PRUDENT: A Sequential-Decision-Making Framework for Solving Industrial Planning Problems

Wei Zhang

Boeing Phantom Works
P.O. Box 3707, MS 7L-66
Seattle, WA 98124-2207
wei.zhang@boeing.com

Abstract

Planning and control are critical problems in industry. In this paper, we propose a planning framework called PRUDENT to address many common issues and challenges we are facing in industrial applications, including incompletely known world models, uncertainty, and very large problem spaces. This framework considers planning as sequential decision-making and applies integrated planning and learning to develop policies as reactive plans in an MDP-like progressive problem space. Deliberative planning methods are also proposed under this framework. This paper describes the concepts, approach, and methods of the framework.

Introduction

Planning and control are critical problems in industry. At Boeing, we are facing a wide range of problems where effective planning and control are crucial and the key to business success. In manufacturing, we are dealing with highly challenging problems that require integration of the functions from plan generation to execution from low-level, largely automated factory control to high-level enterprise resource planning (ERP) and supply-chain management (SCM). In the autonomous-vehicles business sector, we face challenges in solving a variety of planning and control problems in designing unmanned vehicles for us in air, space, ground, and underwater. In enterprise computing network protection and security, our business must deal with the challenges of building effective intrusion-detection and system-monitoring *policies*—like *universal plans* (Schoppers 1987)—that can ensure the security of a computing environment as well as accurate, timely response to unpredictable events and novel patterns.

While *operator sequencing* is an important family of techniques that can be applied for building plans in these task domains, it does not necessarily address all of the important technical challenges. In a completely deterministic world, it is possible to build a plan perfectly before execution, thus when the plan is executed—following the pre-planned or scheduled sequence of actions—the desired outcome will result. In the real world, however, incompletely foreseen events are often considered normal, thus a useful planning system must be able to know what to do when things go unexpected and for many circumstances must consider such uncertainty as a *regular structure* as opposed to the exception.

This nature is shared by all the problems listed above. To provide more competitive solutions, we take a broader view of planning where the tasks of a planner go inside all the stages of problem solving, including initial planning and possibly many iterations of replanning (interleaved with plan execution) to build and continuously improve plans and problem-solving policies. With this view, we turn our attention to the contents of planning, or plans, as opposed to the activity itself (a one-step task-arrangement activity), thus, the focus of planning can be perfectly described as *sequential decision making*—the process of determining sequences of actions for achieving goals. We refer to this view of planning as *process-based planning* so as to emphasize continuous policy (plan) improvement throughout a whole problem-solving process.

This paper presents a framework for dealing with real-world planning problems from this point of view. The framework is called PRUDENT, short for **P**lanning in **R**egular **U**ncertain **D**ecision-making **EN**vionment**T**, designed for addressing problems with regular uncertain structures across their whole problem space. A major contribution of the PRUDENT framework, from the technical point of view, is the introduction of sequential-decision-making techniques—specifically, *partial-policy reinforcement learning* techniques—to perform both *reactive* planning and *deliberative* planning in a process parallel to plan execution. From the practical standpoint, with integrated planning and learning, PRUDENT provides a promising tool for solving the problems described above. While reactive plans—plans reacting to a sensed environment—are the primary means to act in non-deterministic environments, adding deliberative plans may improve problem-solving capability significantly, particularly when facing problems requiring *timely* response to unpredictable events. A purely reactive plan lacking carefully pre-planned sequences of actions is slow and often fail to proceed when problems occur during sensing and data processing.

The paper is organized as follows. The following section first provides some necessary background for the PRUDENT

framework and then describes basic PRUDENT concepts. The main body of the paper describes an approach proposed using partial-policy reinforcement learning for developing reactive plans, world models, and deliberative plans. The paper concludes with a brief summary.

Planning as Sequential Decision Making and PRUDENT

Background

Markov Decision Processes (MDPs), originated in the study of stochastic control (Bellman 1957), is a widely applied, basic model for describing sequential decision making under uncertainty. In general, an MDP can be considered as an extension of the deterministic state-space search model for general problem solving. This extension allows modelling of non-deterministic state transitions, which are described as stochastic processes with *static* probabilistic state transition functions. The model comprises five components: (1) a finite state space $S = \{s_i | i = 1, 2, ..., n\}$, (2) an action space that defines actions that may be taken over a state space: $A = \{A(s) | \forall s \in S\}$ where $A(s)$ is a finite set of actions defined on state s, (3) a probabilistic state transition function $P(s_i, a, s_j)$ describes the probability of making a state transition from any one arbitrary state s_i to a state s_j (which maybe the same) when an action a defined on $A(s_i)$ is taken, (4) a reward function (or cost function) $R(s_i, a, s_j)$ over the problem space that specifies an instant reward that the agent will receive after an action is performed (under a corresponding state transition) , and (5) a discrete time space $T = \{0, 1, 2, ...\}$. Note the form of state-transition functions above says that the possible next states depend and only depend on the current state, independent of the previous ones. This characteristic is called the *Markov property*.

The task for an *agent* in an MDP environment is to determine, for a given future *time horizon* $H \in T$ (where H may be finite or infinite), a *policy* to apply over time that results in the maximal expected total future reward. This policy is referred to as an *optimal policy*. Specifically, a policy specifies an action to be taken in each state. At any state s, taking the action provided by an optimal policy, specified with respect to its time horizon H, guarantees maximizing the expected total future reward in the time frame.

While MDPs provide a powerful way to allow modeling state-space search under uncertainty, they also possess mathematical beauties to allow structured, efficient policy computation. With limited space, we summarize these algorithmic aspects as follows.

- Value function: A value function V of a policy π defines the *value*—the expected total future reward with respect to a time horizon H—of a state $s \in S$ using this policy over all states: $V_H^\pi(s) = E[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1})]$. V_H^π can computed recursively from V_{H-1}^π: $V_H^\pi(s) = \sum_{s' \in S | a = \pi(s)} P(s, a, s')[R(s, a, s') + \gamma V_{H-1}^\pi(s')]$. Here γ, $0 \leq \gamma \leq 1$, is the discounting factor controlling the influence of rewards in the past with a degree of *exponential decay*.
- Value iteration: The value iteration algorithm, or dynamic programming, for computing an optimal policy π^* is developed using *Bellman update* of *optimal value function* V^* (Bellman 1957): $V_H^*(s) := \max_{a \in A(s)} \left(\sum_{s' \in S} P(s, a, s')[R(s, a, s') + \gamma V_{H-1}^*(s')] \right)$. Once the optimal value function (with respect to a time horizon) is computed, an optimal policy can be obtained by executing a *one-step greedy lookahead* search using the optimal value function. This means knowing the V^* is equivalent to knowing a π^* (Note V^* is unique but it may correspond to multiple π^*s).
- Infinite time horizon with discounted rewards: Under the infinite time horizon, $0 \leq \gamma < 1$ should be applied. The optimal value function for $H \to \infty$ converges by value iteration under various conditions. While there are many interesting theoretical convergence results, our interest lies in real-world problems where limited time space is concerned.
- Policy iteration: When H is large, it may be more efficient to use the *policy iteration* algorithm. Policy iteration starts with an arbitrary policy π and then repeats the following policy evaluation-improvement steps: (1) evaluation: compute V^π, and (2) improvement: obtain greedy policy $\pi(s) := \arg\max_{a \in A(s)} \sum_{s' \in S} P(s, a, s')[R(s, a, s') + \gamma V_{H-1}^\pi(s')]$.

In the last decade, the MDP framework has been heavily revisited and studied in AI and machine learning communities, leading to the advances in reinforcement learning (Barto *et al.* 1995, Kaelbling *et al.* 1996, Sutton and Barto 1998) and decision-theoretic planning (Dean *et al.* 1995, Boutilier and Puterman 1995). The PRUDENT framework is developed based on these advances.

PRUDENT Concepts

PRUDENT is designed to address real-world problems that share the following common properties and challenges.

- **Incompletely known world model:** PRUDENT considers real problems where the world model is not completely known but underlying structures of the model exist and these structures may be explored and learned.
- **Uncertainty:** PRUDENT deals with problems where uncertainty is considered normal, possibly appearing throughout a problem space. This makes an MDP-like state-space model a favorable choice. In a quite *static* environment, knowledge of environment can be gained relatively easily by executing a process. This knowledge normally results in a reduced level of uncertainty for a learned model by eliminating unlikely transitions and making other transitions more certain. In a rather *dynamic* environment, however, new problems may occur during execution. This could introduce additional uncertainty into a model.
- **Non-Markov problems:** Problems are not Markovian under a natural view. For example, in a manufacturing process we collect sensor data every second. In the natural representation that uses the original state configurations (based on sensing and other conditions) and a regular time scale (by second), we find it clear that dependencies exist between future states and historical conditions.

The problems we face under a natural view normally are not Markovian.

- **Very large problem space:** Problems are complicated and require use of massive states to describe all the details. Such a large state space makes it impossible to build a complete universal plan. Standard dynamic programming and policy iteration for computing policies are not feasible.
- **Progressive state space:** A *progressive* state space is not an *ergodic* space where any state in the space can be reached from any other with finite steps. States are largely partially ordered. Making moves in a progressive space without a purposeful plan (say following a random walk) is likely to lead an agent from one end of a space to the other (the finish) end. In general, a progressive state space allows inclusion of a relatively small number of loops for modeling often occurred UNDOs and REDOs of a task or a sequence of tasks.

Accordingly, the PRUDENT design is baded on the following key concepts.

- **Planning:** The PRUDENT architecture is built on the MDP-like state-space structure. This makes PRUDENT a reactive planner. A partial-policy reinforcement learning approach is developed for this architecture to incorporate deliberative planning into this reactive-planning based framework. This paper argues that such a design is a natural choice for addressing the type of the problems discussed above.
- **Sensing:** Sensing is a basic requirement for reactive planning. PRUDENT utilizes sensing for three purposes: getting environment state information for a reactive plan, providing possibly useful information for a deliberative plan, and learning to better describe world models.
- **Learning:** The data received from sensing enable learning. Learning can be performed either during real-time or off-line. The task of learning is two-fold: (1) learning to better describe world models under various degrees of world dynamics, from quite static to more dynamic, and (2) coordinating with planners to learn to build and improve plans to act properly and more optimally in an environment.
- **Problem solving:** As a generalized planning system, PRUDENT supports *iterative problem solving*. We refer to a *goal-oriented* task from a start state to a goal (finish) state as a single problem-solving process. This process in PRUDENT supports interleaved planning (including replanning) and execution with incorporated learning functions. Such a process may continue for many iterations, possibly with different start points and different goals and change of conditions.
- **Problem formulation and transformation:** PRUDENT also provides functions for transforming original planning and control tasks into a state-space model, facilitating formulation of an MDP-like problem. A well-formulated problem can avoid many difficulties for planning and learning algorithms. It is important to notice that a non-Markov state space often may be transformed into a Markovian one by using a different state representation. Dependencies between future states and historical conditions may be removed by grouping temporally-dependent states and restructuring a state space using generalized states.

Approach and Methods

PRUDENT planning and learning follow the partial-policy reinforcement learning paradigm. This section first presents some important preparation issues, followed by the major elements of the PRUDENT approach: (1) learning partial policies as reactive planning, (2) learning world models, (3) real-time learning, and (4) planning sequences of actions.

Preparation Considerations

Applying PRUDENT planning first requires formulating an MDP-like problem space, describing states, actions, state-transition relations, and problem objectives in the form of reward function, time scale, and search horizon. We say the PRUDENT problem-space structure is MDP-like because it adopts the same fundamental elements as MDPs.

One major difference between PRUDENT and MDPs is that PRUDENT does not assume it has a complete knowledge of state transitions and its model is learned and updated during execution. Therefore, there is no need to carefully study and hand-engineer the state-transition probabilities at the beginning. An initial state-transition model can be quite rough.

Another difference is that a PRUDENT problem space does not require satisfying the Markov property. However, as an important principle, PRUDENT encourages use of more MDP-like structures whenever possible, maximally removing the dependency between future states and the history. A more MDP-like problem space can make planning and learning much easier.

For many problems it may be quite straightforward to come up with an MDP-like problem space for PRUDENT. But in other cases, various difficulties may be encountered, making it hard to completely remove historical dependencies for a transformed model. Typical problems causing these difficulties include historical dependencies across long-time periods, historical dependencies in variable time scale, incomplete sensing (the world may be partially observable), and incorrect sensing (errors and noise in sensing).

Learning Partial Policies

This function learns a *partial policy* as a reactive plan off-line under a fixed state-transition function.

When building a plan for a task involving in a very large problem space, one basic strategy is *divide-and-conquer*. Set a number of sub-goals in an order (a partial order) and accomplish these sub-goals in the defined order. PRUDENT learns partial policies using the same strategy. Table 1 shows the procedure.

The algorithm is a modified value iteration procedure, which learns a partial value function to obtain a partial policy—the policy greedy to this partial value function. It is

Table 1: Partial Policy Learning Algorithm

procedure PARTIALPOLICYLEARNER(S,A,P,R,G,s_0,σ)
inputs:
$S=\{s_i|i=1,2,...,n\}$ // *a finite state space*
$A=\{A(s)|\forall s\in S\}$ // *an action space*
$P=\{P(s,a,s')|\forall s\in S\&\forall a\in A(s)\}$ // *a state-transition function*
$R=\{R(s,a,s')|\forall s\in S\&\forall a\in A(s)\}$ // *a reward function*
$G=\{S_g,O_g\}$ // $S_g=\{g_i|i=1,2,...,m\}\subseteq S$ *is a set of goals* // *and* O_g *is a partial order of the goals*
s_0 // *a start state*
σ // *a set of scope rules*

INITVALUE() // *initialize value function* $V(s):=0,\forall s\in S$
repeat until (STOPPINGRULES()) // *repeat until stopping rules are satisfied*
for all $g\in S_g$ // *select g backward according to* O_g
BACKWARDUPDATE(g,S,A,P,R,σ) // *perform backward updates from g*
FORWARDUPDATE(s_0,S,A,P,R,σ) // *perform forward updates from* s_0
for all $g\in S_g$ // *select g forward according to* O_g
FORWARDUPDATE(g,S,A,P,R,σ) // *perform forward updates from g*
end repeat

end procedure

designed for goal-oriented problems with progressive problem spaces. For problems with this structure, rewards (or major rewards) are typically received when a goal or sub-goal is achieved. In real applications, Tesauro's backgammon programs applied zero rewards on all states until they reach the end of a game when the agent receives reward 1 if it wins or -1 if it loses (Tesauro 1992). In reinforcement learning applications for space shuttle processing for NASA, the program presented in (Zhang and Dietterich 1995) applies a measure of the quality of a schedule as a reward when a final feasible solution (a sub-goal) is obtained, while for other states, all operations (repairing steps for modifying and improving a current schedule) are assessed with a constant small penalty to encourage developing feasible solutions with the smallest number of repairs.

The procedure works as follows. Initially, the value for each state is set to 0. The main procedure updates values following a backward-forward update process iteratively until a stopping rule encoded in the function STOPPINGRULES is satisfied. Ideally, the procedure stops when the value function converges. Other rules may be included to allow a process to stop at other conditions, such as running out of time.

Each iteration first updates values backward. The backward update process starts with a final goal and then works successively backward on the rest of the goals according to the provided order of the goals O_g. When the BACKWARDUPDATE subroutine is called for selected goal state g, it starts with state $s:=g$ and updates its value,

$$V(s):=\max_{a\in A(s)}\left(\sum_{s'\in S}P(s,a,s')[R(s,a,s')+\gamma V(s')]\right).$$

Then it selects all state $s^{\#}$ that can directly lead to s with a single action ($P(s^{\#},a,s)>0$) and for all $s:=s^{\#}$ updates $v(s)$ using the same formula. This backward-update step proceeds until a scope rule in σ is satisfied. σ works as a set of heuristic rules. If it is possible to estimate the pariwise distance between all successive goals as well as the distance between s_0 and the first set of sub-goals, one possibly good σ rule is "set update steps to half of the largest distance". This rule expects that for any pair of successive goals, $V(s)$ can be computed by a backward process in the second half of the space and for the first half the values can be computed by a forward process.

After a backward update process is finished, in the same iteration, a forward update process starts. Forward update starts with s_0 and works forward through the goal states. Each FORWARDUPDATE call starts from the first state s (s_0 or a sub-goal g) and finds all possible next states s' of s and puts s' into a pool. Then for all s in the pool, it pops s and repeats the same step, putting all possible next states of s into the pool. This state-space growth process continues until a scope rule in σ is satisfied. All processed states are selected. After the state space is determined, FORWARDUPDATE sorts the states according to their current values, from the largest to the smallest, then updates values for all states using this order. This allows efficient use of updated values on the states that have been connected to goal states, because only states connected to certain goals can receive large values.

Learning World Models

This function learns state-transition functions in the form of world models. Learning utilizes existing incomplete models to try to improve them to better describe the world.

Learning state-transition functions is based on observations of state transitions made during system execution. PRUDENT employs the following three sets of learning parameters for learning and improving world models.

- Degree of environment dynamics. In a quite static environment, historical observations over a long period of time can be employed. In a quite dynamic environment, however, only data collected in a short history is used. This set of parameters determines the time period when data is selected for learning.
- Degree of observation reliability. This addresses real problems with incomplete sensing and incorrect sensing. Observations are carefully reviewed and selected. This set of parameters controls selection of observations individually.
- Conditions of variations. Variations of state transitions and their conditions are carefully studied. This attempts to find conditions for non-deterministic state transitions. If possible, states may be redefined by adding more conditions to existing specified states and splitting them. This may effectively remove many uncertain state transitions. This set of parameters determines if states need to be restructured.

Once correct, relevant data are selected, updating state transition probabilities is straightforward. PRUDENT applies

the standard maximum-likelihood method:

$$p(x,a,y) = \frac{n^a_{xy}}{n^a_x},$$

where n^a_{xy} is the number of cases that the environment switches to state y after action a is taken at state x and n^a_x is the number of cases in the selected observations where action a is taken at state x.

Restructuring states is a difficult task. Presenting techniques for accomplishing this task exceeds the scope of this paper.

Real-Time Learning

This function is developed for applications with fairly poor understanding of environments or quite dynamic environments. In such environments, since the current policy and world model are not reliable and often fail, adjusting them in real time by making use of current experience immediately is considered as a wise choice.

PRUDENT applies the *real-time dynamic programming* paradigm, or RTDP, developed by Barto et al (Barto *et al*. 1995). This employs trajectory-based reinforcement learning to learn partial policies. For learning world models in real time, PRUDENT applies the maximum-likelihood method described above as well as the *adaptive real-time dynamic programming* method developed by Barto et al as well.

Planning Sequences of Actions

This function attempts to develop deliberative plans based on the state-space based reactive planning paradigm. It is developed for applications with quite static environments where there are various deterministic sub-problems or substructures or knowledge can be learned to allow removal of various uncertain structures in a world model.

There are basically two conditions preventing making a deliberative plan from the MDPs based reactive planning framework: uncertainty and the needs for sensing. These two conditions are related. When state transitions are not deterministic, sensing becomes necessary in execution because of the need for determining states. And this is true vice versa.

While deliberative planning for an MDP-like environment may not be applicable in general, special problems in such an environment often exist that make building such a plan important. Here are three families of such problems.

- Planning for worst possibilities. For example, playing chess is a non-deterministic process. For quick response, it is important for an agent to have a deliberative plan to play against opponent's best moves. Planning for the worst possibility with a single worst case is a deterministic problem. In this case, the sequence of actions can be pre-determined without sensing.
- Planning for situations where the same sequence of actions is often applied. This involves part of a space where state transitions are quite deterministic. Making a deliberative plan can help fast execution by possibly avoiding most expensive step-by-step sensing and data processing activities.
- Planning for situations where sensing often fails. In this case, a deliberative plan can provide a backup plan that doesn't depend on sensing, replacing reactive plans.

PRUDENT considers developing deliberative plans for these three families of problems. Additional steps are added to the partial-policy reinforcement learning methods presented above to allow learning sequences of actions to come up with a deliberative plan. PRUDENT basically provides two methods. The first method returns sequences of actions for dealing with worst possibilities. The second method returns all sequences of actions corresponding to all possible trajectories for a quite deterministic sub-space. Each returned sequence of actions employs the greedy policy to the learned values of the states along a trajectory. If too many trajectories are generated, a useful parameter for controlling the number is selecting only the k most-likely trajectories (e.g., for planning a chess game, consider the trajectories that your opponent is most likely to adopt).

Selecting the k most-likely trajectories for PRUDENT is straightforward, because state transition probabilities are available. PRUDENT employs a lookahead parameter κ (usually $\kappa \leq 5$) to deal with possible combinatorial explosions. In the lookahead region, it performs a κ-step exhaustive search and computes the joint probability of state transitions for each of the returned trajectories. After κ-step trajectories are generated, the method extends each of them by performing 1-step lookahead greedy search, returning a single "most-likely" trajectory (in terms of the greedy heuristic) for each trajectory length, $\kappa+1, \kappa+2, ..., N$ (N is a given limit for the length). Finally, the k most-likely trajectories are returned from the generated pool. Since longer trajectories result in smaller joint probabilities, we compare trajectories by grouping them by the length. When comparing trajectories of different lengths, we use a simple normalization method that multiplies the likelihood value for an n-step trajectory by 2^n.

Summary

In summary, this paper proposed the PRUDENT planning framework to address many common issues and challenges we are facing in industrial applications, including incompletely known world models, uncertainty, and very large problem spaces. This framework considers planning as sequential decision-making and applies integrated planning and learning to develop policies as reactive plans in an MDP-like progressive problem space. Deliberative planning methods are also proposed under this framework.

Application of this framework to real-world problems is in practice. Our practices are conducted mainly for the problems present in three domains: manufacturing, autonomous systems, and security and network management. With increased capability of collecting massive data from domain processes, opportunities for applying integrated planning and learning are increasingly large.

This paper is a work-in-progress. We expect to release part of our application results in public in a near future.

References

A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

C. Boutilier and M. L. Puterman. Process-oriented planning and average-reward optimality. In *IJCAI-95*, pages 1096–1103, 1995.

T. L. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 1-2(76):35–74, 1995.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of AI Research*, 4, 1996.

M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039–1046, 1987.

R. S. Sutton and A. G. Barto. *Introdution to Reinforcement Learning*. MIT Press, 1998.

G. J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, 1992.

W. Zhang and T. Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI-95*, pages 1114–1120, 1995.