Agents

- Agents are designed to learn patterns and behave intelligently.
- Common Traits:
  - Autonomous, and thus be proactive.
  - Adaptive, be reactive or learn from social settings.
  - Collaborative.
  - Mobile.
Agent Technologies (1)

- Generally not plug and play.
  - Agents should learn over time and assessment is an integral component of how well the agents are doing. Their performance should improve.
- Rule Based Reasoning Techniques. Rules are IF THEN statements.
  - IBM’s RAISE (Reusable Agent Intelligence Software Environment) of rule based reasoning. RAISE is the inference engine of IBM’s Agent Building Environment (ABE) developer’s toolkit. Problem with rule based systems is that they must be kept up to date manually.

Agent Technologies (2)

- Knowledge Based Reasoning Techniques.
  - Cyc being built by Doug Lenat (AI pioneer) with over 1 million common sense rules. Cyc can be used to build Cyc agents.
- Simple Statistical Analysis – Tools recognize the co-occurring patterns, and numbers frequently called.
Agent Technologies (3)

- Neural Networks – Neuron architecture
- Evolutionary Computing – example best policy.

Agent Standards

- Agent Communication Languages: ACL is being standardized by FIPA – Foundation for Intelligent Physical Agents.
- Open Profiling Standards or OPS.
- Mobile Agents being standardized by OMG (Object Management Group).
Agent Design Issues (1)

- Misunderstanding your intentions.
- Security. Agent friendly systems leave themselves open to security risks.
- Resource Consumption
- Mobility – raises specters with system administrators, as they perform the work where the data is located. Robots.txt file.
- Answer is erroneously assumed to be the best answer (example, bargainfinder.com).

Agent Design Issues (2)

- Trust:
  - Will the agent do what I want or misinterpret my instructions.
  - Will it distribute my credit card information to unauthorized people.
  - Will it act effectively? (i.e. find best bargain, search useful information...)
  - Can we trust someone else’s agent to be who they say they are?
Agent Design Issues (3)

- Complexity: How much intelligence does agent need to have?
- Examples of
  - likes and dislikes,
  - scheduling meeting preferences,
  - rules for trashing e-mail based on previous actions, etc.

Agent Development Process

- Identify a realistic opportunity
- Identify the content sources that are required
- Consider appropriate agent technologies.
- Ensure that the application would be well-behaved. (unlike agents in MS-Word).
- Make the application smart - do what you want, not what you ask for.
- Slowly build a relationship with the human boss. Start with generic rules, gradually build user’s trust.
Agent Application Examples (1)

- Filtering Tools
- Analysis Tools – compare documents and create logical clusters of information. Identify themes and detect anomalies.
- Competitive intelligence systems. Monitor news to discover by matching news articles to a generic profile of a competitor.
- Interface agents that will learn your preferences for styles, report formats, etc.
- Decision support systems that couple database and agent systems.

Agent Application Examples (2)

- Crisis Management Systems.
- E-Commerce Agents – Compare prices, and choose the best price. Negotiate transactions.
- Medical agents that monitor a patient data over time and decide if a situation has become critical.
- Training agents would guide the users through new processes.
Agent Application Examples (3)

- Data Mining agents using statistical correlation and rule based reasoning for cause and effect and track trends over time.
- Travel agents - track your travel preferences and book flights as directed. Watch fares and convey the best time to buy your tickets.
- Meeting agents - juggle schedules.

Agent Application Examples (4)

- Self Customizing Agents/Interfacing Agents
- Agents will manage security on the system. They will be able to detect intrusion and unauthorized uses.
Other Examples

- Softbots for Matchmaking
- Life Like Characters (LLCs) in Virtual Meeting Rooms
- Personal Web Assistants
- Process Planning
- Integration: Virtual Reality and Agents.

Humanoid Agents – Virtual Reality and Agent Integration
Module I
Knowledge Representation in Agents

Contents
- Agent Architecture and Rational Agents
- Reflexive Agent
- Belief Desire and Intention Architectures
- Agent Building Environments:
  - Java Agent Toolkit
  - IBM’s Aglets
  - Microsoft’s Agent Toolkit
Agent Architecture

```java
Environment e;
RuleSet r;
while (true) {
    state = senseEnvironment (e);
    a = chooseAction (state, r);
    e.applyAction (a);
}
```

Knowledge Representation

- Several techniques may be adopted:
  - Rule Bases
  - Logical Assertions
  - Semantic Networks
  - Frames
  - Neural Networks
  - Genetic / Evolutionary Algorithms
Rational Agents (1)

- Agent is anything that can be **perceiving** its environment, through **sensors** and **acting** upon its environment through **effectors**. (compare to human agent).
- A **rational agent** is one that does the right thing. What’s the right thing - basically anything that makes the agent most successful. So, the new issues are - how and when to evaluate the agent’s success.

Rational Agents (2)

- So - objective functions need to be developed. These are often hard.
  - Example - a vacuum cleaning agent. We can not measure how clean the floor is. We can only measure how much dirt was picked up. (And perhaps how little energy and low noise was generated).
  - What if the agent began to put dirt on the floor to maximize their success!
- Issue about when to measure success is equally important. The success should be factored over long periods, and its constancy is important.
Rational Agents (3)

- Distinguish omniscience from rational.
  - Omniscience is impossible to achieve in reality.
- Rationality deals with expected success given what has been perceived.
  - Example of an agent crossing the road—not perceiving an object falling from the sky.
  - Another agent, capable of making that perception, and having the ability to deal with it by repelling it with a steel cage (taking action), is not necessarily more rational.
- Cannot blame the agent for not taking into consideration what it cannot perceive, nor for not taking the action that it is incapable of taking (such as repelling the falling object).

Softbots

- Softbots- software agents is a term used to generally describe complex software agents that exist in rich, unlimited domains.
  - For example, a softbot would be designed to fly an airplane - responding to inputs in real-time, and taking actions selected from a large number of possibilities.
- Distinction is also needed to be made between “real” and “artificial” environment.
  - For example, for the bot crossing the road, the resolution of the camera dictates the artificial perception of the environment.
Agent’s Environment

- **Environment Models** need to be considered as well.
  - Accessible or Non-Accessible?
  - Deterministic or Non-Deterministic?
  - Episodic or Non-Episodic?
  - Static or Dynamic?
  - Discrete or Continuous?

Reflexive Agents

*Do not utilize precept history*

```java
if car-in-front-is-braking then initiate-braking

Environment e;
RuleSet r;
while (true) {
    state = senseEnvironment (e.getPrecepts());
    a = chooseAction (state, r);
    e.applyAction (a);
}
```
Agents With State Information

**maintain state information**

Environment e;
RuleSet r;
State s; // Description of the current world

while (true) {
    state = senseEnvironment (s, e.getPrecepts());
    a = chooseAction (state, r);
    s = updateState (state, a);
    e.applyAction (a);
}

Agents With Goals and Utilities:
Means Ends Analysis

Utility is a measure of how happy the agent would be in a given anticipated/possible state.

This helps the agent determine the most suitable action to take.
**Belief Desire Intention (BDI) Agent Architecture**

Intentions:
- Drive the means-end analysis
- Intentions constrain future deliberation
- Intentions persist
- Intentions influence beliefs

brf: ? (Bel) ? P ? ? (Bel): Belief Revision

Desires: ? (Bel) ? ? (Int) ? ? (Des) - Options

filter (deliberation):
? (Bel) ? ? (Des) ?? (Int) ? ? (Int) filter must satisfy:
? (Int) filter (B,D,I) ? I ? D

execute: ? (Int) ? A

**Example**

Let L be the set of sentences in first order logic,
and let D = (? (L) be the set of the sets of L formulae.
The internal state of an agent is then
? 1 ? 2 ... ? I ? D. Reasoning:
? ? ? ?
see: S ? P
next: D ? P ? D
action: D ? A

**action (? : D): A**
for each a ? A do if ? ? ? Do(a) then return a;
for each a ? A do if ? ? ? ? Do(a) then return a;
return null;

**Predicates:** In (x,y), Dirt (x,y), Facing (d)

**Old (? ) = {P(t1, t2, ...tn) |**
P ? {In, Dirt, Facing} and
P(t1, t2, ...tn) ? D

**New:** D ? P ? D

**Next** (? , p) = (? \ old(?))? new (? , p)

**Rules:**
In (x,y) ? Dirt (x,y) ? Do (suck)
In (0,0) ? Facing (north) ? ? Dirt (0,0)
? Do (forward) ...
In (0,2) ? Facing (north) ? ? Dirt (0,2)
? Do (turn)
Module Conclusions

- Agents are characterized by precepts, effectors and reasoning.
- Agents Interact with the Environment.
- Rational Agents take the best decision – defined by some objective functions (or utilities).
- Agents performance must be evaluated on a continuous basis.
- Agents may be based on reflexive actions, utilize state information, or exhibit desires and intentions.

Module II

Multi-Agent Systems and Society of Agents
Contents

- KQML
- Examples of Multi-agent Cooperation
- Truth Maintenance Systems
- Examples:
  - Java KQML – jKQML

Multi-Agent Systems: Society of Agents

- Social Dependence
  (SocialDependence x y a p) ?
  (Goal x p) ? ~ (CanDo x a) ? (CanDo y a) ?
  ((DoneBy y a)? Eventually p)

- Mutual Dependence (example, Cooperation)
  ? p (SocialDependence x y ay p) ?
  (SocialDependence y x ax p)

- Reciprocal Dependence: (example Social Exchange)
  ? pxpy (SocialDependence x y ay px) ?
  (SocialDependence y x ax py)
**Multi-Agent Systems**

- Cooperative Teams:
  - All agents adhere to a common goal
  - Each agent is required to do its share to achieve common goal.
  - Each agent adopts a request to do its share
- Agents must be developed to “speak a common language”
- KQML is one such language.

**Agent Languages**

- Agent Communication Languages - ACL
- KQML
- KQML language is divided into three layers:
  - The Content Layer
  - The Message Layer
  - The Communication Layer
**KQML: Content Layer**

- Holds the actual content of the message – in program’s own representation language.
- It does not matter what the content is. The KQML can accommodate any content – both ASCII and binary information is possible.
- KQML system simply needs to know where the message content ends.

**KQML: Communication Layer**

- Carries information about the
  - sender address
  - receiver addresses,
  - the message identifier etc.
**Message Layer**

- Encodes the message that one agent would transmit to another.
- Forms the core of KQML, and determines the kinds of interaction one KQML-speaking agent can have with another.
- The message level defines a speech act or performative which sender attaches to the content
  - Such as that it is an assertion, a query, a command, or any set of known performatives.
- Since the content is opaque in KQML, this level includes optional features which describe the content language, the ontology it assumes, and some type of description of the content.

**Examples – Sender**

```lisp
(ask-one
 :sender joe
 :content (PRICE IBM ?price)
 :receiver stock-server
 :reply-with ibm-stock
 :language LPROLOG
 :ontology NYSE-TICKS
 )

(tell
 :sender stock-server
 :content (PRICE IBM 25)
 :receiver joe
 :in-reply-to ibm-stock
 :language LPROLOG
 :ontology NYSE-TICKS
 )
```

- KQML performative is ask-one, and tell
- Communication Level: sender, receiver, and reply-with
- Message Level: language, ontology.
KQML also introduces a small number performatives which are used by agents to describe the meta data.
KQML also introduces a special class of agents called communication facilitators.
A facilitator agent provides various useful communication services,
- maintaining a registry of service names,
- forwarding messages to named services,
- routing messages based on content,
- match-making between information providers and clients,
- providing mediation and translation services, etc.

Example (1)

The Query - Agent Q1 asks the broker B1 to recruit all agents who can answer the question specified as the content.

(recruit-all
  :sender Q1
  :receiver B1
  :reply-with id1
  :ontology Transportation-Domain
  :language Predicate-Calculus
  :content (Weight (Automobile ?x)))
**Example (2)**

- The Broker forward the advertise message from the Information-Agent IA3 (for example) to the Query Agent Q1.

```
(forward
  :to Q1
  :from B1
  :sender IA3
  :receiver Q1
  :in-response-to id1
  :content
    (advertise
      :send IA3
      :receiver Q1
      :reply-with id2
      :content (ask-all (weight (Automobile ?x)))
    )
)
```

**Example (3)**

- The Query Agent Q1, subsequently asks the information agent for the data about the weights of the automobiles:

```
(ask-all
  :sender Q1
  :receiver IA3
  :in-response-to id2
  :reply-with id3
  :ontology Transportation-Domain
  :language Predicate-Calculus
  :content (Weight (Automobile ?x))
)
```

- The Information Agent IA3 would next respond message containing a tell performative.
**Example (4)**

**INFORMATION AGENT**

- send: tell
- receive: ask all
- send advertise

**QUERY AGENT**

- send: recruit all
- receive: advert
- receive: forward (to QA)

**BROKER**

- receive: forward (from QA)

**Contract Nets (1)**

- Client Agent asks for bids to perform a specific task.
- Server agents respond with the cost (example, turnaround time, solution completeness, etc.).
- One of the server agents gets the contract.
- The Client/Server Agent communication facilitated through KQML.
- May not get a response (i.e. no one bids). Aversion strategy:
  - Broadcast and contracting
  - Retry
  - Revision
  - Alternate decomposition
**Contract Nets (2)**

Manager Agent

\[
\text{(evaluate :sender manager :receiver agent1 :reply-with id1 :content (WorkDescription))}
\]

Worker Agent

\[
\text{(reply :sender agent1 :receiver manager :in-reply-to id1 :reply-with id1 :content (Can-Do Cost, Utility))}
\]

\[
\text{(tell :sender manager :receiver agent-k :in-reply-to id-k :content (Proceed with Work))}
\]

**Multi-Agent Truth Maintenance (1)**

Agent A (weather prediction agent) had previously told Agent B (using :tell performative), that it is going to rain tomorrow.

However, Agent A now does not believe that it will rain tomorrow.

It must therefore inform Agent B of the same (using the :retract performative).
Multi-Agent Truth Maintenance (2)

- Agent B may in turn retract this fact from other agents C, D whom it may have told.
- Upon the retraction of the “rain fact” Agent C may retract another derived fact from Agent B.
- Which may in turn lead to the retraction of facts from Agents X, Y.
- The systems needed for preserving overall rationality of facts are called TMS.

Example (TMS)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact1: P</td>
<td>fact1: P</td>
<td>fact1: R</td>
</tr>
<tr>
<td>status: (IN)</td>
<td>status: (EX)</td>
<td>status: (IN)</td>
</tr>
<tr>
<td>shared with: (B)</td>
<td>shared with: (A)</td>
<td>shared with: (C)</td>
</tr>
<tr>
<td>justification: (PREMISE)</td>
<td>justification: (PREMISE)</td>
<td>justification: (PREMISE)</td>
</tr>
<tr>
<td>fact2: Q</td>
<td>fact1: P</td>
<td>fact1: R</td>
</tr>
<tr>
<td>status: (IN)</td>
<td>status: (EX)</td>
<td>status: (IN)</td>
</tr>
<tr>
<td>shared with: (nil)</td>
<td>shared with: (A)</td>
<td>shared with: (nil)</td>
</tr>
<tr>
<td>justification: (fact1, rule1), (fact4, rule2)</td>
<td>justification: (PREMISE)</td>
<td></td>
</tr>
<tr>
<td>fact3: W</td>
<td>fact2: Q</td>
<td>fact5: S</td>
</tr>
<tr>
<td>status: (IN)</td>
<td>status: (IN)</td>
<td>status: (IN)</td>
</tr>
<tr>
<td>shared with: (nil)</td>
<td>shared with: (nil)</td>
<td>shared with: (nil)</td>
</tr>
<tr>
<td>justification: (fact2, rule4)</td>
<td>justification: (fact1, rule1), (fact4, rule2)</td>
<td>justification: (fact4, rule3)</td>
</tr>
</tbody>
</table>
**TMS Example – Belief Revision**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th>B</th>
<th></th>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fact1: P</td>
<td>status: (OUT)</td>
<td>shared with: (B)</td>
<td>justification: (PREMISE)</td>
<td>fact1: P</td>
<td>status: (OUT)</td>
<td>shared with: (A)</td>
</tr>
<tr>
<td>fact2: Q</td>
<td>status: (IN)</td>
<td>shared with: (nil)</td>
<td>justification: (fact4, rule2)</td>
<td>fact3: W</td>
<td>status: (IN)</td>
<td>shared with: (nil)</td>
</tr>
<tr>
<td>fact4: R</td>
<td>status: (EX)</td>
<td>shared with: (C)</td>
<td>justification: (premise)</td>
<td>fact5: S</td>
<td>status: (IN)</td>
<td>shared with: (nil)</td>
</tr>
</tbody>
</table>

**Mobile Agents**

- **Why:**
  - Reduced network traffic,
  - Multiprocessor calculations
  - Low reliability networks.

- **Lifecycle model:**
  - Creation,
  - Starting,
  - Deactivation,
  - Disposal.
Aglets – Life Cycle of an Aglet

Why Retract Is Needed...
Cooperative Problem Solving through Agent Mobility

Electronic Marketplace

- Consumer Aglets live in Marketplace.
- They monitor prices and make purchases.
- Seller aglets also live here.
- May negotiate ...
Agent Ensembles – Transparency of Execution Nodes

KQML has been implemented as a Java Toolkit – jKQML.

It is a common language for cooperation.

Facilitator agents serve as brokers of services.

Ensemble of agents can cooperatively solve problems without regard to hardware locale.

Conclusions
Module III
Distributed Decision Making

Contents

- Principles of Distributed Decision Making
- Voting
- Auctions
- Social Choice
- Bargaining Theory
- Nash Equilibrium
- Clarke's Tax Algorithm
**Multi-Agent Systems**

- Industrial trend towards virtual enterprises.
  - Dynamic alliance of a number of small, agile enterprises which together achieve the economies of scale.
- Multi-agent technology facilitates such negotiation at the operative decision making level.
  - Automation saves labor for human negotiators.
- Agents are better at sifting through large search spaces and may offer strategic advantages.

**Distributed Decisions**

- Agents have different goals - each is trying to maximize their own good without the concern for a global good.
- Issue of self-interest has to be dealt with in building the IT infrastructure for negotiations.
Cooperative Problem Solving Vs. Multi-Agent Systems

- Cooperative Distributed Problem Solving, the system designer imposes an interaction protocol and a strategy.
- Multiagent Systems, the designer imposes an interaction protocol but agents follow their own strategy.
- Agent will choose the best strategy for itself - so the protocols need to be designed for a non-cooperative, strategic perspective.
- Idea: make the desired strategy for the agent to be its best strategy - forcing the agent to use it.
  - For example, if the protocol (Clark's Tax Algorithm) is designed such that the best strategy for the agent is to speak the truth - so the agent speaks the truth.

Negotiation Protocols

- Negotiation Protocols are evaluated according to many criteria:
  - Maximize Social Welfare: Total sum of the payoff (utilities) to all agents.
  - Enable us to reach Pareto Efficient solutions: A solution $x$ is Pareto Optimal, such that there is no other solution $x'$ such that at least one agent is better off in $x'$, and no other agent is worse off.
  - Individual Rationality: This criteria states that the agent should be better off by participating in the negotiation, than by simply not participating. (Example of this the Nash Equilibrium situation.)
  - Stability or Non-manipulable. Should cause the agents to behave in a desired manner.
**Nash Equilibrium**

1. Because if the agent can behave in a non-desired manner, it will do so. Dominant Strategies are those that the agents use, no matter what the others are doing.
2. When dominant strategies do not exist, stability criteria is needed. One such criteria is Nash Equilibrium.

The strategy profile SA= \(<S_1^*, S_2^*, S_3^*, \ldots S_{|A|}^*>\) amongst agents \(|A|\) is in Nash Equilibrium if for each agent \(i\), \(S_i^*\) is the best strategy—given the others choose \(<S_1^*, S_2^*, S_3^*, \ldots S_{i-1}^*, S_{i+1}^*, \ldots S_{|A|}^*>\).

**Social Choice (1)**

- **Plurality Protocol:** All alternatives compared simultaneously. One with the highest number of votes wins.
- **Binary Protocols:** Pairwise comparisons. The order of comparisons influences the final outcome.
Social Choice – Borda’s Protocol

- **Equilibrium of a Protocol:**
  - Assumes that insincere agents are participating in the protocol.
  - Outcome is the same when the agents reveal their true choices.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a&gt;b&gt;c&gt;d=e</td>
</tr>
<tr>
<td>2</td>
<td>b&gt;c&gt;d&gt;a=e</td>
</tr>
<tr>
<td>3</td>
<td>c&gt;d&gt;a&gt;b=e</td>
</tr>
<tr>
<td>4</td>
<td>a&gt;b&gt;c&gt;d=e</td>
</tr>
<tr>
<td>5</td>
<td>b&gt;c&gt;d&gt;a=e</td>
</tr>
<tr>
<td>6</td>
<td>c&gt;d&gt;a&gt;b=e</td>
</tr>
<tr>
<td>7</td>
<td>a&gt;b&gt;c&gt;d=e</td>
</tr>
</tbody>
</table>

Groves-Clark tax Algorithm

- Let the outcomes be:  
  $O = (g, ?, ?, \ldots, ?_{|A|})$
- Let the utility for the outcome:  
  $U_i(O) = V_i(g)$
- If $V_i^{\text{gross}}(g)$ be the utility of reaching outcome $g$, and let $\Phi = (-P/|A|)$ be the equally distributed cost of reaching $g$, the utility of $g$ is:  
  $U_i(O) = V_i^{\text{gross}}(g) - \Phi$
- Tax is computed as:  
  - Every agent $i$, reveals its valuation in every choice $V_i^*(g)$ for every possible $g$.
  - Social Choice is $g^* = \arg \max_g \sum_{j \neq i} V_j^*(g)$
  - Tax levied:
    $\text{Tax}_i = \sum_{j \neq i} V_j^*(g^*) - \sum_{j \neq i} V_j^*(\arg \max_g ?_{k \neq i} V_k^*(g))$
Example – Groves Clarke Algorithm

<table>
<thead>
<tr>
<th></th>
<th>Without Collusion</th>
<th>With Agents 1 and 2 Colluding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$V_i^{\text{max}}$</td>
<td>$V_i(1)$</td>
</tr>
<tr>
<td>1</td>
<td>5000</td>
<td>2000</td>
</tr>
<tr>
<td>2</td>
<td>4000</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>-2500</td>
</tr>
</tbody>
</table>

- Cost of the pool is $9000.
- Valuation of Agents 1 and 2 is high for this choice (5000 and 4000 respectively).
- If they collude, and rate their valuations 500 units more than actual, they reduce taxes.

Conclusions

- Interaction protocols need to be designed such that the dominant strategy of the agents is to tell the truth.
- Although, this is possible in principle, generally agents may collude and “beat the system”.
Module IV
Learning in Multi-Agent Systems

Contents
- Bayes Theorem
- Bayesian Networks
- Utility Functions
- Reinforcement Learning
- Examples
  - Motion Planning
  - Routing
Belief Networks

- Based on Bayes’ Rule.
- Belief Network is used to represent dependence between variables and give a concise specification of the joint probability distribution.
- A belief network is the graph for which the following holds:
  - A set of random variables make up the nodes in the network
  - A set of directed links or arrows connects pairs of nodes. The meaning of an arrow from X to Y is that X had direct influence on Y.
  - Each node has a conditional probability table that quantifies the effects that the parents have on the node. (The parents of a nodes are all those that point to it).
  - The graph had no cycles (hence it is a directed, acyclic graph, or a DAG).

Belief Networks – Benefits

- It is easy for an expert to establish what influences what.
  - Thus, the topology is easy to establish in comparison to the specification of the exact probabilities.
- After the topology is defined, only the probabilities for the direct dependence need to specified.
  - The others are computed using Bayes’ rule.
Example

| Burglary (B) | Earthquake (E) | \( P(\text{Alarm|B,E}) \) | \( 1 - P(\text{Alarm|B, E}) \) |
|-------------|---------------|-----------------|-----------------|
| True        | True          | 0.95            | 0.05            |
| True        | False         | 0.94            | 0.06            |
| False       | True          | 0.29            | 0.71            |
| False       | False         | 0.001           | 0.999           |

Updated Belief Network

Making Decisions with Belief Networks

Joint Probability Distribution
\[
P(X_1 = x_1 \land X_2 = x_2 \land \ldots \land X_n = x_n) = \prod_{i=1}^{n} P(x_i | \text{Parents}(X_i))
\]

Example: Compute the probability that an alarm has occurred but there is no earthquake, and both John and Mary has called.

\[
P(\text{J} \land \text{M} \land \text{A} \land \neg \text{B} \land \neg \text{E}) = P(\text{J}|\text{A}) \cdot P(\text{M}|\text{A}) \cdot P(\text{A}) \cdot P(\neg \text{B}|\neg \text{E}) \cdot P(\neg \text{E})
\]
\[
= 0.9 \cdot 0.7 \cdot 0.001 \cdot 0.999 \cdot 0.998 = 0.00062
\]
Learning in Belief Networks

Utility Based Decisions

- We denote $U(S)$ to be the utility of the agent’s being in state $S$.
- A non-deterministic action $A$ will have possible outcome states $\text{result}_i(A)$, where the index "$i$" ranges over the different outcomes.
- Prior to executing the action, the agent assigns a probability $P(\text{result}(A) | \text{Do}(A), E)$ to each outcome, where $E$ summarizes the agent’s available evidence about the world, and $\text{Do}(A)$ is the proposition that $A$ is executed in the current state.
- the expected utility of the action $A$ given the evidence is $EU(A | E)$ defined as follows:
Utility Principle

- The principle of Maximum Expected Utility (MEU) states that a rational agent should choose the action that maximizes the agent’s expected utility.
- If an agent’s preferences obey the axioms of utility, then there exists a real valued function $U$ that operates on states such that $U(A) > U(B)$ if and only if $A$ is preferred to $B$, and $U(A) = U(B)$ if and only if the agent is indifferent between $A$ and $B$.

Lotteries

- Complex scenarios are called lotteries to emphasize the idea that the different attainable outcomes are like different prizes, and the outcomes are determined by chance.
- The lottery $L$, in which there are two possible outcomes—state $A$ with probability $p$ and state $B$ with the remaining probability, is written as
  
  $L = [ p, A; 1-p, B ]$
Utility Functions – Expressed as Lotteries

- **Maximum Expected Utility Principle**: The utility of a lottery is the sum of probabilities of each outcome times the utility of that outcome:

\[ U(p_1, S_1; \ldots; p_n, S_n) = \sum p_i U(S_i) \]

- Utility is a function that maps from states to a real number.
- For example, an agent may exhibit a monotonic preference for money.
- This is not sufficient to guarantee that money behaves as a utility function.

Example

- Consider a game, where you can have won a $1,000,000 prize. You can either take that away, or choose to toss a coin. If the coin comes up heads, you take home nothing. However, if it comes up tails, you take home $3,000,000.
- **Expected Monetary Value if you choose to toss the coin is** = $1,500,000
- **So why would most people decline to toss the coin?**
- **We can explain using the utility functions:**
  - Assuming that you assign a utility of 5 to current financial status, and utility of 8 to winning the $1,000,000 and utility of 10 to winning $3,000,000. (Usually, the utility of the first million would be more than the remaining 2 millions).
  - **No Toss has** U of 8, while toss has 0.5x5 + 0.5x10 = 7.5
Complex Decisions

In sequential decision problems we are concerned with the situations where the utility depends upon a sequence of decisions.

Decision making process returns a policy – that is, a set of situation-action rules for each state- arrived at by calculating the utility for each state.

Other than the states marked +1 and –1, there is no utility associated with any state.

Stochastic Case

In the deterministic version of the problem, each action reliably moves one square.

Each action achieves the intended result with the probability of 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction.

For example, from the start cell (1,1),
- action North moves the agent to cell (1,2) with the probability of 0.8,
- but with the probability 0.1, it moves East to cell (2,1),
- and with probability 0.1, it moves West, bumps into the wall, and stays in cell (1,1).
Utility of a Path

- Utility function must be based on the sequence of states – an environment history rather than a single state.
- Let us assume that the utility of a state will be the value of the terminal state minus 1/25th the length of the sequence.
- Thus, a sequence of length 6 that leads to the +1 cell, has a utility of $+1 - 6/25 = 0.76$.

Sequence of Actions

- Then, one could simply apply the Maximum Expected Utility principle to sequences.
- The rational action would be to the first action of the optimal sequence. This is how a search algorithm would really work.
- This approach has the fundamental flaw – it requires the agent to commit to an entire series of actions.
- Thus, an agent may end up action irrationally based on the new sensory information received after every move.
**Policy**

- The term **transition model** is used to refer to the probabilities associated with the possible transitions to the state after a given action.
- The notation $M_{ij}^a$ denotes the probability of reaching state $j$ if action $a$ is done in state $i$.

Policy offers an agent the opportunity to choose a new action after each step – based upon the new information provided by its sensors.

- The problem of calculating the optimal policy in an accessible, stochastic environment with known transition model is called a **Markov Decision Problem (MDP)**.
- Value iteration is an algorithm for computing the optimal policy. The basic idea is to calculate the utility of each state, $U(\text{state})$, and then use the state utilities to select an optimal action in each state.
- The notation $H(\text{state}, \text{policy})$ denotes the history tree starting from the state and taking an action according to the policy. Then the utility of the state $i$ is given by the expected utility of the history beginning at that state and following an optimal policy:

$$U(i) = \mathbb{E} \left[ U(\text{H}(i, \text{policy}^*)) \mid M \right] = \sum P(\text{H}(i, \text{policy}^*) \mid M) U_h(\text{H}(i, \text{policy}^*))$$

Where $\text{policy}^*$ is an optimal policy defined by the transition model $M$ and the utility function on histories $U_h$. 
Value Iteration (1)

The simplest form of utility functions are additive, as shown below. Here $R(s_0)$ is the reward associated with state $s_0$. In practice, utility functions are assumed to be additive.

$$U_h([s_0, s_1, \ldots, s_n]) = R(s_0) + U_h([s_1, \ldots, s_n])$$

Given an additive utility function, we can recover the standard Maximum Expected Utility principle that an optimal action is one with maximal expected utility of outcome states:

$$policy^*(i) = \arg \max_a \max_j \mathbb{E}U(j)$$

Value Iteration (2)

Also, the utility of a state can be expressed in terms of the utility of its successors:

$$U(i) = R(i) + \max_a \mathbb{E}U(j)$$

A simple algorithm for approximating the utilities of states to any degree of accuracy utilizes the following iterative procedure:

$$U_{i+1}(i) = R(i) + \max_a \mathbb{E}U_{i+1}(j)$$
Value Iteration (3)

As $t \to \infty$, the utility values will converge to a stable set of values.

Reinforcement Learning

Reinforcement learning (q-learning) is based upon the Temporal Difference equations that are generally utilized to learn utility of each state. Note that $R(i)$ is the reward of state $i$.

Thus, for example, if we transitioned from State $i$ to State $j$, with $U(i) = -0.5$, and $U(j) = +0.5$, we should consider increasing the utility of state $i$ according to the TD (temporal difference) formula above.

Passive learning requires a set of Training Sequences such as:

1. $(1, 1) \to (1, 2) \to (1, 3) \to (1, 2) \to (1, 1) \to (2, 1) \to (3, 1) \to (4, 1) \to (4, 2) \to 1$
2. $(1, 1) \to (1, 2) \to (1, 3) \to (2, 3) \to (3, 3) \to (4, 3) \to +1$
3. $(1, 1) \to (1, 2) \to (1, 1) \to (1, 2) \to (1, 3) \to (2, 3) \to (3, 3) \to (4, 2) \to 1$
4. $(1, 1) \to (2, 1) \to (3, 1) \to (2, 1) \to (1, 1) \to (1, 2) \to (1, 3) \to (2, 3) \to (3, 3) \to (4, 3) \to +1$
5. $(1, 1) \to (2, 1) \to (1, 1) \to (1, 2) \to (1, 3) \to (2, 3) \to (3, 3) \to (3, 2) \to (4, 2) \to 1$
Passive Learner

The following simple algorithm is utilized, where $e$ is the environment, to implement TD. Percepts is a sequence of states that the agent has visited.

1. $J = \text{State}(e)$
2. if $J$ is a terminal state
   $U(J) = \text{Reward}(J)$
3. else (if length (percepts) > 1)
   $I = \text{Previous State}$
   $U[I] = U[I] + \alpha (\text{Reward}(I) + U[J] - U[I])$

---

Exploratory Learner (1)

- The agent does not have any training sequences at its disposal and must explore and learn the utility of each state.
- The agent learns its optimal policy (i.e. the best action for a given state) using the Q-values that it derives.
- The idea is to maintain a list of past actions (i.e. move N, move S, move E, and move W) and use the associated previous rewards to learn the Q-values iteratively.
- In relation to the passive learner, $Q(a, i)$ represents the utility of taking the action $a$ in state $i$, i.e.:

$$U(i) = \max_a Q(a, i)$$

$$Q(a, i) = R(i) + \beta \max_a \max_{J} Q(a', J)$$

$$M_{ij} = a'$$
Exploratory Learner (2)

- In contrast, Q-Learning enables the agent to learn $Q(a,i)$ without explicitly learning the stochastic environment model $M_j^a$.
- Temporal Difference is utilized to learn the Q-value for each state:

$$Q(a, i) = Q(a, i) + \alpha (R(i) + \max_{a'} Q(a', j) - Q(a, i))$$

- $Q(a,i)$ is updated after each transition from state i to j. The following algorithm is executed at each state transition. It returns the best action to be taken in the current state.

Conclusions

- Agents compute utilities for making decisions in a stochastic environment.
- Complex decisions requiring a sequence to decisions are sometimes necessary.
- The method of choice is to develop state based policies to implement these.
- Q-Learning is used for enabling an agent to explore its environment.
Tools to Get Started

- Aglets – A Mobile Agent toolkit from IBM.
  - Aglets move around the network.
  - One may specify an itinerary of “places” where aglet server is running.
- Microsoft – ActiveX Agent Building Environments.
  - BOTS, SPIDERS (C++)
- Swarm – Multi-agent simulations
- Intelligent Agent Factory – Java
- JESS – CLIPS implementation in Java for building forward/backward reasoning.

Tools to Get Started

- FIPA: Foundation of Intelligent Physical Agents.
  - www.fipa.org
- DAML Agent Ontologies
  - http://www.daml.org/ontologies/
- jKQML
**Project: QoS for Distributed Simulation**

- Maintained configuration / performance information for each node in ensemble
- Used AI techniques for mapping distributed simulation components to machines.

**Projects: Visual Realism in Driving Simulator - Planning**
Visual Realism in Driving Simulator

Workshop Summary

- Discussed basic architectures – reflexive and BDI.
- Described languages for inter-agent communication.
- Discussed issues for making decisions that affect all agents – social choice and Clarke’s tax algorithm.
- Outlined strategies for making simple, and complex (sequence) for decisions, and approaches utilized for creating “learning agents”.