Artificial Intelligence
Study Manual for the Comprehensive Oral Examination

Hendrik Fischer
Formerly of:
Artificial Intelligence Center
The University of Georgia
Athens, Georgia 30602
henne@uga.edu

Vineet Khosla
Formerly of:
Artificial Intelligence Center
The University of Georgia
Athens, Georgia 30602
vinkhosla@gmail.com

Brian A. Smith
Artificial Intelligence Center
The University of Georgia
Athens, Georgia 30602
brian.allen.smith@gmail.com

2003 - 2005

Contents

Abstract

Most of this guide is written by Hendrik except for few sections written by Vineet. You are free to use this guide and make changes to it. The LaTeX source file can be obtained by emailing either author. We are not going to charge anything but a few beers would be nice :) Best of Luck !

1 General Remarks

1. Be able to write short programs in PROLOG and LISP
2. Explain different programming concepts like *tail recursion*, recursion vs. iteration, working with *lists* (cdr/cad)
3. You won't have to formally prove any equations or formulas
4. Have examples ready for everything (e.g. a known toy problem to be solved by a PROLOG program)
5. Explain soundness and completeness and that kind of stuff about logic
6. What is Occam's Razor?

2 Symbolic Programming - CSCI/ARTI 6540

2.1 Basics PROLOG

Ques 1. What are the four terms of Prolog?
Atom = begin with lower case and includes all predicates.
Structures = atom (arg1, arg2, ... , argN).
Variables = begin with uppercase or underscore.
Numbers
Ques 2. What isNegation by Failure?
We cannot have true logical negation in Prolog because a Prolog is based on Horn Clause logic and in Horn clause we can have atmost one positive literal. Since we cannot state a negative fact, we instead try to get the positive fact to fail, hence having negation by failure.
Ques 3. What is the difference between a red cut and a green cut?
A green cut does not alter the output of the program but makes it more efficient. Red cut alters the possible output of the program.
Ques 4. What are the advantages of using Prolog over other languages?

Inbuilt inference engine.
Can allocate memory dynamically.
Can modify itself.

2.2 Algorithms in PROLOG

Subset of FOL - restricted to Horn Clauses
Inference by resolution theorem proving
Unification
Depth-First-Search
Backward-Chaining Approach

Resolution is semi-decidable for FOL:
KB = $\{P(f(x)) \rightarrow P(x)\}$ and let's say we're trying to prove $P(a)$. Resolution refutation assumes the negated goal - $\sim P(a)$ and from $\sim P(f(x)) \vee P(x)$ we can derive $P(f(a))$ by substituting a for x. But then we can go on and resolve again to get $P(f(f(a)))$ etc.
Does this contradict the claim that resolution refutation is complete? NO - because completeness states that any sentence Q that can be derived by resolution from a set of premises P must be entailed by P. If it is not entailed by P, then we might no be able to prove that!

Propositional calculus is decidable.

Horn Clause: a clause (disjunction of literals) with at most one positive literal (in Prolog: q :- $p_1, p_2, \ldots, p_n$ stands for the Horn Clause $q \vee \sim p_1 \vee \sim p_2 \vee \ldots \vee \sim p_n$

Prolog syntax is based on Horn clauses, which have the form of $p_1, p_2, \ldots, p_n \rightarrow q$. In Prolog, this is mapped to an "if-then"-clause:
q :- $p_1, p_2, \ldots, p_n$ (q, if p_1 and p_2 and ...)
This can be viewed as a procedure: (1) A goal literal matches (unifies) with q; (2) The tail of the clause $p_1 \ldots p_n$ is instantiated with the substitution of values for variables (unifiers) derived from this match; (3) The instantiated clauses serve as subgoals that invoke other procedures and so on.

Theorem-proving method of Prolog is resolution refutation (assume the negated hypothesis and try to resolve the empty clause) - if you succeed, then it's safe to assert the hypothesis, otherwise not. Resolution simply works on contradictory literals (e.g. not p,q,p stands for "if socrates is a man (p) then he is mortal (q)" in CNF (p implies q becomes not p or q);

now if we assume that socrates is indeed a man, then by resolution, this implies q).

If we have a goal q, then it might make sense to assert the negated goal for resolution refutation, which is essentially proof by reductio.

This is a form of backward reasoning with sub-goaling: we assert the negated goal and try to work backwards, unifying and resolving clauses until we get to the empty clause, which allows us to claim that the Theory implies our original hypothesis (by reductio).

```
Input = Query Q and a Logic Program P;
Output = "yes" if Q follows from P, "no" otherwise;

    Initialize current goal set to {Q}
    While(the current goal set is not empty) do

        Choose a G from the current goal set (first)
        Look for a suitable rule in the knowledge-base;
        unify any variable if necessary to match rule;
            G :- B1, B2, B3, ...
        If (no such rule exists) EXIT
        Else Replace G with B1, B2, B3...

    If (current goal set is empty)
        return NO.
    Else
        return YES. (plus all unifications of variables)
```

Cuts (!) in Prolog are NOT sound!

Prolog itself (even if not considering the "failure by negation" (\ +) and the cut is NOT complete.

Prolog uses Linear Input Resolution (resolve axiom and goal, then axiom and newly resolved subgoal, etc. + never resolve two axioms).

2.3 Algorithms in LISP

CDR yields rest of the list, CAR yields head of the list. Lisp programs are S-expressions containing S-expressions. So basically, a LISP program is a

linked list processing linked lists.

2.4 Factorial

```
factorial(1,1).
factorial(N,Result) :- NewN is N-1,
               factorial(NewN,NewResult),
               Result is N * NewResult.

tail recursion:

factorial(N,Result) :- fact_iter(N,Result,1,1).
fact_iter(N,Result,N,Result) :- !.
fact_iter(N,Result,I,Temp) :- I < N,
                              NewI is I+1,
                              NewTemp is Temp * NewI,
                              fact_iter(N,Result,NewI,NewTemp).

(defun factorial (n)
    (cond
        ((<= n 1) 1)
    (* n (factorial (- n 1)))))

(defun fact_tail (n)
    (fact_iter 1 1 n))

(defun fact_iter (result counter max)
    (cond
        ((> counter max) result)
        (T (fact_iter (* result counter) (+1 counter) max))))
```

2.5 Fibonacci

```
fib(1,1). fib(2,1).

fib(N,Result) :- N > 2,
         N1 is N-1,
         N2 is N-2,
         fib(N1,F1),
```

```
        fib(N2,F2),
        Result is F1 + F2.


tail:

fibo(N,Result) :- fib_iter(N,Result,1,1,1).

fib_iter(N,Result,N,_,Result) :- !.

fib_iter(N,Result,I,F1,F2) :- I < N,
                  NewI is I+1,
                  NewF1 is F2,
                  NewF2 is F1+F2,
                  fib_iter(N,Result,NewI,NewF1,NewF2).

(defun fib (n)
    (cond
        ((<= n 2) 1)
    (+ (fib (- n 1)) (fib (- n 2)))))

(defun fib_tail (n)
    (fib_iter 1 1 1 1 n))

(defun fib_iter (result counter f1 f2 max)
    (cond
        ((> counter max) result)
        (T (fib_iter (+ f1 f2) (+1 counter) f2 (+ f1 f2) max)))
```

2.6 Recursion

2.6.1 Standard Recursion

Program calls itself recursively from within the program. Uses a stack to store intermediate states that "wait" for a result from a lower level. The recursion goes down to a stopping condition, which returns a definite result that is passed on to the next higher level until we reach the top level again. The final result is computed along the way.

2.6.2 Tail Recursion

In tail recursion, the recursive call is the last call in the program and there are no backtrack points. This allows us to build up the result while we're going down in the recursion until we hit the bottom level; along the way, we compute the final result, such that it is available once we reach the stopping condition. "Smart" compilers recognize tail recursion and stop the program once we hit the bottom - but some compilers "bubble" the result up to the top level before returning it.

2.7 Other Examples

```
% FLATTEN
(defun flatten (thelist)
       (if
         ((null thelist) nil)
         ((atom thelist) (list thelist))
         (t (append (flatten (car thelist)))
         (flatten (cdr thelist)))))

% REVERSE
(defun myrev (thelist)
       (if
         ((null thelist) nil)
       (append (myrev (cdr thelist)) (list (car thelist))

%  reverse function...reverses a list...non tail recursive
my_reverse([ ],[ ]).
my_reverse([Head|Tail],Result) :-
                              my_reverse(Tail,RevTail),
                              append(RevTail,[Head],Result).

%  reverse function...reverses a list...tail recursive
fast_reverse(Orignal,Result):-
       nonvar(Orignal),
       fast_reverse_aux(Orignal,[ ],Result).

fast_reverse_aux([Head|Tail],Stack,Result) :-
          fast_reverse_aux(Tail,[Head|Stack],Result).
```

```
fast_reverse_aux([ ],Result,Result).

% DELETE ELEMENT (Prolog)
del(X,[],[]) :- !.
del(X,[X|Tail],Tail1) :- !, del(X,Tail,Tail1).
del(X,[Y|Tail],[Y|Tail1]) :- del(X,Tail,Tail1).

% FLATTEN (Prolog)
flat([],[]). flat([Head|Tail],Result) :- is_list(Head),
                            flat(Head,FlatHead),
                            flat(Tail,FlatTail),
                            append(FlatHead,FlatTail,Result).

flat([Head|Tail],[Head|FlatTail]) :- \+(is_list(Head)),
                                     flat(Tail,FlatTail).

% SET STUFF
% Union
union([],Set,Set).

union([Head|Tail],Set,Union) :- member(Head,Set),
                                union(Tail,Set,Union).

union([Head|Tail],Set,[Head|Union]) :- \+(member(Head,Set)),
                                       union(Tail,Set,Union).

% PERMUTATION
permut([],[]).
permut([L|H],R):- permut(H,R1),add(L,R1,R).

add(X,L,[X|L]).
add(X,[L|H],[L|R]):- add(X,H,R).

% POWER SET (the set of all possible subsets)
power_set([],[]).

power_set([Head|Tail],Result) :-
                  power_set(Tail,TailResult),
                  power_set_aux(Head,TailResult,[],TempResult),
                  append(TailResult,TempResult,Result).
```

```
power_set_aux(Head,[S|T],Stack,Result) :-
                    is_list(S),
                    append([Head],S,R),
                    power_set_aux(Head,T,[R|Stack],Result).

power_set_aux(Head,[S|T],Stack,Result) :-
                     \+ is_list(S),
                     append([Head],[S],R),
                     power_set_aux(Head,T,[R|Stack],Result).

power_set_aux(X,[],Stack,[X|Stack]).
```

3 Logic - CSCI(PHIL) 4550/6550, PHIL 6510

3.1 Definitions for Predicate Logic

An **atomic formula** is a sentence letter or an n-place predicate, e.g. **A** or **R**abx.
A **literal** is an atomic formula or the negation of an atomic formula.
A **clause** is a disjunction of literals.
A **unit clause** is a disjunction with one disjunct / the same as a literal.
A **negative clause** is a clause with no positive literals.
A **definite clause** is a clause with exactly one positive literal.
A **Horn clause** is a clause with at most one positive literal.
An **empty clause** is a disjunction with no disjuncts.
A **ground term** is a term that contains no variables, e.g. a constant or f((f(f(c)))).
A **ground literal** is a literal with no variables.
A **ground clause** is a clause with no variables / a disjunctions of ground literals.

3.1.1 Entailment $\models$

A set of sentences **A** semantically entails a sentence **B** iff in every model in which all sentences of **A** are true, **B** is also true or **A** $\models$ **B**. That is, every model of **A** is also a model of **B**.

A set of sentences **A** logically entails a sentence **B** means that "**B** can be

proven by **A**". That is, **B** can be derived from **A** by applying the according derivation steps (depends on the logic system used).

A set of sentences **A** truth-functionally entails a sentence **B** iff there is no truth-value assignment (TVA) on which every member of **A** is true and **B** is false.

Example: Wumpus world; Lets say that according to the KB, we know that there's a breeze in [2,1]. Hence possible models are: there is a pit in [1,3] and [2,2], or [2,2] only or [1,3] only. Then α_1 ="There is no pit in [1,2]" is entailed by the KB, because in every model in which KB is true, α is also true. If we consider α_2 ="There is no pit in [2,2]", then we can say that α_2 is *not* entailed by the KB, because in some models in which the KB is true, α_2 is false. Hence the agent cannot conclude that there is no pit in [2,2], nor can it conclude that there indeed IS a pit in [2,2]!

Entailment is only semi-decidable in FOL (i.e. we can show that sentences follow from premises if they do, but we cannot always show it if they do not).

To check an entailment claim, we have to check if for every model for p that it is also a model for q. This might be infeasible and that's why we use inference rules (derivations) to proof these claims (e.g. Modus Ponens, Resolution, etc.)

3.1.2 Model Checking

m is called a model for a sentence α if α is true on m. In propositional logic, a truth value assignment for a given sentence is an interpretation (similarly, in FOL, an interpretation is a definition of all terms, predicates etc.). Now, such an interpretation, or TVA, is a model for α iff α is true on that TVA.

Model Checking is a type of infernece algorithm which enumerate all possible models to check if α is true for all models in which the knowledge-base is true.

A model is generally a commitment to the truth of its components.

3.1.3 Decidability

Decidable
A set **S** is decidable iff there is an algorithm that determines for every input if the input is in the set or not.
Decidable = recursive set (roughly speaking)
In terms of turing machine a set **S** is recursive iff there is a Turing machine that for any input, halts and outputs 1 if the input is in **S** and halts and outputs 0 if the input is not in **S**. It can always tell you if something is a theorem and also if something is not a theorem.
A QUESTION is decidable, semi-decidable, or undecidable. A SET or a FUNCTION is recursive, recursively enumerable, or neither. If a SET S is recursive, then the QUESTION "Is x a member of S?" is decidable, etc.

Non-Decidable
A set **S** is non-decidable iff it is not decidable.

Semi-Decidable
A set **S** is semi-decidable iff there is an algorithm that determines for every input if the input is in the set, but can't determine if an input is not in the set.
Semi-Decidable=recursive enumerable. (roughly speaking)
In terms of Turing machine a set **S** is recursive enumerable iff there is a Turing machine that for any input halt and outputs 1 if the input is in **S** and runs forever if the input is not in **S**. It can always tell you if something is a theorem, but not that something is not a theorem.

???: *Is the set of semi-decidable problems a subset of the non-decidable problems?*

3.1.4 Unification

Unification takes two atomic sentences p and q and returns a substitution that would make p and q look the same or else fails. If we have "Knows(John, Jane)" in our KB and the sentence "Knows(John, x)", then we can unify both by substituting x with "Jane" in the second sentence, making them look alike. This is used for the **generalized modus ponens**. If we have the rule "Knows(John, x) $\Rightarrow$ Hates(John, x)", the above unification can be used to deduce that John hates Jane.

This is the main concept on which PROLOG is based on: the binding of uninstantiated variables with an atom or a term. The assignment with another uninstantiated variable (or even itself) is forbidden in modern PROLOG. This protection takes the form of an *occurs-check* (Otherwise A = f(A): A could be unified with f(f(f(...))), which is not desirable).

3.1.5 Equivalence, Validity and Satisfiability

Two sentences α and β are **logically equivalent** if they are true in the same set of models. In other words, if $\alpha \models \beta$ and $\beta \models \alpha$, then they are equivalent.

A sentence is **valid** if it is true in all models in which the premises (or knowledge base) is true.

A sentence that is true in *all* models is known as a **tautology**. For example, $P \vee \neg P$ is valid and a tautology.

A sentence that is not valid on *any* model is known as a contradiction. An example of such a sentence is $P \wedge \neg P$.

A sentence is **satisfiable** if it is true in some model. Validity and satisfiability are connected in the following manner: α is valid iff $\neg \alpha$ is unsatisfiable. $\alpha \models \beta$ iff the sentence $(\alpha \wedge \neg \beta)$ is unsatisfiable.
The above is called proof by **refutation** or proof by **contradiction**.

3.2 Inference Procedure

Patterns of inference that can be applied to derive chains of conclusions that lead to a desired goal.

3.2.1 Soundness or Truth Preservation

A derivation system is sound iff if $\Gamma \vdash P$, then $\Gamma \models P$, for all Γ, P.

Soundness is a property of an inference procedure and an inference procedure is sound iff derivations only produce entailed sentences. That is, a system is sound if all derived results are entailed by the knowledge base used in the derivation. If the system (claims to) prove something true, it really is true! That is, if the conclusion is true in all cases where the premises are true. This is sometimes called “truth-preserving”.

A logical argument is **sound** iff the argument is valid and all of its premises are true.

An unsound inference rule is abduction (a $\rightarrow$ b, and b, then a might be true, or not).

All men are mortal.
Socrates is a man
Hence Socrates is mortal.

This argument is valid and it is sound, since its premises are true. Whereas

All animals can fly.
Pigs are animals.
Therefore, pigs can fly.

is valid, but it is not sound, since the first premise is false.

An example of a sound, but incomplete derivation system would be one containing only &I and &E. All derivable sentences in such a system are logically entailed, but there are many logically entailed sentences which are underivable.

3.2.2 Completeness

A derivation system is complete iff if $\Gamma \models P$, then $\Gamma \vdash P$, for all Γ, P.

An inference procedure is complete iff derivations can produce all entailed sentences. That is, if something really is true, the system is capable of proving it.

An example of a complete but unsound derivation system would be SD with the addition of abduction as an inference procedure. Such a system can, indeed, derive all logically entailed sentences. But, it can also derive sentences which are not logically entailed.

???: *Can SD with the addition of abduction as an inference procedure derive all sentences?*

3.2.3 Types of Inference Procedures

Modus Ponens P $\supset$ Q and P, therefore Q.
Modus Ponens is sound.
Modus Ponens is incomplete.
Modus Ponens is complete for Horn KB's.

Modus Tollens P $\supset$ Q and $\neg$ Q, therefore $\neg$ P.
Modus Tollens is sound.

And-Elimination P & Q, therefore P.
And-Elimination is sound.
And-Elimination is incomplete.

Resolution Resolution is sound.
Resolution is complete.
Resolution works as a proof by reductio, based on a set of sentences in conjunctive normal form (CNF). If we got two sentences $A \vee B$ and $\sim A \vee C$ in CNF, then we can use resolution to get $B \vee C$. This also works with substituting ground terms for variables before the actual resolution. Proof by resolution refutation is assuming the negation of a hypothesis and then showing that we can resolve to the empty clause in some way (which essentially depicts a proof by reductio, or contradiction).
Resolution is **refutation complete** means if the set of sentences is unsatisfiable, then you will always be able to derive a contradiction with resolution.

3.3 Propositional or Boolean Logic

Ontological commitment: there exist only "facts" in the world.
Epistemological commitment: true, false and unknown.

Pros of Propositional Logic:
(1) declarative - pieces of syntax correspond to facts.
(2) compositional - the meaning of A $\vee$ B is derived from the meaning of its constituents A and B.
(3) context-independent.

Cons of Propositional Logic:
(1) lacks the expressive power of higher logical languages like FOL (predicate logic).

Propositional Logic consists of atomic sentences (A, B, C ...) and a set of logical connectives (not, and, or, conditional, biconditional). Each atom can be assigned a truth value (TVA). The truth value of the main connective determines the "meaning" of the sentence (i.e. its truth value).

3.4 First-Order Logic

Ontological commitment: there exist facts, objects and relations between those objects
Epistemological commitment: true, false and unknown.

FOL consists of variables (x, y, z ... = variable terms), constants (a, b, c ... = constant terms), first-order predicates (P, Q, R ... = define a relation among objects of the universe of discourse), functions (legOf, fatherOf, +, *, ... = take an object as input and return an object) and logical connectives (same as propositional logic + $\forall$ and $\exists$ quantifiers).

Universal Instantiation: every instantiation of a universally quantified sentence is entailed by it

Existential Instantiation: we can substitute a new constant (that does not appear anywhere else in the KB) for the existentially quantified variable of a sentence. This is called a *Skolem* constant. The new sentence is entailed by the old one.

If we instantiate all sentences in all possible ways, we have reduced the FOL problem to propositional logic and can use its tools for inferences.

FOL is **semi-decidable**: it stops if a formula follows, but could run forever without ever stopping if it does not.
First-order logic is **sound**: means if K yields/derives P, then K entails P ($K \vdash P$, then $K \models P$).
Basically it means that it derives only entailed sentences.
First-order logic is **complete**: If K entails P, then K yields/derives P ($K \models P$, then $K \vdash P$).
Basically it means that it can derive any sentence that is entailed.

3.5 Defeasible Logic

Defeasible logic is a subset of non-monotonic logic.

Non-monotonic logic - meaning: in monotonic logic, the line of reasoning usually is piling up propositions; but in non-monotonic logic, the propositions are defeasible: the number of valid propositions can decrease if justification for one of them is retracted. This line of reasoning resembles the scientific method (hypothesis, gather evidence, modify hypothesis according to new findings).
Nonmonotonic reasoning: In Nonmonotonic reasoning we can reject an earlier conclusion on the basis of new information, even if the old conclusion was justified by the evidence we had at that time. This reasoning system lets us draw likely conclusions with less than conclusive evidence. Human reasoning is not monotonic. In the normal course of human reasoning we often arrive at conclusions which we later retract when additional evidence becomes available. The additional evidence defeats our earlier reasoning and much of our reasoning is nonmonotonic and Defeasible in this way

???: *Barring the retraction of rules, is it true that the set of entailed sentences can only increase asinformation is added to the knowledge base?*

3.5.1 Components

Undercutting defeaters : Undercutting defeaters are too weak to support an inference on their own and their sole purpose is to call into question an inference we might otherwise be inclined to make. It is important to note here that they don't suggest the opposite conclusion or for that matter any conclusion at all. They stop you from making a wrong conclusion. Generally these rules can be identified by the word 'might'. For example, "A damp match might not burn."
Strict rules : These are the rules which can never be defeated or undercut. They don't have exceptions and represent necessary truth conditions.
An example would be "All crows are black" or "Everyone born in Atlanta was born in Georgia".
Defeasible rules : These are the weaker rules which can be defeated or undercut. For example "Birds fly" or "Penguins live in Antarctica."
Defeasible rules can be defeated or rebutted by a strict rule or another defeasible rule which supports the opposite inference.
Defeasible rules can be undercut by undercutting defeaters which point out

a situation in which the rules they defeat might not apply.

???: *Can defeasible rules undercut one another, or is there a definite hierarchy?*

3.5.2 Rule conflict

Strict Rule vs Defeasible Rule : The strict rule is always superior to the defeasible rule, even if the condition of the strict rule is defeasibly derivable while the condition of the defeasible rule is strictly derivable.
Defeasible Rule vs Defeasible Rule : When two defeasible rules reach a contradicting conclusion, we need a way to break that loop. We can either declare competing rules as "incompatible" or declare one rule "superior" to other.
Undercutting defeaters : Undercutting defeaters are used to defeat defeasible rules. Undercutting defeaters are small pieces of knowledge, which stop us from making conclusions we normally would have made.

3.6 Default Logic

Behaves like standard logic if complete information is available, but lets us also infer things if information is not complete. For example, in standard logic, if we know that something is a bird, we don't infer that it can fly, because there are birds that don't fly. In default logic, we would infer that it flies, until we get evidence that it might be otherwise.

If our initial premises are:

Bird(Condor), Bird(Penguin), ∼Flies(Penguin), Flies(Airplane) and our default rule is

$W = \left\{ \frac{Bird(X):Flies(X)}{Flies(X)} \right\}$

then we can safely infer that a condor flies, because condor is a bird and we don't have evidence that condors can't fly. However, we cannot infer that penguins fly, because it is inconsistent with our knowledge.

3.7 Sample derivations

3.7.1 SD derivation

Given assumptions 1-3 below, derive H ⊃ J.

1	(H & T) ⊃ J	Assp
2	(M ⊃ D) & (¬D ⊃ M)	Assp
3	¬T ≡ (¬D & M)	Assp
4	| H	Assp
5	| | ¬J	Assp
6	| | | ¬D	Assp
7	| | | ¬D ⊃ M	2&E
8	| | | M	6,7⊃E
9	| | | M ⊃ D	2&E
10	| | | D	8,9⊃E
11	| | D	6-10¬E
12	| | | T	Assp
13	| | | H & T	4,12&I
14	| | | J	1,13⊃E
15	| | | ¬J	5Reit
16	| | ¬T	12-15¬I
17	| | ¬D & M	3,16≡E
18	| | ¬D	17&E
19	| J	5-18¬E
20	H ⊃ J	4-19⊃I

3.7.2 PD derivation

Derive (∀x)(Bi ⊃ Ax) ≡ Bi ⊃ (∀x)Ax.

1	| (∀x)(Bi ⊃ Ax)	Assp
2	| | Bi	Assp
3	| | Bi ⊃ Ac	1∀E
4	| | Ac	2,3⊃E
5	| | (∀x)Ax	4∀I
6	| Bi ⊃ (∀x)Ax	2-5⊃I
7	| Bi ⊃ (∀x)Ax	Assp
8	| | Bi	Assp
9	| | (∀x)Ax	7,8⊃E
10	| | Ac	9∀E
11	| Bi ⊃ Ac	8-10⊃I
12	| (∀x)(Bi ⊃ Ax)	11∀I

13 | (∀x)(Bi ⊃ Ax) ≡ Bi ⊃ (∀x)Ax 1-6,7-12≡I

4 Artificial Intelligence - CSCI(PHIL) 4550/6550

4.1 Overview

Except for the search part, almost everything else covered in this class is covered in detail in a specialized class e.g. "expert systems", "genetic algorithms","neural networks" etc.

4.1.1 What is AI?

Artificial Intelligence is the field that strives to *manufacture* machines that exhibit intelligence. This leads to two questions - what is artificial, i.e. what can be built, and what is intelligence or intelligent behavior.

There are numerous definitions of what artificial intelligence is.
We end up with four possible goals:

1. Systems that think like humans: Cognitive modeling
   (focus on reasoning and human framework)
2. Systems that think rationally: Laws of thought
   (focus on reasoning and a general concept of intelligence)
3. Systems that act like humans: Turing test
   (focus on behavior and human framework)
4. Systems that act rationally: Rational agent
   (focus on behavior and a general concept of intelligence)

What is **rationality**? An *ideal* concept of intelligence, "doing the right thing" simply speaking.

Definition:
"The art of creating machines that perform functions that require intelligence when performed by humans" (Kurzweil)

(1) Involves cognitive modeling - we have to determine how humans think in a literal sense (explain the inner workings of the human mind, which requires experimental inspection or psychological testing) - Newell and Simon: GPS (General Problem Solver).

(2) Deals with "right thinking" and dives into the field of logic. Uses logic to represent the world and relationships between objects in it and come to conclusions about it. Problems: hard to encode informal knowledge into a formal logic system + theorem provers have limitations (if there's no solution to a given logical notation).
(3) Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool a human person (=**Turing Test**). Avoided physical contact to the machine, because physical appearance is not relevant to exhibit intelligence. The "total Turing Test" includes this by encompassing visual input and robotics as well.
(4) The rational agent - achieving one's goals given one's beliefs. Instead of focusing on humans, this approach is more general, focusing on agents (which perceive and act). More general than strict logical approach (=thinking rationally).

4.1.2 Strong AI vs. Weak AI

Strong AI = sentience : machine-based artificial intelligence that can truly reason and becomes self-aware (sentient), either human-like (thinks and reasons like a human mind) or non-human-like (different form of sentience and reasoning) - coined by John Searle: "an appropriately programmed computer IS a mind" (see Chinese Room).

Weak AI = problem-tailored : machine-based artificial intelligence that can reason and solve problems in a limited domain. Hence it acts as if it were intelligent in that domain, but would not be truly intelligent or sentient.

4.1.3 Strong methods v. Weak methods

Strong methods : Makes use of deep knowledge regarding the environment and possible situations that may be faced. Strong methods are usually fine-tuned to the problem at hand and make use of a problem model.

Weak methods : Makes use of general problem solving structures such as logic and automated reasoning. Weak methods are not usually fine-tuned to a specific problem but are applicable to a broad range of different problem classes.

4.1.4 Turing Test

A human judge engages in a natural language conversation with two other parties, one a human and the other a machine; if the judge cannot reliably

tell which is which, then the machine is said to pass the test. It is assumed that both the human and the machine try to appear human.

Objections:
(1) A machine passing the test could simulate human behavior, but this might be considered much weaker than intelligence - the machine might just follow a set of cleverly designed rules. Turing rebutted this by saying that maybe the human mind is just following a set of cleverly designed rules also.
(2) A machine could be intelligent without being capable of conducting conversations with humans
(3) Many humans might fail this test, if they're illiterate or inexperienced (e.g. children).

???: *How applicable is the third objection?*

4.1.5 Chinese Room Experiment

Tried to debunk the claims of "**Strong AI**". Searle opposed the view that the human mind IS a computer and that consequently, any appropriately construed computer program could also be a mind.

Outline: Man who doesn't speak Chinese sits in a room and is handed sheets of paper with Chinese symbols through a slit. He has access to a manual with a comprehensive set of rules on how to respond to this input. The output is also in Chinese and to an observer on the outside of the room, the room appears to give perfectly reasonable responses to the inputs. Searle argues, that even though he can answer correctly, he doesn't understand Chinese; he's just following **syntactical** rules and doesn't have access to the meaning of them. Searle believes that "mind" emerges from brain functions, but is more than a "computer program" - it requires intentionality, consciousness, subjectivity and mental causation.

One objection to Searle is the *systems response*. Of course the man doesn't know Chinese and neither does the room, but the man-room *system* clearly knows Chinese. This response points out the serious flaw in Searle's argument: he appears to require a single, irreducible part of the system that is responsible for intentionality. He refuses to allow for the possibility that intentionality emerges from the interaction of several parts.

4.1.6 Mind-Body Problem

The problem deals with the relationship between **mental** and **physical events**. Suppose I decide to stand up. My decision could be seen as a mental event. Now something happens in my brain (neurons firing, chemicals flowing, you name it) which could be seen as a physical, measurable event. How can it be, that something "mental" causes something "physical". So it's hard to claim that both are completely different things. One take on this is to say that these mental events are, in fact, physical events. My decision IS neurons firing, but I'm not aware of this - I feel like I made a decision independently from the physical. Of course this works the other way round to: everything physical could, in fact, be mental events. When I stand up after having made the decision (mental event), I'm not physically standing up, but my actions cause mental events in my and the bystanders minds - physical reality is an illusion.

There are two extremes of ontological commitment in relation to this problem. *Material reductionism* holds that all mental activity is reducible to interactions taking place in an objective, physical reality. *Radical idealism* holds instead that physical reality is instead entirely reducible to mental activity in some consciousness. Descartes is famous for his treatment of the issue and his is sometimes known as *Cartesian duality*. It holds that mind (or spirit) is non-physical while the body is physical. Just how the mind and body could interact under such an interpretation was a mystery even to Descartes, though he did suggest that the pineal gland in the brain might somehow be responsible.

4.1.7 Frame Problem

It is the question of how to determine which things remain the same in a changing environment.

4.1.8 Problem-Solving

Occurs whenever an organism or artificial intelligence is at some current state and does not know how to proceed in order to reach a desired **goal** state. This is considered to be a problem that can be solved by coming up with a series of actions that lead to the goal state (the "solving").

4.2 Search

4.2.1 Overview

In general, search is an algorithm that takes a problem as input and returns with a solution from the search space. The **search space** is the set of all possible solutions. We dealt a lot with so called "state space search" where the problem is to find a goal state or a path from some initial state to a goal state in the state space. A state space is a collection of states, arcs between them and a non-empty set of start states and goal states. It is helpful to think of the search as building up a search tree - from any given node (state): what are my options to go next (towards the goal).

Complete search : Search is complete if it is guaranteed to find a solution if a solution exists.
Optimal search : A search method is optimal if it finds the cheapest solution.
Complexity of search: It is of two types:

1. *Time :* How long does it take to find a solution.
2. *Space :* How much memory is used to find a solution.

Name	**Complete**	**Optimal**	**Time**	**Space**
Depth First	Yes*	No	$O(b)^m$	$O(b*m)$
Breadth First	Yes	Yes**	$O(b)^d$	$O(b^d)$
Iterative Deepening	Yes	Yes	$O(b)^d$	$O(b*d)$
Greedy Search	No	No	$O(b)^m$	$O(b)^m$
A-Star	Yes***	Yes***		
Uniform Cost				

d = depth
m = max depth
* if search tree is finite
** only when cost is uniform
*** if heuristic is admisssible, i.e it never overestimates

4.2.2 Uninformed Search

Uninformed search (blind search) has no information about the number of steps or the path costs from the current state to the goal. They can only distinguish a goal state from a non-goal state. There is **no bias** to go

towards the desired goal.

1. Breadth-First-Search:
Starts with the root node and explores all neighboring nodes and repeats this for every one of those (expanding the "depth" of the search tree by one in each iteration). This is realized in a *FIFO queue*. Thus, it does an exhaustive search until it finds a goal. **BFS** is complete (it finds the goal if one exists and the branching factor is finite). **BFS** is optimal (if it finds the node, it will be the shallowest in the search tree). Space: $O(b^d)$. Time: $O(b^d)$.

Open list: nodes yet to be explored, closed list: nodes that have been explored.

2. Depth-First-Search:
Explores one path to the deepest level and then backtracks until it finds a goal state. This is realized in a *LIFO queue, or stack*. **DFS** is complete (if the search tree is finite). **DFS** is *not* optimal (it stops at the first goal state it finds, no matter if there is another goal state that is shallower than that). Space: $O(bm)$ (much lower than **BFS**). Time: $O(b^m)$ (Higher than **BFS** if there is a solution on a level smaller than the maximum depth of the tree). Danger of running out of memory or running indefinitely for infinite trees.

3. Depth-Limited / Iterative Deepening:
To avoid that, the search depth for **DFS** can be either limited to a constant value, *depth-limited* search. DLS is not complete, as the goal state may be deeper than the search depth. *Iterative deepening* Repeatedly applies depth-limited search, each time with an increased search depth. This is repeated until finding a goal. Iterative deepening combines the advantages of **DFS** and **BFS**. **ID** is complete. **ID** is optimal (the shallowest goal state will be found first, since the level is increased by one every iteration). Space: $O(b \cdot d)$ (better than **DFS**, d is depth of shallowest goal state, instead of m, maximum depth of the whole tree). Time: $O(b^d)$.

4. Bi-Directional Search:
Searches the tree both forward from the initial state and backward from the goal and stops when they meet somewhere in the middle. Di-directional search requires a *predecessor function* in addition to the successor function required by all uninformed search methods. It is complete and optimal.

Space: $O(b^{\frac{d}{2}})$. Time: $O(b^{\frac{d}{2}})$.

5. Graph Search:
Graph search avoids making repeated steps and is therefor useful when the search tree is a true tree, but rather a cyclic graph. Graph search maintains a *closed list* of expanded nodes and an *open list* of unexpanded nodes on the fringe. Graph search can be applied to any of the tree search algorithms previously discussed.

4.2.3 Informed (Heuristic) Search

In **informed** search, a **heuristic** is used as a guide that leads to better overall performance in getting to the goal state. Instead of exploring the search tree blindly, one node at a time, the nodes that we could go to are ordered according to some **evaluation function** $h(n)$ that determines which node is probably the "best" to go to next. This node is then expanded and the process is repeated ("**Best First Search**"). **A*** is a form of **BestFS**. In order to direct the search towards the goal, the evaluation function must include some estimate of the cost to reach the closest goal state from a given state. This can be based on knowledge about the problem domain and the description of the current node, the search cost up to the current node. **BestFS** optimizes **DFS** by expanding the most promising node first. Efficient selection of the current best candidate is realized by a *priority queue*.

1. Greedy Search:
Minimizes the estimated cost to reach the goal. The node that is closest to the goal according to $h(n)$ is always expanded first. It optimizes the search locally, but not always finds the global optimum. It is not complete (can go down an infinite branch of the tree), it is not optimal. Space: $O(b^m)$ for the worst case, same for time, but can be reduced by choosing a good heuristic function.

2. A* Search:
Combines **uniform cost search** (expand node on path with lowest cost so far) and **greedy search**. Evaluation function is $f(n) = g(n) + h(n)$ (or estimated cost of the cheapest solution through node n).
$g(n)$ = Cost to reach the node.
$h(n)$ = Cost to get from node to goal.
It can be proven that A* is complete and optimal if h is "**admissible**" - that is, if it *never overestimates* the cost to reach the goal. This is optimistic,

since they think the cost of solving the problem is less than it actually is. Examples for h: Path-Finding in a map: Straight-Line-Distance. 8-Puzzle: Manhattan-Distance to Goal State. Everything works, it just has to be admissible (e.g. $h(n) = 0$ always works, but transforms A* back to uniform cost search). If a **heuristic function** h_1 always estimates the actual distance to the goal better than another **heuristic function** h_2, then h_1 *dominates* h_2.
A* maintains an open list (**priority queue**) and a closed list (visited nodes). If a node is expanded that's already in the closed list, stored with a lower cost, the new node is ignored. If it was stored with a higher cost, it is deleted from the closed list and the new node is processed.

h is **monotonic**, if $h(n) - h(n') \leq g(n') - g(n)$, so that the difference between the heuristics of any two connected nodes does not overestimate the actual distance between those nodes. If h is monotonic and there is only one goal state, then h must be admissable. Example of a **non-monotonic** heuristic: n and n' are two connected nodes, where n is the parent of n'. Suppose $g(n) = 3$ and $h(n) = 4$, then we know that the true cost of a solution path through n is at least 7. Now suppose that $g(n') = 4$ and $h(n') = 2$. Hence, $f(n') = 6$. First off, the difference in the heuristics (2) overestimates the actual cost between those nodes (1). However, we know that any path through n' is also a path through n and we already know that any path through n has a true cost of at least 7. Thus, the f-value of 6 for n' is meaningless and we will consider its parent's f-value.

h is consistent, if the h-value for a given node is less or equal than the actual cost from this node to the next node plus the h-value from this next node (triangular inequality).

If h is **admissible** and **monotonic**, the first solution found by A* is guaranteed to be the optimal solution (open/close list bookkeeping is no longer needed).

Finding Heuristics: (a) Relax the problem (e.g. 8-puzzle: Manhattan distance). (b) Calculate cost of subproblems (e.g. 8-puzzle: calculate cost of first 3 tiles).

3. Uniform Cost Search: Use only the "g" value. Is optimal.

4.2.4 Adversarial Search (Game Playing)

Adversarial search makes use of a competitive environment with multiple agents that have conflicting goals. Minimax is for deterministic games with perfect information. Minimax is complete if the game tree is finite and optimal against an optimal opponent. Non-Deterministic games will use the **expectiminimax** algorithm.

1. Minimax Algorithm:
A two player **zero-sum game**, in which bothplayers make a move alternately, is defined by an initial state (e.g. board positions), a set of operators that define legal moves, a terminal test that defines the end of the game and a **utility function** that determines a numeric value for the outcome of the game (e.g. "1" for win, "-1" for loss).

If it was a search problem, one would simply search for a path that leads to a terminal state with value "1". But the opponent (Min) will try to minimize one's (Max) outcome. The **minimax** algorithm generates the whole **game tree** and applies the **utility function** to each terminal state. Then it propagates the utility up one level and continues to do so until reaching the start node. Propagation works like this: Min is assumed to always chose the option that is the worst for Max (minimum utility value). If we have three terminal states in one branch with 1, 2 and 3 as their utility values, then Min would chose 1. Hence, "1" is propagated to the next level and so forth. Max can then decide upon his opening move (the "minimax decision" = maximizing utility under the assumption that the opponent will play perfectly to minimize it).

For games that are too complex to compute the whole game tree, the game tree is cut off at some point and the utility value is estimated by a heuristic evaluation function (e.g. "material value" in chess).

2. Alpha-Beta Pruning:
We don't have to look at each node of the game tree: Minimax with α β pruning yields the same result as a complete search, but with much greater efficiency (reduces its effective branching factor to its square root). Here we store the best value for Max' play found so far in α and the best value for Min's play found so far (i.e. the lowest value) in β. If, at any given point, α becomes smaller than β, we can prune the game tree at this point. Therefore, it is required to evaluate at least on set of leaf nodes branching

off of one of the deepest Min nodes and then use the acquired alpha-value to prune the rest of the search.

3. Non-determinism:
Introduces "chance nodes" to the decision points, based on what kind of chance is introduced (e.g. a 50/50 chance for flipping a coin), which yield the "expected value" (average - add up utility values weighted by the chance to achieve them).

???: *Is expected value always the best way to go? What about cases where there will not be multiple trial?*

4.2.5 Constraint Satisfaction Problems

For CSP's, states in the search space are defined by the values of a set of variables, which can get assigned a value from a specific domain, and the goal test is a set of **constraints** that the variables must obey in order to make up a valid solution to the initial problem. Example: 8-queens problem; variables could be the positions of each of the eight queens, the constraint to pass the goal test is that no two queens can be such that they harm each other.

Different types of constraints: (a) **unary** (value of a single variable), (b) **binary** (pairs of variables, e.g. x ¿ y), (c) **n-ary** constraints. In addition to that, constraints can be absolute or preference in nature (the former rules out certain solutions, the latter just says which solutions are preferred). The domain can be discrete or continuous. In each step of the search, it is checked if any variable has violated one of the constraints - if yes: **backtrack** and rule out this path.

Forward checking looks one step ahead, removing some of the possible assignments from variables linked by constrainst. It checks if any decisions on yet unassigned variables would be ruled out by assigning a value to a variable. It deletes any conflicting values from the set of possible values for each of the unassigned variables. If one of the sets becomes empty, then backtrack immediately. **Constraint propogation** occurs after forward checking. It applies constraints "outward" from each of the variables altered during forward checking to the variables they are linked to by some constraint.

There are also heuristics for making decisions on variable assignments.

Selecting the next variable:
most-constrained-variable works together with forward checking (checking which values are still allowed for each unassigned variable): the next variable to be assigned a value is the one with the *fewest* possible values (reduces branching factor).
Most-constraining-variable assigns a value to the variable that is involved in the *largest* number of constraints on other yet unassigned variables.
After selecting the variable:
least-constraining-value assigns a value that rules out the smallest number of values in variables connected to the current variable through constraints.

CSP's that work with iterative improvement are often called "**heuristic repair**" methods, because they repair inconsistencies in the current configuration. **Tree-structured CSP's** can be solved in linear time.

4.2.6 Other

1. Hill Climbing (Gredy local search):
Evaluate all neighboring nodes and move to the best one. If there is no better node than the current node, declare this node to be the optimum and halt. Will get stuck at local maxima. Plateauxs in the search space allow for the possibility of loops. Selecting the successor node randomly when plateauxs are encountered may allaw the algorithm to break out of such a loop.

2. Simulated Annealing:
Escape local maxima by allowing "bad" moves (descent), but gradually decrease their size and frequency (read Genetic Algorithm section for more detail).

4.3 Logic

4.3.1 Logical Agents

Usually in a knowledge-based approach agents are seen as "knowing" about the environment and being able to "reason" about possible course of actions. This combines the "perceive-act" cycle of agents with logical deduction as their means of inference. Inference is the process of producing new knowledge based on available knowledge (knowledge base + percept/input).

Validity and Satisfiability:
A sentence is valid (necessarily true) iff it is true under all possible interpretations in all possible worlds (e.g. $A \vee \sim A$, a tautology).
A sentence is satisfiable iff there is *some* interpretation in *some* world for which it is true (e.g. $A \wedge B$ might be true, but is not valid).
A sentence is **unsatisfiable or false** if it is not satisfiable (e.g. $A \wedge \sim A$, a contradiction).

"Logic" in general, is a formal language for representing information such that conclusions can be drawn. The syntax defines the way sentences are set up in this language. The semantics define the "truth" of a sentence in a world (meaning).

4.3.2 Wumpus World

Observable - no (only local perception)
Deterministic - yes (outcomes are specified)
Episodic - no (sequential at the level of actions)
Static - yes (Wumpus and pits do not move)
Discrete - yes
Single-Agent - yes (Wumpus is not an agent).

4.3.3 Model Checking

m is called a model for a sentence α if α is true on m. A model is generally a commitment to the truth of its components. In propositional logic, a truth value assignment for a given sentence is an interpretation (similarly, in FOL, an interpretation is a definition of all terms, predicates etc.). Now, such an interpretation, or TVA, is a model for α iff α is true on that TVA.

Model Checking
Take the current knowledge base and enumerate all possible models that could lead to the KB. Check if α is true for all models in which the knowledge-base is true.

4.3.4 Horn Clauses

A Horn clause is a disjuntion of literals, of which at most one is positive. A Horn clause can be written as an implication where the premise is a conjunction of positive literals and the conclusion is a single positive literal.

Horn clauses allow inference through forward or backward chaining. Given a knowledge base of Horn clauses, entailment is decidable in linear time relative to the size of the KB.

4.3.5 Knowledge-Base

A logical knowledge-base is a conjunction of Horn clauses. A Horn clause consists of either a proposition symbol or an implication like $(A_1 \wedge A_2 \wedge \ldots A_n) \rightarrow B$. Logical agents apply inferences to the knowledge-base to come to new knowledge about what course of action to take.

4.3.6 Forward- and Backward-Chaining

Forward-Chaining is based on the idea to add the conclusions of any implications whose premises are satisfied by the available facts to the knowledge base. The new facts will probably be part of other implications whose premises can be satisfied and so on. This is a data-driven approach which might lead to redundant work.

Backward-Chaining works backwards from a query: it checks all implications which conclude the query q and tries to find facts that satisfy the premises of these implications or looks for other implications that conclude those premises. This is a goal-driven approach, appropriate for problem-solving.

Forward- and Backward-Chaining are both sound and complete (BC is incomplete if we don't check for infinite loops). May not terminate if α is not entailed (semi-decidable)

4.3.7 Resolution

1. *CNF* Convert all premises in the knowledge base into *conjunctive normal form*, a conjunction of disjuncions of literals.
2. *Break apart* Split the KB into its component axioms on the $\wedge$ symbol.
3. *Negate goal* Add the negation of the goal to the list of axioms.
4. *Combination* Take two axioms that contain contradictory literals such as $A \vee B$ and $\sim B \vee C$ and combine them, removing the contradictory terms the get $A \vee C$, a new axiom. Repeat this step, always aiming to eliminate the number of literals.

5. *Empty set* Resolution is complete when an empty set is derived. This implies that the goal is entailed by the knowledge base.

4.4 Intelligent Agents

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives". It has sensors, with which it percepts its environment and effectors to manipulate the environment in order to proceed towards a desired goal or objective.

Ideal rational agent: for each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence (history) and whatever built-in knowledge the agent has.

4.4.1 Simple Reflex Agents

Consists of simple "condition-action rules": the sensors tell the agent what the environment looks like at a given time and the rules will control the agents behavior and take some action based on what rule was triggered by the conditions that match the current perceived situation.

4.4.2 Model-Based Agents

Works with a knowledge-base and is grounded in some system (e.g. propositional logic or predicate logic). It still perceives its environment and acts according to the appropriate rule derived by the inference rules given by the logic system. It can make conclusions about its environment and the best action to take in the current state by reasoning on the currently known facts, the percept and the model incorporated in it (Wumpus world). If it draws a conclusion, it is guaranteed to be correct, given the available information and the underlying model are correct.

4.4.3 Goal-Based Agents

In simple reflex agents, the programmer has to determine the right action that should be taken in a certain situation. Goal-based agents involve consideration of the future: information about the ultimate goal and how to get there. This includes information about the outcome of the actions and their evaluation in respect to proceeding towards the goal. This makes it a lot

more flexible (e.g. a car driving agent could be given a new destination as a goal and it would adjust its turn-actions accordingly - whereas we would have to rewrite a great deal of rules of the reflex agents to account for the changed sequence of turns).

4.4.4 Utility-Based Agents

Just having a goal is not enough to generate high-quality behavior (e.g. there are many ways to get to a destination). Utility-based agents add to that by assigning a utility value to the possible actions, thus choosing the (hopefully) optimal action out of the set of all possible actions.

Performance Function : Used by an outside observer to evaluate the performance of an agent in an environment.

Utility function : Maps the world state to a real value allowing for the comparison of distinct world states. Used by an agent to better its own performance.

4.4.5 Learning Agents

The learning agent learns from experience with respect to its assigned task and some performance measure. It strives to increase its performance over time, with increasing experience.

4.4.6 Ideal Rational Agents

A rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by percept sequence and whatever built in knowledge the agent has in it.

4.4.7 Environments

1. Accessible vs. Inaccessible:
Sensors give access to complete state of the environment (hence no need to keep track of the world).

2. Deterministic vs. Non-Deterministic:
If the next state of the environment is completely determined by the current state and the actions selected by the agent, then the environment is deterministic.

3. Episodic vs. Non-Episodic:
In episodic environments, the agent's experience is divided into "episodes". Each episode is a perceive-act cycle. Evaluation of behavior is limited to the current episode - no real look-ahead.

4. Static vs. Dynamic:
The environment is dynamic, if it can change while the agent is working on figuring out the next step.

5. Discrete vs. Continuous:
If there are a limited number of distinct, clearly defined percepts and actions, we say that the environment is discrete.

4.5 Planning

4.5.1 Overview

Planning is the task of coming up with a sequence of actions that will achieve a goal. Search-based problem-solving and logical planning agents involve planning. A classical planning environment is deterministic, finite, fully observable, static (it changes only through actions) and discrete.

Closed-world assumption = any conditions that are not mentioned are assumed false.

4.5.2 Search vs. Planning

Search-based problem-solvers are overwhelmed by irrelevant actions. They will search for the appropriate action - which can be a problem if the search space is large. Planning agents have the necessary knowledge to directly decide what action has to be taken to achieve a goal.

Planning agents can treat a goal as a conjunction of subgoals and so forth - coming up with a general heuristic is therefore easy. Search agents usually require a much more complex heuristic that is problem-dependent.

Planning agents can decompose the problem into a set of subproblems, which a search-based agent cannot do.

4.5.3 Planning as Search

Forward-Search:
Progression planning. Starts with the initial state and performs a search for a path through the search tree to the goal node. Considers irrelevant actions and is inefficient without a good heuristic.

Backward-Search:
Regression planning. Easy to implement in STRIPS, since the goal state consists of a condition that has to be satisfied (by the effects of some action). Doesn't necessarily consider irrelevant actions - efficient! In addition to requiring certain actions to satisfy certain (pre-)conditions of further actions or the goal state, we also require them to not undo any desired literals. An action that satisfies this restriction is called **consistent**.

Both backward and forward search are called total-order planning - they explore strictly linear sequences of actions that lead from the initial state to the goal state and thus cannot take advantage of problem decomposition.

4.5.4 Partial-Order Planning

POP starts with an empty plan, only containing the start state and the finish state (start = set of literals as effects, no preconditions; finish = no effects, set of literals as preconditions). We then work out subplans for subgoals and put them into the plan, establishing a certain order based on how the actions of each subplan effect the state of the world - partial order: for some subplans it wouldn't matter which one comes first. POP can be **linearized** into total-order plans by fixing the actual ordering. Delaying that kind of choice is the **least-commitment strategy**

A plan consists of a set of "actions" (steps from Start to Finish), ordering constraints (e.g. $A \prec B$ = A before B - cycles are usually forbidden), a set of causal links (e.g. $A \rightarrow_p B$ = A achieves p for B, which essentially protects "p" from being negated until action B was executed) and finally a set of *open* preconditions (from which the next "to do" item is arbitrarily chosen).

A solution is defined by a plan that has no remaining open preconditions and no conflicts with any causal links and no cycles. Conflicts are resolved by demoting/promoting the conflicting action to somewhere outside the pro-

tection interval of another set of actions (e.g. if C conflicts with $A \rightarrow_p B$, then the ordering constraint added is either $B \prec C$ or $C \prec A$).

Planning is more of a plan refinement strategy than search among states.

Heuristics:
Could be based on number of open preconditions after taking an action, or the **most-constrained-variable** heuristic known from CSP's (take the open condition that can be satisfied in the *fewest* possible ways). All in all, coming up with a heuristic for partial-order planning is hard, because there is no real measure of how far away we are from the goal state, since we're dealing with plans, not with real actions in the world.

4.5.5 STRIPS

Russel and Norvig, p. 379 presents a figure which summarizes STRIPS well.
Actions are defined by a signature (e.g."Fly(P,A,B)" = fly plane P from A to B), a conjunction of preconditions (e.g. "At(P,A)") and a conjunction of effects (e.g. "At(P,B)"). Both the preconditions and the effects must consist of function free positive literals.

Closed-world assumption

Goals are conjunctions of ground literals.

No equality function, no types.

ADL - Action Description Language adds equality test, typing, goals can be mixed conjunctions/disjunctions and be quantified, Plus: open world assumption, allows negative literals in states.

5 Expert Systems - CSCI 8050

5.1 Overview

5.1.1 What Are Expert Systems?

An Expert System is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice (Jackson).
May either completely substitute the human expert or play the role of an

assistant.
It is a part of AI, because it is concerned with the design and implementation of programs capable of emulating human cognitive skills.

Characteristics:
Simulates human reasoning.
Performs reasoning over representation of human knowledge.
Solves problems using heuristics.
Deals with realistic complexity normally requiring human expertise.
Must exhibit high level of performance
Must be capable of explaining/justifying its behavior

Knowledge-Based System:
Any system which performs a task by applying rules of thumb to a symbolic representation of knowledge instead of employing mostly algorithmic methods (Jackson).

Knowledge Engineering:
The process of constructing an expert system

Knowledge Acquisition:
The transfer and transformation of potential problem solving expertise from some knowledge source to a program/machine.

Knowledge Representation:
Mostly concerned with finding ways in which large bodies of useful information can be formally described for the purposes of symbolic processing. Main issues are *logical adequacy* (= you can capture all logic relations in the KB), *heuristic power* (= quick way to solve problems and perform searches in the KB) and *notational convenience* (= easy of use / accessibility, structured objects).

Expert Systems encode the domain-dependent knowledge of an expert and use it to solve problems.

Metaknowledge - Knowing what one knows and knowing when and how to use it is an important part of expertise.

> There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns. There are things we don't know we don't know. - Donald Rumsfeld

> An expert system is a computer program intended to embody the knowledge and ability of an expert in a certain domain. - McCarthy

5.1.2 Components of an Expert System

On an abstract level, Expert Systems are concerned with:
Knowledge Representation.
Knowledge Acquisition.
Control of Reasoning.
Explanation/Justification.

On the implementation level, Expert Systems consist of
1. **Knowledge-base** consisting of facts and production rules (knowledge)
2. **Working memory** (current state) which holds data and intermediate results
3. **Inference engine** that reasons based on the rules stored in the knowledge-base.

5.1.3 Deep vs. Shallow Expert Systems

Depends on the domain model and how the domain specific knowledge is encoded into the system:
Shallow Expert Systems:
based on simple associations between things in the domain (stimulus and response) and (empirical) heuristics ("rules-of-thumb"); limited scope; usually highly specialized; *Example*: MYCIN - takes for granted that the user of the system has enough understanding of the domain;
Deep Expert Systems:
embrace much larger body of knowledge; more detailed domain model that actually incorporates the underlying causal model of the problem in question; Example: the CYC Project - goal is to come up with detailed causal model which reflects upon what could be called "common sense"; ultimate goal: explain the justifications of the reasoning process, beyond the simple "why"-explanation of certain rule firings.

5.1.4 Why and When Using Expert Systems?

Typical tasks for an expert system:
Interpretation of data
Diagnoses (MYCIN)
Structural analysis of complex objects
Configuration of complex objects
Planning sequences of actions.

When to use an expert system?

1. Domain is characterized by use of expert knowledge, judgement and expertise
2. Conventional programming approaches are not satisfactory
3. Recognized experts in solving the problem exist
4. Experts are better than amateurs in solving the problem
5. Expertise is scarce, expensive and depends on overworked experts
6. Heuristics, rules-of-thumb, dealing a lot with uncertainty make extremely large number of possibilities easy to handle
7. Amount of knowledge is large enough to make knowledge base developed interesting, but the number of important concepts is reasonable
8. Task is decomposable
9. Task is teachable and some knowledge can be acquired from text books
10. Experts could decide whether or not the results are good

5.2 Symbolic Computation

A symbol is a designator for "something" (e.g. a physical object, a concept etc.). In symbolic computation, we need a way to define symbols, create associations between symbols and provide operators to manipulate symbols (syntactic rules + transformation rules).

Physical Symbol Hypothesis: Newell and Simon: "A physical symbol system has the necessary and sufficient means for general intelligent action"

Structure based on logical/mathematical systems - hence we're dealing with sets and sequences, best represented as lists (as realized in LISP). Why? Because these formal structures are well understood.

Pattern matching:
Can tell if two symbols are equal to each other. Advantage: thanks to unification (substitution), it can match "wildcards" or variables to a instantiation.

Why LISP isn't good for knowledge representation:
because it lacks means to organize the use of knowledge, it just imposes a certain structure.

5.3 Rule-Based Systems

5.3.1 Production Systems

A production system consists of:
1. A set of **production rules**
2. **Rule interpreter** that decides which rules to apply and when
3. **Working memory**Contains the current state of the problem: known facts/data, current sub-goals, intermediate results. These data are used by the rule interpreter to activate the rules.

This is similar to the tree elements of an expert system: knowledge base, inference engine, and working memory.

Rules are triggered by this data and the interpreter controls the activation and selection of rules at each **recognize-act cycle**.
An advantage of Production System over regular computer language is that they provide an **immediate response**

A rule is of the form $P_1, \ldots, P_m \rightarrow Q_1, \ldots, Q_m$ where P_i are the premises (antecedents) and Q_i are the actions to be performed (consequents). A rule of this form "produces" the Q_i.

Example:

> $\alpha_1\$_1 \ldots \alpha_m\$_m \rightarrow \beta_1\$_1 \ldots \beta_n\$_n$
> This tries to match the content of the working memory to the premise and produces the action depicted by the right-hand side

of the rule.

$\$_1 0 \rightarrow \$_1 STOP$
$\$_1 1 \rightarrow \$_1 STOP$
This cuts any given binary number in half by right-shift.

CALC $1\$_1$RESULT$\$_2 \rightarrow$ CALC $\$_1$RESULT$\$_2\$_2$
CALC RESULT $\$_2 \rightarrow \$_2$ DONE
$\$_1 0 \rightarrow$ CALC $\$_1$ RESULT 1.
Calculates 2^n by transforming any binary number into a "calc (binary number) result 1" scheme (initialization, since $2^n = 1$ for $n = 0$).

5.3.2 Conflict Resolution

The interpreter for a set of production rules can be described in terms of the *recognize-act cycle*:
1. MATCH the premise patterns against elements in working memory
2. Choose one out of all selected rules (CONFLICT RESOLUTION)
3. APPLY the rule

The computation usually halts if no more rules can fire or a rule explicitly issued a *halt*-command.

Selected rules in step 2 are called *instantiations*. If the rule set is designed such that no conflicts can emerge, it is called *deterministic* or else *non-deterministic*.

Control regimes: Global - domain-independent (refractoriness, recency, specificity); Local - domain-dependent (meta-rules, based on expert knowledge on how to resolve those conflicts)
Control regimes have two main characteristics: sensitivity (how quickly does it respond to conflicts) and stability (are the decisions consistent). In addition to that, we can create salience groups for rules to determine their ordering. All this adds *heuristic power* to expert systems!

Appropriate Domains: Computer System Assembly, Circuit Fault Diagnosis, Route planning, mostly narrow domains. Inappropriate: Classification of extremely large, complex and ambiguous data, scientific calculations, multiple fault diagnosis.

5.3.3 Forward vs. Backward Chaining

Forward-Chaining is data-driven: we chain forward from true/satisfied conditions towards the goal and use newly added facts to further chain forward if possible.

Backward-Chaining is goal-based: we look for the satisfaction of the premises that conclude our goal and take it further from there (sub-goaling).

Example:
FC = generate palindromes from a given start;
BC = recognize palindromes from a given one.

5.4 Turing Machines - Computational Power and Expressiveness

It can be shown that Turing Machines and Production Systems are equally expressive. That is, any computation that can be performed by a Turing Machine can be translated to a production system and vice versa.
A language that is accepted by a TM that halts on all inputs is known as a recursive set.
Recursive languages are decidable.
Recursively enumerable languages are not (if word is in language, then TM halts, if not, then the TM might run halt or run forever).
Halting Problem: Is there an algorithm which can determine whether a TM eventually halts on input w? (No, as proved by Turing in 1936.)
(see 3.1.3 for more on decidability)

Non-deterministic Turing Machine:
A TM that allows a number of possible transitions instead of only one. DTMs have a single computation path, while NTMs have a computation tree. Apparently, NTMs seem to be more powerful than DTMs, but every NTM can be transformed into a DTM, which would compute the same, but takes a lot longer. How much longer is not known in case of the open $NP = P?$ question.

5.4.1 Time Complexity

P: Class of problems that can be solved in polynomial time on a deterministic TM - usually taken as the group of tractable problems, or problems for which an efficient algorithm exists.

NP: Class of problems that can be verified in polynomial time on a Nondeterministic Turing Machine. This class contains many problems that people would like to be able to solve effectively, including the Boolean satisfiability problem, the Hamiltonian path problem and the Vertex cover problem. All the problems in this class have the property that their solutions can be checked effectively.
A P-problem (whose solution time is bounded by a polynomial) is always also NP

NP-complete: Most difficult problems in NP; A problem is NP-complete if it is in NP and every other problem in NP is reducible to it (reduction = there is an transformation algorithm that runs in polynomial time and transforms any problem to the current problem).
A problem which is both NP (verifiable in nondeterministic polynomial time) and NP-hard (any other NP-problem can be translated into this problem). Examples of NP-hard problems include the Hamiltonian cycle and traveling salesman problems.
For example, given a logical expression, is there an assignment of truth values to the variable which will make the expression true?
Another example of an NP-complete problem is the subset sum problem which is: given a finite set of integers, determine whether any non-empty subset of them sums to zero. A supposed answer is very easy to verify for correctness, but no one knows a significantly faster way to solve the problem than to try every single possible subset, which is very slow.

NP-hard:A problem is NP-hard if an algorithm for solving it can be translated into one for solving any other NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem, " although it might, in fact, be harder.
All problems that are NP-complete or harder (formally: all NP-problems can be reduced to it, but it is not necessarily in NP itself, so it must be outside P, if P $\neq$ NP).

5.5 Dealing With Uncertainty

5.5.1 MYCIN-Style Certainty Factors

Certainty factors can be used to

1. Guide program reasoning.

2. Prune unpromising goals.

3. Rank hypotheses after all evidence has been considered.
endenumerate They should not be viewed as probabilities.

   Certainty factor (CF) : (Measure of Belief in hypothesis H based on evidence E) − (Measure of Disbelief in hypothesis H based on this evidence)
   Increase in degree of belief:

$$\frac{P(h \mid e) - P(h)}{1 - P(h)}$$

   Increase in degree of disbelief:

$$\frac{P(h) - P(h \mid e)}{P(h)}$$

   Problem with this method:
   P(d_1) = 0.8 , P(d_1|e) = 0.9 , P(d_2) = 0.2 , P(d_2|e) = 0.8
   Increase in belief in d_1

$$\frac{0.9 - 0.8}{0.2} = 0.5$$

   Increase in belief in d_2

$$\frac{0.8 - 0.2}{0.8} = 0.75$$

   We should have favored d_1 intuitively, but d_2 is the winner.

   Post-1977 MYCIN focused on CFs.

   Antecedent Rule - combine user's certainty on the provided evidence.

   Serial Rule - combine aggregated user's certainty and the expert's certainty, to give a fired rule a overall CF.

   Parallel Combination Rule - if several rules support the same hypothesis, their respective CFs have to be combined using this rule. Any good parallel combination rule should be associateive (so order of the

arguments is unimportant), commutative (so the sequence order is unimportant), and symmetric (so equal but opposite evidence cancels. Given CF_1 and CF_2, certainty factors for two rules,

$$CF = \begin{cases} CF_1 + CF_2 - CF_1 * CF_2, & CF_1, CF_2 > 0 \\ CF_1 + CF_2 + CF_1 \cdot CF_2, & CF_1, CF_2 < 0 \\ (CF_1 + CF_2)/(1 - min(\mid CF_1 \mid, \mid CF_2 \mid)), & CF_1 \cdot CF_2 \end{cases} .$$

5.5.2 Fuzzy Sets (Possibility Theory)

Possibility theory is a theory dealing with non-random type of uncertainty. Based on the concept of membership instead of simple switch-off/switch-on approach

1. Fuzzify input
2. Apply logical rules
3. Apply implication (min)
4. Aggregate Results
5. Defuzzify (Centroid, Bisector, Maximum...).

5.5.3 Conditional Probabilities (Bayesian)

P(A given B) = $\frac{P(A \wedge B)}{P(B)}$. Probability tree useful to come up with the cases (see Bob-Example: $P(A \wedge B)$ is 0.15 (0.9 x $\frac{1}{6}$), whereas $P(B)$ also takes into account the cases where initially no six came up, and Bob lied.

This is different from Certainty Factors, because here a belief in something commits one to disbelief in the negated hypothesis.

Bayesian methods can be used to determine the most probable hypothesis, in a sense that no other hypothesis is more likely.

5.5.4 Dempster-Shafer

Based on belief functions, rather than pure probabilities, they resemble the MYCIN certainty factors and are believed to combine probabilities with the flexibility of rule-based systems. Dempster-Shafer theory allows us to specify a degree of ignorance, the likes of which are not

allowed in Bayesian theory. Also not that the measure of belief and disbelief of an outcum need not equal one.

First, uncertainties are sorted into independent items of evidence. Then Dempster's rule is applied computationally.

Based on two things: computing belief functions and combining belief functions derived from different pieces of evidence

"m" is a basic probability assignment; m(0) = 0 and m(sumofall hypotheses-subsets) = 1; Total belief in any focal element A (a subset containing a number of hypotheses) is the sum of all bpa's (m's) of its subsets $B_1 \dots B_n$. Lower bound is the "support" of A, which is the total belief in A; Upper bound is the "plausibility" of A, given by $1-\text{Bel}(A^c)$ - (A^c being the complement of A) - i.e. this is the degree to which we don't believe in $\sim A$, hence $\text{Pls(A)} = 1-\text{Bel}(\sim A)$.

Belief interval: from "total belief in A" (support) up to "the extent that A could possibly be true" (plausible).

Demptser's combination rule: computes new belief function based on two existing belief functions derived from different evidence.

Doesn't distinguish between prior and posterior probabilities like Bayesian belief. Belief in something doesn't mean that the remaining belief is committed to its negation ($P(H) = 1-P(\sim H)$). In Dempster-Shafer, belief in some subset A (focal element) does not force the remaining belief to be committed to its complement. Hence Bel(A)+Bel(∼A) can be less than 1. The remainder is the degree of ignorance concerning A.

5.6 Knowledge Representation

5.6.1 Frames

Minsky's theory of frames - how the mind works: A frame is a data structure for representing stereotyped situations. Attached to each frame are several kinds of information (usage, predictions, exceptions, etc.). The human mind stores this kind of situational information in memory structures called frames, which are to be adapted to fit reality

by changing the details as necessary (frame problem: how to determine which things remain the same in a changing environment).

We can think of a frame as a network of nodes and relation (in fact, they are a subset of semantic networks). Collections of related frames are linked together in frame-systems. Actions lead to transformations between the frames of a system. A frame's terminal slots are normally filled with default assignments - presumptions and expectations. These are replaced once new and better information comes in.

Scripts are a type of frame. They specify a sequence set out in time.

5.6.2 Semantic Networks / Associative Nets

Semantic Networks are a subset of associative networks. They consist of nodes and links between nodes that express a relationship between each other. Basic idea is that the meaning of concepts come from their semantic relationship to other concepts. This meaning can be illustrated using nodes and labeled connections. SNs are a way to represent knowledge.

5.6.3 Ontologies

Explicit specification of some topic; formal and declarative representation which includes the names for referring to the terms in that domain and logical statements that describe what the terms are, how they are related to each other and how they can or can't be related to each other. (In philosophy the term accounts for what exists - in AI: what "exists" can be represented; knowledge can be represented in a declarative formalism and the set of objects is what's called the universe of discourse). An ontology is used to share domain knowledge among applications, agents, etc.

5.6.4 Belief Revision / Truth Maintenance Systems

Rule-based systems consist of a set of rules, which contribute to achieving a certain hypothesis. We keep adding supporting knowledge and evidence. TMS on the other hand keep better track of the reasoning process:

dependencies between rules and pieces of evidence - newly added evidence not only supports or negates a hypothesis, but can also have an impact on other hypotheses via a dependency network;
Facilities to maintain and revise current knowledge about the world - monotonic (always increasing the set of beliefs) or non-monotonic (allows retraction of beliefs). The former is what rule-based systems do, the latter is a system for knowledge justification, which allows commonsense reasoning;
Detects inconsistencies and contradictions in interdependent hypotheses and can make justified assumptions or retract justifications for pieces of knowledge.
This kind of information is propagated to all nodes of the network, making the reasoning a less straight-forward process.

5.7 Knowledge Acquisition

"Transfer and transformation of potential problem solving expertise from a knowledge source to a program"

Knowledge Acquisition is a generic term, which is neutral in respect to how this knowledge is gathered. Knowledge Elicitation is knowledge extraction from a human expert.

In general, the idea is to elicit knowledge in some way (e.g. provide example cases), then store the knowledge by designing a domain model of sorts, then compile the knowledge into production rules.

Identification - Identify problem characteristics
Conceptualization - find concepts to represent knowledge
Formalization - design structure to organize knowledge (domain model)
Implementation - formulate rules to embody knowledge
Testing - validate rules by validating the results

Testing leads to refinements of the rule base, redesigns of the domain model and reformulations of the initial concepts and characteristics.

Note that this should be an ongoing process as the system goes over to the maintenance phase after being deployed. Hence the expert system should allow to easily modify the rules and the domain model.

5.8 Case-Based Reasoning

Rule-based reasoning relies on "rules of thumb", i.e. heuristic knowledge acquired from an expert. Another method to solve problems is to remember how we solved it in the past: case based reasoning. Examples: Law - based on precedence; Medicine - based on previously experienced sets of symptoms.

Current "case" is matched to the most similar existing case (using vector algebra) and a decision is being made. The new case is then stored into the case database. The new case can also be modified to reflect differences from the existing cases. Case-based decisions also encompass decision tree instance based learning - coming across a new case can be seen as learning from example: something rule-based KBS cannot do; on the other hand - instead of modeling the domain and have a thorough understanding of it (i.e. a theory), case-base reasoning is rather dull/blind.

5.9 Explanation

Some facility to store the WHY and the HOW of the reasoning process. The user can ask, why the system is asking a certain question and the system should respond with some insight in the current line of reasoning. The user can ask how the system came to a certain conclusion and the system should respond with the inferences that lead to the current situation.

5.10 Famous Problems

5.10.1 Monty-Hall Problem

Always go for the switch, it doubles your chances of winning.
Initially: pick one from 3 doors gives your $\frac{1}{3}$ of a chance of winning the car. Now Monty opens one of the doors with the goats. Since he knows where the car is, he just gave you an important hint: Consider the other two doors as one entity of something you didn't pick. The chance that the car is behind one of those two doors is $\frac{2}{3}$ ($+\frac{1}{3}$ of your selection adds up to 1.0, so far so good) - hence this "entity" containing these two doors has a $\frac{2}{3}$ chance of winning. As soon as Monty opens

one of the doors of that "entity", you know that the chances that the car is behind that particular door becomes 0, but the "entity" still has a $\frac{2}{3}$ chance of winning. Hence, the remaining door in that "entity" has a $\frac{2}{3}$ chance of winning, whereas your original selection remains at $\frac{1}{3}$ – SWITCH!

Variant: The Big Deal
Monty invites 2 players to play the big deal: each player picks a door (different ones); Monty ejects the player with the goat behind his or her door or one of them randomly if both picked a goat-door (but he won't tell the players that of course); he then offers the remaining player to switch – should he accept? NO! In $\frac{2}{3}$ of the cases, the sticker will already have selected the door with the car. Only in the case that both players selected a goat-door will the sticker lose.

5.10.2 3 Glasses of Water

3 glasses of water - the first is potable with probability of 0.9, the second one is potable with certainty factor 0.9 and the third one is potable with possibility or fuzzy membership 0.9. Which one should you drink? Answer: the third one. Why? The issue is that probabilities and fuzzy memberships convey a totally different type of information. The 0.9 probability tells us that over a long run of experiments, 1 out of 10 bottles would contain a deadly poison. Essentially, you have a 10% of dying after choosing this bottle. The fuzzy number on the other hand quantifies the similarity of an object to imprecisely defined properties. A 0.9 membership means that the liquid in that bottle contains a liquid that is fairly similar to a perfectly potable liquid. The membership for a poisonous liquid would not be that high, so the risk of dying is completely ruled out. Hence this is the better choice.

5.11 Intelligent Information Systems

5.11.1 Blackboard Architecture

Addresses the issues of information sharing among multiple problem-solving agents in a heterogeneous environment. The blackboard is the central repository for all shared information and knowledge. Agents are considered to be some sort of expert that uses information from

various knowledge sources to contribute to the solution of the problem. An "arbiter" controls the access to the blackboard, allowing only the most promising contributions to be written to it (this process goes on until a solution has been found). The blackboard can therefore be thought of as the working memory of a problem-solver.

5.11.2 Intelligent Software Agents

Program designed for performing specific tasks on behalf of the user or other agents (in collaborative multi-agent environments).

5.11.3 Data Warehousing

Integrating multiple heterogeneous information sources into a single repository that can be accessed and queried directly in order to perform analysis or manipulate data through a single interface.

It is usually used for accessing databases with different formats and data models (oftentimes legacy databases) in order to provide a "single version of the truth", i.e. a single way to represent data and make it available to the global system

5.11.4 Active Database

Regular databases that embed situation-action-rules (i.e. react to a certain situation occurring)

5.11.5 Data Mining

Data mining essentially is a knowledge discovery mechanism for data bases (e.g. customer data). Uses statistics and pattern recognition as its main tools. It extracts implicit, previously unknown and potentially useful information from large data sets. The "patterns" that are searched for are usually hidden causal links between data.

6 Genetic Algorithms - CSCI(ENGR) 8940, CSCI(ARTI) 8950

6.1 Overview

6.1.1 What is a Genetic Algorithm

GAs model genetic evolution based on the Theory of Evolution.

6.1.2 Components of a Genetic Algorithm

Problem encoded in some way (chromosome); Population containing a large number of chromosomes; Genetic operators (selection, crossover, mutation); additional operators (elitism, seeding); Fitness function (encoding the optimization goal).

6.1.3 Why Using Genetic Algorithms?

Noisy, discontinuous, non-differentiable problems seem to be appropriate for GA/EAs, whereas continuous and differentiable problems are more suitable for traditional optimization.

Difference between traditional approach and evolutionary approach:
Traditional optimization uses deterministic rules to browse the search space, does a sequential search from one particular starting state, carefully adjusting this state to move towards the goal. TO is guided by some derivative information (e.g. logic)
Evolutionary optimization uses probabilistic transition rules, does a parallel search from various initial starting states. EO is guided solely by the fitness values of individuals.

6.1.4 No Free Lunch Theorem

For any algorithm, any elevated performance on one class of problems is paid for in performance on another class. Generally speaking, even if there's something like a "free lunch" on an individual basis, *someone* will have to pay for it.

6.1.5 Intractable Problems

A class of problems is called **intractable**, if the time required to solve instances of the class grows at least exponentially with the size of instances. This is important, because it means that even mid-sized instances of a problem class cannot be solved in reasonable time. Generally speaking, a problem is intractable if no algorithm exists that can compute all instances of it in **polynomial time**

One should try to **divide** an intractable problem into tractable subproblems or **reduce** a given problem to an already solved one (principle of reduction).

How to recognize an intractable problem class? NP-completeness theorem: any problem class to which an **NP-complete** problem class can be reduced is likely to be intractable.

6.2 Fundamentals

6.2.1 Theory of Evolution - Survival of the Fittest

Guess what...

6.2.2 Schemata

Schema theorem: It indicates the expected number of individuals containing schema H in the next generation.

$m(S, nextGen) \geq m(S, thisGen) \cdot \frac{f(S)}{f(all)} \cdot 1-$surviving crossover and mutation

The selection factor $\frac{f(S)}{f(all)}$ expresses the fact that above average schemata have a higher chance of being in the next generation than below average schemata. This factor (selective term) is responsible for exponential growth of above average schemata, and exponential decay of below average schemata.

The loss term (surviving-stuff at the end) filters some of these schemata from the exponential selective process. Schemata that are short and of

low order and are emphasized by the selective term have a high probability of passing this filter. This is due to two things: crossover is less likely to destroy low-order schemata with few crosspoints; mutation is less likely to destroy a low-order schema if the defining length is short.

These schemata are called "building-blocks".

6.2.3 Building-Block Hypothesis

The GA favors certain building blocks (short, low order schemata) through selection and applies the genetic operators on them. The assumption is, that combining short, low order schemata leads to highly fit larger building blocks and eventually highly fit complete designs.

6.2.4 Implicit Parallelism

The GA processes way more schemata than it's population size. The challenge is to come up with an accurate number of schemata the GA usefully processes. However, in a chromosome, there are many schemata encoded. And even if not all of them are useful building blocks, so many get processed that the GA approach becomes really useful. The examination of schemata goes on implicitly and in parallel.

The number of schemata that are evaluated, selected and recombined is atleast $O(n^3)$ when n structures are processed for a moderate population size.

It shows that although not all building blocks do conform to the building block hypothesis, the GA looks at a large number of them. It will find one of many paths to a highly fit solution.

6.2.5 Importance of Diversity

Diversity makes sure that we're still looking at numerous places in the search space. The diversity should be high in the beginning, when we're scanning the search space and should decrease when the fitness gets close to the optimal, because then we're near the global maximum.

6.2.6 Premature Convergence

Often prevents finding the global maximum, because the chromosomes converge to a local maximum and give up the search of the remaining search space.

6.2.7 Exponential Growth of "Good" Solutions

The GA processes schemata (building blocks) and looks at a large number of them implicitly and in parallel. In addition to that, the GA also makes a large number of decisions between schemata. Each schema belongs to a partition of schemata - * * d encodes the schemata * * 1 AND * * 0 for example. Hence the GA is implicitly selecting between schemata based on average fitness. The GA will exponentially assign all individuals to a given schema within a partition (ie. focuses on one particular, highly fit building block).

6.2.8 Deception

A problem contains deception if the winner of some partition has a bit pattern that is different from the bit pattern of the winner of a higher-order subsumed partition. For example, if the winner of the schema *11* is different from the subsumed partition *110 (the winner being the schema with the highest average fitness).

A problem is fully deceptive at order N if, given an order-N partition P, the winners of all subsuming lower-order partitions lead toward a deceptive attractor, a hyper- plane in P that is different from the winner of P. For example, given the schema *ddd (order-3 partition), *all* subsuming partitions (e.g. *dd*) point to a different direction, the deceptive attractor.

A problem is consistently deceptive at order N if, given a order-N partition P, the various winners of all subsuming lower-order partitions lead toward hyperplanes that are different from the winner of P.

GAs containing deception are sometimes labeled "GA-hard", even though it is not the only thing that makes a problem hard (other factors are: number of local optima in the fitness landscape, sampling

error, relative differences in disjoint desirable schemata).

Decption can lead to poor performance of the GA, because the low-order schemata do not contain enough information, such that crossover would gradually improve the average fitness - i.e. crossover is rendered ineffective.

6.2.9 Baldwin Effect

If some trait is LEARNED by an individual, then the fact that learning was used to extend the chances of survival, then the ability to learn something is passed on to the offspring.

6.3 Genetic Operators

6.3.1 Chromosome Representation

There is evidence that it should be binary, but continuous or discrete values would work too (as we've done it in the MSE project).

6.3.2 Population

Bunch of chromosomes, each encoding a possible solution to the problem.

6.3.3 Selection

Selection is based on the fitness value of an individual chromosome. The more fit it is, the higher should be the chances to be selected. This can be achieved by roulette wheel selection, which assigns a portion of the roulette wheel according to the fitness. This might lead to premature convergence, when fit guys take over a huge portion of the r-wheel. Other options include tournament selection and rank-based selection.

6.3.4 Crossover

1-point, 2-point, uniform crossover, or other crossover techniques that are cleverly adjusted to suit the problem.

6.3.5 Mutation

Just as in biological evolution, there is a slight chance that some gene within a chromosome mutates (e.g. flipping the bit value). Some say that mutation is everything, and crossover is not that important. Other say it the other way round.

6.4 Advanced Genetic Algorithms

6.4.1 Elitism

Keeping the best guy as part of the next generation to preserve the currently best known solution.

6.4.2 Seeding

Either feed previously acquired partial/not-optimal solutions to start the search from "better" positions within the search space or use domain knowledge to initialize the first generation in a way that it reflects this knowledge, i.e. create chromosomes that contain a guess regarding the solution

6.4.3 Hill Climbing / Steepest Ascent

Start somewhere, then evaluate the fitness values of all neighbors and proceed to the best one. Problem: might get stuck at local maxima easily. Solution: Random restart, increase "neighborhood size".

6.4.4 Taboo Search

Mark positions you already searched or that are not worth to be evaluated to begin with. Maintain a list of those nodes and exclude them from the search. Nodes on that list might be allowed to put back into the search space under certain conditions.

6.4.5 Simulated Annealing

Based on the annealing in metallurgy - The goal there is to achieve a state of the internal structure of the steel that has the lowest energy

level possible. The goal is to bring the system from a arbitrary state to an optimal state (optimization problem). SA considers a range of neighboring states in the search space at each step. If the step it could take is better, it will take it, if it is worse, then there is a chance of taking that step based on the temperature function. Initially, the odds are pretty high, and decreases over time until reaching zero.

The metropolis algorithm defines the probability of taking a worse step: initially, the probability to make a step towards a better solution is 1 and the probability to make a worse step $e^{\frac{-\delta E}{kT}}$ (k is the Boltzmann constant). It's easy to see that the lower the temperature is, the smaller the chances to take a worse step. When T approaches 0, only steps that are better are taken (changing the whole thing to a hill-climber). Eventually, the system "freezes" in a steady state.

The decrease in temperature is carefully planned and laid out in the annealing schedule, which decreases the temperature after each step a little bit.

SA can be part of a GA. How about letting the metropolis algorithm decide if a given solution can be part of the next solution; i.e. a better solution (chromosome with above average fitness) is always accepted, a worse solution is accepted with some probability based on difference in fitness and current temperature.

6.5 Other Evolutionary Strategies

6.5.1 Genetic Programming

GP represents individuals as executable programs (as trees). For each generation, each evolved program is executed to measure its performance within the problem domain (= quantify the fitness of that program).

6.5.2 Evolutionary Programming

Instead of aiming at the genotypic evolution that GAs and GP deal with, EP focuses on the phenotypic evolution, ie. EP is derived from the simulation of adaptive behavior in evolution. The goal is to find

a set of optimal behaviors from a space of observable behaviors. The fitness function therefore measures the "behavioral error" with respect to the environment.

Originally designed to evolve finite state machines.

6.5.3 Evolutionary Strategies

Idea: biological processes have been optimized by evolution. But evolution itself is a biological process. Hence evolution must also optimize itself. That's what ES are shooting for: evolution of evolution.

6.5.4 Swarm Intelligence

A swarm is a structured collection of interacting organisms (e.g. Ant colonies, Fish schools, Bird flocks...). For example, in ant colonies, individuals specialize in one set of simple tasks and altogether, the swarm maintains a large set of complex tasks; i.e. the global behavior of a swarm emerges from the behavior of all individuals. Interaction is determined genetically or through social interaction. Changes to one individual in a swarm is influenced by the experience/knowledge of its neighbors.

7 Neural Networks - CSCI(ENGR) 8940, CSCI(ARTI) 8950

7.1 Overview

7.1.1 What is a (Artificial) Neural Network?

A network made of artificial neurons that provide a learning method that is robust to noise in the training data. It can learn real-valued and vector-valued functions over continuous and discrete attributes.

7.1.2 Components of a Neural Network

Inputs, Neurons, Hidden Layer, Output Layer, weights for each connection, bias node, activation function, net input, sigmoid activation

function, feed-forward, backpropagation algorithm, gradient descent to minimize the error; learning rate; momentum; recurrent networks, self-organizing maps (unsupervised learning)

7.1.3 Why Using Neural Networks?

Good at pattern recognition and classification of all kinds.

7.2 Fundamentals

7.2.1 Concept of a Perceptron

A artificial neural "unit" is called a perceptron. It takes a vector of inputs, calculates a linear combination of them and outputs 1 if the activation value is above a certain threshold or -1 otherwise. The input to the perceptron is called net input and the output is called activation.

A perceptron can classify any training set in which the individual instances are linearly separable. This allows it to learn stuff like the AND function, but not functions like XOR.

7.2.2 Inductive Bias

7.2.3 Learning From Example

Training takes place by comparing the error to the desired target and adjusting the weights accordingly. There are various ways to do that. The most popular being the **gradient descent / backpropagation algorithm**, which requires the activation function to be sigmoid (such that it is continuous and differentiable, because we use partial differentials to come up with the weight change). The steepest descent in the error surface can be found by computing the derivative of E with respect to each component in the weight-vector:

$\nabla E(\overrightarrow{w}) \equiv \left[\frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}, \ldots, \frac{\delta E}{\delta w_n}\right]$.

So for each weight w_i, the weight change is $\Delta w_i = -\eta \frac{\delta E}{\delta w_i}$.

If we change the weights after **each** training example, we're doing

stochastic gradient descent. This can lead to quicker convergence. Otherwise we change weights only after all training examples have been checked.

In multi-layer networks, we have to use backpropagation, because it is easy to see how the weights have to be changed for the incoming weights at the output layer (they're directly related to the output and the error respectively) – but determining the weight change for the hidden layer is more of a challenge. That's when we need the sigmoid function:

$\sigma(y) = \frac{1}{1+e^{-}y}$, where y is the input.

Learning Rate : Learning rate is how much of the calculated error is applied to the NN.
Smaller learning rate is preferred because even though more iterations are required to reach the local minimum, the search path resambles a gradient.
With a high learning rate, it is highly probable that we keep oscillating without ever touching the local minimum.
With a very small learning rate we can get stuck in the first local minimum with no mechanism to move out.
Momentum : Momentum is how much of the previous weight change influences the current weight change. The key idea is to average the weight changes so that the search path is in the average downhill direction. It helps us avoid local optima.
Stopping Criteria :
1. Maximum number of epochs have been reached.
2. Mean Squared Error (MSE) on training set is small enough.
3. When overfitting is observed, i.e. when training data is being memorized.

7.2.4 Noisy Data

Some training examples might be erroneous and misleading. Neural Networks are very robust regarding those kind of things.

7.2.5 Overfitting

Same as decision tree - training set error gets "too" good and the performance on the test set and any unseen examples will get worse.

7.3 Structure

7.3.1 Layers

Input layer; Hidden Layer; Output Layer;
Recurrent Networks permit feedback loop from hidden (Elman) or output layer (Jordon).

7.3.2 Net Input

$y = w_1x_1 + w_2x_2 + \ldots + w_nx_n$

7.3.3 Activation Function

Step-Function is the simplest; We use sigmoid or logistic function since it is differentiable and hence it allows to use the backpropagation algorithm and gradient descent (delta rule).

7.4 Types of ANN (Supervised Learning)

7.4.1 Feed-Forward NN

A standard FFNN has three layers: input, hidden and output, with direct connections between the input layer and output layer.
They can represent any boolean function, bounded continuous function and approximate arbitrary functions to some extent. Feed-forward provides the hypothesis space for the backpropagation algorithm.

7.4.2 Backpropagation NN

Changes the weights in the network with respect to the determined error from the feed forward stuff. Each combination of weights represents a hypothesis than could be considered. The inductive bias of this algorithm is to find a smooth interpolation between data points.

7.4.3 Functional Link NN

7.4.4 Product Unit NN

7.4.5 Simple Recurrent NN

7.4.6 Time Delay NN

7.5 Hidden Units - What Are They Good For?

They are used to form decision boundaries in the search space. If it is known that we have 5 different possible classifications, one should use at least 5 hidden units to take this into account. If less hidden units are used, the classification error will increase.

7.6 Unsupervised Learning

7.6.1 Self-Organizing Maps

Developed by Kohonen. They take the input and effectively cluster them through a competitive learning process, while maintaining the topological structure of the input space. The map is usually a 2D grid. Then the magic happens and the SOM will cluster similar input patterns in the map.

8 Machine Learning - CSCI(ARTI) 8950

8.1 Overview

8.1.1 What is (Machine) Learning?

A computer program is said to learn from experience with respect to some task and a performance measure, if its performance at the task improves the more experience it has.

Issues: how much training is needed; what algorithms can we use; how can we use prior knowledge.

8.2 Concept Learning

Inferring a boolean-valued function (does the example belong to the concept class - yes or no) from training examples of its input and output. We learn a concept over a set of items (instances). We map each instance to either 1 or 0 (belongs to concept, or doesn't). Now we have a set of hypotheses - all possible mappings from the available inputs to the output (concept). The goal is to find a hypothesis h that correctly maps all training examples to the concept function (and hopefully all unseen examples too).

8.2.1 Inductive Learning Hypothesis

If we find a hypothesis that covers a large enough set of training examples well enough, then this hypothesis will do well for unseen examples too.

8.2.2 Concept Learning As Search

Can be seen as a search for a specific hypothesis within a large space of possible hypothesis (usually just the consistent ones - the Version space). The goal of the search is to find the hypothesis that best fits the training set.

8.2.3 General-To-Specific Ordering of the Hypothesis Space

A hypothesis h_g is more general than or equal to a hypothesis h_j if and only if any instance that satisfies h_s also satisfies h_g.

8.2.4 Find-S Algorithm

Initialize h to the most specific hypothesis in H.
For each positive training instance x: for each feature constraint a_i in h, if it is satisfied by x, cool, if not, then replace a_i in h by the next more general constraint that is satisfied by x.
Output hypothesis h.

This will find the maximally specific hypothesis.

8.2.5 Version Space

The Version Space is the set of hypotheses that are consistent. A hypothesis is consistent with a set of training examples if and only if $h(x) = c(x)$ for each example. That is, it correctly classifies all the training examples. The version space is a subset of the hypothesis space.

8.2.6 Candidate Elimination

We start with the set of maximally general hypotheses G (all positive) and the set of maximally specific hypothesis S (all negative). For each positive example, we remove from S and G inconsistent h's and make S a little bit more general. For each negative example, we remove from S and G inconsistent h's and make G a little bit more specific.

This algorithm converges to the right hypothesis if there are no errors in the training set and there actually is some hypothesis that describes the target concept correctly.

8.2.7 Inductive Bias

Inductive bias includes anything that allows a system to generalize on unseen examples. The number of unseen examples over which a learning algorithm generalizes increases along with the strength of the bias. Induction bias includes representational bias (the assumption that the representation scheme used is capable of expressing a true model) and preferences as to a superior form of the solution (e.g., Occam's razor or the maximum specificity of the Find-S algorithm.

8.2.8 Futility of Bias-Free Learning

No bias - no learning. Exception: memorization $\rightarrow$ deduction

8.3 Decision Trees

Requirements:
Instances are represented by attribute-value pairs
Target function has discrete output values

Training data may contain errors.
Might contain examples with missing attributes.

Best classifier - use ID3 (C5) algorithm: it selects the attribute that is the most useful for classifying examples, based on the information gain.

Bias for decision tree: small trees and maximum information gain.

Decision Tree Learning does a search over the hypothesis space in form of a simple hill-climbing search until it finds a tree that correctly classifies the training data. The evaluation function that guides this search is based on the information gain.

Overfitting is an issue for training decision trees.

8.3.1 Why prefer short hypotheses?

Occam's Razor.

8.3.2 Overfitting

Doing better and better on the training set is not always helpful, because at some point, the decision tree, or the hypotheses or the neural net will start to perform worse on other (unseen) data. That's why we use test sets to double check the results: if the error in the test data starts getting worse, it's time to stop the search, because overfitting has occurred.

8.3.3 Entropy and Information Gain

Entropy is a measure of the heterogeneity of the classifications of a collection of examples. If each possible classification has an equal number members within the set, entropy is maximized. Entropy is zero when the entire collection belongs to a single class. The formula for entropy of a collection S is given by

$$Entropy(S) = \sum_i p_i \cdot \log_2 p_i,$$

where p_i is the proportion of the elements of S that are members of class i.

The information gain of subdividing the elements of a set S along attribute A is given by

$$InfoGain(S, A) = Entropy(S) - \sum_v \frac{|S_v|}{|S|} \cdot Entropy(S_v),$$

where v are the values, or classes, of attribute A, S_v is the set of the elements of S which are members of the class v, and $|S|$ is the size of set S. When ID3 is deciding what attribute to branch its decision tree on next, it selects the attribute that maximizes the information gain.

8.4 Bayesian Learning

8.4.1 Bayes' Rule

Mitchell, Machine Learning, p. 156-158

T he posterior probability of a hypothesis h given a data set D, $P(h|D)$, is given by

$$P(h|D) = \frac{P(D|H \cdot P(h))}{P(D)}.$$

$P(h)$ is the prior probability of the hypothesis h, $P(D)$ is the prior probability of a data set D being observed, and $P(D|h)$ us the probabilty of the data set D being observed if h is true.

T he maximum *a posteriori* (MAP) hypothesis, h_{MAP}, in the hypothesis space H is simply the hypothesis which maximizes the *posterior probability*, $P(h|D)$. By applying Bayes' rule, we see that

$$h_{MAP} = argmax_{h \in H} P(D|h) \cdot P(h).$$

F inding the maximum liklihood hypothesis requires specifying the priors. If we assume that the priors of all hypotheses are equal, then the MAP hypothesis is the same as the maximum liklihood (ML) hypothesis,

$$h_{ML} = argmax_{h \in H} P(D|h).$$

8.4.2 Bayes Optimal Classifier

Mitchell, Machine Learning, p. 174-176

If we are interested in predicting the probability of a classification value, v, given some training data set D, and set of hypotheses, we calculate

$$P(v|D) = \sum_h P(v|h) \cdot P(h|D).$$

The $P(v|h)$ may well be 1 or 0, which occurs in the case of a deterministic hypothesis. The Bayes optimal classifier assigns from a set of classifications V, the one which maximizes this value.

$$v_{BOC} = argmax_{v \in V} \sum_h P(v|h) \cdot P(h|D).$$

The Bayes classifier is known to be optimal over any method employing the same hypothesis space.

B ut the optimality comes with a cost. It is necessary to calculate the effect of *each* hypothesis on *each* classification value.

8.4.3 Gibbs Classifier

Mitchell, Machine Learning, p. 176

If the cost of applying the posterior probabilities of the hypothoses to all of the classifications is too high, introducing an element of randomness can speed the process along, albeit at a cost on performance. The Gibbs classifier simply makes a weighted random draw (roulette wheel selection) from the set of hypotheses on the basis of the posterior probabilities. The classification is simply that given by the randomly selected classifier.

8.4.4 Naive Bayes Classifier

Mitchell, Machine Learning, p. 177-179

If the optimal classifier is too expensive and Gibbs sampling not accurate enough, the naive Bayes classifier may be a Goldilocks contender.

If the classification instance can be described by a set of attributes, then the probability of a particular classification given the attributes, by Bayes' rule is

$$P(v|a_i...) = \frac{P(a_i...|v) \cdot P(v)}{P(a_i)}.$$

Since, given a particular input instance, the denominator of the right hand side is the same, finding the classification that maximizes the probability is a mattre of maximizing the numerator. The classifier is "naive" because, in estimating $P(a_i...|v)$, it assumes that all attributes are independent. So the probability of their conjunction in light of a classification v is simply the product of the probabilities of each attribute in light of v. So the formula for the naive Bayes classifier is

$$v_{NB} = argmax_{v \in V} P(v) \cdot \prod_a P(a|v).$$

A s an example consider spam detection. The possible classifications are v_{spam} or v_{mail}. The attributes are words or strings of words. In this case

$$P(v_{spam}) = \frac{Numberofspammessages}{Numberofmessages}$$

and $P("GrandPrizeWinner"|v_{spam})$ is equal to the percent of all spam messages with that phrase occuring.

N ote that the result of $P(v) \cdot \prod_a P(a|v)$ is not to be interpreted as a probability. To calculate the *conditional probabilty* of a particular classification v_x given an instance

$$P(v_x) = \frac{P(v_x) \cdot \prod_a P(a|v_x)}{\sum_v P(v) \cdot \prod_a P(a|v)}.$$

9 Philosophy of Language PHIL(LING) 4300/6300

This class gave an overview on the achievements in analytical philosophy during the last century. We started with Frege, who tried to analyze the question of how we refer to things in a more formal way. Hence he developed a general framework, which later served as the basis of what we call first order logic today. He claimed that reference

takes place on different levels - any proposition has a referent and a sense. The referent is the actual object in the real world, the sense is an intermediate step between that and our subjective apprehension. The sense expresses the commonly shared features of a proposition (e.g. "The Morning Star" has the planet Venus as its referent, and it's sense is something like "the celestial object that is the last to be seen in the morning, before sunrise"). Russell rejected senses, and favored a more formal method using a completely logical framework to express the senses (e.g. "the present King of France is bald" can be formally expressed by $\exists x$ such that x is the present king of France, and no other x is the present king of France, and x is bald"; as you can see, since it is a conjunction and the first term is false, the whole proposition is false.

Five Puzzles about reference:
1 - Reference to Nonexistent (e.g. King of France)
2 - Negative Existentials (e.g. The present king of France does not exists)
3 - Identity Statements (e.g. Mark Twain = Sam Clemens)
4 - Substitutivity of co-referring terms in belief contexts
5 - Law of Excluded Middle (e.g. Santa Clause is obese vs. Santa Clause is not obese - problem: there is no referent, so Santa Clause seems to be neither)

We then went on to analyze definite descriptions further. Strawson objected Russell's theory to some extent by claiming that the speaker's intentions and the context in which a proposition is expressed play a crucial role regarding its truth value (e.g. the above example might have been true if uttered in the 17th century). Strawson sees referring as being something the speaker DOES while uttering a proposition, whereas Russell thought referring is more like making an existential claim about some object (i.e. denoting)

Donnellan distinguished between attributive and referential use of definite descriptions. Attributive use = implies uniqueness of a certain object; e.g. "the murderer of Smith is insane" refers to that one person, whoever it may be, who killed Smith. Referential use = directly pick out an object, e.g. during a conversation, I point to Jones who is the alleged killer standing in the dock and utter the same sentence "The murderer of Smith is insane", this time directly referring to Jones,

even if it turns out he is not guilty. Russell rebutted this furiously.

This gave way to a new discussion about speaker's intentions and speaker's reference. However, Kripke criticized Donnellan's theory, because it relied on semantic ambiguity, where there should be none. The word "the" seems to be pretty straight-forward, and there is no indication of when it is used referentially and attributively according to Donnellan. That's why Kripke distinguished between speaker's reference and semantic reference instead, which was seen as a more pragmatic approach (speech acts). Neale did the same thing and came up with definitions of propositions expressed and propositions meant, which essentially try to cover Donnellan's discoveries with Russell's original theory of definite descriptions.

Another interesting discussion was about Externalism: how does the external world, a speaker's environment etc. have an impact on his inner mental states. In other words - to what extent does the real world really exists independently from a human mind. Kripke came up with a new Theory of Causal Reference, in which he claimed that in order successfully refer to something, a causal link has to be established from the origin of the name to the current use. The name has to be introduced by a "mental ceremony of naming" like baptizing. This causal link is external to the mind, yet determines the inner mental state in some way. Then Putnam came up with the whole Twin Earth crap: even though both people from Earth and Twin Earth use "water" for that clear liquid, they actually refer to different chemical structures (H20 and XYZ). For Putnam, that means that meaning of natural kind terms like water are external to the mind - the term "water" has different extensions on each planet (H20 and XYZ), even though the intensions are the same (clear liquid, thirst-quenching...).

Extension of a proposition is the set of referents of that proposition; Intension is the meaning or the set of characteristics of a given proposition. Extension usually happens externally, whereas Intension is something that happens within the human mind (according to Putnam).

The second part of the semester dealt with propositional attitudes like beliefs and also belief reporting and the problems related to this - Londres est jolie vs. London is pretty; Paderewski has musical talent vs. Paderweski has no musical talent; the phone booth situation.

10 Fuzzy Logic - CSCI(ENGR) 8940

10.1 Overview

10.1.1 What is Fuzzy Logic?

It is a superset of boolean logic, which deals with the concept of "partial truth" or degrees of truth (instead of just yes/no, 1/0, black/white as in classical logic). It was introduced by Latfi Zadeh. Mathematicians criticize it because it seems to contradict the principle of bivalence (any statement must be either true or false).

10.1.2 Components of Fuzzy Logic

Fuzzy input functions, rule logic, output function, defuzzification method.

10.1.3 Why Using Fuzzy Logic?

It is more expressive than classical logic. It can handle imprecise properties like coldness easily. It is a way of dealing with uncertainty - that's why their not only used in fuzzy controllers, but also in expert systems.

10.2 Fuzzy Systems

10.2.1 Fuzzy Sets / Membership Functions

Depicts the degree of membership for a given object to a general concept. E.g. "tall" might be a continuous curve going up to 6', where it reaches 1 - this means that the closer you get to 6', the closer you are to being part of the concept "tall" (e.g. 0.9 tall might be 5'9"). Everybody taller than 6' is a definite member of this concept.

Fuzzy sets are characterized by the membership function which determines if a given item of the UD should be in the set *to a certain degree* (instead of just yes/no).

10.3 Fuzzy Inference

1. Fuzzify according to the membership functions; For example in the water tank controller, there is an input from the sensor that measures how full the tank is. This could be projected onto two membership functions: Fullness and Emptyness. If the tank is 0.5 full, it is right in the middle between those two concepts, so emptyness could be 0.2 and fullness could be 0.2 as well. Draw diagrams, which can take various shapes.

Apply logic: IF ... THEN rules. If premises are ANDed, we take the minimum fuzzy membership value of all included memberships. Let's say the rule is IF old-car AND american-car THEN piece-of-junk. And the fuzzy value of old-car is 0.4 and the one for american-car is 0.8, then the piece-of-junk conclusion is 0.4 (the MIN of the two values). If the premises are ORed, we take the maximum, if they are negated, we take the complement.

Apply implication: this essentially is applying the MIN function to the fuzzy output membership function. Let's say in the above example, the implication yields 0.4 piece-of-junk, and the output function is some sort of singleton (cone in the middle of the diagram), then we mark that cone at 0.4 and cut off the tip. Easy as pie.

Aggregate outputs: In case there are more than one rules firing for the same output (e.g. the water tank has 2 rules that each say something about the valve output function, the first says something about how OPEN it should be, the other one about how CLOSED it should be, then we put both results into one diagram).

Defuzzify: Take the centroid of the combined output graphs to get a crisp output value (e.g. in the water tank example, we get a definite value for the valve position).

10.4 Fuzzy Controllers

10.4.1 Mamdani

Inputs are fuzzizied, then the logical operators are applied. The implication is projected onto a fuzzy output membership function which

is usually singleton (like single cone in the output diagram). After aggregating all outputs of the rule firings, the result is defuzzified usually using the centroid of the area that the output function covers.

10.4.2 Sugeno

Same as Mamdani, except for the way it deals with the output. Instead of having fuzzy output functions, the output is a linear function of the inputs. So if P1, P2 and P3 are my fuzzy inputs, the the output is not the centroid of some dubious output area, but a function $f(P1, P2, P3)$ that leads to a crisp output value.

11 Knowledge Representation - PSYC 8290

This class was all about reading papers, so I imagine it might be different from year to year. These are the major competing theories to explain human intelligence.

Intelligence with or without representation?
According to Rodney Brooks, representation is not paramount to intelligence. His problem with representation is that it is something we give, not what exists. We can have intelligent behavior in robotics which rely on simple response-to-stimuli mechanism. Steels is pretty much in the same category. He believes in emergent intelligence....intelligence is in the eye of the observer, not in the system itself. Keywords ... Embodiment and Dynamics, Situatedness and Ecology, Emergent behavior. He builds layers upon layers of behaviour which seem intelligent.

The vision binding problem :
How do we recognize an object? How is it stored in our brain? Then what happens when we want to retrieve it? So when I look at a coffee mug, do I take the cylinder body, and the hooked arm and then bind them together in some way to make it a coffee mug? That is do I have a dynamic binding of elemental shapes or do I have static image of a complete coffee mug. Hummel etc believe in static-dynamic binding. When I am attending to the object, then it is dynamic, else static.

Categories and Hierarchies : Humongous paper by Humphreys

and Forde which explores all the category specific disorders in the human brain and asks, are there any real categories or is it simply that same region of brain is used for similar functions, hence leading us to believe in categories. As with everything in psychology and philosophy, there is no right or wrong; pick your side of the fence.

Physical Symbol System : Newell's PSS. Physical Symbol Hypothesis states that the physical symbol systems performing symbol manipulation is a necessary and sufficient condition for intelligence. The SS consists of a memory, a set of operators, a control, an input and a output.
Grounding problem : How do we ground these symbols to reality? Are these symbols just arbitrary meaningless symbols? Harnard's response is to look at symbols from ground up and not top down, i.e symbols arise from the nature of objects and our interaction with them. They are not arbitrary, hence computers will never get it.

Connectionist Network a.k.a NN : Pros:
1. Look like our brain.
2. Have the power to learn from examples which makes them dynamic.
Cons:
1. Cannot generalize outside the training space, hence have no concept of "Universals"; something which occurs naturally in SS.

Perceptual Symbol Systems They define symbol not as byte of sound or marks on paper, but as a sum of all our senses. So a symbol for an object would include representation of its shape, touch, smell etc. This is the most effective way to get over the symbol grounding problem. But is it? Now we have to deal with how the binding of so many different regions of brain happen to produce a symbol or recognize a symbol.

Latent semantic Analysis This theory uses statistical models to explain human cognition. They believe that using this model they can explain Plato's problem. Plato's problem = We know more words than we normally encounter.

:

12 Decision Support Systems

This class mainly dealt with the development of Executive Information Systems. This entailed the use of DSS and Expert Systems in those systems. The main questions centered around how such a system suits a company's needs and how to deal with executives who normally have little understanding of IT requirements. So oftentimes the question arises: why do we need it?

EIS consits of data bases, a comprehensive and intuitive user interface (different for different users - thin client, thick client), drill-down capability, extensive reporting, monitor performance on any detail level; compare, analyze and highlight trends of variables etc.

This requires the extensive use of company data, so data warehousing often is an important part of any EIS

A good EIS gives the managers are comprehensive overview of what is going on in the company, whereas traditionally they had to rely on paper-based reports that took longer to be created. Ideally, the executive can simply browse through the important information concerning the company and identify areas of improvement.

EIS - Executive Information System: A computer-based system that helps with information and decision-making of senior executives, commonly considered a specialized form of DSS

DSS - Decision Support System: A computer-based information system that helps with the task of decision-making. It usually lacks the reasoning mechanism of expert systems, even though their domain is narrow too. DSS are more like tools that help to understand a certain problem or domain better, such that the human expert can make a decision more easily, instead of going through tons of data.

MIS - Management Information System: company-wide information systems based on the network of communication channels within an organization (includes hardware, software, people and communication systems such as phones, email, fax etc. and of course the data itself). It involves collecting, manipulating and disseminating data among this

network.

13 Rapid Application Development

Rapid application development (RAD), is a software programming technique that allows quick development of software applications. Some RAD implementations include visual tools for development and others generate software frameworks through tools known as "wizards." (CASE tools, code generators...)

While RAD tools significantly cut down in software development time, they sometimes do it while sacrificing application execution speed or efficiency. Solutions developed via RAD techniques may not be the optimal solutions for any given problem set.

Goals: Short development time, high quality and low cost.

Fundamentals:
1. Tools - powerful enough to create "good-enough" prototypes
2. People - well-trained on using the tools and understanding of methodology; small teams
3. Management - managed for speed; extract user's needs quickly; iterative deployment
4. Methodology - activity planning and documentation; quick ways to solutions should be formalized.

RAD is divided into stages - requirements, design, implementation and transition. The requirements and design phase are often combined. Implementation and transition is sometimes combined (Extreme Programming - iterative release of prototypes, feedback loop).

Prototyping:
Acquire requirements, build prototype - review prototype - refine requirements - revise prototype - review prototype loop; Allows to discover fundamental errors earlier, due to high user involvement.

Essential Analysis and Design:
Based on event tables, data flow diagrams and context diagrams (DFD = entities, events and data storage; context diagrams are entities and

how they interact with the system).

Rest of the class dealt with an introduction to ASP.NET, including connecting to a database and writing a basic web service with .NET

14 Advanced Data Management (XML)

Task was to write an XML textbook for wikibooks. Each student created one chapter and edited another one.

We learned about how to express one-one, many-one, many-many and recursive relationships in XML. We learned how to use XML schema and style sheets. We learned how to use XPATH expressions. We learned how to use Java to parse XML files and write input to a SQL data base. We learned about XUL, a GUI scripting language based on the Gecko engine that comes with Mozilla. We learned about Web Services. We learned about SMIL, an XML based way of describing animations and slide shows. We learned about RDF (resource description framework).

15 Actual Questions

(a) (Dr Potter): Show an assignment of variables that would make this expression false? The expression was a sentential logic implication. Can you write a computer program that could solve this? Could you write one that could solve the problem faster than hard crunching? What is NP/NP hard/NP complete.

(b) (Dr Nute): What is decidable/semi-decidable/undecidable and give examples of each. Draw a Venn diagram of how they relate. What does it mean to be decidable (more technical definition). What does it mean for FOL to be semi-decidable.

(c) (Dr Potter): Write a tail recursive equation in prolog (factorial).

(d) (Dr Potter): How does a GA work? Why will it find an optimal solution.

(e) (Dr Potter): Explain simulated annealing in terms of GA

(f) (Dr Potter): How GA works in general? (generational vs steady state)

(g) (Dr Potter): Explain crossover (Just simple one)

(h) (Dr Potter): What is schema theory?

(i) (Dr Potter): What is implicit parallelism?

(j) (Dr Potter): How should you deal with a large number of variables when designing a GA?

(k) (Dr Potter): What is the growth rate of schemas of high fitness from generation to generation according to the schema theorem?

(l) What is the diagonalization argument?

(m) What papers did you read in Phil lang and mind (not followed up by any further questions).

(n) Speed round: Chineese room ; Turing test ; Name three AI guys ; Turing machine and decidability

(o) (Dr. Potter): What is the basic technique of AI?

(p) (Dr. Potter): Explain different types of search

(q) (Dr. Potter): Explain alpha-beta pruning
Explain with graph
What is min-max

(r) (Dr. Potter): Explain Iterative Deepning in detail

(s) (Dr. Potter): Explain simulated annealing in detail

(t) (Dr. Potter): Do you know D* search?

(u) (Dr. Potter): A* search
Explain
Why good?
Can it get into infinite loop?
Open List or Closed List?

(v) (Dr. McClendon): Explain why it is good to use sigmoid function as an activation function in Neural Net? Explain it in terms of how you change the weight.

(w) (Dr. McClendon): Explain error and weight relation (1 and 2 weights)

(x) (Dr. McClendon): Explain how to calculate change of weight when you have 2 weights

(y) (Dr. McClendon): said "Well we have to train NN until the error is reduced as much as possible" So you need to stop him and then explain.

a. Why this is not the case
b. How do you know when to stop the training?
c. Why do you have to use 3 different sets, not 2?
d. Do you change weights according to Testing and Production set?

(z) (Dr. McClendon): Assume there are 2 inputs (X and Y) and 1 output
a. Draw membership functions
b. What membership function means?
c. How do you deal with AND and OR operator?
d. Does out put function have to be truncated or is there another way to handle output? (Mamudani vs Sugeno)

() (Dr. Potter): Assume you are in dessert for a week without any drink. Now there are 3 cups of potable drink with Certainty Factor= 0.8, Probability = 0.8, and Fuzzy = 0.8. Which one do you chose to drink and why.

() " What are the major blind search strategies in AI?
What is complexity
What is the time/space complexity for the blind searches
Are they complete, optimal?
Under what condition are they complete or optimal?

() What are some informed search strategies (A*)
What is an admissible heuristic?
What is a consistent heuristic?
Are they complete, optimal?
Under what conditions?

() What is the Stroop effect?
How does this relate to Automacity?
Does a small child have the Stroop effect?

() What is encoding specificity?

() Mechanisms of memory encoding and retrival
Recall vs Recognition

() What is a clause, literal, term, function, predicate, definite clause, negative clause, horn clause?

() What is resolution?
How does it work?
Is it complete?
How does prolog work in relation to these concepts?

() What is a generative grammar?

() Create an algorithm (generative grammar) for writing specific sentences.

() Valid/Sound arguments
The syntactical structures of the arguments?
What is the line b/w the premises and conclusion called?

() Name some of the major techniques discussed in Machine learning.
You will be given specific 'consulting' situations and asked what machine learning techniques you would use/recommend to your employer and why?

() Create and understand an ANN on the board.
What measure of error does back prop of error use?
How does back prop training of ANN weights compare to using a GA to evolve the weights?
What are two problems associated with evolving ANN weights with an evolutionary strategy (GA, EA)?
(Answer) A large number of weights might be too complex for GA
(Answer) Aliasing

() Explain Turing Test. b. If program passed Turing Test, can you say it is performed as well as human (Answer in terms of game context)?