CCPS 721 Artificial Intelligence I

Ilkka Kokkarinen

Chang School of Continuing Education
Toronto Metropolitan University

Version of December 29, 2024

This document contains the outline and course management form for the course **CCPS 721 Artificial Intelligence I**, as prepared and taught by Ilkka Kokkarinen for Chang School of Continuing Education at Toronto Metropolitan University, Canada.

Ilkka Kokkarinen acts as the Subject Matter Expert for the purposes of virtualization of this course. The latest versions of the files and documents used in this course are always available in the public GitHub repository `ikokkari/AI`.

Learning outcomes

Upon completion of this course, the students have acquired the following knowledge.

- Think of decision problems in terms of agents operating inside an environment to maximize its expected utility, and classify such problems according to the relevant aspects of the problem.
- Understand blind and informed search algorithms for problem solving in state spaces of one-player games of complete information.
- Use the minimax algorithm to play two player games of complete information, and know both general and game-specific techniques to speed up the search by pruning the game tree.
- Can model problems as constraint satisfaction problems to be solved with consistency and backtracking algorithms, and again know both general and problem-specific optimizations to prune branches of the search tree.
- Understand logical reasoning and programming in the Prolog programming language enough to solve non-trivial coding problems by converting them into Prolog logic formulas, especially problems in parsing and combinatorial search that would be problematic to solve within the traditional imperative programming paradigm.
- Know the syntax and semantics of propositional logic and predicate logic, and can apply mechanistic inference rules to logical formulas.
- Represent uncertainty with Bayesian probabilities, and perform both causal and diagnostic computations for two or three unknown by hand, and for a more realistic number of unknowns with belief networks.
- Combine utilities and probabilities for decision theory to be applied to both simple and complex decisions.
- Understand on a theoretical level the core ideas, principles and essential algorithms of machine learning, both supervised and reinforcement kind, so that they are ready to learn how these techniques are put in practical use in their work with `scikit-learn` and other modern machine learning frameworks.

Course outline

This course content consists of twelve modules. These modules have been naturally grouped into four parts of three modules each, each part consisting of three modules that together share and build on the same central theme.

Part One: Thinking in states

- Module 1: Agents and environments
- Module 2: Adversarial search
- Module 3: Constraint satisfaction

Part Two: Coding in Prolog

- Module 4: Prolog programming language
- Module 5: Solving problems with Prolog
- Module 6: Structure and interpretation of Prolog programs

Part Three: Thinking in symbols

- Module 7: Propositional logic
- Module 8: Predicate logic
- Module 9: Bayesian probability

Part Four: Acting in uncertainty

- Module 10: One-shot decisions
- Module 11: Supervised learning
- Module 12: Reinforcement learning

Reading material

Course textbook

The official textbook for this course is the **fourth edition** of the classic textbook "Artificial Intelligence: A Modern Approach", henceforth called **AIMA4**. (amazon.ca hardcover, Pearson online) This fourth edition is a massively streamlined improvement over the already magnificent third edition that was used in the previous semesters of this course, and is recommended for all students without hesitation during and after this course. Keeping up with the times, AIMA4 comes with a public GitHub repository `aimacode` chock full of all kinds of useful material. The repository `aima-python` contains example code as Python modules and Jupyter notebooks that summarize the important material of AIMA4 in a convenient interactive format.

This course covers approximately half of the material in the textbook. The lecture outline follows the usual, these days almost quaint outline for the first "Good Old-Fashioned Artificial Intelligence" course duplicated in the stratified air of Princeton and Berkeley all the way down to the practical can-do spirit of Wossamotta U. However, this course concentrates on theory and deep principles, and leaves out the classic practical applications of artificial intelligence in robotics, natural language and computer vision. This course also does not get into the current buzz about deep learning. These topics are best left for later, more specialized courses.

The repository `aima-exercises` offers a large collection of exercises, ranging from theoretical to practical, about the topics of each module. In fact, AIMA4 no longer contains any exercises inside the dead tree edition. These exercises live and evolve in this repository so that those pages can be used for talking about the important concepts.

Each scheduled session nominally consists of three hours of lecture followed by one hour of lab. However, seeing that these virtualized courses liberate learning from the tyranny of time and place, our lectures and lab sessions are also integrated so that during these lectures, selected lab problems are examined at whichever point of the entire four hour session seems the most appropriate to bring in that problem.

To fill in some narrative gaps that existed in earlier versions of AIMA and especially its old overhead slides that have not been updated since the first edition of this textbook, the instructor has compiled over time a set of additional unofficial notes, available as the PDF document "CCPS 721 Artificial Intelligence Extras". These notes were largely inspired by examples, observations and ideas encountered in past lectures that afterwards seemed just a little bit too good to be left behind to be the sole property of the people of the past.

Prolog material

As with all classic textbooks of computer science, the intention of AIMA4 is, of course, to soar above any particular programming language used to actually implement the low-level details of its high-level ideas. Therefore this tome does not contain any material on Prolog programming, beyond some cursory mentions of its connection to predicate logic and constraint satisfaction. The mainstream programming languages of the 2020's have generally reached the sweet spot between the imperative, functional and logical programming paradigms to make explorations on artificial intelligence topics natural. Therefore the same as in other applications of programming, AI programming these days is best done in high-level imperative languages such as Python, usually aided with powerful frameworks of numpy and scikit-learn.

Originally discovered back in the very different world of the seventies, Prolog no longer holds much sway in artificial intelligence the way it did back when its practical alternatives were basically C and various other low-level string grinders designed for efficiency of low-level operations instead of clarity and brevity of thinking in higher levels of abstraction. However, due to its unique nature in that the Prolog language with its modern extensions is practically *sui generis* as its own programming paradigm, Prolog still shines brightly in problems that consist of analyzing and processing symbolic structures, and also in discrete optimization problems that can be expressed and solved using the standard Constraint Logic Programming extension to the language. Prolog should still therefore be the language of choice for parsing and analyzing the meaning of both natural and artificial languages.

Even outside its applications in linguistic analysis and combinatorial search and optimization with constraints, the Prolog programming language still has value in expanding the concept of programming from the restrictive imperative form to more flexible logical and functional forms. In addition, Prolog makes otherwise ossified formal logic come to life when logical expressions get applied to arbitrary symbolic terms such as lists and integers. This drills into the worldview of computer programs as special cases of symbolic formulas and their entailed consequences in the domain of integers.

The instructor also has a series of YouTube videos on Prolog, collected in the playlist unimaginatively named "Prolog".

To supplement the lectures and example programs of this course, students may choose to study either one of the following Prolog tutorials. "Learn Prolog Now!" is simpler and behind that link even embedded with SWISH-Prolog for interactive play with examples. "The Power of Prolog" is more abstract and theoretical, but nobody can deny how well it makes its case, presented with the famous Teutonic efficiency and style that we all know and love. The author also maintains an eponymous YouTube channel "The Power of Prolog" of his lecture videos.

For additional clarification of all these new terms flying around here, students can also check out The Prolog Dictionary. For the Constraint Logic Programming extension that makes Prolog unique among all programming languages in solving all sorts of discrete optimization problems, you can check out the CLP(FD) Tutorial, a part of the larger site "Real World Programming in SWI-Prolog".

The collection "Ninety-Nine Prolog Problems" and their solutions to these problems can be used to practice problem solving in Prolog before you start the Prolog labs.

SWI-Prolog is the official Prolog environment used in this course. Students may choose to work with some other flavour of Prolog, but the lab submissions will be tested and graded in SWI-Prolog. Distributed under GNU General Public Licence, SWI-Prolog is completely free for download and unlimited use. Instead of installing this system locally on their computers, students can also use the free and interactive cloud-based version SWISH-Prolog that is used to demonstrate the language during the lectures.

Same as with working in any other programming language, you should keep the SWI-Prolog reference manual page open in one browser tab for quick access to the documentation of built-in predicates. The page "Notation of predicate descriptions" explains the conventional notation used to document the instantiation expectations and behaviour of parameters.

Grading and policy on cheating

The grading for this course consists of the following four components:

1. Python state space search labs, 30%
2. Prolog labs, 20%
3. In-person midterm exam, 20%
4. In-person final exam, 30%

The final course grade of the sum of the scores for these four components. Furthermore, as required by TMU School of Computer Science, to pass the course, the student must get at least half the marks for the final exam, regardless of their marks for the other three components.

This course has **zero tolerance** in students sharing their coding solutions in both the Python state space search labs and the Prolog labs. It is acceptable to discuss the problems at the level of ideas, but any sharing of code whatsoever will result in zero mark for those labs for all students who participate in sharing.

This course has **zero tolerance** in students submitting solutions generated by AI models. Any student submitting even one such solution will lose all the marks for those particular labs.

1. Python state space search labs

In these labs, students can solve **up to six problems of their choice** from the following list of problems that involve recursive backtracking and various state space search techniques for both one and two player games. Each problem, solved correctly so that it passes cleanly the teacher-provided automated tests without any errors or warnings, is worth **five points** to your course grade. The following Python problems from the collection "Additional Python problems" from the author's collection "109 Python Problems for CCPS 109" are acceptable for these labs:

- 70. "Baker–Norine dollar game"
- 72. "Balsam for the code"
- 74. "Maximal disk placement"
- 75. "Nice sequence"
- 78. "Independent dominating set"
- 79. "Vertex cover"
- 81. "Card row game"
- 84. "Tailfins and hamburgers"
- 86. "Tower of cubes"
- 89. "Balance of Power"
- 94. "Tower of Babel"

- 96. “Kayles”
- 97. “Out where the buses don't run”
- 98. ”SMETANA interpreter”
- 99. “The sharpest axes”
- 100. “Vidrach Itky Leda”
- 103. ”How's my coding? Call 1-800-3284778”
- 104. “Inverse pair sums”
- 105. “Blocking pawns”
- 106. “Boggles the mind”
- 107. “The round number round”
- 111. “Casinos hate this Toronto man!”
- 112. “Word bin packing”
- 113. “Probabilistic tic-tac-toe”
- 114. “Bandwidth minimization”
- 115. “Set splitting”
- 116. ”Domino poppers”
- 117. “Knight jam”
- 118. “Cubes on the trailer”
- 119. “Tom and Jerry”
- 120. “Pinch to a pound”
- 121. “Minimal Egyptian fractions”
- 122. “Unity partition”

To start solving these problems, download all the files from the GitHub repository linked above. Write all your solution functions in the same Python source code file `labs109.py`, so that you can run the automated tester script `tester109.py` to verify that your solutions are correct.

There are no partial marks given for labs that fail the automated tests. Only the problems listed above are accepted for these labs. Do not bother to ask if some other problem would be accepted.

Students should start working on these labs some time around Module Three after the state space technique lectures have been fully digested, and from there continue thinking about and working on these problems on their own time and convenience during the semester. These Python labs are due all as a single bunch during Module 12, to be uploaded to D2L as a single file `labs109.py`.

2. Prolog labs

The Prolog labs consist of a collection of problem specifications given in the PDF document "CCPS 721 Prolog Problems". Of the collection of problems given, **students get to freely**

choose up to ten problems to solve. Each problem is to be solved by defining the appropriate Prolog predicate whose name and arity are specified in the title of the problem specification. Working on these labs makes the students practice and apply their Prolog skills and techniques that were taught in Part Two of this course outline.

Every correctly solved problem that passes the instructor's test predicate (as given in the Prolog file `tester721.pl`) with flying colours is worth **two points** to the course grade, up to the maximum of thirty points for successfully solving ten Prolog problems.

Students should start working on these labs some time around Module Six after the Prolog lectures have been fully digested, and from there continue thinking about and working on these problems on their own time and convenience. These Prolog labs are due all as a single bunch during Module 12, to be uploaded to D2L.

3. Midterm exam

The midterm exam will be done in person in week 6. The midterm exam will be a multiple choice exam covering the course material from modules 1 to 3, the Prolog materials not being included in the midterm. The midterm exam is closed book, writing equipment only, and lasts two hours.

4. Final exam

The final exam of this course is done in-person at school, during the last scheduled session of the course. The final exam will be a multiple choice exam covering the course material from modules 7 to 12. The final exam is closed book, writing equipment only, and lasts three hours.

Individual modules

In the table presentation for each module, the textbook section titles listed in the row "AIMA 4" summarize the important ideas and concepts that form the learning goals of that module in this course. The links in the row "Central terms" lead to the corresponding Wikipedia articles for additional information about the central terms of each module.

Some of these related concepts are close enough to the actual course topics to warrant inclusion in these tables. However, such topics are marked with the symbol ♛ to indicate that they are not part of the main material.

Note that the chapter numbering of `aima-exercises` follows the third edition of AIMA, and can therefore be off by one compared to the chapter numbers of AIMA4.

1. Agents and environments

Central terms	Intelligent agent (classes) Rational agent Depth-first search Breadth-first search Iterative deepening A* Bidirectional search
AIMA 4	2.1, "Agents and Environments" 2.2, "Good Behaviour: The Concept of Rationality" 2.3, "The Nature of Environments" 2.4, "The Structure of Agents" 3.1, "Problem-Solving Agents" 3.3, "Search Algorithms" 3.4, "Uninformed Search Strategies" 3.5, "Informed (Heuristic) Search Strategies"

`aima-python`	`agents4e.py`
	`agents.ipynb`
	`search.py`
	`search.ipynb`

`aima-exercises`	2. Intelligent agents 3. Solving problems by searching

2. Adversarial search

Central terms	Minimax algorithm (negamax) Alpha-beta pruning Transposition table Principal variation search ♛ Expected value Solved game ♛ Combinatorial game theory ♛
AIMA 4	5.1, "Game Theory", 5.2, "Optimal Decisions in Games" 5.3, "Heuristic Alpha-Beta Tree Search" 5.4, "Monte Carlo Tree Search" 5.5, "Stochastic Games"
`aima-python`	`games4e.py` `games4e.ipynb`
`aima-exercises`	5. Adversarial search

3. Constraint satisfaction

Central terms	Hill climbing Simulated annealing Tabu search Genetic algorithm Backtracking Look-ahead Local consistency AC-3 algorithm Eight queens puzzle Verbal arithmetic SWI-Prolog Constraint Logic Programming ♛
AIMA 4	4.1, "Local Search and Optimization Problems" 6.1, "Defining Constraint Satisfaction Problems" 6.2, "Constraint Propagation: Inference in CSP's" 6.3, "Backtracking Search for CSP's" 6.4, "Local Search for CSP's" 6.5, "The Structure of Problems"
`aima-python`	`csp.py` `arc_consistency_heuristics.ipynp`
`aima-exercises`	4. Beyond classical search 6. Constraint satisfaction problems

4. Prolog programming language

The reading material of modules Four to Six consists of the example programs listed in each module, in addition to the Prolog tutorial chosen by the student from the earlier list of Prolog materials.

Example programs	`listdemo.pl`
	`builtindemo.pl`
	`pokerhand.pl`

5. Solving problems with Prolog

Example programs	tailrecdemo.pl bst.pl fractions.pl

6. Structure and interpretation of Prolog programs

Example programs	`crypt.pl`
	`homoiconic.pl`
	`clpfddemo.pl`

7. Propositional logic

Central terms	Logic Propositional logic (argument forms) What the Tortoise said to Achilles ♛ Law of excluded middle (not to be confused with the princ bivalence) List of rules of inference ♛ (these are only for reference for inte students, since the only inference rules that we need in this are *Modus Ponens* and *resolution*) Conjunctive normal form (Tseytin transformation ♛**)** Satisfiability (2-Satisfiability ♛) Forward chaining Backward chaining Resolution DPLL algorithm
AIMA 4	7.3, "Logic", 7.4, "Propositional Logic: A Very Simple Logic" 7.5, "Propositional Theorem Proving" 7.6, "Effective Propositional Model Checking"
`aima-python`	`logic4e.py` `logic.ipynp`
`aima-exercises`	7. Logical agents

8. Predicate logic

Central terms	First-order predicate logic Interpretation Horn clause Skolem normal form
AIMA 4	8.1, "Representation Revisited" 8.2, "Syntax and Semantics of First-Order Logic" 8.3, "Using First-Order Logic" 9.1, "Propositional vs. First-Order Inference" 9.2, "Unification and First-Order Inference" 9.3, "Forward Chaining" 9.4, "Backward Chaining" 9.5, "Resolution"
`aima-python`	`logic4e.py` `logic.ipynp`
`aima-exercises`	8. First-order logic 9. Inference in first-order logic

9. Bayesian probability

Central terms	Probability (axioms) Dutch book Conditional probability (Confusion of the inverse, Base rate fallacy) Independence Conditional independence Bayes' theorem Monty Hall problem (Principle of Restricted Choice ♛) Bayesian network Probability interpretations ♛
AIMA 4	12.1, "Acting Under Uncertainty", 12.2, "Basic Probability Notation" 12.3, "Inference Using Full Joint Distributions" 12.4, "Independence" 12.5, "Bayes' Rule and Its Use" 13.1, "Representing Knowledge in Uncertain Domain" 13.2, "The Semantics of Bayesian Networks" 13.4, "Approximate Inference for Bayesian Networks"
`aima-python`	`probability4e.py` `probability4e.ipynp`
`aima-exercises`	13. Quantifying uncertainty 14. Probabilistic reasoning

10. One-shot decisions

Central terms	Preference Utility (cardinal) Allais paradox Decision network ♛ Value of perfect information Game theory Simultaneous game Nash equilibrium (online bimatrix solver for arbitrary matrix games ♛) Matching pennies, Prisoner's Dilemma, Chicken (list of other g ♛)
AIMA 4	16.1, "Combining Beliefs and Desires under Uncertainty" 16.2, "The Basis of Utility Theory" 16.3, "Utility Functions" 16.6, "The Value of Information" 18.1, "Properties of Multiagent Environments" 18.2, "Non-Cooperative Game Theory" 18.3, "Co-operative Game Theory"
`aima-python`	`making_simple_decision4e.py`
`aima-exercises`	16. Decision theory

11. Supervised learning

Central terms	Machine learning Supervised learning (Inductive bias) Training, test and validation sets Bias-variance tradeoff Decision tree learning (Random forest) Overfitting Probably approximately correct learning Naive Bayes classifier Ensemble learning
AIMA 4	19.1, "Forms of Learning" 19.2, "Supervised Learning" 19.3, "Learning Decision Trees" 19.4, "Model Selection and Optimization" 19.5, "The Theory of Learning" 19.8, "Ensemble Learning" 20.1, "Statistical Learning"
`aima-python`	`learning4e.py`
`aima-exercises`	-

12. Reinforcement learning

Central terms	Markov decision process (MDP) Reinforcement learning Softmax function Temporal difference learning Q-learning TD-Gammon ♛ AlphaGo ♛ Kelly criterion ♛
AIMA 4	17.1, "Sequential Decision Problems" 17.2, "Algorithms for MDP's" 22.1, "Learning from Rewards" 22.2, "Passive Reinforcement Learning" 22.3, "Active Reinforcement Learning" 22.4, "Generalization in Reinforcement Learning"

`aima-python`	`mdp4e.py`
	`reinforcement-learning4e.py`
	`reinforcement-learning4e.ipynp`

`aima-exercises`	17. Making complex decisions 21. Reinforcement learning