Reinforcement Learning and Automated Planning: A Survey

Ioannis Partalas, Dimitris Vrakas and Ioannis Vlahavas
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{partalas, dvrakas, vlahavas}@csd.auth.gr

Abstract

This article presents a detailed survey on Artificial Intelligent approaches, that combine Reinforcement Learning and Automated Planning. There is a close relationship between those two areas as they both deal with the process of guiding an agent, situated in a dynamic environment, in order to achieve a set of predefined goals. Therefore, it is straightforward to integrate learning and planning, in a single guiding mechanism and there have been many approaches in this direction during the past years. The approaches are organized and presented according to various characteristics, as the used planning mechanism or the reinforcement learning algorithm.

1. INTRODUCTION

Reinforcement Learning and Automated Planning are two approaches in Artificial Intelligence that solve problems by searching in a state space. Reinforcement Learning is a subcategory of the Machine's Learning field, an Artificial Intelligence's area concerned with the computer systems design, that improve through experience. Automated Planning is one of the most important areas of Artificial Intelligence; it deals with the design and implementation of systems (planners), that automatically produce plans (i.e sequences of actions) which lead the world from its current state to another one that satisfies a set of predefined goals.

Machine Learning has been widely used to support planning systems in many ways. A detailed survey can be found in (Zimmerman & Kambhambati, 2003), where the authors describe numerous ways of how learning can assist planning. Furthermore, the existing techniques over the past three decades are categorized with respect to both the underlying planning and the learning component. More specifically, the techniques are organized according to: a) the planning approach (state space search, plan space search), b) the planning-learning goal (speed up planning, improve plan quality, learn or improve domain theory), c) the learning phase (before planning, during planning process, during plan execution) and d) the type of learning (analytic, inductive, multistrategy).

In (Vrakas et al., 2005), a compact review of planning systems, that utilize machine learning methods is presented, as well as background information on the learning methodologies, that have been mostly used to support planning systems. Especially, this work concerns with the problem of the automatical configuring of a planning system's parameters, and how this problem is solved through machine learning techniques. In addition, it presents two different adaptive systems that set the planning parameters of a highly adjustable planner, based on the measurable characteristics of the problem instance.

Recently, Bonet and Geffner (2006) introduced a unified view of Planning and Dynamic Programming methods. More specifically, they combined the benefits of a general dynamic programming formulation with the power of heuristic-search techniques for developing an algorithmic framework.

In this chapter, we review methods that combine techniques from both Reinforcement Learning and Planning. The rest of the chapter is structured as follows: Section 2 provides the basic theory of Automated Planning and Reinforcement Learning respectively. The next section reviews surveys that have been presented in the bibliography, about the research work related to the combination of Machine Learning and Planning. In section 4 the related articles are reviewed and they are categorized according to certain criteria; also their main aspects are presented. Section 5 discusses the findings of the survey and concludes the chapter.

2. Background

2.1 Planning

Planning is the process of finding a sequence of actions (steps), which if executed by an agent (biological, software or robotic) result in the achievement of a set of predefined goals. The sequence of actions mentioned above is also referred to as plan.

The actions in a plan may be either specifically ordered and their execution should follow the defined sequence (linear plan) or the agent is free to decide the order of executions as long as a set of ordering constraints are met. For example, if someone whishes to travel by plane, there are three main actions that he/she has to take: a) buy a ticket, b) go to the airport and c) board on the plane. A plan for traveling by plane could contain the above actions in a strict sequence as the one defined above (first do action a, then b and finally c), or it could just define that action c (board on plane) should be executed after the first two actions. In the second case the agent would be able to choose which plan to execute, since both a→b→c, and b→a→c sequences would be valid.

The process of planning is extremely useful when the agent acts in a dynamic environment (or world) which is continuously altered in an unpredictable way. For instance, the auto pilot of a plane should be capable of planning the trajectory that leads the plane to the desired location, but also be able to alter it in case of an unexpected event, like an intense storm.

The software systems that automatically (or semi- automatically) produce plans are referred to as Planners or Planning Systems. The task of drawing a plan is extremely complex and it requires sophisticated reasoning capabilities which should be simulated by the software system. Therefore, Planning Systems make extensive use of Artificial Intelligence techniques and there is a dedicated area of AI called Automated Planning.

Automated Planning has been an active research topic for almost 40 years and during this period a great number of papers describing new methods, techniques and systems have been presented that mainly focus on ways to improve the efficiency of planning systems. This can be done either by introducing new definition languages that offer more expressive power or by developing algorithms and methodologies that produce better solutions in less time.

2.1.1 Problem Representation

Planning Systems usually adopt the STRIPS (Stanford Research Institute Planning System) notation for representing problems (Fikes and Nilsson, 1971). A planning problem in STRIPS is a tuple $<I,A,G>$ where I is the Initial state, A a set of available actions and G a set of goals.

States are represented as sets of atomic facts. All the aspects of the initial state of the world, which are of interest to the problem, must be explicitly defined in I. State I contains both static and dynamic information. For example, I may declare that object John is a truck driver and there is a road connecting cities A and B (static information) and also specify that John is initially located in city A (dynamic information). State G on the other hand, is not necessarily complete. G may not specify the final state of all problem objects either because these are implied by the context or because they are of no interest to the specific problem. For example, in the logistics domain the final location of means of transportation is usually omitted, since the only objective is to have the packages transported. Therefore, there are usually many states that contain the goals, so in general, G represents a set of states rather than a simple state.

Set A contains all the actions that can be used to modify states. Each action A_i has three lists of facts containing:

a) the preconditions of A_i (noted as $prec(A_i)$)

b) the facts that are added to the state (noted as $add(A_i)$) and

c) the facts that are deleted from the state (noted as $del(A_i)$).

The following formulae hold for the states in the STRIPS notation:

- An action A_i is applicable to a state S if $\text{prec}(A_i) \subseteq S$.
- If A_i is applied to S, the successor state S' is calculated as:

$S' = S \setminus \mathrm{del}(A_i) \cup \mathrm{add}(A_i)$

- The solution to such a problem is a sequence of actions, which if applied to I leads to a state S' such as $S' \supseteq G$.

Usually, in the description of domains, action schemas (also called operators) are used instead of actions. Action schemas contain variables that can be instantiated using the available objects and this makes the encoding of the domain easier.

The choice of the language in which the planning problem will be represented strongly affects the solving process. On one hand, there are languages that enable planning systems to solve the problems easier, but they make the encoding harder and pose many representation constraints. For example, propositional logic and first order predicate logic do not support complex aspects such as time or uncertainty. On the other hand there are more expressive languages, such as natural language, but it is quite difficult to enhance planning systems with support for them.

The PDDL Definition Language

PDDL stands for Planning Domain Definition Language. PDDL focuses on expressing the physical properties of the domain that we consider in each planning problem, such as the predicates and actions that exist. There are no structures to provide the planner with *advice*, that is, guidelines about how to search the solution space, although extended notation may be used, depending on the planner. The features of the language have been divided into subsets referred to as requirements, and each domain definition has to declare which requirements will put into effect.

After having defined a planning domain, one can define problems with respect to it. A problem definition in PDDL must specify an initial situation and a final situation, referred to as goal. The initial situation can be specified either by name, or as a list of literals assumed to be true, or both. In the last case, literals are treated as effects, therefore they are added to the initial situation stated by name. The goal can be either a goal description, using function-free first order predicate logic, including nested quantifiers, or an expansion of actions, or both. The solution given to a problem is a sequence of actions which can be applied to the initial situation, eventually producing the situation stated by the goal description, and satisfying the expansion, if there is one.

The initial version of PDDL has been enhanced through continuous extensions. In version 2.1 (Fox and Long, 2003) the language is enhanced with capabilities for expressing temporal and numeric properties of planning domains. Moreover the language supports durative actions and a new optional field in the problem specification that allowed the definition of alternative plan metrics to estimate the value of a plan. PDDL 2.2 (Edelkamp and Hoffmann 2004) added derived predicates, timed initial literals, which are facts that become true or false at certain time points known to the planner beforehand. In PDDL 3.0 (Gerevini and Long, 2005) the language was enhanced with constructs that increase its expressive power regarding the plan quality specification. These constructs mainly include strong and soft constraints and nesting of modal operators and preferences inside them.

2.1.2 Planning Methods

The roots of Automated Planning can be traced back in the early '60s with the General Problem Solver (Newell and Simon 1963) being the first planning system. Since then a great number of planning systems have been published that can be divided into two main categories: a) classical and b) neo-classical planning.

Classical Planning

The first approaches for solving Planning problems included Planning in state-spaces, planning is plan-spaces and hierarchical planning.

In state-space planning a planning problem is formulated as a search problem and then heuristic search algorithms, like Hill-Climbing or A* are used for solving it. The search in state-space planning can be forward (move from the initial state towards the goals), backward (start from the goals and move backward towards the goals) or bi-directional. Examples of this approach include the STRIPS planner among with some older systems such as GPS.

Plan-space differs from state-space planning in that the search algorithms do not deal with states but with incomplete plans. The algorithms in this category start with an empty plan and continually improve it by adding steps and correcting flaws until a complete plan is produced. A famous technique that is used by plan-space planners is the *Least Commitment Principle*, according to which the system postpones all the commitments (e.g. variable grounding) as long as it can. Such systems are the SNLP (McAllester and Rosenblitt 1991) and UCPOP (Penberthy and Weld 1992).

Hierarchical Planning is mainly an add-on to either state-space planning according to which the problem is divided into various levels of abstraction. This division requires extra knowledge provided by a domain expert, but the total time needed for solving the problem can be significantly reduced. The most representative examples of hierarchical planners are the ABSTRIPS (Sacerdoti 1974) and the ABTWEAK (Yang and Tenenberg 1990) systems.

Neo-classical Planning

The second era in planning techniques, called neo-classical planning, embodies planning methodologies and systems that are based on: planning graphs, SAT encodings, CSPs, Model Checking and MDPs and Automatic Heuristic Extraction.

Planning graphs are structures that encode information about the structure of the planning problems. They contain nodes (facts), arrows (operators) and mutexes (mutual exclusion relations) and are used in order to narrow the search for solution in a smaller part of the search space. The first planner in this category was the famous GRAPHPLAN (Blum and Furst 1995) and it was followed by many others such as IPP (Koehler et al. 1997) and STAN (Fox and Long 1998).

Another neoclassical approach for planning is to encode the planning problem in a propositional satisfiability problem, which is a set of logic propositions containing variables. The goal in such problems is to prove the truth of the set by assigning Boolean values to the referenced variables. The most representative systems based on satisfiability encodings are the SATPLAN (Kautz and Selman 1992) and BLACKBOX (Kautz and Selman 1998).

Transforming the planning problem in a constraint satisfaction problem is another neoclassical technique, able to handle problems with resources (e.g. time). By encoding the planning problem in a CSP one can utilize a number of efficient solving techniques and therefore obtain a good solution very fast. There are many systems and architectures based on this model such as OMP (Cesta, Fratini and Oddi 2004) and CTP (Tsamardinos, Pollack and Vidal 2003).

Encoding the planning problem as a Model Checking or Markov Decision problem are two neoclassical approaches, able to support uncertainty in planning problems. Uncertainty is a key factor especially when planning in dynamic real word situations, as the environment of a robot. These two neoclassical techniques have proven to be quite efficient for uncertain environments with partial observability capabilities (Boutilier, Reiter and Price 2001, Giunchiglia and Traverso 1999).

The last neoclassical methodology is to perform domain analysis techniques on the structure of the problem, in order to extract knowledge that is then embedded in heuristic functions. The first system towards this direction was McDermott's UNPOP (McDermott 1996) which used the means-end-analysis in order to analyze the problem and create a heuristic mechanism that was then used by a state-space heuristic search algorithm. The example of UNPOP was followed by a large number of systems, with the most important of them being HSP (Bonet and Geffner 2001), FF (Hoffmann and Nebel 2001) and AltAlt (Nguyen, Kamphambati and Nigenda 2000). The reader should consider (Ghalab et al. 2004) for a more comprehensive view of Automated Planning.

2.2 Reinforcement Learning

2.2. 1 Introduction in Reinforcement Learning

Reinforcement Learning (RL) addresses the problem of how an agent can learn a behaviour through trial and error interactions with a dynamic environment (Kaelbling, Littman & Moore

1996), (Sutton & Barto 1999). RL is inspired by the reward and punishment process encountered in the learning model of most living creatures. The main idea is that the agent (biological, software or robotic) is evaluated through a scaled quantity, called reinforcement (or reward), which is received from the environment as an assessment of the agent's performance. The goal of the agent is to maximize a function of the reinforcement in the next iteration (greedy approach). The agent is not told which actions to take, as in most forms of Machine Learning, but instead it must discover which actions yield the most reward by trying them.

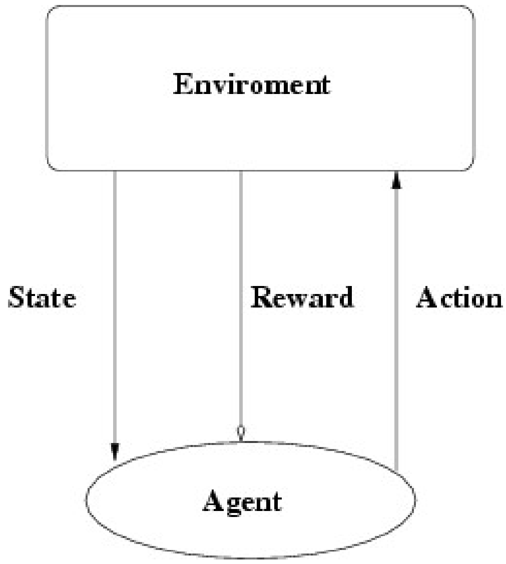
RL tasks are framed as Markov Decision Processes (MDP), where an MDP is defined by a set of states, a set of actions and the dynamics of the environment, a transition function and a reward function. Given a state s and an action a the transition probability of each possible successor state s' is $P^{a}_{ss'} = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ and the expected value of the reward is $R^{a}_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$. Each MDP satisfies the Markov property which states that the environment's reponse at t+1 depends only on the state and actions representations at t, in which case the environment's dynamics can be defined by specifying only $\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\}$, for all s', at, $r_t, \ldots, r_1, s_0, a_0$. Based on this property we can predict the next state and the expected next reward given the current state and action.

2.2.2 Reinforcement Learning Framework

In an RL task the agent, at each time step, senses the environment's state, $s_t \in S$, where S is the finite set of possible states, and selects an action $a_t \in A(s_t)$ to execute, where $A(s_t)$ is the finite set of possible actions in state s_t. The agent receives a reward, $r_{t+1} \in \Re$, and moves to a new state s_{t+1}. Figure 1 shows the agent-environment interaction. The objective of the agent is to maximize the cumulative reward received over time. More specifically, the agent selects actions that maximize the expected discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma r_{t+k+1},$$

where $\gamma, 0 \leq \gamma < 1$, is the discount factor and expresses the importance of future rewards.



The agent learns a *policy π*, a mapping from states to probabilities of taking each available action, which specifies that in state *s* the probability of taking action *a* is *π(s,a)*. The above procedure is in real time (online) through the interaction of the agent and the environment. The agent has no prior knowledge about the behavior of the environment (the transition function) and the way that the reward is computed. The only way to discover the environment's behavior is through trial-and-error interactions and therefore the agent must be capable of exploring the state space to discover the actions which will yield the maximum returns. In RL problems a crucial task is how to trade off exploration and exploitation, namely the need of exploiting quickly the knowledge that the agent acquired and the need of exploring the environment in order to discover new states and bear in a lot of reward over the long run.

2.2.3 Optimal Value Functions

For any policy *π*, the value of state *s*, $V^{\pi}(s)$, denotes the expected discounted return, if the agent starts from s and follows policy π thereafter. The value $V^{\pi}(s)$ of s under π is defined as:

$$V^{\pi}(s) = E_{\pi}\left\{R_t \middle| s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\},$$

where s_t and r_{t+1} denote the state at time t and the reward received after acting at time t, respectively.

Similarly the *action-value function*, $Q^{\pi}(s,a)$, under a policy *π* can be defined as the expected discounted return for executing *a* in state *s* and thereafter following policy *π*:

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \middle| s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\}.$$

The agent tries to find a policy that maximizes the expected discounted return received over time. Such a policy is called an *optimal* policy and is denoted as π^*. The optimal policies may be more than one and they share the same value functions. The optimal state-value function is denoted as:

$$V^*(s) = \max_a \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V^*(s')\right]$$

and the optimal action-value function as:

$$Q^*(s,a) = \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma \max_{a'} Q^*(s',a')\right].$$

The last two equations are also known as the *Bellman optimality equations*. A way to solve these equations and find an optimal policy is the use of dynamic programming methods such as *value iteration* and *policy iteration*. These methods assume a perfect model of the environment in order to compute the above value functions. In real world problems the model of the environment is often unknown and the agent must interact with its environment directly to obtain information which will help it to produce an optimal policy. There are two approaches to learn an optimal policy. The first approach is called *model-free* where the agent learns a policy without learning a model of the environment. A well known algorithm of these approaches is Q-learning which is based on value iteration algorithm. The second approach is called *model-based* where the agent learns a model of the environment and uses it to derive an optimal policy. Dyna is a typical algorithm of model-based approaches. These techniques will be presented in detail in the following sections as well as the way that planning techniques are integrated them.

2.3 Learning and Planning

Machine learning algorithms have been extensively exploited in the past to support planning systems in many ways. Learning can assist planning systems in three ways (Vrakas et al., 2005), (Zimmerman & Kambhambati, 2003):

- *learning to speed up planning*: In most of the problems a planner searches a space in order to construct a solution and many times it is forced to perform backtracking procedures. The goal of learning is to bias the search in directions that is possible to lead to high-quality plans and thus to speed up the planning process.
- *improving domain knowledge*: Domain knowledge is utilized by planners in pre-processing phases in order to either modify the description of the problem in a way that it will make easier for solving or make the appropriate adjustments to the planner to best attack the problem. So there are obvious advantages to a planner that evolves its domain knowledge by learning.
- *learn optimization knowledge*: Optimization knowledge is utilized after the generation of an initial plan, in order to transform it in a new one that optimizes certain criteria, i.e. the number of steps or usage of resources.

2.4 Reinforcement Learning and Planning

The planning approaches used by deliberative agents rely on the existence of a global model which simulates the environment (agent's world) during time. For example, if a state and an action is given, the model predicts the next state and can produce entire sequences of states and actions. The actions which the agent executes specify the plan, which can be considered as a policy.

Planning methods involve computing utility functions (value functions) that represent goals, in order to guide the agent in selecting the next action (i.e. policy). Reinforcement learning methods are also based on the estimation of utility functions, in order to improve the agent's policy. This common structure led researchers to integrate planning and reinforcement learning methods in order to improve their efficiency.

In reinforcement learning methods, the aim of the agent is to maximize the expected sum of the rewards received over time, while acting in a dynamic environment. This process is often very slow; in fact in real situations, where the complexity is very high, it is very hard to find an optimal policy. A method that can speed up process of learning is the use of planning algorithms; through this we can achieve a good policy with fewer environmental interactions, as they can improve the policy using the model of the environment.

Moreover, in planning under uncertain and partial observability, where the state of the environment is only partially visible at run-time, there is an impervious need to find ways to overcome the problems that arise in such environments. A solution is to model the problem of planning under the reinforcement learning framework and then to use the appropriate techniques in order to solve it.

3. APPROACHES COMBINING PLANNING WITH RL

We shall organize the various methodologies that combine planning and RL in two major categories: a) Planning for Reinforcement Learning and b) Reinforcement Learning for Planning. These categories will be further divided according to the RL and the planning methods that are used in the various approaches.

3.1 Planning for Reinforcement Learning

In this section we present the main aspects of the methods, that combine RL and Planning techniques in order to accelerate the learning process. The methods presented in the following section are based on two architectures that integrate planning and reinforcement learning, called Dyna, originally proposed by Sutton (1990), and Prioritized Sweeping (Moore & Atkenson, 1993). The various methodologies are organized in the two following categories: a) Dyna-based Methods and b) Prioritized Sweeping-based Methods. There are also methods that do not fall in these categories and will be presented in the Other Methods section.

3.1.1 Dyna Architecture

The most common and often used solution to speed up the learning procedure in RL is the Dyna framework (Sutton 1990, Sutton 1991). The basic idea of Dyna is that it uses the acquired experience in order to construct a model of the environment and then use this model for updating the value function without having to interact with the environment. Figure 1 shows the Dyna framework (Sutton & Barto 1999).

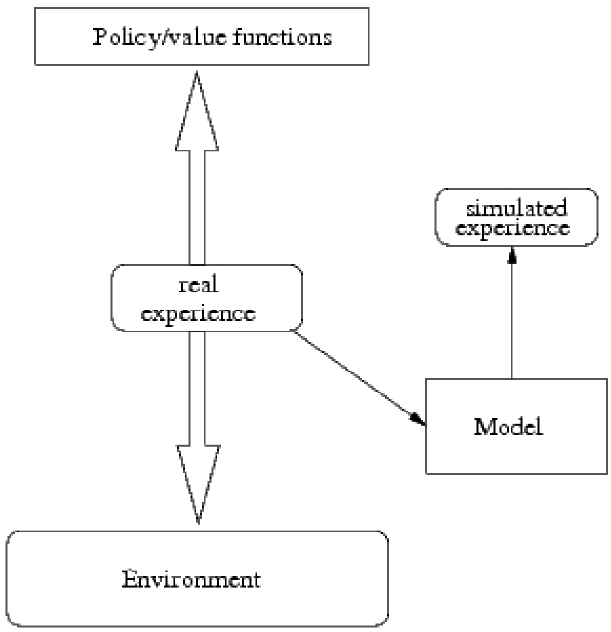


Figure 2: The Dyna framework.

3.1.2 Dyna based Methods

The dyna architecture forms as the basis of other techniques for combining reinforcement learning and planning. Sutton (1990) proposed a method that engages Dyna and Q-learning while in (Peng and Ronald, 1993) presented a prioritized version of Dyna. Lin (1992) extended the Adaptive Heuristic Critic and Q-learning with the internalization of Dyna. Finally, Zhao et al. (1999) extended the queue-Dyna algorithm by adding a component called exploration planning.

Dyna-Q

In (Sutton 1990) a method that combines Dyna and Q-learning (Watkins 1989) is proposed. Q-learning is a model free method which means that there is no need to maintain a separate structure for the value function and the policy but only the Q-value function. The Dyna-Q architecture is simpler as two data structures have been replaced with one. Moreover, Dyna-Q handles environments that are dynamic and may change over time. This is accomplished by adding a new memory structure that keeps the time steps n that elapsed since the last visit of every state-action pair. Then each state-action pair is given an exploration degree proportional to the square root of n. In this way the agent is encouraged to take actions that have not been chosen for a long time and thus to explore the state space.

Queue-Dyna

Peng and Ronald (1993) introduced queue-Dyna, a version of Dyna in which not all the state action pairs are considered to be useful and thus there is no need to be updated. In this way the algorithm is focused on places that may require updating through the planning process. The basic idea is that the values of the states are prioritized according to a criterion and only those having the highest priority are updated at each time step. A critical procedure of the proposed algorithm is the identification of the *update candidates*, i.e. places where the value function may require updating. The algorithm uses the Q-learning as the RL mechanism for updating the values of the state-action pairs. In order to determine the priority, the authors defined the *prediction difference* as the difference of the Q-value between the present state-action pair and the successor state. If this difference exhibits a predefined threshold the update candidate is placed in to the queue for updating. After this step the planning process is performed. The first update candidate from the queue is extracted and updated. Then the predecessors of the state updated are examined if their prediction differences are significant, and if they are then the states become update candidates and placed in to the queue. We must mention here that the priority which is assigned at the update candidates is based on the magnitude of the prediction difference which means that large differences get high priorities. The experimental results showed that the proposed method improves not only the computational effort of Dyna (Sutton 1990, Sutton 1991) but also the performance of the agent.

AHC-Relaxation Planning & Q-Relaxation PLanning

In (Lin, 1992), two extensions of the Adaptive Heuristic Critic (AHC) (Barto et al. 1983) and Q-learning are presented. Adaptive heuristic critic architecture consists of two components: a method that learns a value function for a fixed policy, and a method that uses the learned value function to adjust the policy in order to maximize the value function. Lin (1992) proposes a framework that combines AHC and a variation of relaxation planning (Sutton 1990). The proposed planning algorithm addresses some problems of relaxation planning such as the effort that is spend in hypothetical situations that will not appear in the real world by executing actions from states actually visited and not from randomly generated states. The disadvantage of this approach is that the number of the hypothetical experiences that can be generated is limited by the number of states visited. Additionally, the planning steps are performed selectively, i.e. when the agent is very decisive about the best action the planning process is omitted while in the other hand when the agent cannot decide accurately about the best action the planning process is performed. At the beginning of learning the planning process is performed frequently as the actions are been chosen with equal probability. As the agent acquires knowledge the planning steps are performed less often. The author also proposed a similar algorithm which combines Q-learning and relaxation planning. We must mention here that the above approaches adopt function approximation methods in order to tackle the problem of the large state space. The value functions are represented using neural networks and learned using a combination of temporal difference methods (Sutton 1988) and the backpropagation algorithm.

Exploration Planning

In (Zhao et al. 1999) the authors proposed an extension of the queue-Dyna architecture by adding a module called *exploration planning.* This module helps the agent to search for actions that have not been executed in the visited states in order to accelerate the learning rate. This is achieved by defining state-action pairs as *sub-goals* according to a metric similar to that of prediction difference that is used in queue-Dyna. When the agent detects a sub-goal, the path from the initial state is drawn. Then, starting from the sub-goal the Q values of each state-action pair in the path are updated in a backward form.

3.1.3 Prioritized Sweeping – based Methods

This section reviews methods that make use of the Prioritized Sweeping algorithm originally proposed by Moore and Atkenson (1993). In (Andre et al, 1998) presented an extension that scales up to complex environments. Dearden (2001) proposed a structured version of prioritized sweeping that utilizes Dynamic Bayesian Networks.

Prioritized Sweeping

The idea behind prioritized sweeping (Moore & Atkenson, 1993) is that instead of selecting randomly the state-action pairs that will be updated during the planning process is more efficient to focus in state-action pairs that are likely to be useful. This observation is similar, but has been developed independently, as in queue-Dyna where some state-action pairs are more urgent to be updated. At the beginning of the learning process the agent has not yet acquired enough knowledge in order to construct an accurate model of the environment and thus planning is not efficient and spends time in making wasteful backups. This problem becomes noticeable in large domains where the planning process with randomly generated state-action pairs would be extremely inefficient.

The prioritized sweeping algorithm tries to determine which updates are likely to be interesting. It is assumed that the states belong in two disjoint subsets, the non-terminal and the terminal states. When the agent visits a terminal state it can not left it. An update is interesting when there is a large change between the values of the state and the terminal states. The algorithm uses a queue where it stores the interesting states according to the change. When a real world observation (state) is interesting then all its predecessors are placed near to the top of the queue in order to be updated. In this way the agent focuses in a particular area but when the states are not interesting then it looks for other areas in the state-space.

Experiments on discrete state problems shown that there was a large improvement in the computational effort that needed other methods like Dyna or Q-learning. But a question that arises at this point is how prioritized sweeping would behave in more complex and non-discrete environments. In this case there is the need to alter the representation of the value functions and utilize more compact methods like neural networks instead of the common tabular approach.

Generalized Prioritized Sweeping

Andre, Friedman and Parr (1998) presented an extension of prioritized sweeping in order to address the problems that arise in complex environments and to deal with more compact representations of the value function. In order to scale-up to complex domains they make use of the *Dynamic Bayesian Networks* (Dean & Kanazawa, 1990) to approximate the value functions as they were used with success in previous works. The proposed method is called Generalized Prioritized

Sweeping (GenPs) and the main idea is that the candidate states for been updated are the states where the value function change the most.

Structured Prioritized Sweeping

In (Dearden 2001) the *structured policy iteration* (SPI) algorithm applied in a structured version of prioritized sweeping. SPI (Boutilier et al. 1995) is used in decision-theoretic planning and is a structured version of policy iteration. In order to describe a problem (which is formalized as an MDP) it uses DBNs and conditional probability trees. The output of the algorithm is a tree-structured representation of a value function and a policy. The main component of the SPI algorithm is a regression operator over a value function and an action that produces a new value function before the action is performed. This is similar to planning that takes a goal and an action and determines the desired state before the action is performed. The regression operator is computed in a structured way where the value function is represented as a tree and the results is another tree in which each leaf corresponds to a region of the state space.

Structured prioritized sweeping is based on generalized prioritized sweeping (Andre et al., 1998). The main difference is that in the proposed method the value function is represented in a structured way as described in to the previous paragraph. Additionally, another major change is that the updates are performed for specific actions whereas in prioritized sweeping for each state that is been updated all the values of the actions in that state must be recomputed. This feature gives a great improvement in the computational effort as only the necessary updates are performed. The structured prioritized sweeping algorithm provides a more compact representation that helps the speed up of the learning process.

3.1.4 Other Methods

Several other methods have been proposed in order to combine Reinforcement Learning with Planning. In (Grounds & Kudenko, 2005) presented a method that couples a Q-learner with a

STRIPS planner. Ryan and Pendrith (1998) proposed a hybrid system that combines teleo-reactive planning and reinforcement learning techniques while in (Ryan 2002) planning used to automatically construct hierarchies for hierarchical reinforcement learning. Finally, Kwee et al. (2001) introduced a learning method that uses a gradient-based technique that plans ahead and improves the policy before acting.

PLANQ-learning

Grounds & Kudenko (2005), introduced a novel method in reinforcement learning in order to improve the efficiency in large-scale problems. This approach utilizes the prior knowledge of the structure of the MDP and uses symbolic planning to exploit this knowledge. The method, called *PLANQ-learning*, consists of two components: a STRIPS planner that is used to define the behavior that must be learned (high-level behavior) and a reinforcement learning component that learns a low level behavior. The high-level agent has multiple Q-learning agents to learn each STRIPS operator. Additionally, there is a low-level interface from where the low level percepts can be red in order to construct a low-level reinforcement learning state. Then this state can be transformed to a high-level state that can be used from the planner. At each time step the planner takes a representation of the state according to the previous interface and returns a sequence of actions which solves the problem. Then the corresponding Q-learner is activated and it receives a reward for performing the learning procedure. Experiments showed that the method constrains the number of steps required by the algorithm to converge.

Reinforcement Learnt-TOPs

In this work reinforcement learning is combined with teleo-reactive planning to build behavior hierarchies to solve a problem like robot navigation. Teleo-reactive planning (Nilsson, 1994) is based on teleo-operators (TOPs) which each of them is an action that it can be a simple primitive action or a complex behavior. Teleo-reactive plans are represented as trees which each node is a state, and the root is the goal state. The actions are represented as connections between the nodes and when an action is performed in a node then the conditions of the parent node are achieved. In the proposed method the reinforcement learnt behaviors are represented as a list of TOPs and then a Teleo-Reactive planner is used to construct a plan in the form of a tree. Then an action is selected to be executed and the result is used to determine the reinforcement reward to feed the learner and the process repeats by selecting the following actions until the goal is achieved. In this way, behavior hierarchies are constructed automatically and there is no need to predefine these hierarchies. Additionally, an advantage of the proposed architecture is that the low-level TOPs that are constructed in an experiment can be used as the basis of solving another problem. This feature is important as the system does not spend further time in the reconstruction of new TOPs.

Teleo-Reactive Q-learning

Ryan (2003) proposed a hybrid system that combines planning and hierarchical reinforcement learning. Planning generates hierarchies to be used by an RL agent. The agent learns a policy for choosing optimal paths in the plan that produced from the planner. The combination of planning and learning is done through the Reinforcement Learning Teleo-operator (Ryan & Pendrith, 1998) as described in to the previous section.

3.2 Reinforcement Learning for Planning

This section presents approaches that utilize reinforcement learning to support planning systems. In (Baldassarre 2003) presented a planner that uses RL to speed up the planning process. In (Sun & Sessions, 1998), (Sun 2000) and (Sun et al., 2001) proposed a system that extract the knowledge from reinforcement learners and then uses it to produce plans. Finally, Cao et al. (2003) uses RL to solve a production planning problem.

3.2.1 Forward and Bidirectional Planning based on Reinforcement Learning

Baldassarre (2003) proposed two reinforcement learning based planners that are capable of generating plans that can be used for reaching a goal state. So far planning was used for speed up learning like in Dyna. In this work reinforcement learning and planning are coordinated to solve problems with low cost by balancing between acting and planning. Specifically, the system consists of three main components: a) the learner, b) the forward planner and c) the backward planner. The learner is based on the Dyna-PI (Sutton 1990) and uses a neural implementation of AHC. The neural network is used as a function approximator in order to anticipate the state space explosion. The forward planner comprises a number of experts that correspond to the available actions, and an action-planning controller. The experts take as input a representation of the current state and output a prediction of the next state. The action-planning controller is a hand-designed algorithm that is used to decide if the system will plan or act. Additionally, the forward planner is responsible for producing chains of predictions where a chain is defined as a sequence of predicted actions and states.

The experiments that contacted showed that bidirectional planning is superior to the forward planning. This was a presumable result as the bidirectional planner has two main advantages. The first is that the updates of the value function is done near the goal state and leads the system to find the way to the goal faster. The second advantage is that the same values are used to update the values of other states. This feature can be found in (Lin 1992).

3.2.2 Extraction of Planning Knowledge from Reinforcement Learners

In (Sun & Sessions, 1998), (Sun 2000) and (Sun et al., 2001) presented a process for extracting plans after performing a reinforcement learning algorithm to acquire a policy. The method is concerned with the ability to plan in an uncertain environment where usually knowledge about the domain is required. Sometimes is difficult to acquire this knowledge, it may impractical or costly and thus an alternative way is necessary to overcome this problem. The authors proposed a two-layered approach where the first layer performs plain Q-learning and then the corresponding Q values along with the policy are used to extract a plan. The plan extractor starts form an initial state and computes for each action the transition probabilities to next states. Then for each action the probabilities of reaching the goal state by performing the specific action are computed and the action with the highest probability is been chosen. The same procedure is repeated for successor states until a predefined number of steps are reached. Roughly speaking, the method estimates the probabilities of the paths which lead to the goal state and produces a plan by selecting a path greedily.

From the view of reinforcement learning, the method is quite flexible because it allows the use of any RL-algorithm and thus one can apply the algorithm that is appropriate for the problem in hand. Additionally, the cost of sensing the environment continuously is reduced considerably as planning needs little sensing from the environment. In (Sun 2000) the aforementioned idea was extended with the insertion of an extra component for extracting rules when applying reinforcement learning in order to improve the efficiency of the learning algorithm.

3.2.3 A Reinforcement Learning Approach to Production Planning

Cao et al. (2003) designed a system that solves a production planning problem. Production planning is concerned with the effective utilization of limited resources and the management of material flow through these resources, so as to satisfy customer demands and create profits. For a medium sized production planning problem is required to set thousands of control parameters which leads to a computational overhead. In order to make the problem tractable the authors modeled it as an MDP and used a Reinforcement Learning based approach to solve it. In details, RL, and more specifically Q-learning, is applied to learn a policy that adjusts the parameters of the system. To improve the efficiency of learning, two phases of learning used. In the first phase, the goal is to find the values of the parameters that are likely to lead to a near-optimal solution. In the second phase, these values are used to produce a plan.

4. DISCUSSION

This chapter deals with two research areas of Artificial Intelligence that aim at guiding an agent in order to fulfill his goals. These areas, namely Automated Planning and Reinforcement Learning attack the same problem from different perspectives, but at the bottom line share similar techniques and methodologies. The goal of both areas is to provide the agent with a set of guidelines for choosing the most appropriate action in each intermediate state between its initial position and the goal state.

Planning has been traditionally seen as a hierarchical paradigm for agent autonomy. In hierarchical models the agent senses the environment, constructs a model of it, searches for a plan achieving the problem goals and then starts acting in the real world. In order for the produced plan to be useful, two conditions must be met: a) the model of the world must be well defined and informative and b) the world must not deviate from the maintained model at any time point. In other words, the agent must either assume that the world is static and it can only change by its own actions, or it should periodically sense the world and use a sophisticated algorithm for incorporating the perceived changes in the model and updating the plan accordingly.

On the other hand, Reinforcement Learning is closer to the reactive paradigm of agent autonomy since it is based on continuous interactions with the real world. The model is not purely reactive since the exploration process can be guided by the acquired experience, but it does not involve complex reasoning on a model. RL is very efficient for dynamic worlds that may change in an unpredictable way and it eliminates the need for storing an informative and yet compact model of the world (frame problem). However, in complex problems the reactions with the environment are expensive and the search space may become intractable for exploration.

The systems surveyed in the chapter combine the two approaches in a manner similar to the hybrid paradigm of autonomy in order to provide faster and better guidance to the agent. The key idea is to equip the agent with two modules, a planner and a learner that cooperate either in a sequential or in a concurrent way. From this point view, there are three different approaches in combining planning and reinforcement learning:

a) First Plan then Learn

Planning is used to construct macro-operators, fixed sequences of actions, that can facilitate the value propagation and therefore speed-up the learning process. These macro-operators can be used in various forms: the simplest way is to pre-compile compound actions from simple ones. This can be either done manually or automatically using a planning process. This sort of hierarchical solving may rely on region based decomposition of MDPs (*Hauskrecht et al 1998, Dietterich 2000, Szita and Takacs 2003*) or on the idea of Teleo Reactive Planning (Nilsson 1994) that solves the problem by producing abstract plans in the form of action trees (Ryan and Pendrith 1998, Ryan 2002, *Grounds and Kudenko 2005*).

b) First Learn then Plan

Planning is combined with Reinforcement Learning in a two-step algorithm. In the first step a Reinforcement Learning algorithm is used in order to estimate the value function for each problem state. In the second phase, a planning algorithm is deployed in order to produce a plan according to the traditional formulation of AI planning. The plan that is produced is a complete control policy consisting of an explicit sequence of actions that requires minimal or zero environmental feedback during execution. The advantage of this approach is to improve the usability of the results. RL produces closed-loop policies that rely on moment-to-moment sensing, while the sequences of actions are very useful in situations where continuous sensing is difficult, unreliable or undesirable.

A representative example of this approach is the work by Sun, Sessions and Peterson (Sun and Sessions 1998, Sun 2000, Sun, Peterson and Sessions 2001, Sun and Sessions 2000). The algorithm they use is a simple heuristic search starting from the initial state and moving forward. At each step the planning algorithm examines concurrently a constant number of promising states and thus performs a beam search on the space of states.

c) Interchange Learning and Plannning

The last way to combine Planning with Reinforcement Learning is to maintain an internal model of the world and update the value function estimates by simulating transitions on the world model. This works concurrently with an actual reinforcement learning algorithm (e.g. Q - learning) that updates the same value functions by real interactions with the agents world. The values found by the two models are accumulated in the global policy and value function. This helps the value function to converge faster since the simulation of the interactions (performed by the Planning module) is done much faster than the actual interactions. Examples of this category include the DYNA family (Sutton 1990, Sutton 1991, Szita, Takacs & Lorincz 2003) and the Prioritized Sweeping (Moore and Atkenson 1993, Andre et al 1998, Dearden 2001).

REFERENCES

Andre D., Friedman N., Parr R. (1998) Generalized Prioritized Sweeping. Advances in Neural Information Processing Systems, MIT Press.

Baldassarre G. (2003). Forward and Bidirectional Planning on Reinforcement Learning and Neural Networks in a Simulated Robot. Adaptive Behaviour in Anticipatory Learning Systems, Butz M., Sigaud O., Gerard P. (Eds), Springer-Verlag, ch. 11, 179-200

Barto A. G., Sutton R. S., Anderson C. W. (1983) Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13(5), 834-846.

Bonet B. & Geffner. H. (2006) Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and its application to MDPs. Proc. 16th Int. Conf. on Automated Planning and Scheduling (ICAPS-06).

Boutilier, C., Reiter, R. & Price, B. (2001), Symbolic Dynamic Programming for First-Order MDPs. *Proceedings of the 17th Joint Conference on Artificial Intelligence*. pp. 690-700.

Cao H., Xi H., Smith S., F. (2003). A Reinforcement Learning Approach to Production Planning in the Fabrication Fulfillment Manufacturing Process. Winter Simulation Conference, 2, 1417-1423.

Cesta, A., Fratini, S. & Oddi, A. (2004), Planning with Concurrency, Time and Resources, A CSP-Based Approach. *Intelligent Techniques for Planning*, Vlahavas and Vrakas (eds), Idea Group Inc. pp. 259-295.

Bonet, B. and Geffner, H. (2001), Planning as Heuristic Search. *Artificial Intelligence, Special issue on Heuristic Search*. Vol 129 (1-2), pp. 5-33.

Boutilier, C., Dearden, R., Goldszmidt, M. (1995). Exploiting structure in policy construction. Fourteenth International Joint Conference on Artificial Intelligence.

Blum, A. & Furst, M. (1995), Fast planning through planning graph analysis, In *Proceedings of the 14th International Conference on Artificial Intelligence*, Berlin, Germany, pp.636-642.

Dean T., Kanazawa K. (1990). A model for reasoning about persistence and causation. Computational Intelligence, 5 (3), 142-150.

Dearden R. (2001) Structured Prioritized Sweeping. International Conference on Machine Learning, 82-89.

Fikes R. E., Nilsson N. J. (1974). STRIPS: A new approach to theorem proving in problem solving. Artificial Intelligence, 2, 189-208.

Ghallab M., Nau D. & Traverso P. (2004). Automated Planning: Theory and Practice. Morgan Kaufmann Publishers.

Giunchiglia, F. & Traverso, P. (1999), Planning as model checking. In *Proceedings of the 5th European Conference on Planning*, Durham, UK, pp. 1-20.

Grounds M., Kudenko D. (2005). Combining Reinforcement Learning with Symbolic Planning. Fifth European Workshop on Adaptive Agents and Multi-Agent Systems, Paris.

Hoffmann, J., & Nebel, B. (2001), The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research,* Vol. 14, pp. 253-302.

Kaelbling, L., Littman, M., Moore, A. (1996). Reinforcement Learning: A survey. Journal of Artificial Intelligence Research, 4, 237-285.

Kautz, H. & Selman, B. (1992), Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence,* Vienna, Austria, pp. 359-363.

Kautz, H. & Selman, B. (1998), BLACKBOX: A new Approach to the application of theorem proving to problem solving, In *Proceedings of the AIPS-98 Workshop on Planning as Combinatorial Search*, Pittsburgh, Pennsylvania, pp. 58-60.

Koehler, J., Nebel, B., Hoffmann, J. & Dimopoulos, Y. (1997), Extending planning graphs to an ADL subset. In *Proceedings of the 4th European Conference on Planning*, Toulouse, France, pp. 273-285.

Kwee I., Marcus H., Schmidhuber J. (2001). Gradient-based Reinforcement Planning in Policy-Search Methods. Fifth European Workshop on Reinforcement Learning. 27, 27-29.

Lin, Long-Ji. (1992). Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. Machine Learning, 8, 293-321.

Long, D. & Fox, M. (1998), Efficient Implementation of the Plan Graph in STAN, *Journal of Artificial Intelligence Research*, Vol. 10, pp. 87-115.

McDermott, D. (1996), A Heuristic Estimator for Means-End Analysis in Planning. In Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems, Eddinburgh, UK, pp. 142-149.

Moore A. W., Atkenson C. G. (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. Machine Learning, 13, 103-130.

Newell, A. & Simon, H.A. (1963), GPS, a Program That Simulates Human Thought. *Computers and Thought*, eds, Feigenbaum and Feldman, McGraw-Hill, N.Y, pp. 279-293.

Nguyen, X., Kambhampati, S. & Nigenda, R. (2000), AltAlt: Combining the advantages of Graphplan and Heuristics State Search. Proceedings of the 2000 International Conference on Knowledge-based Computer Systems, Bombay, India.

Nilsson, N. J. (1994). Teleo-reactive programs for agent control. Journal of Artificial Intelligence Research, 1, 139-158.

Peng, J., Ronald J. W. (1993). Efficient Learning and Planning Within the Dyna Framework. Adaptive Behavior, 1 (4), 437-454, MIT Press.

Ryan M. R. K. (2002). Using Abstract Models of Behaviours to Automatically Generate Reinforcement Learning Hierarchies. 19th International Conference on Machine Learning, 522-529.

Ryan M. R. K, Pendrith M. D. (1998). RL-TOPs: An Architecture for Modularity and Re-Use in Reinforcement Learning. 15th International Conference of Machine Learning, 481-487.

Sacerdoti, E. (1974), Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, Vol 5, pp. 115-135.

Sun R. (2000). Beyond Simple Rule Extraction: The Extraction of Planning Knowledge from Reinforcement Learners. International Joint Conference on Neural Networks, Como, Italy. IEEE Press.

Sun R., Sessions C. (1998). Extracting Plans from Reinforcement Learners. International Symposium on Intelligent Data Engineering and Learning. Springer-Verlag.

Sun R., Sessions C. (2000). Learning Plans without a priori knowledge. Adaptive Behavior, 18 (3), 225-253.

Sun R., Peterson T., Sessions C. (2001). Beyond Simple Rule Extraction: Acquiring Planning Knowledge from Neural Networks.

Sutton, R. S. (1988) Learning to predict by the methods of temporal differences. Machine Learning, 3, 9-44.

Sutton, R. S. (1990) Integrated architectures for learning, planning and reacting. Proceedings of the Seventh International Conference on Machine Learning, 216-224

Sutton, R.S. (1991) Planning by incremental dynamic programming, Proceedings of the Eighth International Workshop on Machine Learning, 353-357

Sutton, R.S., Barto, A.G. (1999) Reinforcement Learning, An Introduction. MIT Press

Tsamardinos, I., Vidal, T. & Pollack, M. (2003), CTP: A New Constraint-Based Formalism for Conditional, Temporal Planning. *Consraints 8 (4), Special Issue on Planning,* pp. 365-388.

Vrakas, D., Tsoumakas, G., Bassilliades, N., Vlahavas, I. (2005) Machine Learning for Adaptive Planning. Intelligent Techniques for Planning, Vlahavas, I. and Vrakas, D. (Eds), IDEA GROUP PUBLISHING, Ch. 3, pp. 90-120

Watkins, C. J. J. H. (1989) Learning with Delayed Rewards. PhD thesis, Cambridge University Psychology Department

Yang, Q. & Tenenberg, J. (1990), AbTWEAK: Abstracting a nonlinear least commitment planner. In *Proceedings of the 8th National Conference on Artificial Intelligence*, Boston, MA, pp. 204-209.

Zhao G., Tatsumi S., Sun R. (1999) RTP-Q: A Reinforcement Learning System with Time Constraints Exploration Planning for Accelerating the Learning Rate. IEICE TRANSACTIONS on Fundamentals of Eloctronics, Communications and Computer Sciences, Vol.E82-A, pp. 2266-2273.

Zimmerman, T., Kambhambati, S. (2003) Learning-assisted automated planning: looking back, taking stock, going forward. AI Magazine. 24, 2, 73-96