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.