

Personnel

Lecture 1

Artificial Intelligence (0270) 2002

In our first lecture, we will have a general overview of the course, and a few essential ideas about Artificial Intelligence in general.

References

- Chapter 1: Section 1.1, 1.4
- Chapter 2: The whole chapter

Lecturer: Dr. Isaac To Kar Keung (kkto@csis.hku.hk), CYC407.
Office hour: Thu 1pm–3pm.

Tutors:

- Felix, Cheung Kai Man (kmcheung@csis.hku.hk), Office CYC317.
Consultation hour:

Web site: <http://www.csis.hku.hk/~c0270/>

Textbook:

- “*Artificial Intelligence, A Modern Approach*”, 2nd Edition. Russell and Norvig, Prentice Hall. Web site: <http://aima.cs.berkeley.edu/>.

AI(0270)

AI(0270)-1.1

Calling for help

If you have difficulties, you can do one of the followings:
In the order of preference...

- Post a news article to our **newsgroup**, hku.csis.CSIS0270. This is preferred: all of you can answer, and all of you benefit from the answer. Questions asked in any other means may be answered (only) in newsgroup, so **please keep watching the newsgroup**.
- Send us a **mail** at c0270@csis.hku.hk. The mail will end up in the mailbox of me and all tutors.
- Contact us directly. Send a personal mail before visiting our office to make an **appointment**.
This is the order in which I deal with them. I.e., news are first cleared, and only when all news are dealt with I'll deal with mails to c0270, and only after that I'll look at personal mails.

AI(0270)-1.2

Lecture conduct

Lectures are free-form, relaxed but serious.

- **Talking** is allowed **only on course matters**.
- **Attend lectures** only if you are willing to learn.
Nobody force you into lectures anyway.
- **Everything** about the lecture **needs to be understood**.
Preferably during the lecture, and absolutely no later than the end of the day.
- **Ask** whenever you find something you don't understand.
Don't hesitate: what you don't understand is probably not understood by many others as well, and you're helping others by just asking.
- **Stop the lecture** if you need time to digest. This is **vitaly** important: it is usually the only way for me to know that you need more time.
This is the way in which you can understand everything during the lecture.

AI(0270)-1.3

Assessment

Assignments (35%)

- 4–5 assignments with written and programming parts.
Language: C++ (unless otherwise specified).
Platform: either Linux or our departmental servers.

Quiz (15%), To be announced

- 1 hour quiz of easy questions. Should be very easy if you have done the assignments.
Big hint! Quiz is on assignments, not on other materials of the course.

Exam (50%)

Advices on learning

- **Learn the concepts first (most important)**. Implementation details are not as important as the concepts.
We usually show psuedo-code for illustration purpose. But if you are not familiar with computer code, it is okay to skim them (until you need to write programs).
- **Understand, not just memorize**. Find the underlying ideas that give rise to concepts. Find unifying themes.
- **Keep up with lectures**. Each lecture builds on top of the previous ones, so if you miss just one lecture, it will be significantly more difficult to catch up with upcoming lectures.
- **Do the assignments early**. Programming assignments are usually more difficult than what you first think. And assignments clear up your concepts for easy understanding of the coming lectures.
- **Read the book**. Lecture is too short to explain everything in the book, and the function of lectures is simply to guide you to read.

AI(0270)-1.4

AI(0270)-1.5

What is Artificial Intelligence?

AI is the study of **making intelligent systems**.

Note the word "making": we will write quite some programs.

But what is intelligent? Different approaches:

Think like human (cognitive modelling) Automate human thinking, e.g., decision making, problem solving, learning.	Think rationally (laws of thought) Perform computations that makes it possible to perceive, reason and act.
Act like human (Turing test) Do things that requires intelligence when performed by people.	Act rationally (rational agent) Explain and emulate intelligent behaviour in terms of computational process.

Note that we didn't specify the problem domain. This vagueness is due to the fact that we want **general strategies**, not specific to particular problems.

AI(0270)-1.6

Our approach

We adopt the "**Rational agent**" approach:

- Require **rational** behaviour: avoid the unclear idea about which behaviour is more "human-like". In contrast, what is rational is more well-defined (what is best given the current knowledge).
- Require **actions**, not just thinking: avoid the philosophical question about whether machines can think. It is also more general: we need the knowledges advocated by "thinking" camp in order to "act" rationally.
Agent: anything that obtains "percepts" (through "sensors") to predict the environment and produces "action" (through "actuators") in to modify it.

Combining the two advantages, we can ask whether an agent is better or worse than another in a scientific way.

AI(0270)-1.7

Rational is not omniscience

I see my old friend across the street, and want to chat with him. Noting that there is no traffic ,and I've nothing else to do, I started crossing the street... only to get flattened in the middle of the road, by a cargo door falling off from a 747. Am I rational?

- **Rationality is not perfection.** The choice of an agent can depend only on percept sequence **to date**.
Otherwise we want a crystal ball, not a computer!
- But what if, in our environment, cargo door falls every 2 minutes?
- Then we better look up first! It is **too dangerous** to cross the street **without first acquiring the knowledge** about whether doors are falling...

AI(0270)-1.8

What is rational

Rationality: **expected** success **given what has been perceived**. It is determined by

1. The **performance measure** that is determined in advance (sometimes, averaged over a period of time).
2. What is known by the agent about the **environment**.
3. What the **actuators** can do to the environment.
4. What the **sensors** perceived so far.

An **ideal rational agent** will maximize the expected performance measure for **every possible percept sequence**, on the basis of the percepts and its built-in knowledge about the environment.

Not always possible, e.g., due to limited resource.

AI(0270)-1.9

Let's look it up!

Input is some percept, output is some action. Thus our agent is simple:

```

percepts = []          # An empty percept sequence
table = [ ... ]        # Maps percept sequences to actions
def TableDrivenAgent(p):
    percepts.append(p)
    return table[percepts]

```

Note: We use a syntax that clearly demonstrate the concepts. When you implement it, you will need to deal with the restriction imposed by the actual language. BTW, the syntax looks a bit like a scripting language called Python.

- Note that we don't need to put the environment and the evaluation functions here. These information are **encoded into the table**.

AI(0270)-1.10

The failure of lookup

The problem is that **it won't work for most situations**.

- The **table** will be **huge**, spending a lot of memory.
E.g., to play "reverse it", the table will need 60! entries.
- It will take a **long time** for the designer to **fill the table entries**.
Both because the table is large, and because it requires some intelligence.
- It is very **difficult** for the program to **learn** by itself, because it needs to learn all entries of the huge table.
- If the environment or the evaluation function **changes** a little bit, **the table needs to be completely redesigned**.
In other words, the degree of automation is very low.

Our aim: add (a reasonable amount of) code to make it more **practical**.

AI(0270)-1.11

Reflex agent: making lookup more reasonable

- One way to improve the situation: make **if-then rules** to “compress” the table. The resulting program has the following form:

```
if I can make a line of 3 now:  
    play the move which makes the line of 3  
elif the opponent can make a line of 3 during the next move:  
    play the move to block it  
elif ...
```

- Such agents are called **reflex agent**: like reflex in neuroscience, it is encoded into the program, and performed without reasoning.
- Example: to play a game of tic-tac-toe using table lookup, we would need $3^9 = 19683$ entries. For a **rule-based agent**, 5 rules suffice .

So a rule-based agent is sufficient to solve Tic-Tac-Toe. But if we play Chess using a rule-based agent, the agent plays rather poorly.
The rules are all too specific!

AI(0270)-1.12

How to do better?

So what can we do to improve the situation?

- Write code that **predicts the results** of a possible action.
- Explicitly code the **evaluation function** into our program.
- Write code to **plan action sequences** to maximize performance.
- Have some way to **learn** the above by the itself without explicitly coding it, e.g., let the agent learn what is a good evaluation function.
Unluckily, the course does not have enough time to cover this.

Since an evaluation function is used to predict the **utility** of a state, we call such agents **utility-based**.

AI(0270)-1.13

Example: Chess

We would write a program to include code which can compute these.

- Given a current configuration, determine **what moves are valid**.
- Given a configuration and a valid move, determine **the next configuration** after making that move.
- Given a configuration, determine (at least approximately) **how likely** will that configuration lead to **winning**—utility.

Then our program will have additional functions to...

- **Try a sequence of moves** to see what is the utility after the moves.
- **Consolidate the result of a lot of such sequences** and determine what is the best next move.

Ideally, we also want a program which automatically find a good evaluation function that actually reflect whether a configuration will lead to winning.

AI(0270)-1.14

Problem-specific and General code

In the last page...

- The first group of functions are **problem specific**, but more or less trivial (at least, for simple games).
- The second group of functions is not as trivial and is the core of the “reasoning”. They are also **generic**: they probably work for all two-player games that is “similar”.
- In lectures, we will explore the generic methods that allows reasoning to be done. When actually using them to build a complete program, one must add problem specific information to make these functions work.

AI(0270)-1.15

About the environment

We say the same method can play all “similar” games. What is meant by similar? We mean a game which is...

- **fully observable**, i.e., the whole environment is known. This is as opposed to “partially observable” where some information is hidden.
- **deterministic**, i.e., the result of every action is fixed. This is as opposed to “stochastic” (or “probabilistic”) where there is an element of chance.
- **discrete**, and has only **a few possible** percept and action values, as opposed to “continuous” where there are many or infinite .
- **static**, i.e., the environment does not change while the agent thinks, as opposed to “dynamic” where it changes when the agent is thinking.

Not every AI problem is this simple. Confronting more difficult AI problems, the programmer and the computer need to pay more effort.

And we probably need to accept a solution that is not as good as we want.

AI(0270)-1.16

What's ahead

- Our primary business in this course is to learn **how to reason efficiently** with mechanical code.
- The Chess example shows one of them: **by searching for plans** to succeed. We will learn how to search efficiently .
- Searching will be combined with a method to **represent knowledge** of the environment: logic.
- We will see how new knowledge can be generated from old ones, and how to combine it with searching to **plan in the logical world**.
- Finally, we will learn how reasoning has to be done when there is **uncertainty** in our knowledge, which is common in expert systems.

What you should do now: get the book, and skim through the first two chapters.

AI(0270)-1.17

Course overview

- Wk 1: Introduction, searching
- Wk 2: Informed search (A1)
- Wk 3: Constraint satisfaction problems
- Wk 4: Games (A2)
- Wk 5: Logic
- Wk 6: First Order Logic
- Wk 7: Chaining, Prolog (A3)
- Wk 8: Resolution, Quiz
- Wk 9–10: Planning with make-up class.
- Wk 11: Hierarchical planning (A4)
- Wk 12: Probabilistic Reasoning
- Wk 13: Belief Network (A5?)
- Wk 14: Markov Model