

CCPS721 Artificial Intelligence I

This document contains the outline and course management form for **Fall 2020 term** for [CCPS 721 Artificial Intelligence I](#), as prepared and taught by [Ilkka Kokkarinen](#) for the Chang School of Continuing Education at Ryerson University, Toronto.

Ilkka Kokkarinen acts as the Subject Matter Expert for the purposes of virtualization of this course. The latest versions of all original 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 consists of twelve modules, followed by a take-home final exam to be completed over the final week. These modules are grouped into four parts, each part consisting of three modules that share their central theme. These four parts are largely independent of each other, and could be studied in almost any order.

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 "[Artificial Intelligence: A Modern Approach](#)", henceforth called **AIMA4**. This new fourth edition is a massive streamlined improvement over the already magnificent third edition, and recommended for all students without hesitation. Keeping up with the times, AIMA4 naturally 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 ([index](#)) that summarize the important material of AIMA4 in an alternative interactive format.

This course covers approximately half of the material in this book. The lecture outline follows the slightly old-fashioned course outline that practically every AIMA-based first course on artificial intelligence around the world has followed since the nineties, from the stratified air of Princeton and Berkeley down to the more can-do practical attitude of Wossamotta U. Our first course leaves out the classic AI applications 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 a more specialized second course after this.

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 actual book, so 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 official overhead slides, 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 implement its high-level ideas. Therefore this tome does not contain any material on Prolog programming beyond 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. Same as in other applications of programming, AI programming is these days best done in high-level languages such as Python aided with the frameworks of [numpy](#) and [scikit-learn](#).

Originally discovered back in the very different world of the seventies, Prolog no longer holds as much sway in artificial intelligence as it did back when its practical alternatives

were basically C and some other low-level grinders designed for efficiency of low-level operations instead of clarity and brevity of low-level thinking. 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 modifying symbolic structures, and discrete optimization problems that can be expressed and solved using the Constraint Logic Programming standard extension to the language. Prolog is therefore the language of choice of parsing and analysing 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 in the worldview of computer programs as special cases of symbolic formulas and their entailed consequences in the domain of integers.

Several excellent Prolog tutorials are available online. 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 theoretical, but nobody can deny how well it makes its case.

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 character codes used to document the instantiation expectations and behaviour of parameters.

Grading

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

1. Python project, 20%
2. Prolog labs, 30%
3. Take-home final exam, 50%

The final course grade of the sum of the scores for these three components. There is no requirement to get at least half the points from any individual component. (For example, a student who for some reason does not submit the Python project at all, could still theoretically get a grade of 80% by acing the Prolog labs and final exam.)

In all three components, students may consult any material they wish before finalizing their answers. However, the code and exam answers of each student must be their own individual work. Students may not ask for any help from any living human inside or outside this course before the results of that component have been tallied and finalized.

It is acceptable to search for and read existing forum posts and online articles about any particular topic, but especially unacceptable to post any new questions about the programming problem or the final exam topics.

1. Python project

The Python project is worth 20% of the total course grade. This project is designed to make the students apply the ideas and algorithms from Part One, search algorithms in state spaces of various computational problems. They should therefore start working with this project pretty soon after completing Module Three. The deadline for the Python project submission is during Module 8.

The specification and all files needed to complete this project are given in the public GitHub repository [ikokkari/ScrabbleRow](https://github.com/ikokkari/ScrabbleRow). The Python project for the Fall 2020 term has students write Python functions to fill in the given one-dimensional row of a Scrabble board with character tiles of their choice. To make this interesting, filling the given row with letters is constrained to produce only words that exist in the given English dictionary, while also having to respect the existing characters that have been already placed in that row.

The automated test script [scrabblerun.py](#) will be used to grade all submissions. Students can run this script any time they want to test the correctness and measure the performance of their chosen search algorithm. For grading purposes, the submitted solutions are run with `rounds = 100` and the same fixed value of `seed` privately chosen by the grading instructor. That way, all submissions will be evaluated with the exact same test cases, and no student can underhandedly optimize their function for some particular fixed value of `seed`.

The submissions are sorted based on their total scores, as calculated and reported by the test script. The highest-scoring submission earns its author the highest possible project mark of 20 points. At the other end of the score spectrum, the lowest-scoring submission that is deemed to have crossed the threshold of being "good enough for government work" gets the lowest passing project mark of 10 points. Project marks for all other students are interpolated between these two fixed points.

2. Prolog labs

The Prolog labs consist of a collection of problem specifications given in the PDF document "[CCPS 721 Prolog Problems](#)". Each problem is to be implemented as the Prolog predicate with the name and arity specified in its title. These labs let the students practice their skills and techniques in Prolog that were taught in modules from Four to Six.

Each correctly solved problem that passes the instructor's test predicate (as encoded in the Prolog file [tester721.pl](#)) with flying colours is worth three points to the course grade, up to the maximum of thirty points granted for solving ten lab problems correctly. Same as in all courses currently prepared or taught by Ilkka Kokkarinen, students get to freely choose which particular problems from this ever-evolving collection they want to work on.

Students should start working on these labs some time around Module Six, and continue thinking about and working on these problems on their own time and convenience. These labs are not to be submitted for grading individually during the course, but are all due as one bunch during Module 12.

3. Final exam

The format of the take-home final exam is **universal open book**. This means that students are free to use any online or printed material to compose their answers. However, the AIMA4 textbook is sufficient in that all questions of the final exam can be answered for full marks using no other material than the sections of AIMA4 listed below in each module description.

The final exam will contain both essay questions and multiple choice questions. There are no questions that require writing any code for the answer. The exam will cover all of the theory topics from Parts One, Three and Four in the course outline. However, the final exam will not contain any coding questions or questions about the theory or practice of Prolog, since that is already covered in the Prolog labs.

The final exam will be made available to students one week before the end of the course.

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

In modules Four to Six of Part Two of this course outline, the reading material consists of the example programs listed in each module, in addition to the Prolog tutorial chosen by the student from the list of Prolog materials.

Example programs	listdemo.pl builtindemo.pl clpfddemo.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 pokerhand.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 principle of bivalence) List of rules of inference 🖐 (these are only for reference for interested students, since the only inference rules that we need in this course are <i>Modus Ponens</i> and <i>resolution</i>) 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 form games 🖱️) Matching pennies , Prisoner's Dilemma , Chicken (list of other games 🖱️)
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