Benjamin Fenster
CS-251 (Wu)
19 December 2005

Notes – Knowledge-Based Systems

I. Expert System
   a. It's an *expert's system*. It's based on the knowledge of some expert!
   b. Deals with ill-structured problems (don't have complete information or a complete algorithm to solve it)
   c. Should give expert-level performance (some people are better at solving a particular problem than others; we should have the best performance)
   d. Need some general domain knowledge as well as expertise.
   e. Expert System Shell: An expert system with the knowledge taken out. This way you can add new knowledge to solve a different problem.

II. Components
   a. Inference Engine: Do the work of processing user input
   b. Working Memory: Store information for a particular user
   c. Tracing Engine: Be able to explain the reasoning applied
   d. Interface to regular users
   e. Interface to domain experts (to provide knowledge)
      i. Knowledge Acquisition Engine
         1. Data Mining tries to get knowledge automatically
         2. Could have computer programmers ask (in code) a set of questions
         3. Knowledge base management system
   f. The Knowledge Base itself: stores the knowledge

III. Knowledge Processing
   a. Knowledge is divided into three parts
      i. Data (evidence of a specific problem, symptoms of one patient)
      ii. Knowledge (from books, doctors, et cetera: applicable to all relevant problems (e.g. all patients)
      iii. Knowledge application (manipulation): Applying knowledge to solve problems
   b. Three Fundamental Techniques
      i. Knowledge representation
      ii. Knowledge acquisition
      iii. Knowledge inference

IV. Knowledge Representation
   a. Introduction
      i. Knowledge has: Significant objects (students, faculty, classrooms) and relations in a domain ("people have two arms")
      ii. Data are evidence about specific problems ("my height is 1.72 meters")
   b. Logic / State Space
      i. In logic, a rule's head holds only when its body can be satisfied
      ii. In state space: Can achieve one state from another by applying operations
   c. Scheme + Medium
      i. Scheme deals with a conceptual design
      ii. Medium deals with implementation of a representation scheme (programming languages are media for algorithm schemes)
   d. Knowledge Representation Schemes
      i. Logic-Based Representation (e.g. first order logic, medium Prolog)
      ii. Procedural Representation: Knowledge is a set of instructions for solving problems. We'll spend significant time on this. If P then G (Start with supports and draw conclusions). Medium: CLIPS
      iii. Network Representation: Knowledge is represented as a graph where nodes are objects and edges are relations
      iv. Structured Representation: Will make sense given a background in object oriented programming
   e. Issues in Choosing a Knowledge Representation Scheme

    i. Expressiveness ("John's car is *redder* than Joe's"). What granularity can one express?
    ii. Extensibility / Modularity
    iii. Clarity
    iv. Naturalness

V. Production Systems
  a. Basic Components
    i. Rule Base: If LHS Then RHS (The LHS determines when the rule may be applied)
    ii. Working Memory
    iii. Inference Engine
  b. Inference Engine
    i. Contrast with Prolog's backtracking mechanism.
    ii. "Four Steps":
      1. Match: Find rules whose LHSs are satisfied from the contents of working memory
      2. Conflict Resolution: Pick a rule to apply based on some conflict resolution strategies (e.g. pick the first applicable rule or pick the one with the most serious consequences).
      3. Act: Update working memory with the RHS of the selected rule (may have to add or remove an item)
      4. Step 4: Repeat these steps
      5. Called the "Recognize-Act" algorithm
    iii. Data-driven or forward chaining. This is different from the goal-driven or backward chaining technique Prolog uses.
    iv. Procedure
      1. Label all rules whose LHSs are satisfied
      2. De-label rules whose RHSs are just repeats of what's already in the working memory.
      3. If there are no labeled rules, exit; otherwise pick the rule with the smallest ordinal.
      4. Clear all labels and repeat.
  c. Conflict Resolution
    i. If you have more than one applicable rule, which one should you use?
    ii. Refraction: Once a rule has fired, it may not fire again until the working memory elements that match its LHS are modified
    iii. Recency:
      1. Rules whose LHSs match the most recently added working memory items are preferred.
      2. This is a heuristic. It may miss some solutions.
      3. It's like a depth-first search rather than a breadth-first search. It may end up adding new rules forever, just as you might search deeper and deeper into a search tree.
    iv. Specificity
      1. A more specific rule (i.e. one with more conjunctive conditions) is preferred over a more general rule.
      2. This gives rules matching fewer working memory situations a better chance.
    v. Heuristic: Provide some other function evaluate each rule.
    vi. RETE
      1. A milestone in AI
      2. Avoids problems with the recognize-act algorithm
      3. Motivations
        a. Don't keep checking the working memory against the rule base.
          i. Store information between cycles
          ii. Memorize which rules were successfully matched

iii. A pattern can be matched only by some working memory elements (e.g. those with the same name)
iv. Make a connection between working memory elements and left sides of rules. When there's a change, only check the affected rules.
v. Keep a list of related working memory items for each rule. Update it whenever rules are added or deleted.
vi. Tokens
1. Identifier
2. Series of name-value pairs
3. (Expression Name Expr4 ^Arg1 X ^Op + ^Arg2 Y)
4. Now we're able to delete with one token and add with another in order to change something

b. Avoid iterating over all rules
i. Organize rules into trees to show their relationships
ii. Find intra-element features (its class, its name-value pairs)
iii. Find inter-element features (shared values)

4. The Sorting Network
a. Have a root node that's not a real fact. Put all possible evidence nodes underneath that.
b. Then build a linear sequence of intra-element features.
c. Next use the inter-element rules to combine related "symptoms". Take two α memory nodes from the end of the sequence and combine them.
d. NB: I have absolutely no idea what this means.

5. Linear Forward-Chaining (LFA)
a. Prof. Wu's algorithm
b. Sort out relationships between different factors
c. Put rules inside rule schemas
d. Start from the bottom-level nodes to make sure each rule is used only once
e. It's more efficient than RETE when both work, but it wasn't published in the top AI journals so it hasn't caught on.

d. Features of Production Systems
i. Modular rules. There's no direct interaction between rules (only through the working memory) so one can add / remove rules. This supports incremental development
ii. The natural and concise if-then structure is efficient to express human problem solving.
iii. Rules are independent from the recognize-act algorithm. This separates knowledge from control and makes programming easier
iv. One can display all selected rules as a mechanism for explaining decisions and tracing

VI. Reasoning Under Uncertainty
a. In knowledge-based systems we don't have *certain* rules. There are poorly formed problems.
b. There's Noise:
i. Ambiguity: A statement may not be entirely clear
ii. Inexact Knowledge: How high is a high fever?
iii. Abductive Reasoning
1. Being too proud leads to failure. Failure is the mother of success. Therefore: Being too proud leads to success

2. Deductive Reasoning: Take general knowledge and apply it to specific problems
3. Inductive Reasoning: From specific cases, induce general knowledge
4. Abduction: Use partial knowledge
5. If you're drunk, you do silly things.
6. Therefore, if there's been a murder and there are no obvious leads, start by looking for drunk people
7. That won't give any kind of definite answer, but it gives a good starting point.

c. Certainty Factors
   i. Have a measure to describe imperfect data (a certainty factor). This could be a probability or "fuzzy membership" or something similar.
   ii. Have a measure to represent imperfect *rules* (conditional probability or rule strength)
   iii. Have a mechanism to apply knowledge in the knowledge base to solve the problem (so integrate the inexact data and inexact rules)
      1. Determine certainty of confusion based on the certainty of supports within the rule
      2. Then determine the final certainty factor of the conclusion as supported by multiple rules
   iv. Certainty Factor Algebra
      1. Evidence described with a probability
      2. Have a measure of belief (MB) and disbelief (MD). MB(H|E) is the measure of belief in hypothesis H given evidence E
      3. CF(H|E) = MB(H|E) – MD(H|E)
      4. CF(P1 and P2) = MIN(CF(P1), CF(P2))
      5. CF(P1 or P2) = MAX(CF(P1), CF(P2))
      6. Combine CFs when multiple rules support a premise
      7. CF(R1) + CF(R2) – CF(R1) + CF(R2) when CF(R1) > 0, CF(R2) > 0
      8. CF(R1) + CF(R2) + CF(R1) + CF(R2) when CF(R1) < 0, CF(R2) < 0

d. Fuzzy Logic
   i. Probability deals with possibilities (a coin lands on either heads or tails)
   ii. Here, instead, we have a deterministic value (Bobbo's grade point average is 3.2) but get different opinions on its meaning (good or bad)
   iii. "I am 22 years of age. Is that young or old?"
   iv. Create a fuzzy membership function. What's the probability that a given age is "young?"
   v. So set membership is now defined with a 0 to 1 value. Rules are combined using MAX for OR and MIN for AND again.

e. Dempter-Shafer's Theory of Evidence
   i. Back to probability
   ii. Distinguish between *uncertainty* (I have an idea but I'm not sure) and *ignorance* (I don't know the answer, so I won't guess)

f. Non-monotonic Logic and Reasoning with Beliefs
   i. Address changing beliefs
   ii. Start with some assumptions, then either uphold or reject them as you get more evidence
   iii. Example: P unless Q -> R. P. R -> S
   iv. Truth maintenance system: Correct erroneous conclusions based on new evidence. *Maintain* what was right and what was wrong.
   v. Monotonic Reasoning (by contrast): Have a set of axioms believed true; infer their consequences).

VII. Associationist Theories

a. Logic vs. Associationist Theories
   i. Logic: Start with some axioms; prove new theorems to add to knowledge
   ii. Associationist:

1. Two basic components: Nodes, associations
2. Like the Entity-Relationship model from databases. (See [BSAD-141], [BSAD-144], or [CS-204] for details)
3. The idea is that sound logic doesn't always make sense. 2+2 = 5 => Elephants are Green (a true statement!)
4. Mathematicians and Philosophers prefer logic.
5. Psychologists prefer associations

b. Semantic Networks

   i. Represent knowledge as a graph. Facts are nodes; arcs are associations between them
   ii. frosty –is-a--> snowman --made of --> snow –form of --> water <-- form of – ice
   iii. Clearly the paradigm supports inheritance.
   iv. Information can be inherited based on semantic labels
   v. The power comes from the definition of links and associated inference rules

c. Conceptual Graphs

   i. Can easily map to this from semantic networks
   ii. It's still a visual format
   iii. No labeled arcs here
   iv. Nodes can either be concepts or conceptual relations (like how the E-R model uses diamonds and boxes, but conceptual graphs doesn't even make that much of a distinction).
   v. Concepts may be abstract in the object-oriented sense
   vi. The graph must be finite (but may be arbitrarily complex)

d. Frames

   i. A popular paradigm in AI
   ii. Have a static data structure for each concept; include as much information as you need
   iii. Provide all the related items for each frame
   iv. Organize knowledge stereotypically based on past experience and revise it to represent differences for new situations.
   v. Example: For a hotel you have some base of experience with different hotels and can form expectations based on that. That's a hotel "Frame."
   vi. A Frame consists of information slots (name, address, et cetera), pointers to other frames, and attached procedures (like methods on a class) for performing functions to get values.
   vii. Default information: If you can't see any information, take a default value. How does this hotel take payments? If you don't know, assume it's by credit card.
   viii. A BNF Description of Frames:
      1. <Frame> :- <FrameName><Slots>
      2. <Slots> :- <Slot> | <Slot><Slots>
      3. <Slot> :- <SlotName><Facets> | <Frame>
      4. <Facets> :- <Facet> | <Facet><Facets>
      5. <Facet> :- <FacetName><FacetValue>
      6. <FacetName> :- range | default | if-needed
      7. <FacetValue> :- <context> | <default-value> | <inherited-value> | <attached-procedure>
   ix. This is a lot like object-oriented programming, but with the significant difference that there's no distinction between a value and a function. There are some other minor differences, but it's otherwise basically the same.
   x. Ben notes: Microsoft .NET allows Properties, which make no distinction between functions and values.
   xi. Example: A Restaurant Frame

```
Specialization-Of: Business Establishment
Types:
      range: (Cafeteria, Seat Yourself, Wait to be Seated)
      default: Wait to be Seated
```

```
            if-needed: If plastic orange counter then Cafeteria
                  if "wait for waitress" sign
                        or reservations made
                        then Wait to be Seated
                  otherwise Seat Yourself
      Location: ...
      Name: ...
      Food Style: ...
```

e. Frames vs. Semantic Networks
    i. Frames make it easier to organize knowledge hierarchically (one frame consists of other frames)
    ii. Think of a frame as a single entity and only consider details when needed
    iii. Procedural attachment: Can create demons (procedures invoked as side effects of some other action)
    iv. Supports inheritance *explicitly* through default values and slots

VIII. Knowledge Acquisition

a. In theory in AI, the system is never finished (unlike software development tasks)
b. Knowledge Bottleneck Problem
    i. Since we want an expert's system, knowledge is power.
    ii. Using knowledge is easy. Acquiring it is *hard*.
    iii. It's difficult to explain knowledge we (humans) have. It's hard to articulate.
    iv. For example: Handwriting recognition is *easy* to a human, but is still very hard for computers.
    v. Likewise theorem proving is still very hard for machines.
c. Three Types of Knowledge Acquisition
    i. By Interview: Engineers write a program after asking questions
    ii. Interactive Knowledge Transfer: The system asks questions of experts by itself
    iii. Automatic Knowledge Acquisition (a.k.a. Machine Learning): Even the experts don't have the knowledge readily usable, so provide case studies of their skill at work and let the machine derive knowledge from that.
d. Knowledge Engineering
    i. This isn't a traditional software development task.
    ii. It's complicated by the interaction between the knowledge engineer and the domain expert.
e. By Interview
    i. Techniques
        1. Questionnaire: "What is the most important factor to diagnose Disease A" (for example)
        2. Problems from the Programmer's Perspective
            a. How to fill in the missing details between separate sets of things discussed
            b. The expert may give different information in response to different questions that are not easily tied together
        3. Problems from the Expert's Perspective
            a. Things that are easy to do are difficult to say
            b. Problems with natural language. A big mouse is smaller than a small elephant.
            c. Problems with logic: Or vs. Xor, if vs. iff, et cetera
    ii. One possible solution: Have a programmer who becomes the expert; then the programmer can write the system by hand.
f. Interactive
    i. TEIRESIAS (in MYCIN): One of the first AI systems; built at Stanford university
        1. Interactive knowledge acquisition
        2. Provide some arbitrary rules to start
        3. Then if the system is wrong (according to an expert), ask specific questions to figure out what the expert knows that the system doesn't.

4. This requires that there's *some* set of initial rules.

ii. SIKT
   1. Developed by Prof. Wu
   2. Doesn't require any initial rules
   3. Build a domain network. At the top: The problem we're solving. Underneath: factors the user (a non-programmer) specifies

iii. Problems
   1. Knowledge representation is fixed, so if the knowledge the expert provides isn't suitable this approach won't work.
   2. When the programmer was doing it s/he had the ability to completely structure the representation around the actual knowledge.

g. Rule Induction
   i. Start from a base of rules and trim out the rules that are unnecessary
   ii. Induction
      1. From specific to general
      2. In mathematics, induction is truth preserving
      3. In statistics, on the other hand, one can perform a regression which may not hold true to the original data.
      4. In AI, induction isn't truth-preserving.
      5. Empirical-Induction: generalization-specialization
   iii. Abduction
      1. Inexact reasoning; rules are incomplete.
      2. Start from something unreliable.
      3. If we have P(a) -> O(b), might conclude O(b) -> P(a)
   iv. Deduction: General to specific; truth-preserving
   v. Knowledge Refinement
      1. Be careful removing redundancy
      2. An element that appears 100 times may be more reliable than one that appears only twice, so redundancy carries some meaning.
      3. Assimilation: Assume knowledge in the knowledge base is correct, so only integrate new knowledge when it's consistent
      4. Accommodation
      5. Generalization
      6. Specialization

h. Learning Strategies
   i. Source information determines the task the learning engine will perform (including which learning strategy)
   ii. Rote Learning: Memorizing, the way a child learns.
   iii. Learning by Being Told
      1. Learning by advice-taking
      2. Like information acquisition for MYCIN
      3. Wait for someone to say, "You were wrong," and then figure out (by asking, perhaps) what exactly went wrong
   iv. Learning from Examples
      1. Input is very detailed (specific cases)
      2. Intelligent Learning Database Systems
         a. Definition (Prof. Wu's 19995/2000 definition):
         b. Have database support, such as a relational/"normal" database (maybe even a plain text file)
         c. Use induction to extract knowledge
         d. Interpret the extracted knowledge to solve problems by deduction
      3. Induction Paradigms
         a. Supervised learning / classification
            i. Need someone to describe the "outcome" of existing sets of data (called the *class label*)

            ii. Given a set of symptoms, name the disease
    b. Association Analysis
        i. Work out which items are highly correlated
        ii. For example: It's been discovered that people who buy diapers also tend to buy beer.
        iii. Want to group related items together in the supermarket.
        iv. Apriori + FT-Growth algorithms (where the latter scans the data only twice!)
        v. No *classes* in association analysis; the data speak for themselves.
        vi. It's also called basket analysis because of the supermarket application.
    c. Unsupervised Clustering: Don't have anybody to give class labels
    d. Discovery of Quantitative Laws
        i. Like statistical regression, but with symbolic values
4. Processing Real-Valued Attributes
    a. Decision trees are built by segregating items into discrete groups
    b. What do we do with real-valued attributes?
    c. Discretization:
        i. Split temperature data into (low, medium, high) and now there are discrete groups.
        ii. Could also draw cut points to create separate classes
        iii. Another approach (Bayesian): Construct a probability curve for each class
        iv. Wherever two curves connect, draw a cut point there (so the highest probability "wins")
    d. Information Gain
        i. This is what's really implemented
        ii. Find a cut point where you get the most information.
        iii. Consider points between any pair of different classes
        iv. Actually split at a point if
            1. The split will produce information gain
            2. The number of examples in the current interval is greater than some threshold
5. Noise Handling
    a. Erroneous, missing, or misclassified data, or a poor distribution of examples in the training data (e.g. a sample of all children and no adults in a medical database)
    b. Redundant data may be helpful
    c. Preprocessing
        i. Replace missing values (?) with the most frequently found value for that data item
        ii. Or: Pick some other sensible means to handle (?)
    d. Induction Time (pre-pruning)
        i. Avoid overfitting by having an overly complicated decision tree.
        ii. A 100% fit may not be a good thing
    e. Post-Pruning: Simplify the induction results further
6. No Match
    a. There are several techniques to resolve when there's no class match.
    b. The easiest: just pick the largest class (the most common diagnosis) and offer that as an answer
7. Multiple Match
    a. First Fit: simplistic (take whichever match you find first)

b. Largest Class: Take whichever match is the largest overall
c. Largest Conjunctive rule
d. Estimate of probability
e. Fuzzy Interpretation of Discretized Intervals in HCV
    i. Separation between “high” and “medium” becomes fuzzy
    ii. This actually helps more to resolve No Match situations
    iii. Where there are multiple matches it actually makes it worse, generally, by allowing even MORE matches. When there’s No Match, allowing a different interpretation may produce a match, and that’s good.

v. Learning by Analogy (case-based reasoning)
    1. Still given example input cases
    2. Look for similar problems in history and find their solutions, then modify those solutions to fit the new problems.
    3. Once a new problem is successfully solved, add its solution to the knowledge base.

vi.