Partially Observable Markov Decision Processes

- A full POMDP model is defined by the 6-tuple
  \[ \Xi = (S, A, Z, R, T, O) \]
  - \( S \) is the set of states (the same as MDP)
  - \( A \) is the set of actions (the same as MDP)
  - \( T \) is the state transition function (the same as MDP)
  - \( R \) is the immediate reward function
  - \( Z \) is the set of observations
  - \( O \) is the observation probabilities

- Solving a POMDP is to find the optimal policy for choosing actions.
  - Finite horizon POMDPs (with or without discount factor)
  - Infinite horizon POMDPs (with discount factor)
Solving MDPs vs. POMDPs

- Finding the optimal policy is to find the actions that maximize the value function.

- Optimal value function for a finite horizon MDP is:

  \[
  V^\pi_n(s) = \max_{a \in A} \left[ r(s, a) + \rho \sum_{s' \in S} \tau(s, a, s') V^\pi_{n-1}(s') \right]
  \]

  While the optimal value function for a finite horizon POMDP can be reformulated in the same format:

  \[
  V^\pi_n(b) = \max_{a \in A} \left[ \omega(b, a) + \rho \sum_{z \in Z} \sigma(b, a, z) V^\pi_{n-1}(b^a_z) \right]
  \]

- A POMDP can be reformulated as a continuous space MDP. Thus theoretical results and algorithmic ideas for solving MDPs can be borrowed to solve POMDPs.
Solving POMDPs

- Approaches to solving POMDPs
  - **Exact** algorithms: “finding all alpha-vectors for the whole belief space which is exact but intractable for large size problems.”
  - **Approximate** algorithms: “finding alpha-vectors of a subset of the belief space, which is fast and can deal with large size problems.”

- Frameworks for solving POMDPs
  - **Value Iteration (VI)**: the dynamic programming approach to find the optimal policy (*iteration in value space*)
  - **Policy Iteration (PI)**: based on an iteration over policies (*iteration in policy space*)
Value iteration structure

\[
\text{valueIteration}(\mathcal{X}, \rho, T)
\]

for each \( s \in \mathcal{S} \)

\[
V_n(s) := 0
\]

end for each \( s \)

for each \( n \in \{1, 2, \ldots, T\} \)

\[
V_n := \text{oneStepDP}(\mathcal{X}, \rho, V_{n-1})
\]

end for each \( n \)

return \( V_T(\cdot) \)

end valueIteration

\[
V_n^{*a}(s) = r(s,a) + \rho \sum_{s' \in \mathcal{S}} \tau(s,a,s')V_{n-1}^{*}(s')
\]
A policy iteration structure

\begin{verbatim}
policyIteration(π, ρ)
  d' := any decision rule
  do
    d := d'
    V := evalPolicy(π, ρ, d)
    d' := improvePolicy(π, ρ, V, d)
  until d = d'
  return d
end policyIteration
\end{verbatim}
Exact Algorithms Using VI (1)

- Sondik’s (1971)
  - First detailed algorithm for finding exact solutions to finite horizon problems and approximate solutions to infinite horizon problems
  - Finite horizon algorithm is slow for all but the smallest problems
  - One-Pass algorithm, Two-Pass algorithm

- Monhan’s (1982)
  - Succeeded Sondik’s algorithm
  - More efficient than Sondik’s algorithm for most problems

- Eagle’s (1984)
  - An optimization of Monhan’s algorithm
  - Still much computation for small problems
Exact Algorithms Using VI (2)

- Cheng’s (1988)
  - Relaxed Region algorithm
    - Very similar to Sondik’s
    - More efficient than Sondik’s one-pass algorithm
  - Linear Support algorithm
    - Approximates the value function until all supports in the value function are found
    - More efficient than Sondik’s one-pass algorithm

- Witness Algorithm (1994)
  - Similar to Cheng’s Linear Support algorithm

- Incremental Pruning algorithm (1996)
  - Like Witness, but differs in constructing each possible vector
Exact Algorithms Using VI (3)

- Generalized Incremental Pruning (1997)
  - The basic structure of the incremental algorithm is the same; the main difference lies in the way the individual PRUNE operations are performed.

  … …

- It was shown by Anthony, for small problems, the Restricted Region Incremental Pruning algorithm is the fastest in most cases among the following four algorithms.
  - Restricted Region Incremental Pruning
  - Normal Incremental Pruning
  - Two-Pass
  - Witness
Available Tools for Solving POMDPs

- **C/C++ source codes**
  - **pomdp-solve** (v5.3, Oct 2005)
  - **MADP** - Multiagent decision process Toolbox (v0.1, Jan 2008) (Just a toolbox)
  - **ZMDP** Software for POMDP and MDP Planning (v1.1.5, Jun 2008)
  - **APPL** - Approximate POMDP Planning (v0.2, Jul 2008)

- **MATLAB source codes**
  - **Perseus** approximate POMDP solving software (v0.1, Sept 2004)
  - **Symbolic Perseus** (Mar 2008) (Needs JAVA)
Other Related Tools

- INRA's matlab MDP toolbox

- Markov Decision Process (MDP) Toolbox for Matlab

- Matlab parser for Tony's POMDP file format
  - http://staff.science.uva.nl/~mtjspaansoftware/pomdp/

- MATLAB parser for ZMDP policy files

- MATLAB parser for pomdp-solve

- RandomMDPs.lisp
  - http://www.cs.ualberta.ca/~sutton/RandomMDPs.html
Pomdp-solve

- Pomdp-solve software
  - Written by Anthony R. Cassandra
  - In C language
  - Latest version 5.3 (2005)
  - Size of source file: 2.29 MB
  - Compile on: Linux / Cygwin

- Implemented algorithms
  - Enumeration (Sondik '71, Monahan '82, White '91)
  - Two Pass (Sondik '71)
  - Linear Support (Cheng '88)
  - Witness (Littman '97, Cassandra '98)
  - Incremental Pruning (Zhang and Lui '96, Cassandra, Littman and Zhang '97)
  - Finite Grid (instance of PBVI) (Cassandra '04)


**Pomdp-solve Inputs**

- **.POMDP file format**
  
  - `discount: %f`
  - `values: [ reward, cost ]`
  - `states: [ %d, <list of states> ]`
  - `actions: [ %d, <list of actions> ]`
  - `observations: [ %d, <list of observations> ]`

  - `T: <action> : <start-state> : <end-state> %f`
  - `O: <action> : <end-state> : <observation> %f`
  - `R: <action> : <start-state> : <end-state> : <observation> %f`

- **Command line options**
Pomdp-solve Outputs

- Output value function format
  
  A
  V1  V2  V3  ...  VN
  A
  V1  V2  V3  ...  VN
  etc  ...

  - A is an action number
  - V1 through VN are real values representing the components of a particular vector. The vector represents the coefficients of a hyperplane representing one facet of the piecewise linear value function.

- Output Policy Graph Format
  
  N  A  Z1  Z2  Z3  ...
  etc  ...

  - N is a node ID
  - A is the action number defined for this node
  - Z1 Z2 Z3 ... are a list of node IDs, one for each observation. The n'th number in the list will be the index of the node that follows this one when the observation received is 'n'.
Policy Graph Examples

Finite horizon policy graph

Infinite horizon policy graph
MADP

- The software
  - Written by Frans Oliehoek and Matthijs Spaan
  - In C++
  - Latest version 0.1 (2008)
  - Size of source file: 2.68 MB
  - Compile on: Linux (needs Boost and libtool)

- MADP
  - A software toolbox for scientific research in decision-theoretic planning and learning in multiagent systems (MASs).
  - A collection of mathematical models for multiagent planning:
    - Multiagent Markov decision processes (MMDPs) [Boutilier, 1996]
    - Decentralized MDPs (Dec-MDPs)
    - Decentralized partially observable MDPs (Dec-POMDPs) [Bernstein et al., 2002]
    - Partially observable stochastic games (POSGs)
  - This toolbox provides classes modeling the basic data types and derived types for the above models.
MADP Inputs

- .dpomdp file format
  - agents: [%d, <list of agents>]
  - discount: %f
  - values: [ reward, cost ]
  - states: [ %d, <list of states> ]
  - actions:
    - [ %d, <list of actions> ]
    - [ %d, <list of actions> ]
    - ...
  - observations:
    - [ %d, <list of observations> ]
    - [ %d, <list of observations> ]
    - ...
  - T: <a1 a2...an> : <start-state> : <end-state> : %f
  - O: <a1 a2...an> : <end-state> : <o1 o2 ... om> : %f
  - R: <a1 a2...an> : <start-state> : <end-state> : <o1 o2 ... om> : %f
MADP Example

- A small example program on the Tiger problem

```c
#include "ProblemDecTiger.h"
#include "JESPExhaustivePlanner.h"
int main()
{
    ProblemDecTiger dectiger;
    JESPExhaustivePlanner jesp(3,&dectiger);
    jesp.Plan();
    cout << jesp.GetExpectedReward() << endl;
    cout << jesp.GetJointPolicy()->SoftPrint() << endl;
    return(0);
}
```
MADP Example Outputs

Output of the program

Value computed for DecTiger horizon 3: 5.19081
Policy computed:
JointPolicyPureVector index 120340 depth 999999
Policy for agent 0 (index 55):
Oempty, --> a00:Listen
Oempty, o00:HearLeft, --> a00:Listen
Oempty, o01:HearRight, --> a00:Listen
Oempty, o00:HearLeft, o00:HearLeft, --> a02:OpenRight
Oempty, o00:HearLeft, o01:HearRight, --> a00:Listen
Oempty, o01:HearRight, o00:HearLeft, --> a00:Listen
Oempty, o01:HearRight, o01:HearRight, --> a01:OpenLeft
Policy for agent 1 (index 55):
Oempty, --> a10:Listen
Oempty, o10:HearLeft, --> a10:Listen
Oempty, o11:HearRight, --> a10:Listen
Oempty, o10:HearLeft, o10:HearLeft, --> a12:OpenRight
Oempty, o10:HearLeft, o11:HearRight, --> a10:Listen
Oempty, o11:HearRight, o10:HearLeft, --> a10:Listen
Oempty, o11:HearRight, o11:HearRight, --> a11:OpenLeft
The software
- Written by Trey Smith
- In C++
- Latest version 1.1.5 (2008)
- Size of source file: 2.30 MB
- Compile on: Linux / Cygwin
- http://www.cs.cmu.edu/~trey/zmdp/

ZMDP
- Implements several heuristic search algorithms for POMDPs and MDPs:
  - Focused Real-Time Dynamic Programming (FRTDP) [2006]
  - Improved Heuristic Search Value Iteration algorithm (HSV12) [2005]
  - Real-Time Dynamic Programming (RTDP) [1994]
  - Labeled RTDP (LRTDP) [2003]
  - HDP [2003]
- Speeds up the usual PWLC value function representation for POMDPs

Input: .pomdp
- The same file format as pomdp-solve

Output: .policy
- Like one of the output files from pomdp-solve, but in different format
{  
  policyType => "MaxPlanesLowerBound",
  numPlanes => 2,
  planes => [
    {
      action => 1,
      numEntries => 3,
      entries => [
        0, 18.7429,
        1, 21.7431,
        2, 18.7427
      ]
    },
    {
      action => 3,
      numEntries => 1,
      entries => [
        1, 22.0287
      ]
    },
  ]
}
APPL

- The software
  - Written by National University of Singapore
  - In C++
  - Latest version 0.2 (2008)
  - Size of source file: 828 KB
  - Compile on: Linux / Cygwin / Windows

- APPL
  - Implements the point-based SARSOP algorithm (Successively Approximating the Reachable Space under Optimal Policies)

- Input: .pomdp
  - The same file format as pomdp-solve

- Output: .policy
  - The same file format as ZMDP
Perseus

- The software
  - Written by Matthijs Spaan
  - In MATLAB
  - Latest version 0.2 (2004)
  - Size of source file: 53 KB
  - http://staff.science.uva.nl/~mtjspaans/pomdp/

- Perseus
  - Implements the Perseus randomized point-based approximate value iteration algorithm

- Input: .pomdp
  - The same file format as pomdp-solve

- Output: a structure called “backupStats”
  - Value functions for each step

- Drawbacks:
  - Need to manually call MATLAB functions step by step (not automatic solving)
  - Need to manually modify MATLAB files for every POMDP problem
Symbolic Perseus

- The software
  - Written by Pascal Poupart
  - In MATLAB and JAVA
  - Latest version 03-24-2008
  - Size of source file: 384 KB
  - Needs JAVA SDK to compile the JAVA code

- Symbolic Perseus
  - Implements the Symbolic Perseus algorithm (a point-based value iteration algorithm that uses Algebraic Decision Diagrams (ADDs) as the underlying data structure to tackle large factored POMDPs)
  - Has been used to solve factored POMDPs with up to 50 million states

- Input:
  - A unique input file format that describes the POMDP model

- Output:
  - Value function: set of vectors
  - Policy: vector of actions for each vector
Choose POMDP Solving Tools

- For exact solutions or small problems
  - pomdp-solve

- For multiagent problems
  - MADP

- For approximate solutions
  - APPL
  - ZMDP

- For approximate solutions using MATLAB
  - Perseus
  - Symbolic Perseus
Discussion

- Limitations of POMDPs
  - It is assumed that the agent knows the complete POMDP model (e.g., transition probabilities, observation probabilities, rewards, etc.), while it is not realistic for many problems.
  - The POMDP model assumes finiteness of the states, actions and observations, while many problems are better modeled with continuous quantities for these.
  - The solution procedures for POMDPs assume that the model is static. It does not permit the model to change over time.
  - It requires time to compute belief state at run time. For certain applications, the resource to maintain and update belief states might not be available.
  - It is impossible to compute the optimal policy for anything but small problems.
Summary

“POMDPs provide a principled mathematical framework for planning and decision-making under uncertainty, but they are notoriously hard to solve.”

“Some point-based approximate algorithms are able to compute reasonably good policies for very large POMDPs with hundreds of thousands states.”

- Available software tools for solving POMDPs are introduced. The SARSOP algorithm (implemented in APPL) may be the fastest existing point-based algorithm for approximate solutions of some problems.
- It may need minutes or hours to solve large POMDPs approximately.
- Many parameters are needed to define a POMDP. In practice, sometimes it is not easy to set values to all of them.
- With the advances in POMDP solution algorithms, POMDPs are gradually becoming practical for non-trivial tasks.
Thank you!