritmos de RL por conta da influência do não-determinísmo do ambiente na exploração que o agente deve realizar no mundo de grades.

Em relação às probabilidades de transição, é importante ressaltar que o $BC+$ e ASP não possuem informações sobre o valor da probabilidade. Estes possuem conhecimento apenas de que estas transições são possíveis, sendo a probabilidade uma informação conhecida apenas pelo ambiente.

Em relação aos resultados para cada um dos mapas, é esperado que para o primeiro mapa (figura 12a) o desempenho dos algoritmos de ASP(RL) sejam muito próximos aos dos algoritmos de RL, uma vez que não há redução no conjunto de estados e a redução no conjunto de ações é quase nula.

Para o segundo e terceiro mapas (figuras 12b e 12c respectivamente), espera-se obter um desempenho melhor em um dos critérios avaliados nos algoritmos de ASP(RL) em relação aos de RL. Enquanto que para o último mapa (figura 12d), por conta da grande redução no espaço de busca, espera-se desempenho melhor para os algoritmos de ASP(RL) para este mapa do que nos 3 mapas anteriores.

A seção 6.4 do próximo capítulo apresenta os resultados obtidos para o experimento descrito nesta seção. Neste experimento, foram comparados número de passos necessários para alcançar o estado objetivo e o retorno obtido por episódio, uma vez que a diferença quadrática da função valor-ação apresentou resultados muito próximos para os quatro algoritmos utilizados.

Este capítulo descreveu os experimentos realizados para avaliar a proposta feita para combinar ASP com MDPs. No próximo capítulo serão descritos os resultados obtidos nestes experimentos.

6 RESULTADOS

Este capítulo apresenta os resultados dos experimentos propostos no capítulo anterior. A descrição dos resultados de cada experimento é feita na seção de cada experimento e uma discussão da proposta é realizada no próximo capítulo (seção 7.1).

6.1 DOMÍNIO DO CLIENTE-SERVIDOR

Os resultados deste experimento foram obtidos a partir da utilização do *Gringo* e *Clasp* para encontrar os *answer sets* do programa lógico descrito nas figuras 8 e 9. Para este experimento, foram encontrados 120 *answer sets* (com parte da saída do programa *Clasp* sendo exibida na figura 14). Como é possível notar na figura 14, a saída do programa mostra em qual servidor `X` o programa `Y` deve ser alocado utilizando `allocate(Y,X)`.

Destas repostas, é obtido um MDP cuja solução ótima é encontrada utilizando o algoritmo MCASE, que utiliza estes 120 *answer sets* como políticas que devem ser amostradas a cada episódio, sendo que, a função valor-ação é atualizada a partir dos reforços recebidos pela execução da política que está sendo amostrada

Figura 14 – Saída exibida pelo programa *Clasp* (algumas linhas foram omitidas)

```
clasp version 2.1.4
Reading from stdin
Solving...
Answer: 1
allocate(1,a) allocate(2,b) allocate(3,c) allocate(4,d) allocate(5,e)
Answer: 2
allocate(1,a) allocate(2,b) allocate(3,c) allocate(4,e) allocate(5,d)
Answer: 3
allocate(1,a) allocate(2,b) allocate(3,d) allocate(4,c) allocate(5,e)
Answer: 4
allocate(1,a) allocate(2,b) allocate(3,d) allocate(4,e) allocate(5,c)
Answer: 5
allocate(1,a) allocate(2,b) allocate(3,e) allocate(4,c) allocate(5,d)
...
Answer: 119
allocate(1,e) allocate(2,d) allocate(3,c) allocate(4,a) allocate(5,b)
Answer: 120
allocate(1,e) allocate(2,d) allocate(3,c) allocate(4,b) allocate(5,a)
SATISFIABLE

Models : 120
Time : 0.003s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
```

Fonte: Autor

A tabela 2 apresenta a função valor-ação aproximada pelo MCASE para o problema descrito na seção 5.1. As colunas representam os servidores que recebem os programas a serem processados, enquanto que cada linha representa um programa a ser submetido. O valor apresentado na coluna `Y` e linha `X` é o valor aproximado pela amostragem de `allocate(X,Y)`. Esta tabela apresenta o valor de $Q(s,a)$ aproximado por uma das execuções, uma vez que, como este problema foi tratado como horizonte infinito, as funções aproximadas por todas execuções possui o mesmo valor.

Tabela 2 – Valores de $Q(s,a)$ aproximados pelo MCASE para o problema do Cliente-Servidor

Programa a ser submetido	Servidores				
	a	b	c	d	e
1	-74.93	75.00	100.00	106.10	42.57
2	-74.41	75.00	100.00	103.97	39.35
3	-75.19	75.00	100.00	104.92	41.55
4	-112.99	112.50	150.00	156.67	63.00
5	-107.32	112.50	150.00	160.75	64.19

Fonte: Autor

Observando a tabela 2 é possível extrair a política ótima a partir dos pares estado-ação que possuem os maiores valores e que não submetam o processo para um servidor que já possui um programa. Desta forma, quando se possui cinco programas a serem processados a primeira alocação deve ser para o servidor `d` se este estiver disponível, uma vez que o valor de $Q(\mathtt{d},\cdot)$ de alocar um processo em `d` é maior do que os valores de alocação para os outros servidores. Em seguida, com quatro programas na fila a alocação é para o servidor `c`, caso `d` não tenha acabado de processar o programa anterior, pois $Q(\mathtt{c},\cdot)$ só não é maior do que o valor de $Q(\mathtt{d},\cdot)$. Da mesma forma, em relação aos valores de $Q(\mathtt{b},\cdot)$, com o terceiro programa a submissão é para o servidor `b` se `d` e `c` já estiverem processando algum programa. A seguir o programa deve ser enviado ao servidor `e`, caso os outros ainda estejam ocupados, uma vez que $Q(\mathtt{e},\cdot)$ só é maior que o valor de $Q(\mathtt{a},\cdot)$. Por fim, o último programa deve ser alocado ao servidor `a` caso todos os outros não estejam disponíveis, uma vez que este servidor possui o menor valor para seus pares estado-ação.

Portanto, apesar do problema ser não-determinístico foi possível utilizar a sua descrição em um programa lógico determinístico para descrever o MDP e fornecer uma formalização para o processo de otimização.

6.2 DOMÍNIO DO SISTEMA DE CONTROLE POR REAÇÃO DE UM ÔNIBUS ESPACIAL

Os resultados obtidos para o problema de um ônibus espacial encontram-se na tabela 3 em que são descritas as quantidades de cada variável que foi obtida para cada MDP descrito (M e $\dot{M}$), além do tempo necessário para realizar o processo de otimização com o algoritmo MCASE passando por todos os *answer sets* obtidos. Vale lembrar que, por conta da grande quantidade de políticas encontradas para cada instância, foi requisitado ao resolvedor ASP (*lparse* e *smodels*) que retornasse apenas 50 *answer sets* para cada um dos 8 passos limites utilizados.

Tabela 3 – Quantidade de cada variável utilizada por cada processo de otimização para o problema do RCS do ônibus espacial

Variável	Sem restrições (M)	Com restrições ($\dot{M}$)	Taxa de redução ($\dot{M}/M$)
Políticas	3.200	400	12,5%
Estados	3.531	499	14,13%
Ações	584	161	27,57%
Tempo (seg)	35.115,19	2.530,19	7,205%

Fonte: Autor

A tabela 3 apresenta uma coluna com os valores obtidos para o MDP M e outra para os valores de $\dot{M}$, com a última coluna sendo a taxa de redução obtida pela comparação dos valores de $\dot{M}$ com M. As variáveis verificadas estão descritas nas linhas da tabela, sendo elas o número de políticas utilizadas para descrever o MDP (primeira linha), o número de estados obtidos para $\mathcal{S}$ e $\dot{\mathcal{S}}$ (segunda linha), o número de ações para os mesmos MDPs (terceira linha) e o tempo necessário para realizar a amostragem de todos os *answer sets* obtidos (última linha).

Observando a tabela é possível notar que houve redução tanto no conjunto de estados (de 3531 em M para 499 em $\dot{M}$) quanto no conjunto de ações (de 584 em M para 161 em $\dot{M}$). Para o caso do conjunto de estados é possível notar que a redução no espaço de busca é de $\mathcal{S}_M/\mathcal{S}_{\dot{M}} = {}^{3531}/_{499} = 14{,}13$ enquanto que para o conjunto de ações a redução é de $\mathcal{A}_M/\mathcal{A}_{\dot{M}} = {}^{584}/_{161} = 27{,}57$. Em relação ao tempo, pode-se notar que com o espaço de busca menor a política ótima foi obtida em tempo menor (7,205% do tempo necessário para M) do que com todo o espaço de busca de M.

Um outro resultado deste experimento são os valores aproximados para a função $Q(s,a)$ que foram obtidos tanto para M quanto para $\dot{M}$. Verificando o retorno obtido pela execução de todos os *answer sets* de cada MDP foi possível notar que, não somente as políticas ótimas

de $\dot{M}$ são as políticas ótimas de M, mas os valores de retorno obtido para todas as políticas são iguais nos dois MDPs, ou seja, os pares estado-ação possuem os mesmos valores para as políticas ótimas de ambos MDPs.

Além disto, nem todas as 400 políticas de $\dot{M}$ são ótimas segundo o critério utilizado. Observando os valores e as políticas (*answer sets*) observa-se o aparecimento de ações (`occurs(flip(X, Y), T)`) e consequências de ações (simplesmente `occurs(X,T)`), fazendo com que políticas que possuem a mesma quantidade de ações que um astronauta tenha que executar possam ter retornos diferentes por conta das consequências de cada ação. Isto é comprovado pelo fato de que as políticas ótimas estão no conjunto de *answer sets* obtidos com número máximo de passos fixado em 3. Dos 50 *answer sets* obtidos, 10 deles formam a política ótima (com retorno de -16) e as outras 40 políticas possuem retorno menor (entre -17 e -19). As políticas obtidas com um número de passos maior possuem retorno entre -24 e -60.

Neste caso, as políticas obtidas não foram comparadas diretamente com as soluções obtidas pelos desenvolvedores do *USA-Adivisor* (BALDUCCINI; GELFOND; NOGUEIRA, 2006; BALDUCCINI et al., 2001; BALDUCCINI et al., 2002; NOGUEIRA et al., 2001), uma vez que as soluções obtidas nestes trabalhos não tentam aproximar uma função valor-ação, mas utilizam o conhecimento de especialistas da USA para escolher qual é a melhor política. Porém, os autores do código citam as políticas com menor quantidade de ações como as que teriam preferência (no caso de não utilizar o conhecimento de um especialista) uma vez que estas podem diminuir o erro humano. Este resultado se compara ao obtido com a aproximação da função valor-ação uma vez que as políticas com apenas 3 passos formam o conjunto de políticas ótimas, porém não são todas as 50 políticas geradas que são ótimas.

Apesar de as 400 políticas que constituem o MDP $\dot{M}$ estarem contidas em M, é importante ressaltar que este espaço de busca não está completo, uma vez que para cada instância utilizada tem-se um grande número de políticas que podem resolver o problema. Por conta desta grande quantidade de soluções, torna-se inviável a utilização de todo o espaço de busca por conta do tempo necessário para encontrá-lo e realizar a busca pela solução ótima. Desta forma é possível que as proporções obtidas para cada variável tenham valores diferentes com o espaço de busca completo ou para outra configuração de estados iniciais.

Portanto, conforme esperado, este experimento apresenta a redução do espaço de busca como uma possível consequência da utilização dos *answer sets* para formalizar um MDP. Consequentemente, o tempo necessário para realizar a aproximação da função valor-ação para o MDP $\dot{M}$ é menor do que para M. Porém, a redução do espaço de busca nem sempre é garantida, como é discutido na seção 7.1 do próximo capítulo.

6.3 DOMÍNIO DO MUNDO DE GRADES NÃO-ESTACIONÁRIO

Os resultados obtidos para cada um dos três testes realizados no domínio do mundo de grades encontram-se na figura 15. Os gráficos mostram a diferença quadrática em cada episódio para os quatro algoritmos utilizados: Q-Learning, SARSA, ASP(Q-Learning) e ASP(SARSA), sendo a diferença quadrática definida como:

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{m=1}^{n} Q_{m,t}(s,a) - Q_{m,t-1}(s,a)}{n}}, \tag{13}$$

em que, para cada instante de tempo t os m valores de $Q_m(s,a)$ de cada ação a que pode ser realizada em cada estado s são comparados com o valor do mesmo par no instante $t-1$ anterior.

É importante ressaltar que Q-Learning e SARSA aproximam a função utilizando o MDP M e que ASP(Q-Learning) e ASP(SARSA) aproximam o MDP $\dot{M}$ obtido a partir de *answer sets* de M. Os gráficos na coluna da esquerda da figura 15 apresentam todos os 200 episódios realizados, enquanto a coluna da direita possui os gráficos com apenas os cinco primeiros pontos de cada fase do aprendizado a fim de facilitar a visualização da região em que uma mudança no domínio ocorre. Estes resultados foram obtidos de 30 testes executados com cada um dos algoritmos. Para este experimento foi utilizada a taxa de aprendizado $\alpha = 0{,}2$, o desconto em $\gamma = 0{,}9$ e a taxa de exploração-explotação em $\epsilon = 0{,}1$.

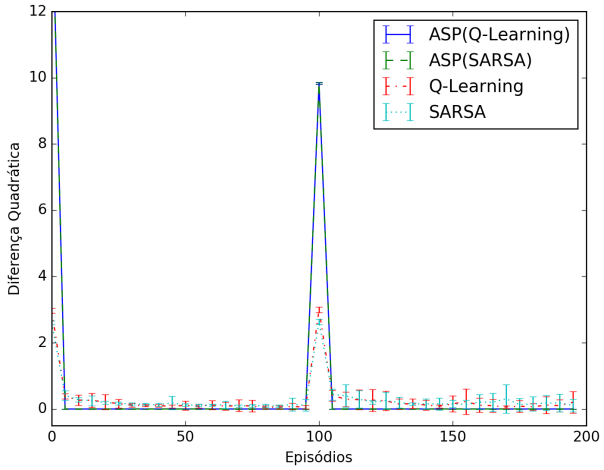
Observando os gráficos das figuras 15a e 15b é possível notar que, assim como no experimento anterior, os algoritmos que aproximam a função valor-ação de $\dot{M}$ (algoritmos ASP(Q-Learning) e ASP(SARSA)) necessitam de menos tempo para encontrar a política ótima do que quando utilizado o Q-Learning e o SARSA somente. Isto se deve pela diminuição do espaço de busca ocorrido, em que as ações que não levam ao estado objetivo foram removidas dos estados que o agente visita. Os estados que não possuem ações que levam ao estado objetivo com o menor número de passos também não fazem parte do MDP $\dot{M}$.

Um outro dado que pode ser extraído é que os algoritmos de ASP(RL) apresentam desvio padrão 0 para todos os pontos, isto se deve pelo fato de que, mesmo quando o algoritmo de Aprendizado por Reforço utilizado escolhe uma ação aleatória, esta ação não deixa de ser ótima no domínio utilizado e de levar o agente para o estado final, desta forma as possíveis políticas que podem ser realizadas com ASP(RL) possuem o mesmo valor de retorno.

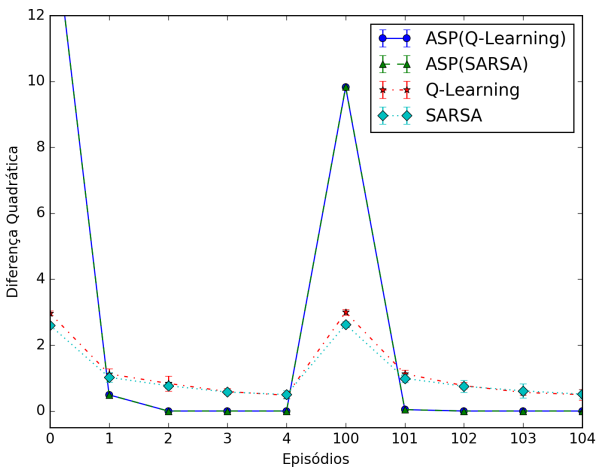
Considerando as mudanças realizadas no domínio durante a aproximação de $Q(s,a)$, é possível notar no episódio 100 que a mudança da função valor-ação dos métodos de ASP(RL)

Figura 15 – Diferença quadrática em cada episódio para os quatro algoritmos testados

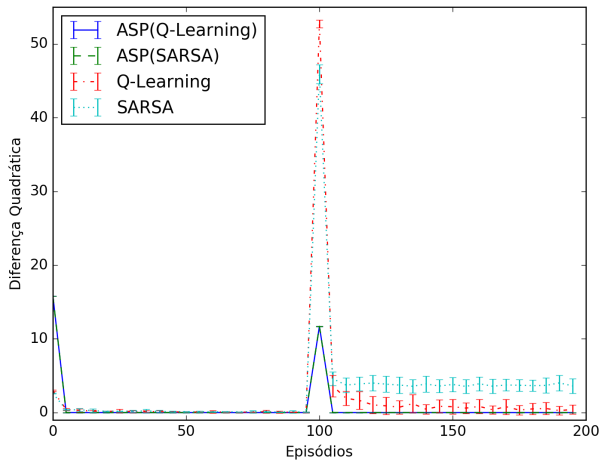
(a) Valores obtidos para o primeiro teste



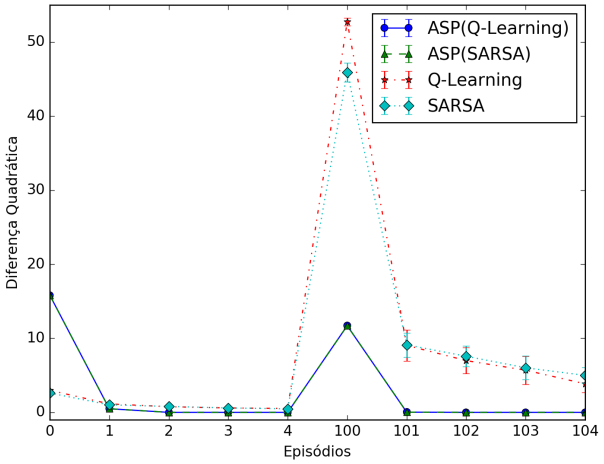
(b) Apenas os cinco primeiros pontos de cada domínio no primeiro teste



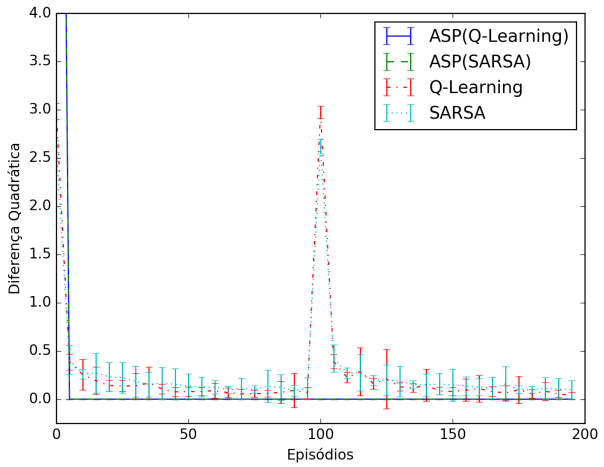
(c) Valores obtidos para o segundo teste



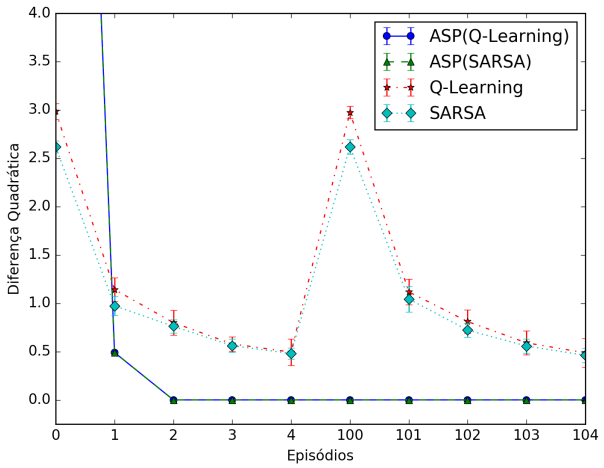
(d) Apenas os cinco primeiros pontos de cada domínio no segundo teste



(e) Valores obtidos para o segundo teste



(f) Apenas os cinco primeiros pontos de cada domínio no segundo teste



Fonte: Autor

produz um valor mais alto quando comparado aos outros valores destes algoritmo. Isto se deve pela mudança realizada na função Q que é diferente entre os mapas, fazendo com que a comparação da função entre os dois mapas gere um valor alto por conta desta diferença. Independente do valor observado no episódio 100, a alteração do ambiente apenas utilizando os *answer sets* de cada fase do mundo de grades foi realizado e o agente conseguiu continuar a aproximação de $Q(s,a)$ a partir dos valores anteriores, sem a necessidade de reiniciar todo o processo.

Nos resultados do segundo experimento observa-se o mesmo comportamento do primeiro, em que ASP(RL) apresenta um tempo de aproximação da função valor-ação menor do que os métodos de RL, sendo que também existe um valor maior do que os demais no episódio 100. Assim como no primeiro teste, é possível observar que a transição entre os mundos de grades foi realizada sem a necessidade de reiniciar o aprendizado.

O terceiro teste apresenta resultados semelhantes aos dois anteriores, porém com a diferença da ausência da mudança de valor no episódio 100. Isto se deve ao fato de que o segundo mundo de grades deste experimento (figura 12d) possui apenas um *answer set* que resolve o problema, e que os algoritmos de ASP(RL) executariam apenas esta política, convergindo rapidamente.

A tabela 4 apresenta em milissegundos o tempo necessário para a execução de cada parte dos testes realizados até que o valor da diferença quadrática seja menor ou igual a 10^{-5}. Nesta tabela são apresentados os tempos do Q-Learning e SARSA, o tempo necessário para realizar a aproximação da função valor-ação utilizando ASP(Q-Learning) e ASP(SARSA), desconsiderando o tempo necessário para encontrar os *answer sets*, além do tempo de execução somente do ASP. O tempo total do ASP(Q-Learning) e ASP(SARSA) é obtido pela soma do tempo do ASP com o valor fornecido para cada um destes algoritmos.

Tabela 4 – Tempo necessário (em segundos) para os algoritmos aproximarem a função valor-ação até chegar a uma diferença quadrática de $DQ \leq 10^{-5}$

Algoritmos	Mapa 12a	Mapa 12b	Mapa 12c	Mapa 12d
Q-Learning	$156{,}63 \pm 38{,}80$	$159{,}30 \pm 31{,}42$	$168{,}74 \pm 46{,}36$	$151{,}95 \pm 32{,}14$
SARSA	$406{,}06 \pm 1028{,}51$	$249{,}78 \pm 697{,}36$	> 1 minuto	$132{,}56 \pm 38{,}03$
ASP(Q-Learning)	$0{,}41 \pm 0{,}03$	$0{,}43 \pm 0{,}03$	$0{,}44 \pm 0{,}03$	$0{,}20 \pm 0{,}01$
ASP(SARSA)	$0{,}42 \pm 0{,}03$	$0{,}43 \pm 0{,}04$	$0{,}44 \pm 0{,}03$	$0{,}21 \pm 0{,}01$
ASP	$4{,}71 \pm 0{,}01$	$1{,}38 \pm 0{,}01$	$0{,}17 \pm 0{,}005$	$0{,}10 \pm 0{,}002$

Fonte: Autor

É possível notar as mesmas características observadas nos gráficos deste experimento e também no experimento anterior; ASP(RL) necessita de menos tempo para convergir do que os

algoritmos de RL, mesmo quando este valor é adicionado ao tempo utilizado para encontrar as políticas ótimas. Em relação ao tempo do ASP, é possível notar que o tempo necessário para encontrar todas as políticas que possuem o menor número de ações é proporcional à quantidade de estados que podem ser visitados pelo agente.

Uma última observação importante é a comparação entre os resultados obtidos para o ASP(Q-Learning) e ASP(SARSA), em que ambos apresentam as mesmas políticas ótimas com os mesmos valores na função $Q(s,a)$ aproximada e que ambos necessitam do mesmo tempo para realizar este processo de aproximação.

Apesar de que ASP(Q-Learning) utiliza um algoritmo *off-policy* e que ASP(SARSA) utiliza um algoritmo *on-policy*, neste caso em que foram removidas ações que levariam o agente a receber alguns reforços (cair em um buraco ou bater em uma parede), um algoritmo *off-policy* apresentou o mesmo comportamento durante a aproximação de $Q(s,a)$ de um algoritmo *on-policy*.

Em relação ao retorno obtido pelos algoritmos de ASP(RL), as políticas encontradas executam 18 ações, sendo a última a que leva ao estado objetivo. Desta forma, o retorno obtido em todos os episódios para os algoritmos de ASP(RL) em qualquer um destes experimentos é de:

$$R(\pi) = R(\pi^*) = r_{s_g} + (n-1) * r_s = +100 + (17 * (-1)) = 83, \tag{14}$$

em que r_{s_g} é o reforço fornecido ao chegar ao estado objetivo, r é o reforço fornecido ao realizar alguma ação que não leve ao estado objetivo e n é o número de passos necessários para alcançar o estado objetivo.

Portanto, este experimento demonstra que ASP(MDP) pode ser utilizado em domínios não-estacionários uma vez que é possível tratar da mudança que ocorre no domínio utilizando ASP. Foi possível observar também que, neste caso em especial em que o algoritmo é determinístico, um algoritmo *on-policy* apresenta o mesmo comportamento durante o processo de aproximação da função $Q(s,a)$ que um algoritmo *off-policy*, necessitando do mesmo tempo de execução e fornecendo a mesma função aproximada. Isto ocorre pelo fato de que os dois algoritmos recebem os mesmos valores de reforço como consequência das ações executadas, fazendo com que a atualização da função Q seja a mesma para ambos.

6.4 DOMÍNIO DO MUNDO DE GRADES NÃO-DETERMINÍSTICO

Os gráficos apresentados na figura 16 mostram os retornos obtidos em cada uma das três situações descritas na seção 5.4 para os quatro algoritmos testados: ASP(Q-Learning), ASP(SARSA), Q-Learning e SARSA. As configurações utilizadas neste experimento seguem as mesmas da seção anterior com a única diferença sendo o não-determinísmo do mundo de grades. Para a mesma configuração, os gráficos contidos na figura 17 apresentam a quantidade de passos necessária para cada um dos quatro algoritmos nas três situações utilizadas.

A partir destes gráficos é possível notar a influência do ASP no método de RL utilizado para encontrar a política ótima e também a redução do espaço de busca, quando esta ocorre.

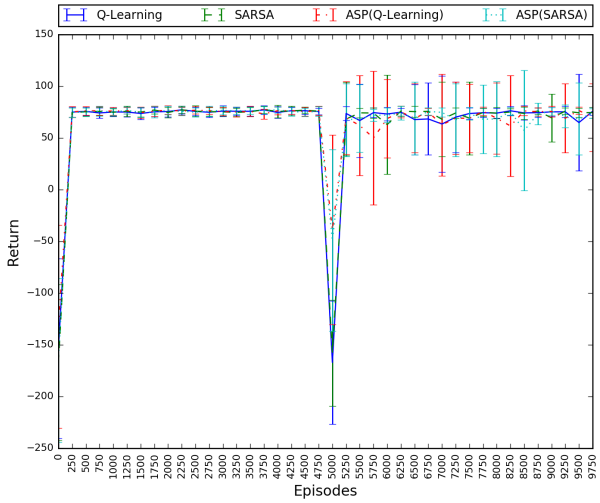
Para a primeira situação, em que o a mudança realizada é do mapa da figura 12a para o mapa da figura 12b, as figuras 16a e 16b apresentam o retorno obtido a cada episódio e as figuras 17a e 17b apresentam o número de passos. É possível notar que para o primeiro mapa a redução no espaço de busca é pequena e, portanto, a diferença entre um método de ASP(RL) e apenas RL não é relevante. Entretanto, quando ocorre mudança no mundo de grades (no episódio 5000), percebe-se que a função aproximada pelo agente para o primeiro mapa auxilia na aproximação da função para o segundo mapa, de forma que os métodos de ASP(RL) apresentam desempenho melhor (menor número de passos e maior retorno) nos primeiros episódios do que os métodos de RL utilizados. Desta forma, conclui-se que o ASP(RL) permite que, mesmo em ambientes não-determinísticos, o agente possa encontrar a política ótima de forma mais rápida.

Entretanto, observando os resultados obtidos para a segunda situação (mudança do mapa da figura 12a para o mapa da figura 12c) percebe-se que a redução do espaço de busca e a aceleração do processo de aprendizado não é sempre garantida. As figuras 16c e 16d apresentam os retornos obtidos para cada algoritmo, enquanto que as figuras 17c e 17d apresentam o número de passos. É possível notar nestas figuras que, mesmo com a utilização dos *answer sets*, os métodos de ASP(RL) possuem desempenho igual ou muito próximo aos que utilizam somente RL. Observando os primeiros episódios do ASP(RL) após as alterações terem sido realizadas no mundo de grades, percebe-se que a função aprendida no mapa anterior auxilia o agente na aproximação da nova função valor-ação uma vez que o retorno obtido nestes primeiros episódios possui valor superior aos retornos obtidos para os métodos de RL, apesar de que ao observar o número de passos, nota-se que os quatro algoritmos apresentam o mesmo desempenho.

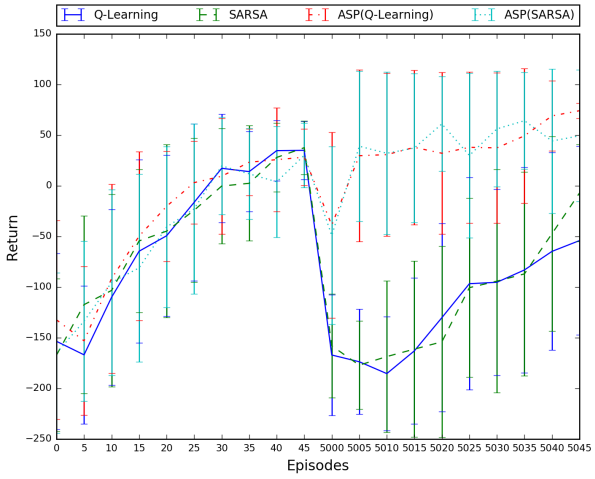
Os retornos obtidos para a última situação encontram-se nas figuras 16e e 16f. O número de passos para cada algoritmo encontram-se nas figuras 17e e 17f. Nestes gráficos nota-se, assim como na primeira situação, que a utilização da função valor-ação aproximada para o primeiro

Figura 16 – Retorno obtido para os quatro obtidos testados

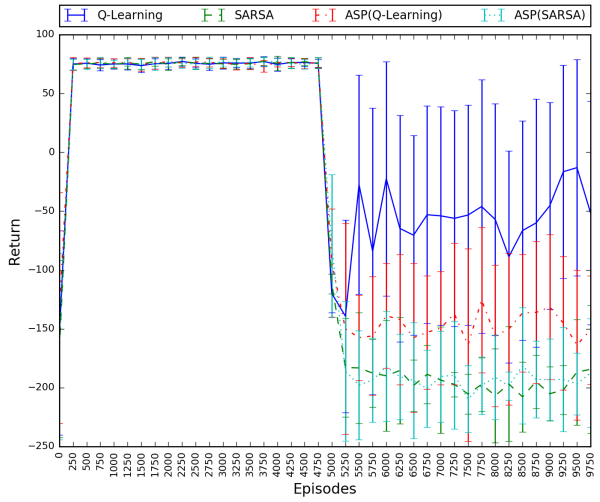
(a) Valores obtidos para o primeiro teste



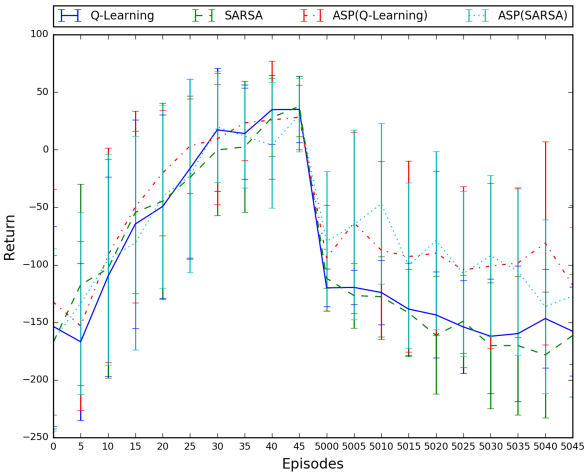
(b) Apenas os cinco primeiros pontos de cada domínio no primeiro teste



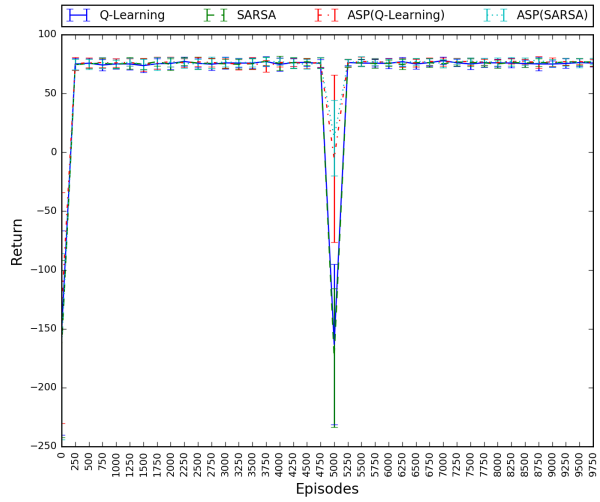
(c) Valores obtidos para o segundo teste



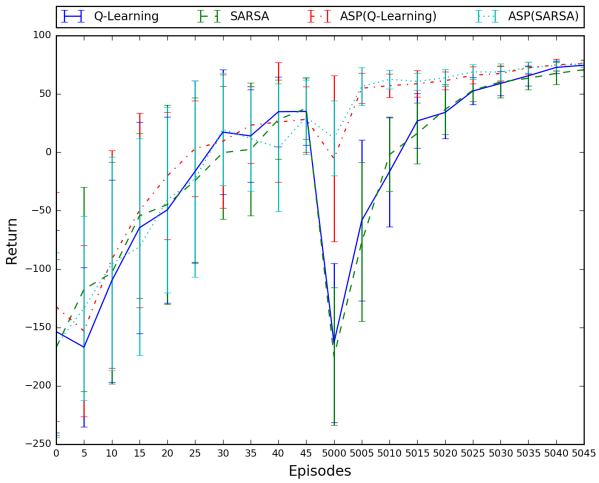
(d) Apenas os cinco primeiros pontos de cada domínio no segundo teste



(e) Valores obtidos para o segundo teste



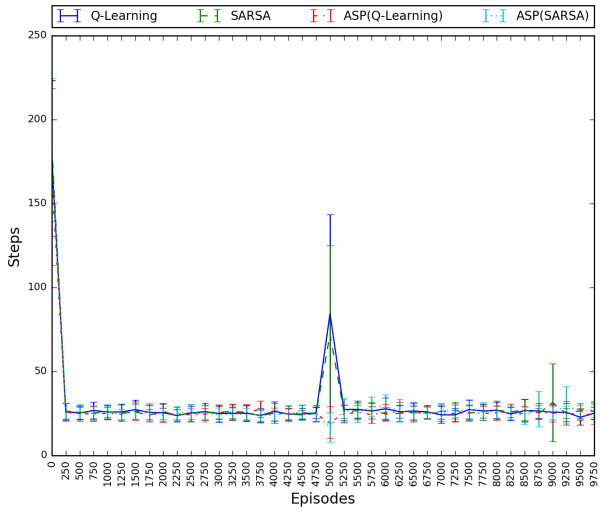
(f) Apenas os cinco primeiros pontos de cada domínio no segundo teste



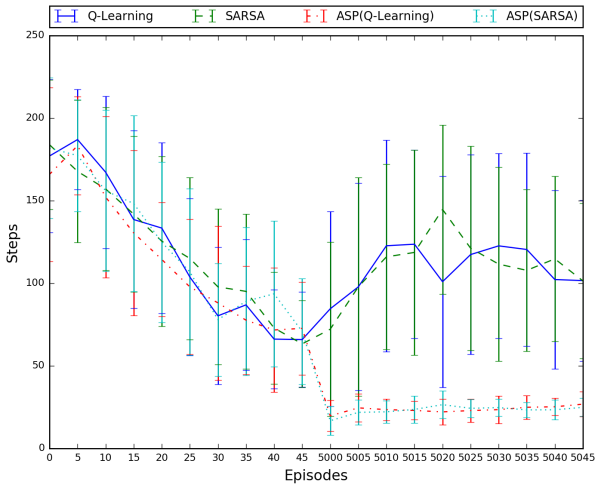
Fonte: Autor

Figura 17 – Número de passos em cada episódio para os quatro algoritmos testados

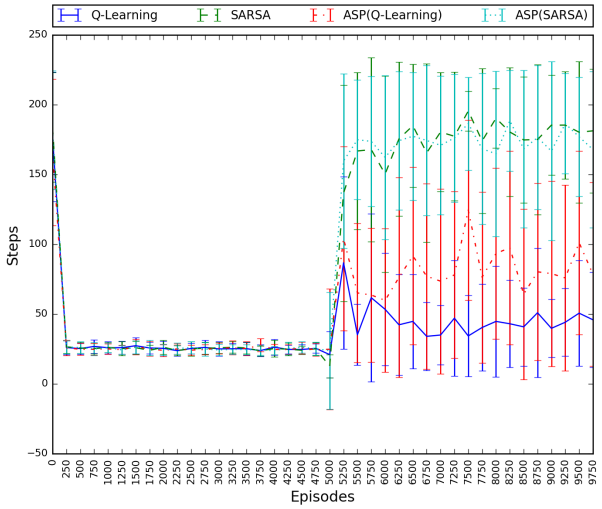
(a) Valores obtidos para o primeiro teste



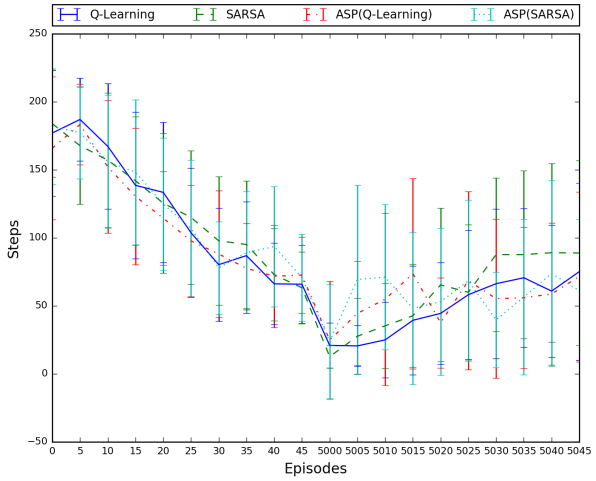
(b) Apenas os cinco primeiros pontos de cada domínio no primeiro teste



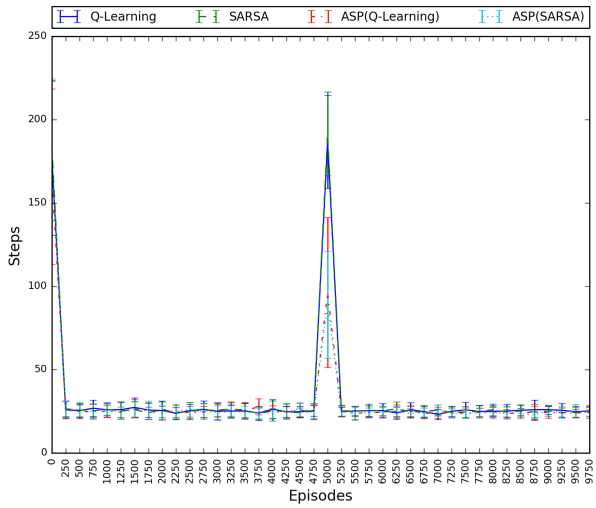
(c) Valores obtidos para o segundo teste



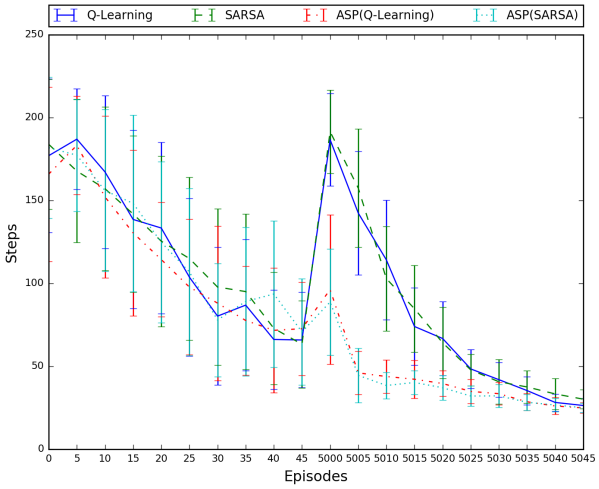
(d) Apenas os cinco primeiros pontos de cada domínio no segundo teste



(e) Valores obtidos para o segundo teste



(f) Apenas os cinco primeiros pontos de cada domínio no segundo teste



Fonte: Autor

mapa auxilia na aproximação da função para o segundo mapa, fazendo com que o ASP(RL) tenha desempenho superior ao RL nos episódios iniciais, convergindo mais rapidamente à política ótima do problema que se deseja resolver.

Portanto, é possível notar que o ASP(RL) pode ser utilizado em problemas não-determinísticos e não-estacionários, porém, como descrito nos lema 1 e no teorema 1, a redução do espaço de busca não é garantida, mas a política ótima do MDP pode ser encontrada tanto com RL quanto com ASP(RL).

Este capítulo apresentou os resultados específicos de cada experimento proposto. O próximo capítulo apresenta uma discussão dos resultados em relação ao algoritmo ASP(MDP) proposto.

7 DISCUSSÃO

Os experimentos do capítulo anterior apresentam os resultados obtidos utilizando o ASP(MDP) para descrever um MDP a partir de *answer sets*. Entretanto, estes experimentos também representam algumas situações específicas do uso do ASP(MDP).

7.1 CASOS ESPECÍFICOS DA UTILIZAÇÃO DO ASP(MDP)

Quando ASP é utilizado para obter a descrição dos conjuntos de estados e ações, alguns casos específicos podem ocorrer, como quando

a) não há nenhuma política que resolve o problema;

b) só existe uma política que resolve o problema;

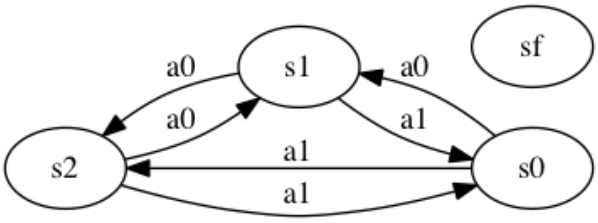
c) a utilização de ASP não reduz o espaço de busca do MDP.

Cada um destes casos será descrito nas próximas seções e a tabela 5, que se encontra ao final dos casos, apresenta o tempo necessário para cada situação.

7.1.1 Não há solução para o MDP

Considerando o MDP da figura 18, percebe-se que não há política que atinge o estado final `sf` a partir de nenhum outro estado existente. Considerando este como o MDP inicial (antes da utilização de ASP), obtém-se resultados diferentes entre a utilização somente de um método de otimização e ASP junto com otimização.

Figura 18 – MDP sem política que chega ao estado final



Fonte: Autor

Dado um problema de planejamento em que ASP fornece a sequência de estados e ações a partir do estado inicial até o estado final, caso não exista nenhuma solução possível, ASP

retornará UNSATISFIABLE e sabe-se que não há como resolver o problema. Portanto, para o MDP da figura 18 a utilização de ASP para obter sua representação retornará que não é possível chegar ao estado final do problema, não sendo necessário utilizar um método de otimização.

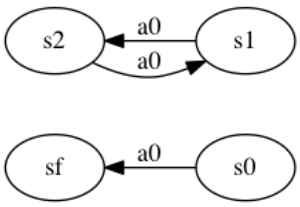
Se apenas a otimização for realizada no MDP da figura 18, será gasto tempo na otimização de uma função que não possui solução (apesar deste fato não ser conhecido a priori). Desta forma, será encontrada uma política ótima para um problema que não possui solução (política ótima que não resolve o problema, mas é ótima segundo o critério de optimalidade e a função de reforço). Neste caso, mesmo que neste MDP nunca se chegue a sf, π^* é encontrada pois depende da função de recompensa $\mathcal{R}$ e o objetivo neste caso é encontrar a política ótima como a sequência de ações que maximiza ou minimiza $\mathcal{R}$.

Portanto, no caso do MDP da figura 18, utilizar somente o ASP, fornecerá no tempo T_{ASP} a resposta de que não há solução para o MDP, enquanto se utilizar apenas um método de otimização necessitaria de tempo T_o para não encontrar uma política que chega ao estado final, uma vez que o ASP(MDP) utiliza ASP para resolver o problema de atingibilidade (ÄSTRÖM; MURRAY, 2008) do estado final antes de realizar o processo de busca pela solução ótima.

7.1.2 Só há uma solução para o MDP

Um caso similar ao anterior em relação ao tempo necessário para encontrar a solução ocorre quando o MDP possui apenas uma solução, como o MDP da figura 19, porém isto não é conhecido a priori.

Figura 19 – MDP com apenas uma política que chega ao estado final



Fonte: Autor

No caso de se utilizar ASP, este retornará apenas uma solução ($\langle$s0, a0, sf$\rangle$), para o MDP da figura 19, pois nenhuma outra ação em nenhum outro estado resolve o problema. Utilizando algum método de otimização, seria necessário testar todas as opções e atualizar a função valor-ação para todas as variáveis para encontrar a política ótima $\pi^* =$ a0, ou seja, a ação a0 deve ser executada no estado s0, levando a sf. Portanto, quando só há uma solução,

ASP retornará esta solução e, assim como no caso anterior, não há a necessidade de realizar o processo de otimização, pois se só há uma solução possível, esta é a política ótima.

Se utilizássemos apenas algum método de otimização, teríamos os valores para todo o MDP e somente após a aproximação da função valor-ação seria obtida a única política capaz de resolver o problema, porém este fato só seria descoberto em tempo infinito, da mesma forma que o caso anterior.

Portanto, quando há apenas uma solução para o MDP, o tempo de aprendizado se torna igual a quando não há solução para o MDP. Ou seja, utilizando apenas um método de otimização seria necessário o tempo T_o para encontrar a solução do problema, enquanto que utilizando ASP o processo de otimização deixa de ser necessário e o tempo necessário é T_{ASP}.

Um exemplo de problema em que esta situação ocorre encontra-se nas terceiras variações dos experimentos do mundo de grade não-estacionário (seções 5.3 e 5.4 respectivamente) em que ao ocorrer alterações no mundo de grades, apenas uma política leva o agente do estado inicial ao estado final.

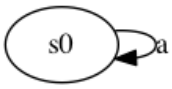
7.1.3 Não ocorre redução no espaço de busca

Como mostrado no teorema 1 e no lema 1, considerando que se está saindo do MDP M no qual se tem descrição de todas as possibilidades para um MDP $\dot{M}$ no qual apenas as soluções que levam ao estado objetivo estão descritas, uma das possíveis consequências é o espaço de busca de $\dot{M}$ ser menor que o de M.

Entretanto, não é sempre que um MDP pode ser reduzido. Um exemplo simples é um MDP com apenas um estado e uma ação, como o da figura 20. Neste caso, não há como reduzir o MDP e este continuará sendo o mesmo após a utilização do ASP, considerando que `s0` é o estado final. É importante ressaltar neste caso a consequência de não haver redução no espaço de busca. Caso isto ocorra, o tempo T_o necessário para executar o processo de otimização será o mesmo com ou sem a utilização de ASP, porém quando o ASP for utilizado antes o tempo total de execução será incrementado pelo tempo T_{ASP} do ASP. Ou seja, se a otimização necessita de T_o para executar, e ASP necessita de T_{ASP}, encontrar a política ótima para o MDP demora T_o e ASP(MDP) necessita de $T_o + T_{\mathrm{ASP}} > T_o$.

Um exemplo desta situação pode ser observada nos resultados obtidos para o primeiro mapa do quarto experimento (seção 6.4), em que a redução no conjunto de estados e ações é pequena. Desta forma, o desempenho dos algoritmos ASP(Q-Learning) e ASP(SARSA) são similares aos dos algoritmos Q-Learning e SARSA.

Figura 20 – MDP com apenas um estado e uma ação



Fonte: Autor

A tabela 5 resume os tempos que são necessários para encontrar a solução ótima para cada um destes três casos apresentados, além do tempo para o caso em que ocorre a redução do espaço de busca e existem mais do que um *answer set* que resolve o problema. Nesta tabela T_o é o tempo necessário para executar um método de otimização que encontra a solução ótima, T_{ASP} é o tempo necessário para encontrar todos os *answer sets* existentes. Os valores para os tempos T_o e T_{ASP} dependem do domínio que está sendo utilizado e não há como garantir uma relação entre eles para todos os casos.

Tabela 5 – Tempos para encontrar a solução ótima para cada um dos casos apresentados

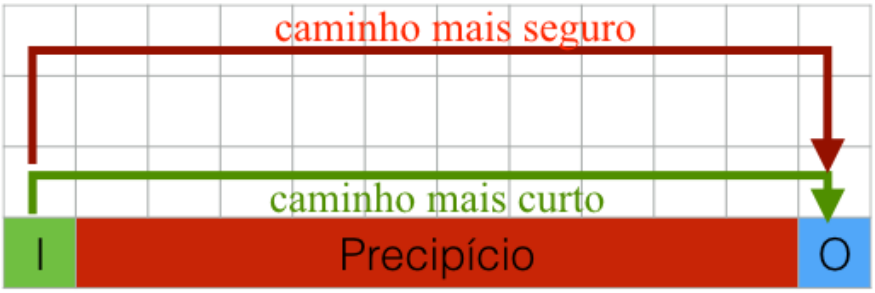
Caso	Otimização	ASP + Otimização
Nenhuma política	T_o	T_{ASP}
Apenas uma política	T_o	T_{ASP}
Sem redução do MDP	T_o	$T_{\text{ASP}} + T_o$
Com redução do MDP	T_o	$T_{\text{ASP}} + T'_o$, com $T'_o \leq T_o$

Fonte: Autor

Como é possível notar pela tabela 5, os casos em que não se tem uma política que resolve o problema ou quando só se tem uma política, são os melhores casos, pois a resposta do ASP é suficiente para saber qual é a solução ótima do problema, quando ela existe. O pior caso para o ASP(MDP) ocorre quando não há redução no espaço de busca, pois além do tempo necessário para encontrar os *answer sets*, é necessário realizar a busca pela política ótima em todo o espaço de busca.

O tempo necessário para realizar o processo utilizado para encontrar a política ótima de um MDP não a única mudança que necessita ser estudada na utilização do ASP(MDP). A utilização de métodos *on-policy* e *off-policy* também necessita de verificação.

Figura 21 – Domínio do precipício apresentado por Sutton e Barto (2015)



Fonte: Autor

7.2 RESULTADO DE MÉTODOS *ON-POLICY* E *OFF-POLICY* APÓS UTILIZAÇÃO DO ASP(MDP)

Como proposto no experimento do mundo de grades (seção 5.3), a utilização do ASP para encontrar os conjuntos de estados e ações pode fazer com que certas partes do domínio não sejam exploradas, de forma que a utilização de um método de aproximação da função valor-ação que realize a atualização de seus valores dependendo da política atual (*on-policy*) possa encontrar uma política diferente daquela que seria encontrada caso o domínio fosse explorado por completo (*off-policy*).

A diferença entre a política ótima encontrada por um método *on-policy* e a de um método *off-policy* é descrita no estudo do domínio do precipício apresentado por Sutton e Barto (2015) de forma que, por conta da forma que é realizada a atualização da função valor-ação, as políticas ótimas aprendidas por cada método são diferentes. O domínio do precipício descrito por Sutton e Barto (2015) apresenta um problema em que o agente deve encontrar o melhor caminho entre o estado inicial (quadrado com a letra “I”) e o final (quadrado com a letra “O”) em um mundo de grades, porém com um precipício (retângulo vermelho) entre estes estados, como mostra a figura 21. Neste domínio, a utilização de um método *on-policy*, como o SARSA, encontra como política ótima o caminho mais seguro, enquanto que um método *off-policy* como o Q-Learning, encontra como caminho ótimo o caminho mais curto.

O que é possível notar nos resultados do experimento da seção 5.3 é o fato de que as respostas obtidas pelo SARSA (*on-policy*) e pelo Q-Learning (*off-policy*) são as mesmas por conta do espaço de estado visitado pelo agente. Portanto, dependendo do domínio em que se procura a solução ótima, os *answer sets* obtidos podem remover alguns pares estado-ação

por conta das restrições existentes no programa lógico que descreve o MDP. Desta forma a atualização feita por um método *on-policy* não percorre todas as possibilidades, mas apenas aquelas que levam à solução, fazendo com que os valores da função valor-ação sejam diferentes dos valores que seriam obtidos caso todos os pares estado-ação estivessem no MDP. No caso de um método *off-policy*, não há alteração nos valores aprendidos, uma vez que a atualização da função valor-ação é feita a partir dos valores existentes na tabela e não da política que está sendo executada.

Portanto, a utilização do ASP antes de um método de otimização não altera a função valor-ação de um método *off-policy*, porém faz com que um método *on-policy* possa encontrar a mesma política ótima que um método *off-policy* como consequência do espaço de estados que será visitado.

7.3 POLÍTICA ÓTIMA E *ANSWER SET* ÓTIMO

Seguindo o princípio da optimalidade de Bellman (definição 1), uma política ótima escolhe a melhor ação para um estado de forma a maximizar (ou minimizar) uma função de recompensa. Portanto, a política ótima é capaz de fornecer a melhor ação em todos os estados que são possíveis de serem visitados.

É possível extrair uma sequência de ações de um *answer set* fornecendo uma política que inicia no estado inicial e termina no estado final. Porém esta política extraída percorre apenas um subconjunto dos estados contidos em $\mathcal{S}$, não fornecendo a melhor ação para todos os estados.

Considerando que um *answer set* é capaz de levar do estado inicial s_0 ao estado final s_g, a política π_{as} fornecida por este possui um retorno $R_{as} = \sum_{s=s_0}^{s_g} r(s,a)$ em que a é fornecida por π_{as} para cada estado. Portanto, o *answer set* que fornece a política com maior (ou menor) retorno é o *answer set* ótimo.

Entretanto, como um *answer set* só fornece a resposta para uma parte dos estados em $\mathcal{S}$, a política π_{as} ótima não é a mesma que a política ótima, porém é uma parte da política ótima. Desta forma, o único caso em que π_{as} ótimo é a política ótima é o caso em que só existe uma política que é capaz de levar do estado inicial s_0 ao estado final s_g.

Considerando um problema que apresenta diversos *answer sets* como resposta, se mais do que um deles apresentar o mesmo retorno R_{as}, então cada um destes é parte da política ótima. Porém, não é possível garantir que a política ótima é formada pela união das políticas

fornecidas por todos estes uma vez que não é garantido que os *answer sets* ótimos percorrem todos os estados existentes no MDP.

Por exemplo, no domínio do mundo de grades (seção 5.3) o último mapa (figura 12d) apresenta o caso em que só existe um *answer set* e, portanto, fornece a política ótima. O primeiro mapa (figura 12a) apresenta um caso em que os *answer sets* passam por todos os estados do MDP, desta forma a política ótima é dada pela união de todos os π_{as}. Nos outros dois casos (figuras 12b e 12c) os *answer sets* não passam por todos os estados existentes no mapa, porém levam do estado inicial ao estado final, portanto, as π_{as} fazem parte da política ótima se considerado que existe a possibilidade de chegar aos outros estados. Entretanto, se o mundo de grades for não-determinístico no programa lógico, os *answer sets* passarão por estes estados como consequência do não-determinismo do domínio e tem-se todas as políticas ótimas para o domínio a partir das π_{as} extraídas dos *answer sets*.

No domínio do ônibus espacial (seção 5.2) tem-se *answer sets* com retornos diferentes. Caso este domínio fosse não-determinístico seria possível que estados que estão em *answer sets* sub-ótimos fossem visitados por conta da chance de visitar um outro estado que não esteja nas políticas π_{as}.

Portanto, um *answer set* ótimo não é equivalente à política ótima, mas o conjunto de *answer sets* ótimos constituem parte ou toda a política ótima, dependendo de quantos estados do MDP os *answer sets* visitam.

7.4 RELAÇÕES ENTRE MDP ESTACIONÁRIOS E NÃO-ESTACIONÁRIOS

Como mostra a figura 1, neste trabalho um MDP não-estacionário é considerado como uma sequência de MDPs estacionários de horizonte finito em que após um determinado período os conjuntos de estados e ações podem sofrer alterações. Neste caso, existem duas situações que devem ser estudadas: quando o horizonte do MDP estacionário é de apenas um episódio e quando ele é infinito.

No primeiro caso, quando apenas um episódio de interação existe antes do MDP ser alterado, torna-se mais complexa a busca pela solução ótima por conta do horizonte ser muito pequeno e não possuir tempo suficiente para realizar a busca por completo. Entretanto, se o MDP não sofrer uma mudança completa (ou quase completa), mas manter uma relação com o MDP do episódio anterior, então pode ser possível realizar a busca pela solução ótima utilizando um algoritmo que trate as mudanças que ocorrem no MDP.

O segundo caso trata apenas de um MDP estacionário com horizonte infinito, ou seja, o MDP deixa de ser não-estacionário. Como os experimentos demonstram, é possível utilizar o ASP(MDP) em um domínio estacionário com horizonte finito.

Portanto, o ASP(MDP) pode ser utilizado independente do horizonte do MDP estacionário que descreve o MDP não-estacionário.

Este capítulo discutiu situações específicas em que o ASP(MDP) pode obter resultados específicos dependendo do domínio. O próximo capítulo apresenta as conclusões deste trabalho.

7.5 HORIZONTE E CONVERGÊNCIA

Considerando um MDP não-estacionário conforme a figura 1, é importante ressaltar a influência do horizonte nos períodos em que tem-se um MDP estacionário.

Uma vez que o ASP(MDP) somente apresenta uma forma de encontrar a descrição do ambiente e a busca pela solução ótima depende do método escolhido, o horizonte necessário para garantir que a solução ótima é encontrada não depende apenas do ASP(MDP), mas também do método escolhido.

Desta forma, um método que necessita de poucas iterações para encontrar a solução ótima também necessitará de um horizonte pequeno para encontrar a solução de um MDP não-estacionário cuja descrição foi obtida com a utilização do ASP(MDP). Da mesma forma, um método que necessita de mais iterações para encontrar a solução ótima de um MDP necessitará de um horizonte maior para encontrar a solução utilizando ASP(MDP). Porém, é importante ressaltar que o horizonte necessário para obter a solução ótima não depende apenas do método utilizado, mas também da complexidade do problema.

Além da escolha do método, um outro fator que altera a garantia de convergência é o próprio horizonte. Caso o horizonte do MDP estacionário seja infinito, se o método escolhido para obter a solução ótima garantir a convergência, então a sua utilização junto ao ASP(MDP) também terá a convergência garantida. Porém, se o horizonte for pequeno (apenas um episódio, por exemplo) então não é possível garantir que a solução é ótima, uma vez que não é possível explorar todo o espaço de busca em apenas um episódio. Entretanto, considerando que o MDP é não-estacionário, caso somente parte deste MDP seja alterada a cada horizonte, é possível que a solução ótima seja encontrada após uma certa quantidade de episódios, uma vez que pequenas alterações em cada horizonte permitem que o espaço de busca seja explorado.

8 CONCLUSÃO

Este trabalho propôs o ASP(MDP), uma forma de combinar *Answer Set Programming* com Processos Markovianos de Decisão de forma que os *answer sets* encontrados como solução para um problema lógico forneçam a descrição de um MDP do qual se busca a solução ótima. Para encontrar a política ótima desta forma foram propostos dois algoritmos que aproximam a função valor-ação utilizando métodos diferentes: o MCASE, que utiliza Avaliação de Política de Monte Carlo para realizar a aproximação, e o ASP(RL), que utiliza aprendizado por reforço para interagir com o ambiente e realizar a aproximação utilizando Aprendizado por Reforço.

Quatro experimentos foram realizados. O primeiro trata da submissão de um processo a servidores que possuem diferentes probabilidades de falha e velocidades de processamento, o que permitiu verificar que é possível utilizar ASP para encontrar a descrição de um MDP. O segundo tem como objetivo realizar uma manobra no ônibus espacial usando um sistema de controle por reação, demonstrando a relação entre um MDP M que descreve o problema e o MDP $\dot{M}$ definido através dos *answer sets* encontrados para M. Neste segundo experimento foi possível verificar que $\dot{M}$ necessitou de menos estados, ações e tempo de execução para encontrar a mesma solução de M. O terceiro se passa em um mundo de grades não-estacionário em que paredes e buracos aparecem e desaparecem ao longo do tempo. Estes três experimentos permitiram testar se é possível utilizar o ASP(MDP) em um ambiente não determinístico (primeiro e quarto experimentos), a redução no espaço de busca que ocorre pela utilização do ASP(MDP) (segundo experimento) e se ASP(MDP) é tolerante a elaboração e permite a busca por soluções de MDPs não-estacionários (terceiro experimento).

Os resultados obtidos para os três experimentos propostos mostram que, no caso médio, o ASP(MDP) é mais rápido do que somente a utilização de um algoritmo que aproxime a função valor-ação de um MDP. Entretanto, no pior caso, em que não ocorre diminuição no espaço de busca, e é necessário realizar tanto a busca com ASP quanto o processo de otimização, o ASP(MDP) levará mais tempo para encontrar a política ótima. Da mesma forma, no melhor caso (em que não há solução para o MDP ou que só há uma política que resolve o problema) o ASP(MDP) será mais rápido que somente aproximar a função $Q(s,a)$ de um MDP por não ser necessário realizar esta aproximação.

Uma propriedade constatada no experimento 5.3 na utilização do ASP(MDP) é a possibilidade de interagir com domínios não-estacionários e que podem ter seus conjuntos de estados e ações modificados ao longo do tempo. A proposta realiza a modificação da função valor-ação de forma que o aprendizado continue ocorrendo, o que mostra a tolerância à elaboração do

ASP(MDP) e permite que mesmo que ocorra mudanças no domínio, é possível continuar o processo de aprendizado a partir do conhecimento que já se possui.

Apesar de ASP não poder utilizar probabilidades de transição na busca por *answer sets*, experimentos apresentados nas seções 5.1 e 5.4 mostram que é possível utilizar a linguagem para descrever um ambiente com as transições esperadas, permitindo que, mesmo quando as probabilidades de transição não sejam conhecidas, é possível utilizar ASP(MDP) para descrever um ambiente não-determinístico e obter o MDP a partir do ASP.

O ASP(MDP) proposto neste trabalho contribui para a forma como o MDP pode ser descrito a partir de uma linguagem de alto nível como ASP ou $BC+$, além de permitir a redução do espaço de estados e também ter a propriedade de ser tolerante a elaboração, permitindo sua utilização em MDPs não-estacionários.

Esta tese contribui na forma como um MDP estacionário pode ser obtido a partir da descrição do problema em uma linguagem de ações e que esta descrição obtida pode reduzir o espaço de busca pela solução ótima através de restrições apresentadas na descrição do domínio. Considerando que o MDP obtido a partir dos *answer sets* constitui um MDP estacionário por um certo período de tempo, o método proposto permita utilização de métodos que são capazes de encontrar a solução de MDPs estacionários sem a necessidade de nenhuma alteração no funcionamento destes métodos. A descrição de um MDP a partir de *answer sets* também permite que, caso ocorra alguma alteração no ambiente, é possível realizar a alteração dos conjuntos de estados e ações do MDP de forma que este seja tolerante à elaboração, desta forma, o ASP(MDP) pode ser utilizado para a descrição de domínio determinístico ou não-determinístico, estacionário ou não-estacionário e com horizonte finito ou infinito.

8.1 TRABALHOS FUTUROS

A partir dos experimentos realizados foi possível observar alguns fatores que ainda devem ser tratados no ASP(MDP).

Outro possível trabalho futuro está também relacionado à adição de novos estados, porém desta vez o MDP não é totalmente conhecido no início da interação com o ambiente, mas este deverá ser construído ao longo desta interação. Uma vez definido como é a descrição de um estado e quais são as ações que o agente pode executar, o objetivo é fazer com que o agente interaja com o ambiente e aprenda uma descrição do MDP. A cada novo estado ou transição observada, o agente retorna à busca pelos *answer sets* que podem descrever o ambiente e formaliza um novo MDP. A partir da formalização deste MDP, o agente busca a política ótima de forma a

realizar a aproximação da função valor-ação. Uma proposta para o algoritmos que resolve este problema é a utilização do $oBC+$ (BABB; LEE, 2015c) que é a versão *online* da linguagem $BC+$ e permite a alteração dos fatos do programa a partir da interação com o ambiente.

Por fim, uma vez que ASP apresenta um linguagem padronizada para ser executada em diversos *grounders* e *solvers*, e $BC+$ também é formalizada de forma a permitir que seja traduzida para ASP, uma última proposta de trabalho futuro é a produção de um aplicativo que realize a tradução da descrição de um MDP para $BC+$ e vice-versa utilizando uma linguagem de descrição de MDPs que seja utilizada em outros projetos, como por exemplo a linguagem proposta por Cassandra (2016).

REFERÊNCIAS

ÄSTRÖM, K. J.; MURRAY, R. M. **Feedback systems**: an introduction for scientists and engineers. Princeton, NJ: Princeton University Press, 2008. Acesso em 30 set. 2016. Disponível em: <http://authors.library.caltech.edu/25062/>.

BABB, J.; LEE, J. Cplus2asp: computing action language C+ in answer set programming. In: CABALAR, P.; SON, T. C. (Ed.). **Logic programming and nonmonotonic reasoning**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. v. 8148, p. 122–134. Acesso em 7 jan. 2016. Disponível em: <http://link.springer.com/10.1007/978-3-642-40564-8_13>.

____. Action language BC+. **Journal of Logic and Computation**, 2015. Acesso em 7 jan. 2016. Disponível em: <http://logcom.oxfordjournals.org/lookup/doi/10.1093/logcom/exv062>.

____. Action language BC+: preliminary report. In: **Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)**. Austin, Texas: [s.n.], 2015. p. 1424–1430.

____. Online action language oBC+. In: CALIMERI, F.; IANNI, G.; TRUSZCZYNSKI, M. (Ed.). **Logic Programming and Nonmonotonic Reasoning**. Cham: Springer International Publishing, 2015. v. 9345, p. 97–111. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/978-3-319-23264-5_9>.

BALDUCCINI, M.; GELFOND, M.; NOGUEIRA, M. Answer set based design of knowledge systems. **Annals of Mathematics and Artificial Intelligence**, v. 47, n. 1-2, p. 183–219, nov. 2006. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/s10472-006-9026-1>.

BALDUCCINI, M. et al. Planning with the USA-Advisor. In: **3rd NASA International workshop on Planning and Scheduling for Space**. Houston, Texas: [s.n.], 2002.

____. The USA-Advisor: a case study in answer set planning. In: GOOS, G. et al. (Ed.). **Logic Programming and Nonmotonic Reasoning**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. v. 2173, p. 439–442. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/3-540-45402-0_39>.

BARAL, C.; GELFOND, M.; RUSHTON, N. Probabilistic reasoning with answer sets. **Theory and Practice of Logic Programming**, v. 9, n. 1, p. 57, 2008. Acesso em 7 jan. 2016. Disponível em: <http://www.journals.cambridge.org/abstract_S1471068408003645>.

BELLMAN, R. The theory of dynamic programming. **Bulletin of the American Mathematical Society**, v. 60, n. 6, p. 503–516, nov. 1954. Acesso em 15 mar. 2016. Disponível em: <http://www.ams.org/journal-getitem?pii=S0002-9904-1954-09848-8>.

____. A markovian decision process. **Indiana University Mathematics Journal**, v. 6, n. 4, p. 679–684, 1957. Acesso em 7 jan. 2016. Disponível em: <http://www.iumj.indiana.edu/IUMJ/fulltext.php?artid=56038&year=1957&volume=6>.

BELLMAN, R. E.; DREYFUS, S. E. **Applied dynamic programming**. 4. ed. Princeton, N.J: Princeton Univ. Press, 1971.

BOCHMAN, A. The Markov assumption: formalization and impact. In: **Proceedings of the Twenty-Third international joint conference on Artificial Intelligence**. Pequim: AAAI Press, 2013. p. 782–788.

BOUTILIER, C.; DEAN, T.; HANKS, S. Decision-theoretic planning: structural assumptions and computational leverage. **Journal of Artificial Intelligence Research**, v. 11, n. 1, p. 94, 1999.

CALIMERI, F. et al. **Fifth answer set programming competition (ASPCOMP 2014)**. [s.n.], 2014. Acesso em 11 nov. 2014. Disponível em: <https://www.mat.unical.it/aspcomp2014/>.

CASSANDRA, A. R. **The POMDP page**. 2016. Acesso em 5 maio 2016. Disponível em: <http://www.pomdp.org/>.

EITER, T. et al. Complexity of answer set checking and bounded predicate arities for non-ground answer set programming. In: **Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR04)**. [S.l.]: AAAI Press, 2004. p. 377–387.

____. Complexity results for answer set programming with bounded predicate arities and implications. **Annals of Mathematics and Artificial Intelligence**, Kluwer Academic Publishers, Hingham, MA, USA, v. 51, n. 2-4, p. 123–165, dez 2007. ISSN 1012-2443. Disponível em: <http://dx.doi.org/10.1007/s10472-008-9086-5>.

____. Answer set planning under action costs. In: GOOS, G. et al. (Ed.). **Logics in Artificial Intelligence**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. v. 2424, p. 186–197. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/3-540-45757-7_16>.

____. A logic programming approach to knowledge-state planning, II: the DLVK system. **Artificial Intelligence**, v. 144, n. 1-2, p. 157–211, mar. 2003. Acesso em 15 mar. 2016. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0004370202003673>.

GEBSER, M.; KAMINSKI, R.; KAUFMANN, B. **Answer set solving in practice**. San Rafael: Morgan & Claypool Publishers, 2013. Acesso em 7 jan. 2016. Disponível em: <http://public.eblib.com/choice/publicfullrecord.aspx?p=1104604>.

GEBSER, M. et al. Potassco: the Potsdam answer set solving collection. **AI Communications**, v. 24, n. 2, p. 107–124, 2011.

GELFOND, M. Answer sets. In: HARMELEN, F. van; LIFSCHITZ, V.; PORTER, B. (Ed.). **Handbook of Knowledge Representation**. [S.l.]: Elsevier, 2008. p. 285–316.

GELFOND, M.; LIFSCHITZ, V. The stable model semantics for logic programming. In: KOWALSKI, R.; Bowen; Kenneth (Ed.). **Proceedings of International Logic Programming Conference and Symposium**. Seattle, Washington: MIT Press, 1988. p. 1070–1080. Disponível em: <http://www.cs.utexas.edu/users/ai-lab/?gel88>.

____. Action languages. **Computer and Information Science**, v. 3, n. 16, 1998.

GELFOND, M.; RUSHTON, N. Causal and probabilistic reasoning in P-log. **Heuristics, Probabilities and Causality. A tribute to Judea Pearl**, p. 337–359, 2010.

GIUNCHIGLIA, E. et al. Nonmonotonic causal theories. **Artificial Intelligence**, v. 153, n. 1-2, p. 49–104, mar. 2004. Acesso em 15 mar. 2016. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S000437020300167X>.

IANNI, G.; KRENNWALLNER, T.; CALIMERI, F. **ASPCOMP 2013**. [s.n.], 2013. Acesso em 11 nov. 2014. Disponível em: <https://www.mat.unical.it/aspcomp2013/>.

JEAN-CHRISTOPHE, B. **Reaction control system**. 2011. Acesso em 13 out. 2016. Disponível em: <https://fr.wikipedia.org/wiki/Reaction_Control_System>.

KHANDELWAL, P. et al. Planning in action language BC while learning action costs for mobile robots. In: **Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014**. Portsmouth, New Hampshire, USA: [s.n.], 2014. Disponível em: <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7940>.

LIFSCHITZ, V. Answer set planning. In: GOOS, G. et al. (Ed.). **Logic Programming and Nonmonotonic Reasoning**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. v. 1730, p. 373–374. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/3-540-46767-X_28>.

____. What is answer set programming? In: **Proceedings of the 23rd International Conference on Artificial Intelligence**. Chicago: AAAI Press, 2008. v. 3, p. 1594–1597.

MAUSAM; KOLOBOV, A. Planning with markov decision processes: an AI perspective. **Synthesis Lectures on Artificial Intelligence and Machine Learning**, v. 6, n. 1, p. 1–210, 2012. Acesso em 7 jan. 2016. Disponível em: <http://www.morganclaypool.com/doi/abs/10.2200/S00426ED1V01Y201206AIM017>.

MCCARTHY, J. Elaboration tolerance. In: **Proc. of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning (Common Sense 98)**. London, UK: [s.n.], 1998. v. 98.

METROPOLIS, N. et al. Equation of state calculations by fast computing machines. **The Journal of Chemical Physics**, v. 21, n. 6, p. 1087, 1953. Acesso em 15 mar. 2016. Disponível em: <http://scitation.aip.org/content/aip/journal/jcp/21/6/10.1063/1.1699114>.

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature**, v. 518, n. 7540, p. 529–533, 2015.

NICKLES, M. A system for the use of answer set programming in reinforcement learning. In: HUTCHISON, D. et al. (Ed.). **Logics in Artificial Intelligence**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 7519, p. 488–491. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/978-3-642-33353-8_40>.

NOGUEIRA, M. et al. An A-Prolog decision support system for the space shuttle. In: GOOS,

G. et al. (Ed.). **Practical Aspects of Declarative Languages**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. v. 1990, p. 169–183. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/3-540-45241-9_12>.

OTTERLO, M. v. **The logic of adaptive behavior**: knowledge representation and algorithms for the Markov decision process framework in first-order domains. maio 2008. Tese (Doutorado) — University of Twente, Enschede. Disponível em: <http://doc.utwente.nl/58987/>.

PUTERMAN, M. L. **Markov decision processes**: discrete stochastic dynamic programming. Hoboken, NJ: Wiley-Interscience, 2005. (Wiley series in probability and statistics).

RUMMERY, G. A.; NIRANJAN, M. **On-Line Q-Learning using connectionist systems**. Engineering Department, Cambridge University, 1994. Technical Report CUED/F-INFENG/TR 166.

SHAKESPEARE, W. **Hamlet**. [S.l.]: University Press, 1904.

SILVER, D. et al. Mastering the game of Go with deep neural networks and tree search. **Nature**, v. 529, n. 7587, p. 484–489, 2016.

SON, T. C.; PONTELLI, E. Planning with preferences using logic programming. In: GOOS, G. et al. (Ed.). **Logic Programming and Nonmonotonic Reasoning**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. v. 2923, p. 247–260. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/978-3-540-24609-1_22>.

SPRAUEL, J.; TEICHTEIL-KöNIGSBUCH, F.; KOLOBOV, A. Saturated path-constrained MDP: planning under uncertainty and deterministic model-checking constraints. In: **Proceedings of 28th AAAI Conference on Artificial Intelligence (AAAI 2014)**. Québec, Canada: [s.n.], 2014. p. 2367–2373.

SRIDHARAN, M. et al. Mixing non-monotonic logical reasoning and probabilistic planning for robots. In: **Workshop on Hybrid Reasoning @ IJCAI 2015**. Buenos Aires: [s.n.], 2015. Disponível em: <https://www.hybrid-reasoning.org/media/filer/2015/07/15/hr_2015_submission_9.pdf>.

SUBRAHMANIAN, V. S.; ZANIOLO, C. Relating stable models and AI planning domains. In: **In Proc. ICLP-95**. [S.l.]: MIT Press, 1995. p. 233–247.

SUTTON, R. S. Learning to predict by the methods of temporal differences. **Machine Learning**, v. 3, n. 1, p. 9–44, ago. 1988. Acesso em 15 mar. 2016. Disponível em: <http://link.springer.com/10.1007/BF00115009>.

____. Dyna, an integrated architecture for learning, planning, and reacting. **ACM SIGART Bulletin**, ACM, v. 2, n. 4, p. 160–163, 1991.

____. Generalization in reinforcement learning: successful examples using sparse coarse coding. In: TOURETZKY, D. S.; MOZER, M. C.; HASSELMO, M. E. (Ed.). **Advances in Neural Information Processing Systems**. Cambridge, MA: MIT Press, 1996. p. 1038–1044.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning**: an introduction. Cambridge, Mass: MIT Press, 1998. (Adaptive computation and machine learning).

____. **Reinforcement learning an introduction – second edition, in progress (draft)**. Cambridge, Mass.: MIT Press, 2015. Acesso em 7 jan. 2016. Disponível em: <https://www.dropbox.com/s/b3psxv2r0ccmf80/book2015oct.pdf?dl=0>.

WALD, A. Foundations of a general theory of sequential decision functions. **Econometrica**, v. 15, n. 4, p. 279–313, 1947. Disponível em: <http://www.jstor.org/stable/1905331>.

____. Statistical decision functions. **The Annals of Mathematical Statistics**, p. 165–205, 1949.

WATKINS, C. J. C. H. **Learning from delayed rewards**. 1989. Tese (PhD thesis) — University of Cambridge England.

YANG, F. et al. Planning in answer set programming while learning action costs for mobile robots. In: **AAAI Spring 2014 Symposium on Knowledge Representation and Reasoning in Robotics (AAAI-SSS)**. Stanford, California, USA: [s.n.], 2014.

ZHANG, S.; SRIDHARAN, M.; WYATT, J. L. Mixed logical inference and probabilistic planning for robots in unreliable worlds. **IEEE Transactions on Robotics**, v. 31, n. 3, p. 699–713, 2015. Acesso em 7 jan. 2016. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7102752>.

ANEXO A – CÓDIGOS UTILIZADOS NO TERCEIRO EXPERIMENTO

Figura 22 – Descrição em $BC+$ das paredes do mundo de grades conforme o mapa da figura 12a

```
:- variables
 WALLS1_X :: 4..4;
 WALLS1_Y :: 7..10;
 WALLS2_X :: 8..8;
 WALLS2_Y :: 1..4;
 HOLE1_X :: 4..4;
 HOLE1_Y :: 4..4;
 HOLE2_X :: 7..7;
 HOLE2_Y :: 8..8.

nonexecutable up if y+1=WALLS1_Y, x=WALLS1_X.
nonexecutable down if y-1=WALLS1_Y, x=WALLS1_X.
nonexecutable right if y=WALLS1_Y, x+1=WALLS1_X.
nonexecutable left if y=WALLS1_Y, x-1=WALLS1_X.

nonexecutable up if y+1=WALLS2_Y, x=WALLS2_X.
nonexecutable down if y-1=WALLS2_Y, x=WALLS2_X.
nonexecutable right if y=WALLS2_Y, x+1=WALLS2_X.
nonexecutable left if y=WALLS2_Y, x-1=WALLS2_X.

nonexecutable up if y+1=HOLE1_Y, x=HOLE1_X.
nonexecutable down if y-1=HOLE1_Y, x=HOLE1_X.
nonexecutable right if y=HOLE1_Y, x+1=HOLE1_X.
nonexecutable left if y=HOLE1_Y, x-1=HOLE1_X.

nonexecutable up if y+1=HOLE2_Y, x=HOLE2_X.
nonexecutable down if y-1=HOLE2_Y, x=HOLE2_X.
nonexecutable right if y=HOLE2_Y, x+1=HOLE2_X.
nonexecutable left if y=HOLE2_Y, x-1=HOLE2_X.
```

Fonte: Autor

Figura 23 – Parte da descrição em $BC+$ das paredes do mundo de grades conforme o mapa da figura 12c. O resto do código foi omitido por conta de sua extensão

```
:- variables
 WALLS1_X :: 3..3;
 WALLS1_Y :: 6..10;
 WALLS2_X :: 6..6;
 WALLS2_Y :: 1..5;
 WALLS3_X :: 8..8;
 WALLS3_Y :: 3..8;

 HOLE1_X :: 1..1;
 HOLE1_Y :: 5..5;

 HOLE2_X :: 2..2;
 HOLE2_Y :: 3..3;

 HOLE3_X :: 4..4;
 HOLE3_Y :: 2..2;

 HOLE4_X :: 5..5;
 HOLE4_Y :: 9..9;

 HOLE5_X :: 6..6;
 HOLE5_Y :: 7..7;

 HOLE6_X :: 9..9;
 HOLE6_Y :: 6..6;

 HOLE7_X :: 10..10;
 HOLE7_Y :: 3..3.

nonexecutable up if y+1=WALLS1_Y, x=WALLS1_X.
nonexecutable down if y-1=WALLS1_Y, x=WALLS1_X.
nonexecutable right if y=WALLS1_Y, x+1=WALLS1_X.
nonexecutable left if y=WALLS1_Y, x-1=WALLS1_X.
```

Fonte: Autor

Figura 24 – Descrição em $BC+$ das paredes do mundo de grades conforme o mapa da figura 12d

```
:- variables
WALLS1_X :: 2..10;
WALLS1_Y :: 1..9.

nonexecutable up if y+1=WALLS1_Y, x=WALLS1_X.
nonexecutable down if y-1=WALLS1_Y, x=WALLS1_X.
nonexecutable right if y=WALLS1_Y, x+1=WALLS1_X.
nonexecutable left if y=WALLS1_Y, x-1=WALLS1_X.
```

Fonte: Autor

Figura 25 – Parte do código em ASP traduzido do programa em $BC+$ da figura 6 que apresenta a *query* e as descrições independente do domínio. A tradução foi realizada pelo programa *CPLUS2ASP*

```
% WARNING: F2LP arrow '<-' found. Treating '-' as classical negation. Please use the
     option -d if you would like '-' to be treated as default negation.
% Standard description-independent declarations and laws to CCalc-ify ASP

% A derived binary relation between a constant and its domain objects,
% connected via constant_sort and sort_object

%------------------------------------------------------------
#base.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------------------------------------------------------------

query_label(query) :- true.
% Selects which query to activate in a based on the value of the const "query"

constant_object(V_constant,Object) :- constant_sort(V_constant,V_sort),sort_object(
    V_sort,Object).

% constant_posobject relates constants with their positive objects
% positive objects are defined to be any object that is not 'none'
% (multi-valued) or 'false' (boolean)

constant_posobject(V_constant,Object) :- constant_object(V_constant,Object),
    constant_sort(V_constant,boolean),Object!=false.

constant_posobject(V_constant,Object) :- constant_object(V_constant,Object),not
    constant_sort(V_constant,boolean),Object!=none.

% constant_negobject relates constants with their negative object
% negative objects are false (boolean) and none (non-boolean)
constant_negobject(V_constant,false) :- constant_sort(V_constant,boolean).

constant_negobject(V_constant,none) :- not constant_sort(V_constant,boolean),
    constant(V_constant).

%------------------------------------------------------------
#base.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------------------------------------------------------------

% Standard, description-independent declarations of sorts and domains

sort(boolean) :- true.
#domain boolean(V_boolean).
sort_object(boolean,V_boolean) :- true.

boolean(true) :- true.
boolean(false) :- true.

sort(computed) :- true.
#domain computed(V_computed).
sort_object(computed,V_computed) :- true.

% Constants hierarchy

% Meta-constants to group categories of constants

sort(constant) :- true.
#domain constant(V_constant).
sort_object(constant,V_constant) :- true.
```

Fonte: Autor

Figura 26 – Código que descreve as constantes utilizadas pela tradução em ASP

```
%----

% Rigid constants
sort(rigid) :- true.
#domain rigid(V_rigid).
sort_object(rigid,V_rigid) :- true.

%---

% Fluent-based constants

sort(fluent) :- true.
#domain fluent(V_fluent).
sort_object(fluent,V_fluent) :- true.

sort(simpleFluent) :- true.
#domain simpleFluent(V_simpleFluent).
sort_object(simpleFluent,V_simpleFluent) :- true.

sort(inertialFluent) :- true.
#domain inertialFluent(V_inertialFluent).
sort_object(inertialFluent,V_inertialFluent) :- true.

sort(sdFluent) :- true.
#domain sdFluent(V_sdFluent).
sort_object(sdFluent,V_sdFluent) :- true.

sort(abFluent) :- true.
#domain abFluent(V_abFluent).
sort_object(abFluent,V_abFluent) :- true.

%---

% Action-based constants

sort(action) :- true.
#domain action(V_action).
sort_object(action,V_action) :- true.

sort(exogenousAction) :- true.
#domain exogenousAction(V_exogenousAction).
sort_object(exogenousAction,V_exogenousAction) :- true.

sort(abAction) :- true.
#domain abAction(V_abAction).
sort_object(abAction,V_abAction) :- true.

sort(attribute) :- true.
#domain attribute(V_attribute).
sort_object(attribute,V_attribute) :- true.

% A regular action is defined to be any action which isn't an abAction or attribute
sort(regularAction) :- true.
#domain regularAction(V_regularAction).
sort_object(regularAction,V_regularAction) :- true.

%---

% Abnormality constants

sort(abnormality) :- true.
#domain abnormality(V_abnormality).
sort_object(abnormality,V_abnormality) :- true.
```

Fonte: Autor

Figura 27 – Descrição das regras e subconjuntos na tradução do código ASP

```
sort(staticAbnormality) :- true.
#domain staticAbnormality(V_staticAbnormality).
sort_object(staticAbnormality,V_staticAbnormality) :- true.

sort(dynamicAbnormality) :- true.
#domain dynamicAbnormality(V_dynamicAbnormality).
sort_object(dynamicAbnormality,V_dynamicAbnormality) :- true.

%---

% Subsort relations

constant(V_fluent) :- true.
constant(V_action) :- true.
constant(V_rigid) :- true.
constant(V_abnormality) :- true.
fluent(V_simpleFluent) :- true.
simpleFluent(V_inertialFluent) :- true.
fluent(V_sdFluent) :- true.
sdFluent(V_abFluent) :- true.
action(V_exogenousAction) :- true.
action(V_abAction) :- true.
action(V_attribute) :- true.

regularAction(V_action) :- not attribute(V_action),not abAction(V_action).

abnormality(V_staticAbnormality) :- true.
abnormality(V_dynamicAbnormality) :- true.

% Note: This isn't necessary for abnormality constants


%-------------------------------------------------------------
#base.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------

%%% Description-independent rules to CCalc-ify ASP @ t=0

step(0) :- true.

%% Handle external abnormality constants
#external ab_h(V_staticAbnormality,0).

% Exogeneity for exogenous actions
exogenous(V_exogenousAction) :- true.

% Exogeneity for attributes
exogenous(V_attribute) :- true.

% Inertia for inertial fluents
inertial(V_inertialFluent) :- true.

% Default false for abFluents
{h(eql(V_abFluent,X),0)} :- constant_negobject(V_abFluent,X).

% anyInitialState: Exogeneity for simple fluents at time 0
{h(eql(V_simpleFluent,Object),0)} :- constant_object(V_simpleFluent,Object).


% exogenous: Grants exogeneity to a constant
```

Fonte: Autor

Figura 28 – Descrição de ações e fluentes na tradução em ASP

```
% Rigids
{h(eql(V_rigid,Value))} :- exogenous(V_rigid),constant_object(V_rigid,Value).

% Fluents (@ t=0)
{h(eql(V_fluent,Value),0)} :- exogenous(V_fluent),constant_object(V_fluent,Value).

% Existence and uniqueness for every constant relative to its domain

% Rigid
 :- not 1{h(eql(V_rigid,Object1)):constant_object(V_rigid,Object1)}1.

% Fluents (@ t=0)
 :- not 1{h(eql(V_fluent,Object1),0):constant_object(V_fluent,Object1)}1.


%-------------------------------------------------------------
#cumulative t.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------
% Keep track of steps we've visited
step(t) :- true.
astep(t-1) :- true.

%%% Description-independent rules to CCalc-ify ASP @ t>0

%% Handle external abnormality constants
#external ab_h(V_staticAbnormality,t).
#external ab_occ(V_dynamicAbnormality,t-1).

% abActions default to false
{occ(eql(V_abAction,X),t-1)} :- constant_negobject(V_abAction,X).

% Default false for abFluents
{h(eql(V_abFluent,X),t)} :- constant_negobject(V_abFluent,X).

% Restriction that attributes must take on the value "none" if and only if their
    linked action does not execute
 :- {not occ(eql(V_attribute,NegObj1),t-1)}0,constant_negobject(V_action,NegObj2),
     action_attribute(V_action,V_attribute),constant_negobject(V_attribute,NegObj1),
     not occ(eql(V_action,NegObj2),t-1).
 :- {not occ(eql(V_action,NegObj2),t-1)}0,constant_negobject(V_action,NegObj2),
     action_attribute(V_action,V_attribute),constant_negobject(V_attribute,NegObj1),
     not occ(eql(V_attribute,NegObj1),t-1).

% exogenous: Grants exogeneity to a constant

% Actions
{occ(eql(V_action,Value),t-1)} :- exogenous(V_action),constant_object(V_action,Value
    ).

% Fluents
{h(eql(V_fluent,Value),t)} :- exogenous(V_fluent),constant_object(V_fluent,Value).

% inertial: Grants inertia to a (non-rigid) fluent

{h(eql(V_fluent,Value),t)} :- inertial(V_fluent),h(eql(V_fluent,Value),t-1),
    constant_object(V_fluent,Value).

% noconcurrency: If stated as a fact, prevents concurrent execution of Boolean
    actions
```

Fonte: Autor

Figura 29 – Configuração de exibição da saída da execução do código em ASP

```
% Fluents (@ t>0)
 :- not 1{h(eql(V_fluent,Object1),t):constant_object(V_fluent,Object1)}1.

% Actions
 :- not 1{occ(eql(V_action,Object1),t-1):constant_object(V_action,Object1)}1.

%-------------------------------------------------------------
#volatile t.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------

% Visit at least minstep before returning a solution
  :- not step(minstep).


%-------------------------------------------------------------
#base.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------

% Etc
  #hide step/1.
#hide astep/1.
#hide constant_object/2.
#hide constant_posobject/2.
#hide constant_negobject/2.
#hide constant_sort/2.
#hide sort_object/2.
#hide action_attribute/2.
#hide sort/1.
#hide sort_object/2.
#hide boolean/1.
#hide constant/1.
#hide fluent/1.
#hide simpleFluent/1.
#hide inertialFluent/1.
#hide sdFluent/1.
#hide abFluent/1.
#hide action/1.
#hide exogenousAction/1.
#hide abAction/1.
#hide attribute/1.
#hide regularAction/1.
#hide abnormality/1.
#hide staticAbnormality/1.
#hide dynamicAbnormality/1.
#hide exogenous/1.
#hide inertial/1.
#hide noconcurrency.
#hide strong_noconcurrency.
#hide query_label/1.
#hide rigid/1.
#hide computed/1.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Fonte: Autor

Figura 30 – Descrição do ambiente para a variável X e resultados das ações nesta coordenada

```
% File "/home/user/gridworld/original/gwbcp"
% gridworld variables
% world size and objective

#base.
sort(numRange_1_15) :- true.
#domain numRange_1_15(V_numRange_1_15).
sort_object(numRange_1_15,V_numRange_1_15) :- true.
#hide numRange_1_15(V_numRange_1_15).

numRange_1_15(1) :- true.
numRange_1_15(2) :- true.
numRange_1_15(3) :- true.
numRange_1_15(4) :- true.
numRange_1_15(5) :- true.
numRange_1_15(6) :- true.
numRange_1_15(7) :- true.
numRange_1_15(8) :- true.
numRange_1_15(9) :- true.
numRange_1_15(10) :- true.
numRange_1_15(11) :- true.
numRange_1_15(12) :- true.
numRange_1_15(13) :- true.
numRange_1_15(14) :- true.
numRange_1_15(15) :- true.
inertialFluent(x) :- true.
constant_sort(x,numRange_1_15) :- true.
inertialFluent(y) :- true.
constant_sort(y,numRange_1_15) :- true.
exogenousAction(up) :- true.
constant_sort(up,boolean) :- true.
exogenousAction(down) :- true.
constant_sort(down,boolean) :- true.
exogenousAction(left) :- true.
constant_sort(left,boolean) :- true.
exogenousAction(right) :- true.
constant_sort(right,boolean) :- true.

sort(numRange_1_15xxx1) :- true.
#domain numRange_1_15xxx1(V_numRange_1_15xxx1).
sort_object(numRange_1_15xxx1,V_numRange_1_15xxx1) :- true.
#hide numRange_1_15xxx1(V_numRange_1_15xxx1).

numRange_1_15xxx1(1) :- true.
numRange_1_15xxx1(2) :- true.
numRange_1_15xxx1(3) :- true.
numRange_1_15xxx1(4) :- true.
```

Fonte: Autor

Figura 31 – Descrição do ambiente para a variável `Y`, resultados das ações nesta coordenada e parâmetros para execução

```
numRange_1_15xxx1(5) :- true.
numRange_1_15xxx1(6) :- true.
numRange_1_15xxx1(7) :- true.
numRange_1_15xxx1(8) :- true.
numRange_1_15xxx1(9) :- true.
numRange_1_15xxx1(10) :- true.
numRange_1_15xxx1(11) :- true.
numRange_1_15xxx1(12) :- true.
numRange_1_15xxx1(13) :- true.
numRange_1_15xxx1(14) :- true.
numRange_1_15xxx1(15) :- true.
#domain numRange_1_15xxx1(X).
#domain numRange_1_15xxx1(Y).

% action transitions
#cumulative maxstep.
h(eql(y,Y+1),maxstep) :- h(eql(y,Y),maxstep-1),occ(eql(up,true),maxstep-1).
h(eql(y,Y-1),maxstep) :- h(eql(y,Y),maxstep-1),occ(eql(down,true),maxstep-1).
h(eql(x,X+1),maxstep) :- h(eql(x,X),maxstep-1),occ(eql(right,true),maxstep-1).
h(eql(x,X-1),maxstep) :- h(eql(x,X),maxstep-1),occ(eql(left,true),maxstep-1).

% world limits
 :- occ(eql(up,true),maxstep-1),h(eql(y,15),maxstep-1).
 :- occ(eql(down,true),maxstep-1),h(eql(y,1),maxstep-1).
 :- occ(eql(right,true),maxstep-1),h(eql(x,15),maxstep-1).
 :- occ(eql(left,true),maxstep-1),h(eql(x,1),maxstep-1).

% only one action at a time
#base.
noconcurrency :- true.

% the only query that matters
% [Query: Label:planning, Maxstep:0::100]
 :- not h(eql(x,1),0),query_label(planning).
 :- not h(eql(y,1),0),query_label(planning).
#volatile maxstep.
 :- not h(eql(x,15),maxstep),query_label(planning).
 :- not h(eql(y,15),maxstep),query_label(planning).
#base.
true.
:- false.
#hide true/0.
```

Fonte: Autor

ANEXO B – TRABALHOS PUBLICADOS

Durante a produção deste trabalho, o seguinte artigo foi publicado:

FERREIRA, L. A.; SANTOS P. E.; BIANCHI, R. A. C. Constraining a MDP's Search Space by Usage of Planning Trajectories. In: WORKSHOP ON HYBRID REASONING @ IJCAI, 2015, Buenos Aires, **Anais eletrônicos...** Buenos Aires: DFG 2015. Disponível em: <https://www.hybrid-reasoning.org/media/filer/2015/07/15/hr_2015_submission_1.pdf> Acesso em: 25 out. 2016

Constraining a MDP's Search Space by Usage of Planning Trajectories

Leonardo A. Ferreira[1] **Paulo E. Santos**[2] **Reinaldo A. C. Bianchi**[2]
1. Universidade Metodista de São Paulo, *leonardo.ferreira@metodista.br*
2. Centro Universitário da FEI, {*psantos,rbianchi*}*@fei.edu.br*

Abstract

Answer Set Planning provides a framework that allows for the description of a sequential decision making problem with non-monotonicity and belief revision capabilities. In this work the sequence of actions to be performed in order to solve a problem is given in answer sets. Markov Decision Process is a formalism that can be used to describe a stochastic sequential decision making problem. The solution for this process is also sequence of actions that leads from an initial state to a goal state. The goal of this work is to use the answer sets found by Answer Set Planning for restricting a Markov Decision Process' search space, such that instead of performing each possible combination of actions in each available state, only the combinations that leads to the goal state are executed. Results shows that using Monte Carlo Policy Evaluation, it is possible to calculate returns for answer sets given as solutions to a planning problem, thus allowing to determine the optimal answer set for a non-deterministic environment.

1 Introduction

There are various methods that can be used to find solutions for sequential decision-making problems [Russell and Norvig, 2004]. For example Monte Carlo (MC) and Reinforcement Learning (RL), with no previous information about the transition probabilities between domain states, are able to obtain optimal policies for non-deterministic domains [Sutton and Barto, 1998]. In contrast, Answer Set Programming (ASP) [Gelfond, 2008; Gebser *et al.*, 2012], when used as Answer Set Planning [Lifschitz, 1999; 2002], is able to describe a sequential decision-making problem and to find every solution that satisfy a set of domain restrictions in the form of answer sets. Informally, answer sets are sets of literals that are minimal models to a non-monotonic logic program. ASP allows a declarative programming whereby a high-level relational description of the environment is the only information a user needs to provide in order to obtain the set of solutions to a particular problem.

The goal of this work is to combine the answer sets obtained from ASP with the return estimate given by an MDP in order to find the optimal answer set considering the transition probabilities of a non-deterministic domain. Given enough samples, this method is capable of estimating return and the transition probabilities and associate them to the answer sets.

An example of an stochastic application domain assumed in this work is the following: given two computer servers ($S1$ and $S2$) and two processes that need to be run ($P1$ and $P2$) in any possible process allocation in the servers, the combinations stated below are possible solutions to this process allocation problem:

- Allocate both processes in one of the servers (either $S1$ or $S2$): $allocate(P1, S1)$ and $allocate(P2, S1)$ or $allocate(P1, S2)$ and $allocate(P2, S2)$.
- Allocate one process in one of the servers: $allocate(P1, S1)$ and $allocate(P2, S2)$ or $allocate(P2, S1)$ and $allocate(P1, S2)$.

The four possibilities given above are answer sets of this problem and can be treated as plans for the sequential process allocation problem. However, servers are not perfect, they may fail due to malfunction, power loss or faulty network connection. Thus, the processes may not finish in the allocated server and need to be reallocated. Therefore, this is a stochastic domain.

Since the probability of the servers functioning correctly is unknown, and ASP does not provide the syntax to describe and update it, a MDP is described from the answer sets and MC policy evaluation is used. If each answer set in the set of answers is considered as a history it is possible to use them to allocate processes in servers. Given enough samples, MC will be able to evaluate each answer set as a policy and it is possible to estimate which is the optimal answer set.

In order to combine these tools, a brief introduction to Sequential Decision Making and Planning will be introduced (Section 2) along with Answer Set Programming and its development for planning (Section 3). The Action Language $\mathcal{BC}+$ (Section 4) was chosen to describe how the connection between Sequential Decision Making and answer sets can

be made (Section 5), allowing an implementation of Monte Carlo Policy Evaluation algorithm to estimate the return value for answer sets (Section 6). This method was validated (Section 7) using the a variation of the process allocation domain presented as an example and it was possible to conclude that it is possible to estimate the optimal answer set by using the proposed method (Section 8).

Along with the description of the proposed method, works related to the proposition made in this work are presented (Section 9).

2 Sequential Decision Making and Planning

Decision making has been studied under different areas, from economics [Wald, 1947; 1950] to Operational Research and Optimal Control [Bellman, 1954; Bellman and Dreyfus, 1962]. A sequential decision function can be defined as a decision function in which the amount of observations necessary for a decision relies on the outcome of the observations [Wald, 1947].

A feasible solution to a sequential decision making process is a selection of transformations in the environment by the execution of decisions, usually called policy or plan (π). From this definition, we are now able to define the Principle of Optimality [Bellman, 1954; Bellman and Dreyfus, 1962], which states that independently of the initial state and the initial decisions that ware made, an optimal policy (π^*) has the property to provide an optimal policy form the resulting state obtained from the decision taken initially.

One formalisation that can be used to describe a sequential decision making problem (or a planning problem) is the Markov Decision Process (MDP), that uses a set of states and a set of actions (decisions) to describe the environment. An MDP can be formalised as:

Definition 1. ***Markov Decision Process:****[Mausam and Kolobov, 2012] A Markov Decision Process (MDP) can be define by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{D}, \mathcal{T}, \mathcal{R} \rangle$, where:*

- *$\mathcal{S}$ is the finite set of all states of a system;*
- *$\mathcal{A}$ is the finite set of every action that the agent can execute;*
- *$\mathcal{D}$ is a finite or infinite sequence of natural numbers in the form $(1, 2, \ldots, D_{max})$ or $(1, 2, \ldots)$, respectively, that represents the time steps in which the actions must be chosen;*
- *$\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \mapsto [0, 1]]$ is the transition function that maps a probability $\mathcal{T}(s_1, a, s_2, t)$ of reaching the state $s_2 \in \mathcal{S}$ by executing the action $a \in \mathcal{A}$ in the state $s_1 \in \mathcal{S}$ in the time step $t \in \mathcal{D}$;*
- *$\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \mapsto \mathbb{R}$ is a reward function that returns a numerical value $\mathcal{R}(s_1, a, s_2, t)$ for reaching state $s_2 \in \mathcal{S}$ by executing $a \in \mathcal{A}$ in $s_1 \in \mathcal{S}$ at $t \in \mathcal{D}$;*

It is worth pointing out that it is usually assumed that the set of actions $\mathcal{A}$ is the same for every state $s \in \mathcal{S}$ [Boutilier *et al.*, 1999]. Nevertheless, it will be shown (Section 5) that in the proposed method, this assumption does not hold.

An important definition for a MDP is the execution history. Informally, the execution of a given policy π in the environment provides a sequence of states visited and actions performed. Therefore the execution history of a MDP can be defined as:

Definition 2. ***Execution history:****[Mausam and Kolobov, 2012] An execution history of an MDP up to the time step $t \in \mathcal{D}$ is a sequence $h_t = ((s_1, a_1), \ldots, (s_{t-1}, a_{t-1}), s_t)$ of pairs formed by states visited and actions executed by the agent for each time step t' in $1 \leq t' < t$ and the state s_t in the current time step.*

There are many methods that are able to find a solution of a MDP, such as Monte Carlo Methods. If one wants to solve a MDP using MC, the transition function $\mathcal{T}$ may be unknown, while the interaction with the domain can provide the reward and allows estimating returns [Sutton and Barto, 1998]. A MC algorithm that can be used to estimate returns of a MDP without knowing the transition probability is the Monte Carlo Policy Evaluation as presented by [Sutton and Barto, 1998; 2012]. The MC described is used in this work as a method to estimate the costs. Nevertheless, since the purpose of the proposed methods is to constrain a MDP, it is possible to select other method to find the optimal policy.

Using the same example of processes allocation as before, there are four solutions to the problem. Once the method has been performed a number of times, it is possible to estimate the cost of allocating such process in the selected server.

The other important point to notice is that, if restrictions are introduced, such as (for instance) "a server must not receive a process if it has just processed one", there are only two policies that can solve the problem (allocating a process in each server). Therefore, from four possible combinations, only two of them are answers to the problem.

Thus, if there is a pre-computed set of policies, Monte Carlo Method may have a smaller computation time to find the optimal policy. If the domain is non-monotonic, or may change over time, the method can iterate over the answers again and find the changes that happened. The purpose of using ASP in this work is exactly to find a pre-computed set of policies that would allow for a search space reduction by using constrains and changes to happen in the domain onwards. Ergo, the next section presents ASP, which is the chosen method to provide this pre-computed policies.

3 Answer Set Programming

Answer Set Programming is a logic programming language that has been shown to be able to express language constructs that are sometimes impossible in classical logic and also to represent some constructs that can be found in some non-mathematical domains [Baral *et al.*, 2004]. By using ASP, one is not describing how the solution of a problem must be found, but only what the problem is [Gebser *et al.*, 2012].

An ASP program is a collection of rules of the form [Baral *et al.*, 2004]:

$$l_0 \; or \; \ldots \; or \; l_k \leftarrow l_{k+1}, \ldots, l_m, not \, l_{m+1}, \ldots, not \, l_n.$$

Where ls are literals formed by atoms defined by p and $\neg p$ [Baral *et al.*, 2004]. Considering that a ground program Π consists of a set of such rules, an answer set of Π can be defined as a set of ground literals that [Gelfond and Rushton, 2010]:

- satisfy the rules of Π;
- is minimal.

One of the roots of ASP is the non-monotonic logics [Gebser *et al.*, 2012; Baral *et al.*, 2004; Gelfond and Rushton, 2010] which allows for the notion of beliefs and the rationality principle which states that "one shall not believe anything one is not forced to believe" [Gelfond and Rushton, 2010], therefore along with *true* and *false*, ASP also has the truth value of *unknown*.

ASP can be used to described a great set of difficult (NP-complete) problems, including planning and scheduling [Lifschitz, 1999; 2002; Khandelwal *et al.*, 2014; Yang *et al.*, 2014].

3.1 Logic Programming and Planning

A planning problem can be described as, given a set of primitive operations (decisions or actions) that an autonomous reasoning agent can execute, a plan is a set of operations that the agent must execute in order to achieve a certain goal [Subrahmanian and Zaniolo, 1995; Lifschitz, 1999]. Also, these primitive operations have fundamental aspects:

1. Precondition: set of facts that are true so that the operation can be executed;
2. Effect: set of facts that were true before the execution of the action that becomes not true after the execution and the set of facts that were not true before the execution and becomes true after the execution of an operation.

As has been shown by [Subrahmanian and Zaniolo, 1995], there is a translation between a planning domain and a logic program, such that iff the planning goal is true in a stable model of the translation, it is possible to find a plan by solving the logic program. Furthermore, when using Answer Set Programming to represent a planning problem, the non-monotonic aspect provided by negation as failure allows for a description of the actions' properties that becomes easier than that of the axiomatisation in classical logic [Lifschitz, 2002].

For example, in the Process Allocation problem assumed in this paper, a set of servers and a set of processes to be run are given. Thus, as already mentioned, it is not necessary to describe how the allocation works or how to evaluate the allocation, it is only necessary to describe when the allocation of a process in a server is possible and let the ASP engine find the answers sets that are solutions to this problem.

An example of an instance and an encoding that can be used for the domain of Process Allocation is given in Listings 1 and 2. The solution found by an ASP Engine to this problem is presented in 3.

In this context of Answer Set Planning, two important definitions are plans and trajectories. Plans are a sequence of actions $P = a_1, \ldots, a_n$ chosen to be performed in a set of actions $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$. A trajectory, as the execution history of an MDP, can be defined as a sequence of states visited, actions performed and states reached by a given plan in the form $T = \langle\langle s_0, A_1, s_1\rangle, \langle s_1, A_2, s_2\rangle, \ldots, \langle s_{n-1}, A_n, s_n\rangle\rangle$, being s_n the goal state and s_0 a legal initial state of the problem [Eiter *et al.*, 2003].

Listing 1: Instance in ASP for the Process Allocation domain.

```
% server configuration
server(a,50,50).
server(b,75,100).
server(c,100,100).
server(d,125,90).
server(e,150,50).
#const nserv=5.
% processes configuration
proc(1..3,100).
proc(4..5,150).
```

Listing 2: Encoding in ASP for the Process Allocation domain.

```
% alias for server and processes
server(S) :- server(S,TP,PP).
proc(P) :- proc(P,TP).
% distribute process in servers
1 <= { run_in(P,S) : server(S) } <= 1
     :- proc(P).
% constrains
:- run_in(P1,S1), run_in(P2,S2),
   S1 == S2,
   P2 = P1 + P, P = 1 .. (nserv - 1).
% formating output
allocate(P,S) :- run_in(P,S).
% output
#show allocate/2.
```

A planning problem can have more than one feasible plan, that is, more than one plan is capable of leading the reasoning agent from a initial state to the goal state. A limitation of the Answer Set Planning approach is that plans found in this way do not consider transition probabilities. As a result, the answer set that is considered minimal may not be optimal in a non-deterministic version of this domain.

With this brief introduction of Sequential Decision Making and ASP, it is possible to notice that both present a method to find a solution to a planning problem, although each one of them has a different approach and allows for differences in representation and evaluation criteria. In order to propose a method that combines them, the Action Language $\mathcal{BC}+$ is used.

4 The $\mathcal{BC}+$ Action Language

A method to describe properties of actions is to use formal models of parts of natural languages, known as Action Languages [Babb and Lee, 2014]. These languages are usually viewed as high-level notation of non-monotonic logics and their semantics describes transition systems. The action language $\mathcal{BC}+$ that has been recently proposed, has a semantics defined in terms of stable model semantics and also allows for some useful ASP constructs, i.e. choice rules and aggregates.

This language includes two sets of symbols that allow for

Listing 3: Output of the ASP engine that found the answer sets for the Process Allocation domain.

```
clasp version 2.1.4
Reading from stdin
Solving...
Answer: 1
allocate(1,a) allocate(2,b) allocate(3,c)
allocate(4,d) allocate(5,e)
...
Answer: 120
allocate(1,e) allocate(2,d) allocate(3,c)
allocate(4,b) allocate(5,a)
SATISFIABLE

Models       : 120
Time         : 0.003s (Solving: 0.00s
1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

a description of a problem: action constants and fluents constants, the latter is divided into regular and statically determined.

In $\mathcal{BC}+$, for every constant c is assigned a finite set called domain, denoted $Dom(c)$ with cardinality ≥ 2. An atom in $\mathcal{BC}+$ is described by a constant c and a value v and has the form $c = v$. If $Dom(C) = \{false, true\}$, then c is considered Boolean.

Formulas are divided in fluent and action formulas. When all constants are fluents, then we have a fluent formula. When there is at least one action constant and no fluent constant, then we have an action formula.

Laws in $\mathcal{BC}+$ are described in two forms. The form

$$caused \text{ F } if \text{ G}$$

where F and G are formulas. If both are fluent formulas, then 4 is a static law. If F is an action formula and G is a formula, then 4 is an action dynamic law. The second form is called fluent dynamic law and has F and G as fluent formulas (where F does not contain statically determined constants) and H as a formula. A fluent dynamic law has the form

$$caused \text{ F } if \text{ G } after \text{ H}$$

and can be used to describe direct effect of actions. An action dynamic law can be used to express causal dependencies between concurrently executed action. A static law can be used to talk about causal dependencies between fluents in the same state.

In $\mathcal{BC}+$, an action description is a finite set of causal law. For a more detailed explanation about $\mathcal{BC}+$ semantics, we direct the reader to the preliminary report on the language [Babb and Lee, 2014].

In a transition system that corresponds to any action description $\mathcal{D}$, a trajectory of length n is represented by the stable model of $\text{PF}_n(\mathcal{D})$ when a sequence of propositional formula $\text{PF}_0(\mathcal{D}), \text{PF}_1(\mathcal{D}), \ldots$ is defined.

In this representation, an interpretation s of a set of fluent constants is a state and $0 : s$ is a stable model of $\text{PF}_0(\mathcal{D})$. A transition is represented by a triple $\langle s, a, s' \rangle$, in which a is an interpretation of a set of actions constants and s and s' are states. A model of $\text{PF}_1(\mathcal{D})$ is represented by $0 : s \cup 0 : a \cup 1 : s'$.

As proved by [Babb and Lee, 2014], a stable model is represented by transitions.

Lemma 1. *For every $m \geq 1$, X is a stable model of $PF_m(\mathcal{D})$ iff $X^0, X^1, \ldots, X^{m-1}$ are transitions, with $X^i (i < m)$ being a triple that consists of a fluent formula, an action constant and a second fluent formula.*

Thus, the action language $\mathcal{BC}+$ can be direct translated to a logic program suitable for an ASP Engine to solve, which would give trajectories as answer sets. Therefore, the next section presents a relation of the action language to a formalisation of a sequential decision making problem.

5 Combining ASP and MDP using Action Theory

Given an transition system $\mathcal{D}$ as described in the previous section, a set of initial states $\mathcal{I}$ and a set of goal states $\mathcal{G}$, a planning problem described by $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, in which the objective is to find the plan that states at any state $s_0 \in \mathcal{I}$ and ends at any state $s_g \in \mathcal{G}$, respecting the transition system described in $\mathcal{D}$ has its solution described by a trajectory . This trajectory is an answer set of an Answer Set Planning problem or a $\mathcal{BC}+$ solution. Also, a trajectory of a planning problem describes the same concept of a history of a MDP.

Theorem 1. *Given a planning problem described by an action language such that $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ describes a set of initial states $\mathcal{I}$, a set of goal states $\mathcal{G}$ and a transition system $\mathcal{D}$, if this description represents the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{T}$ of a MDP $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{D}, \mathcal{T}, \mathcal{R} \rangle$, then a trajectory for $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ is equivalent to a history to M.*

Proof. A MDP $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{D}, \mathcal{T}, \mathcal{R} \rangle$ has a set of states $\mathcal{S}$ which has a subset of possible initial states and a subset of possible final states. Thus the sets $\mathcal{I}$ and $\mathcal{G}$ of a planning problem described by $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ are $\mathcal{I} \subseteq \mathcal{S}$ and $\mathcal{G} \subseteq \mathcal{S}$.

The sets $\mathcal{A}$ and $\mathcal{T}$ of M and the complete set $\mathcal{S}$ are described in the transition system $\mathcal{D}$, such that every $s \in \mathcal{S}$ is a fluent formula in $\mathcal{D}$ and every $a \in \mathcal{A}$ is an action formula in $\mathcal{D}$. The transition function $\mathcal{T}$ is the set of laws in $\mathcal{D}$.

Considering a history h_g that finds the solution to the problem, a history h_g starts in any possible initial state and ends in any possible final state. Therefore a history is equivalent to a trajectory for an planning problem, which is an answer set for an Answer Set Planning problem. □

Each execution history $h_g \in \mathcal{H}_g$ describe subsets of $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{T}$ of an MDP $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{D}, \mathcal{T}, \mathcal{R} \rangle$. Thus, from $\mathcal{H}_g$ it is possible to find an MDP $M' \subseteq M$.

Theorem 2. *The set $\mathcal{H}_g$ of execution histories defines a MDP $M' = \langle \mathcal{S}', \mathcal{A}', \mathcal{T}', \mathcal{R} \rangle$ that is a subset of the MDP $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.*

Proof. A MDP is formalised as $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, which can be described by a logic program Π and each of the answer sets that solves Π is a trajectory to M.

The set of all available states in the problem is described in $\mathcal{S}$, but considering the restrictions in Π, there is a set $\bar{\mathcal{S}} \subseteq \mathcal{S}$ of states that are not possible or allowed to visit. Thus, the set of states that are visited when searching for solutions is defined by $\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' = \mathcal{S} - \bar{\mathcal{S}}$, that is the set of all available states without the states that are restricted in Π.

Similarly, for the set $\mathcal{A}$ of actions, there may be a set $\bar{\mathcal{A}}$ of unavailable actions that are not able to be performed, thus the remaining choices are in the set $\mathcal{A}' \subseteq \mathcal{A} \mid \mathcal{A}' = \mathcal{A} - \bar{\mathcal{A}}$.

Is is not necessary for an agent to be able to execute every action in every state. Thus, the possible reduction in the sets of states and actions does not affect the MDP's description, furthermore, it allows for a planner to be more efficient, since it removes unnecessary states and actions [Boutilier *et al.*, 1999].

The trajectories $h_g \in \mathcal{H}_g$ for M give a sequence of states s and actions a that leads from a initial state $s_0 \in \mathcal{S}$ to the goal state $s_g \in \mathcal{S}$. Since $\mathcal{S}' \subseteq \mathcal{S}$ and $\mathcal{A}' \subseteq \mathcal{A}$ given the restrictions, s_0 and s_g are also in $\mathcal{S}'$, along with every state $s \in \mathcal{H}_g$ and actions $a \in \mathcal{H}_g$ for each trajectory h_g. Furthermore, given the set $\mathcal{H}_g$ of trajectories for M, it is possible to extract $\mathcal{S}'$ and $\mathcal{A}'$ by verifying which states and actions are in $\mathcal{H}_g$.

Along with $\mathcal{S}'$ and $\mathcal{A}'$, the set $\mathcal{H}_g$ of trajectories (or execution histories) give the transition system $\mathcal{D}$. Since an execution history gives the sequence of states and actions , it is possible to notice that the consequence of executing the action a_n in the state s_{n-1} is the next state s_n, for $1 \leq n \leq m$. Thus, by following each of the trajectories $h_g \in \mathcal{H}_g$ it is possible to extract a MDP's transition function $\mathcal{T}'(\mathcal{S}', \mathcal{A}')$.

Given the restrictions in Π this new transition function $\mathcal{T}'$ is not allowed to execute actions in states that it cannot visit ($\bar{\mathcal{S}} \times \mathcal{A}$), to execute actions that are not allowed in any states ($\mathcal{S} \times \bar{\mathcal{A}}$) and to execute actions that are not allowed in states that are forbidden ($\bar{\mathcal{S}} \times \bar{\mathcal{A}}$). Also, it cannot visit forbidden states by executing allowed actions in allowed states ($\mathcal{S} \times \mathcal{A} \mapsto \bar{\mathcal{S}}$). Thus, this new transition function can be defined as:

$$\mathcal{T}'(\mathcal{S}, \mathcal{A}) = \big((\mathcal{S} \times \mathcal{A}) - (\bar{\mathcal{S}} \times \mathcal{A}) - (\mathcal{S} \times \bar{\mathcal{A}}) - (\bar{\mathcal{S}} \times \bar{\mathcal{A}}) - (\mathcal{S} \times \mathcal{A} \mapsto \bar{\mathcal{S}})\big) \mapsto \mathcal{S}'.$$

Therefore, the set $\mathcal{H}_g$ of execution histories for a MDP M defines a states set $\mathcal{S}' \subseteq \mathcal{S}$, action set $\mathcal{A}' \subseteq \mathcal{A}$ and a transition function $\mathcal{T}' \subseteq \mathcal{T}$. With this three new sets and the previously defined return function $\mathcal{R}$, it is possible to describe a new MDP $M' = \langle \mathcal{S}', \mathcal{A}', \mathcal{T}', \mathcal{R} \rangle$ so that $M' \subseteq M$, given the restrictions in Π. □

Once it is known that there is a MDP $M' \subseteq M$ that goes through only the allowed states and executes only the allowed actions, given the restrictions applied to Π, it is important to know that the optimal policy π^*_M for the MDP M is equivalent, considering the sequence of actions, to the optimal policy $\pi^*_{M'}$ of the MDP $M' \subseteq M$.

Theorem 3. *The optimal solution π^*_M for the MDP M is equivalent to the optimal solution $\pi^*_{M'}$ to the MDP $M' \subseteq M$ described by the set $\mathcal{H}$ of histories.*

Proof. The reward function $\mathcal{R}$ that must be maximised or minimised for M and M' is the same, therefore, if there are no restrictions in Π, $M' = M$ and $\pi^*_{M'} = \pi^*_M$. If there are restrictions in Π, since M' is formalised from the execution histories of M, every solution to M is in M' given the restrictions applied by Π to M. □

6 An implementation of MC Policy Evaluation with answer sets

MC Policy Evaluation Algorithm can be used to estimate the returns for the MDP. Using the description given in a logic program that uses answer set semantics, it is possible to build the sets $\mathcal{S}'$ of states and $\mathcal{A}'$ of actions and use the answer sets as set of trajectories (or histories) that need to be evaluated in order to find the optimal policy π^*.

Using the MC Policy Evaluation Algorithm, in the end of the interaction with the environment the $Q(s, a)$ functions is estimated and in order to find the optimal policy π^* it is only needed to extract the action a that maximises (or minimises) the return.

The algorithm that describes the process used in this work is presented in algorithm 1.

Use the Action Theory $\mathcal{BC}+$ to describe the problem.	1
Translate the program in $\mathcal{BC}+$ to a logic program	2
using answer set semantics.	3
Find the answer sets using an ASP engine.	4
Initialize the $Q(s, a)$ function using the instance	5
and the answer sets found in step 3.	6
Repeat forever:	7
For each answer set in the set:	8
Generate a random sample interacting	9
with the environment.	10
Update the $Q(s, a)$ function using	11
the values received by the sample.	12

Algorithm 1: Algorithm for using Monte Carlo for learning probabilities in answer sets.

The cost of executing an action in a domain can be estimated within a certain number of episodes, as it is with MC Methods and Reinforcement Learning [Sutton and Barto, 1998; 2012]. In the case presented above, the cost of performing an action is estimated based on expected average reward. Once the algorithm converges, it can stop interacting with the environment.

Therefore, for a given instance, when the MC Policy Evaluation finishes, we have the optimal policy and the solution for a problem considering transition probabilities. But, if this domain changes overtime, it is possible to use the knowledge already gathered in this instance. If the probabilities or the reward function changes, there is no need to find the answer sets again. One has only to perform the MC Policy Evaluation.

If the domain changes to aggregate more actions or states, there is no need to modify the learning procedure, but it is necessary to find new answer sets.

It will only be necessary to perform the whole process again if the whole transition function changes. In this case, the transitions change, generating a new set of answer sets, which will have different probabilities and a different optimal policy.

Finally, using this method it may be possible to use the solutions found for a given instance to bootstrap the search of a solution for a different instance or for problems that may change over time.

This section presented how the proposed method for combining planning with answer sets and MC Policy Evaluation works. The next sections present the experiments performed with this method and the results obtained.

7 Experiments

In order to verify the functionality of the proposed method, a problem of process allocation in servers (presented as an example in section 2) is used.

In this problem, there is a list of servers that can execute tasks and each server has a different performance (time needed to execute a given process) and are assigned probability of correctness (probability of returning the right answer given the execution of the process or the being able to return any answer at all). Thus, once a process is allocated to a server, it may provide three different responses:

1. Return the right answer in a given time;
2. Return the wrong answer in a given time;
3. Do not return any answer at all.

Along with the server, we have a list of processes that need to be executed and each of these processes has an expected time to be processed.

Therefore, the MDP used to describe this domain is:

- $\mathcal{S}$: the list of processes that have not been processed yet;
- $\mathcal{A}$: the act of delegation of a process to a given server;
- $\mathcal{T}$: in the ASP, the transition is deterministic. Once the process has been allocated to a server, the server will provide an answer and the process will be removed from the list that needs to be processed. In the learning process, there is a chance that the process may be wrongly processed or no answer is given by the server;
- $\mathcal{R}$: the reward provided to the learning process uses the time expected to execute a process (T_e) and the real time needed to execute a process (T_n). In the case that the server provides the right answer, it receives $T_e * (T_n/100)$, in any other case it receives $-(T_e/(T_n/100))$.

For this experiment, a list of five processes (numbered from 1 to 5) was used. The processes 1, 2 and 3 needed 100 units of time and the processes 4 and 5 needed 150. A list of five servers were named by letters from 'a' to 'e' with probabilities to return the right answer of $(0.5, 1.0, 1.0, 0.9, 0.5)$ and expected times for computing of $(50, 75, 100, 125, 150)$ respectively, as described in Listing 1.

If we calculate the number of feasible policies for this MDP [Bellman and Dreyfus, 1962], then we have the number of available states to the power of the number of available actions, which is $5^5 = 3125$ feasible policies. When solving this problem using ASP and applying the restriction as described in Section 3.1 and Listing 2, we were able to find 120 solutions. Thus, by using a planner before the optimisation we are restricting the problem to the 120 solutions that we are interested at.

7.1 ASP Implementation

The ASP implementation uses the encoding in Listing 2 and the instance in Listing 1 to describe part of the MDP that is used to generate the answer sets.

The instance has the description of the processes, with its names and times, the description of the servers, with names, times and probabilities. These will be used to describe the states and actions for the ASP grounder and the MC method. The reward function was left for MC, since in this case ASP will not make use of it.

The encoding has a deterministic transition function that describes the allocation of a process to a server with the condition that before a server can receive an allocation, every other server has to be already processing. This condition is enough to generate the answer sets needed to find the optimal allocation of processes to servers and not to allow the allocation of every process in a single server.

This reward criteria was chosen while considering the MDP that would be solved by a learning process. If it was necessary for the learning process to provide solutions that respects this criterion, than the action of allocating a process to a server that already is processing would receive a negative reward. Nevertheless, if for finding the optimal policy it is necessary to execute every feasible policy, than it would be necessary to spend some time performing this policies that are wanted to be avoided. Thus, the ASP implementation restricts this answers to be executed.

The instance and encoding in Listings 1 and 2 is then used by the ASP engine and, if the instance is satisfiable, answer sets are provided to the MC for Policy Evaluation.

7.2 MC Implementation

The Monte Carlo implementation uses the same instance as the ASP that has the list of five servers and five processes, along with answer sets found by the ASP engine.

The policy evaluation process used consists of generating random samples of each of the obtained answer sets and, using each of these samples, to estimate the $Q(s, a)$ value. Each generation of a sample for each possible answer is considered an episode.

It is important to notice that the probabilities and times informed by the instance in Listing 1 are only used by the domain implementation, and not by the Monte Carlo algorithm. Thus, the probabilities given in the instance are used only to verify if the algorithm has found the correct probability, but not to provide any information to it.

8 Results

The values learned for each state-action pair is given in Table 1 and, to find the optimal solution, it is only needed to look for the actions that maximise the return for each state.

Thus, the first process would be allocated in the server 'd', the next one in the server 'c', followed by the server 'b'. Next, the server 'e' would receive a process to run and, lastly, server 'a' would receive a process. Nevertheless, this is the optimal solution for this Instance of the problem, which is to say that if we change the restrictions applied to the logic program, there is a chance that the optimal solutions is different.

Available States	Available Actions				
	'a'	'b'	'c'	'd'	'e'
{5,4,3,2,1}	-74.93	75.00	100.00	106.10	42.57
{5,4,3,2}	-74.41	75.00	100.00	103.97	39.35
{5,4,3}	-75.19	75.00	100.00	104.92	41.55
{5,4}	-112.99	112.50	150.00	156.67	63.00
{5}	-107.32	112.50	150.00	160.75	64.19

Table 1: $Q(s, a)$ function learned by using the MC and ASP for the problem of process allocation.

9 Related Work

A combination of Reinforcement Learning and First Order Logic has been made by [van Otterlo, 2009], where the sets $\mathcal{S}$ of states and $\mathcal{A}$ of actions are described using FOL and the optimal policy for the problem is found using a Reinforcement Learning method.

The investigation of extending ASP towards probabilistic domains is recent.

A proposal is P-Log [Baral *et al.*, 2004; Gelfond and Rushton, 2010]. P-Log uses a variation of the language used to describe an ASP domain, so that it allows for the description of *a priori* probabilities.

QASP [Nickles, 2012] is a framework that combines Reinforcement Learning (specifically, Q-Learning) with ASP. In this framework, in each time step that the agent selects an action to execute, an ASP engine is called to find answer sets for that step onwards; the agent then selects an action that is available in one of those answer sets.

Proposals that are in the same line as QASP are the ones by [Khandelwal *et al.*, 2014; Yang *et al.*, 2014], which combine ASP (or the action language BC translated to ASP) with actions cost estimation by interacting with an environment.

The work proposed in the present paper is in line with that described in [Khandelwal *et al.*, 2014; Yang *et al.*, 2014] as we do not propose a probabilistic version of ASP, but we advocate a possible way to use answer sets for providing policies for non-deterministic domains. Instead of generating plans as needed to solve a problem, the ASP engine is used as an oracle to provide each feasible policy and use them to describe a MDP that is a subset of a larger problem. This MDP described by the answer sets is them evaluated by the MC policy evaluation. The advantage of using ASP in this fashion is to reduce the MDP's search space to the policies that are actual solutions to the problem at hand.

10 Conclusion

This paper presented a method for estimating policies costs for answer sets described as a Markov Decision Process. The proposed solution is to use the instance passed to ASP with the answer sets found by the ASP engine to perform MC Policy Evaluation. With this combination, it was possible to determine the optimal policy once the transition probability is estimated and the reward is received.

Experiments were performed in a Process Allocation domain. It is possible to notice that ASP was able to provide the policies to the MC Policy Evaluation and, by iterating with the domain using the provided answer sets, it was possible to estimate the cost of performing an action.

Future works shall be considered in three different fronts. The first is the direct usage of $\mathcal{BC}+$ to describe the domain and them translated this description into an ASP code by using the CPLUS2ASP framework [Babb and Lee, 2013].

The second is in the use of other methods along with Monte Carlo, such as Reinforcement Learning, to find the optimal policy. As discussed in Section 2, the purpose of the method proposed in this work is to find a MDP M' that is a subset of a MDP M, which describes the whole problem, being M' search space that has the requested constraints of M. Thus, once M' is determined, the method that will be used to find the optimal policy can be freely chosen.

And third, the generation of a new instance based in the $Q(s, a)$ function estimated by the MC, passed to a new ASP with language directives to minimise costs and maximise transition probabilities, thus allowing for changes in the instance over time, is being considered.

Further experiments will also take place in more complex domains in order to measure the gain that can be achieved in the reduction of the search space, along with a comparison with other methods to estimate the optimal answer set.

Acknowledgments

Paulo E. Santos is partially sponsored by CNPq (grant 307093/2014-0).

References

[Babb and Lee, 2013] J. Babb and J. Lee. Cplus 2ASP: Computing Action Language C+ in Answer Set Programming. In *Logic Programming and Nonmonotonic Reasoning*. Springer, 2013.

[Babb and Lee, 2014] J. Babb and J. Lee. Action Language BC+: Preliminary Report. In *Working Notes of the 7th Workshop of ASPOCP*, 2014.

[Baral *et al.*, 2004] C. Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. In *Logic Programming and Nonmonotonic Reasoning*. Springer, 2004.

[Bellman and Dreyfus, 1962] R. E. Bellman and S. E. Dreyfus. Applied dynamic programming. 1962.

[Bellman, 1954] R. E. Bellman. The theory of dynamic programming. Technical report, DTIC Document, 1954.

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assimptions and Computational Leverage. *Journal of Artificial Intelligence Research*, (11):1–94, 1999.

[Eiter *et al.*, 2003] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Answer set planning under action costs. *JAIR*, 2003.

[Gebser *et al.*, 2012] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer Set Solving in Practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238, 2012.

[Gelfond and Rushton, 2010] M. Gelfond and N. Rushton. Causal and probabilistic reasoning in p-log. *Heuristics, Probabilities and Causality. A tribute to Judea Pearl*, 2010.

[Gelfond, 2008] Michael Gelfond. Answer sets. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, page 285316. Elsevier, 2008.

[Khandelwal *et al.*, 2014] P. Khandelwal, F. Yang, M. Leonetti, V. Lifschitz, and P. Stone. Planning in action language bc while learning action costs for mobile robots. In *ICAPS*, 2014.

[Lifschitz, 1999] V. Lifschitz. Answer set planning. In *Logic Programming and Nonmonotonic Reasoning*. Springer, 1999.

[Lifschitz, 2002] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1), 2002.

[Mausam and Kolobov, 2012] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*, volume 6. Morgan & Claypool Publishers, 2012.

[Nickles, 2012] M. Nickles. A system for the use of answer set programming in reinforcement learning. In *Logics in Artificial Intelligence*. Springer, 2012.

[Russell and Norvig, 2004] S. J. Russell and P. Norvig. *Artificial Intelligence*. Pearson Education India, NJ, USA, 2 edition, 2004.

[Subrahmanian and Zaniolo, 1995] V. S. Subrahmanian and C. Zaniolo. Relating stable models and ai planning domains. In *ICLP*, volume 95, 1995.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement learning: An Introduction* . The MIT press, MA, USA, 1998.

[Sutton and Barto, 2012] R. S. Sutton and A. G. Barto. Reinforcement learning: An Introduction, 2012. Accessed: Nov. 11, 2014.

[van Otterlo, 2009] M. van Otterlo. *The logic of adaptive behavior*. IOS Press, Amsterdam, 2009.

[Wald, 1947] A. Wald. Foundations of a general theory of sequential decision functions. *Econometrica, Journal of the Econometric Society*, 1947.

[Wald, 1950] A. Wald. Statistical decision functions. 1950.

[Yang *et al.*, 2014] F. Yang, P. Khandelwal, M. Leonetti, and P. Stone. Planning in answer set programming while learning action costs for mobile robots. In *AAAI-SSS*, 2014.