AI6th Ed IG Final JH v2 book and working how analysis future study

Artificial intelligence (Iqra University)

Instructor's Manual

Artificial Intelligence:
Structures and Strategies for Complex Problem Solving

Sixth Edition

George F. Luger

For further lecturer material, please visit:
www.pearsoned.co.uk/luger

ISBN 0 321 54591 5

ADDISON-WESLEY

Harlow, England • London • New York • Boston • San Francisco • Toronto • Sydney • Singapore • Hong Kong
Tokyo • Seoul • Taipei • New Delhi • Cape Town • Madrid • Mexico City • Amsterdam • Munich • Paris • Milan

Executive Editor	Michael Hirsch
Acquisitions Editor	Matt Goldstein
Editorial Assistant	Sarah Milmore
Managing Editor	Jeffrey Holcomb
Marketing Manager	Erin Davis
Composition	George F. Luger

ISBN-13: 978-0-321-54591-6

ISBN-10: 0-321-54591-5

Contents

Preface

This informal instructor's guide is intended to offer suggestions for teaching topics in my AI book, **Artificial Intelligence: Structures and Strategies for Complex Problem Solving**. This text, with copyright 2009, is the sixth edition of **Artificial Intelligence and the Design of Expert Systems**, original copyright 1989, and **Artificial Intelligence: Structures and Strategies for Complex Problem Solving**, second edition published in 1993, third edition 1998, fourth edition 2002, and fifth edition 2005. The title change for the second and later editions reflects the fact that this book is a wide ranging AI compendium, covering representation and search issues and stochastic technology in general, as well as the AI applications of machine learning, expert system design, reasoning under uncertainty, automated reasoning, natural language understanding, planning, and much more.

This instructor's manual is written in three sections. The Preface introduces this Instructor's Guide. Section I offers some "top level" suggestions for using the AI book, for instance, possible course outlines, assignments, and sample examinations.

Section II presents an overview and timeline for teaching the introductory material of the text, Chapters 1 to 9 (Parts I – III) plus the conclusion, Chapter 16 (Part VI). These chapters are the most often used parts of the book, and so we offer more detailed comments and teaching suggestions, as well as more fully worked-out exercises.

Section III of this guide presents the advanced chapters of the book. Instructors, after teaching the introductory material often want to sample from the remaining chapters of the book. We present some justification for these topics, an analysis of time required to present the material, and pointers to relevant related earlier parts of the text. The exercises will be less fully developed, as we find that instructors of these chapters often give fewer yet more detailed assignments. There remains a full set of exercises for these chapters, but this manual will answer relatively few of them. Any ideas for further exercises or comments on those already present are most welcomed.

Finally, the language material supporting the sixth edition are now available, in Prolog, Lisp, and Java, both on-line or in our supplementary Addison-Wesley (2009) book: AI Algorithms, Data Structures, and Idioms. Different instructors will use the language materials in different ways. At the University of New Mexico, I give four lectures on Prolog and then assign problems to be solved in Prolog and any other language. This is because the students in my course will already have had Scheme and Java and I feel they need to appreciate the declarative-style semantics of Prolog. What you choose depends on the qualifications of your students and the time allowed for the AI course. The language materials are intended to be tutorial introductions to building AI programs in these languages. Our main goal is to show the learner both the power of these languages as well as demonstrate how they can be used to implement the algorithms of the AI book.

One might ask what our philosophical orientation was for the creation of this book. This question as well as the intellectual orientation reflected in our mix of chapters, the theory and practice, our examples and applications is presented in the preface as well as in the introductory sections of each chapter of the book itself.

All our computer codes are available by anonymous ftp from ftp.cs.unm.edu, and through my homepage:

www.cs.unm.edu/~luger/

The code is organized at this ftp site by the chapter that it appears in the book.

There is also a web site available for students using the book. This site is designed from a chapter-by-chapter perspective that reflects the structure of the book. The site was designed by one of my AI students, Alejandro Cdebaca, and is evolving though student use and input. Over the years of teaching AI many students have contributed their ideas and example code to this repository. I trust this evolution is only beginning! The web site is available through my homepage.

I welcome any and all bug reports and suggestions. Please e-mail me at luger@cs.unm.edu.

We begin the book, Part I (Chapter 1), with an introduction to the philosophical tradition that produced the Artificial Intelligence phenomenon. We feel it important to give students this background to help them appreciate the scope and limits of AI. We end the book, Part VI (Chapter 16) with further comments on these philosophical issues. These comments include the inability to falsify the physical symbol system hypothesis and the nature of interpretation and embodiment that offer hard limits to the notion of artificial intelligence as science.

Finally, this sixth edition is the third edition produced without Bill Stubblefield's assistance as co-author. Although Bill's own career has taken him down different new paths in creative writing and software design, his assistance, guidance, and overall philosophy still infuses this book. I continue to acknowledge Bill's help in my efforts and even more, to appreciate his long friendship. In many situations I will continue to use the plural term "we" to reflect the fact that this book is much more than a one-person effort.

I also acknowledge a deep debt to more than three decades of students who have used and made important suggestions for this work. I also thank the very insightful reviewers and editors who have assisted me at every turn. Many of these students and reviewers are explicitly acknowledged in the prefaces of the various editions of the AI book itself.

May you enjoy using our book as we have enjoyed creating it!

George Luger
Albuquerque
June 2008

Section I

Philosophy, Sample Course Descriptions, and Examinations

SECTION I.1

Our Philosophy

As researchers in the area of artificial intelligence and practitioners in the design of expert systems and many other AI applications, we saw a need for an advanced introduction to the discipline. In creating "Artificial Intelligence: Structures and Strategies for Complex Problem Solving" we had three goals in mind:

1. To present AI technology along with its deep roots in the philosophical, mathematical, and computational traditions. AI as currently practiced is very much both part and product of the western scientific evolution.

2. To offer a broad focus on all AI, the European tradition as well as the American, Lisp language-oriented as well as Prolog, symbol-based, connectionist, and stochastic. A good programmer must be aware of all her tools.

3. Finally, we wished to base AI algorithms and techniques in their rightful place within modern computer science. Much of modern computing is a product of earlier research in AI (recursive data structures, object-based design, semantics of programming languages, and so on). Modern AI practice requires a strong foundation and grounding in traditional computing.

We intended that there be sufficient material in this book for several semesters of study. In the first semester, the foundational material is fairly clear, namely, the first 9 chapters of the book. We present all our introductory algorithms in both Lisp, Prolog, and Java in the supplementary materials; but we have found that, for an introductory quarter or semester, time permits only one language to be covered. At the University of New Mexico our CS majors have all had Lisp/Scheme in their introductory language courses, so in the 400 level AI course we teach only Prolog, and still give programming assignments in both Prolog and another language such as Lisp or Java. At other universities, of course, other options may well be more appropriate.

A second semester course in AI will of necessity be more eclectic. We prefer to cover different topics each time the advanced course is offered. We also feel an advanced course should require students to read and comment on AI research papers, and whenever we offer the advanced AI course, we collect, distribute, and require reading and analysis of 8 or 10 such papers.

In the next section we present a number of curriculum plans. First is a description of an introductory AI course, we call it an "Introduction to Artificial Intelligence". The course is divided into three sections, the first and last with evaluation through an examination, the middle section requiring the student to write a set of programs. After the course description we include two sample examinations, for the first and last thirds of the course. We also describe a typical programming assignment.

SECTION I.2

Sample Course Description: An Introduction to Artificial Intelligence

Textbook (GL), for reference purposes in the following descriptions:

> **Artificial Intelligence: Structures and Strategies for Complex Problem Solving By George F. Luger Addison-Wesley Pearson, 2009**

Week 1: Artificial Intelligence, its roots and scope (GL, ch. 1, Intro Part II)

- AI, an attempted definition
- Historical foundations
- Overview of application areas
- An introduction to representation and search

Weeks 2 & 3: The Predicate Calculus (GL, ch. 2)

- Representation languages
- The propositional calculus and its semantics
- The predicate calculus: syntax & semantics
- Inference: soundness, completeness
- The unification algorithm

Weeks 3 & 4: Structures and strategies for state space search (GL, ch. 3)

- Quick review of graphs
- State space search
- Data-driven and goal-driven search
- Breadth-first, depth-first, and depth-first iterative deepening search

Weeks 4: Heuristic search (GL, ch. 4).

- Priority queues
- A*
- Iterative deepening A*
- Beam search
- Two-person games
- Mini-Max and alpha-beta

Week 5: Stochastic Methods (GL, ch. 5)

- Quick review of counting principles

- Elements of probability
- Applications of the stochastic technology
- Bayes' theorem and its use

Week 6: Architectures for AI problem solving (GL, ch. 6)

- Recursive specification for queues, stacks, and priority queues
- The production system
- The blackboard

Weeks 7 & 8: PROLOG (Part II of AI Algorithms, Data Structures, and Idioms)

- The PROLOG environment
- Relational specifications and rule based constraints
- Abstract data types in PROLOG
- Graph search with the production system
- A PROLOG planner

Week 9: Introduction to AI representational schemes (GL, ch. 7)

- Issues in knowledge representation
- Semantic networks
- Conceptual dependencies
- Frames, scripts, and object systems
- The hybrid design: objects with rule sets

Week 10: Rule-based, case-based, and model-based systems (GL, ch. 8)

- Production system based search
- Rule stacks and the "why" query, proof trees and the "how" query
- Models of inductive reasoning
- The Stanford Certainty Factor algebra
- Knowledge engineering

Weeks 10 & 11: Building expert systems in PROLOG (GL, ch 6, AI Algorithms….)

- Meta-predicates in PROLOG
- The role of a meta-interpreter: PROLOG in PROLOG
- Rule-stacks, proof-trees, and certainty factor algebras in PROLOG
- Exshell, a back-chaining rule interpreter in PROLOG

Week 12: Reasoning in situations of uncertainty (GL, ch. 9)

- Examples of Abductive Inference
- Non-monotonic logic, belief revision
- Certainty factor algebras and fuzzy reasoning
- Stochastic models and Bayesian belief networks

Weeks 13 & 14: Advanced AI applications (GL, select appropriate chapters)

Week 15: Course summary and review (GL, ch. 16)

- The possibility of a science of intelligence
- Limitations and future research

There are two examinations, a mid-term and a final, each one hour long There are three programming assignments:

1. Building graph search algorithms in Prolog

    a) depth-first

    b) breadth-first

    c) best-first search

2. Building graph search algorithms in Lisp

    a) depth-first

    b) breadth-first

    c) best-first search

3. Using EXSHELL to build a rule based expert reasoning system

Course credit: Mid-term and final 40% each, programming assignments 20%. Sometimes a 10-15 page paper is assigned, the AI topic of the student's choice, and then the course credit is each exam 30%, the programs, 30%, the paper 10%.

SECTION I.3

Sample Examinations

Introduction to Artificial Intelligence EXAM Number 1

Name_____

No books or notes. The points for each question and percent of total credit follows the question number. Good luck.

1. (18) Consider the following story: All people that are not poor and are smart are happy. Those people that read are smart. John is wealthy. Helen can read and is wealthy. Happy people have exciting lives. Wealthy people are not poor. Find someone with an exciting life.

   (a) Translate the story into predicate calculus expressions.

   (b) Solve the problem with goal driven reasoning

   (c) Show the solution process with either the iterations of a production system or and/or graph search indicating the unifications and exactly where each is made.

2. (6) Define a production system. How can such a system be used for either data or goal driven problem solving?

3. (6) List three reasons why the production system offers an important "architecture" for computer based problem solving.

4. (8) Give the size, in terms of the branching factor B and the depth of search N, for the open list in each of the searches:

   (a) depth-first

   (b) breadth-first

   (c) best-first search

   (d) What is the size of the closed list in each of these situations?

5. (6) What is depth-first iterative deepening search, and why is it important?

6. (6) Define:

   (a) An A* (A star) algorithm

   (b) Admissibility

7. (6) Prove "A less informed admissible heuristic expands at least as much of the search space as a more informed admissible heuristic".

8. (8)

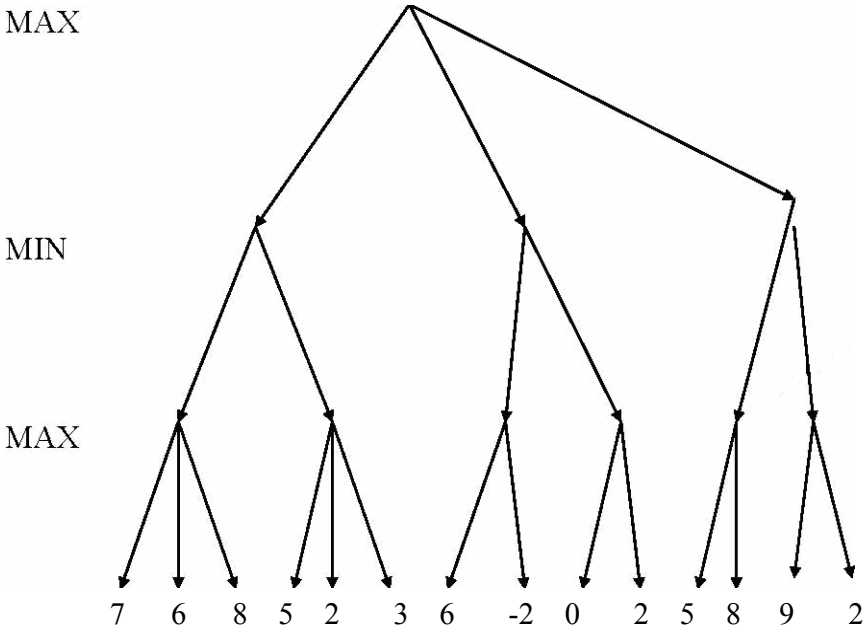
   (c) Define "most general unifier" for two predicate calculus expressions.

   (d) Why is the most general unifier important.

   (e) Trace, by generating the search tree, the unification algorithm on the two following expressions. Show all substitutions if the algorithm succeeds, otherwise show exactly where it fails.

   p(X, george, X) and p(fred, Y, george)

9. (10) Perform an alpha-beta prune of the following tree. Show the direction you are taking, the alpha and beta values at each appropriate place, and exactly where all pruning takes place. You are playing the top node and want to win. Heuristics are at leaf nodes.



10. (10) A blood test is 90% effective in detecting a disease. It also falsely diagnoses that a healthy person has the disease 3% of the time. If 10% of those tested have the disease, what is the probability that a person who tests positive will actually have the disease?

11. (14) TAKE HOME, Two hours work should do it. Bring it back in two days.

Take the sliding tile puzzle, problem 5, page 162. Use the legal moves described with the problem. Identify two heuristics for searching this space. Show whether or not each is admissible, monotonic, and whether or not one is more informed than the other.

Sample answers to exam 1:

1. This question is taken from Section 13.2.3 with a predicate calculus representation and search worked out in that same section. Goal driven reasoning would start with the predicate exciting(W).

2. See the definition in Section 6.2.3. This section also describes different control regimes for production systems.

3. See list of eight reasons in Section 6.2.

4. For a) and b) see the discussion in Section 3.2.3, for c) see the discussion in Section 4.2.1. For best-first, open is less than $B \times L$; how much less, depends on the pruning provided, on average, by the heuristic. d) the closed list always comes to approximate the size of the entire search space, $O(B^L)$ except in best-first where pruned off states don't become part of closed.

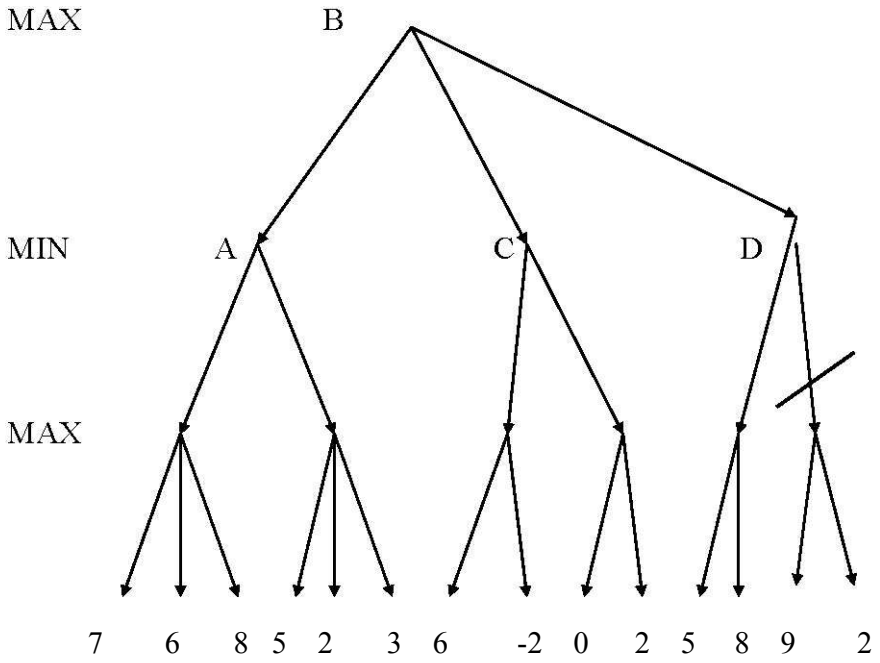
5. See Section 3.2.4 for this discussion.

6. See Section 4.3.1 for this discussion.

7. See one proof of this written up in Section 4.3.3.

8a) See definition, Section 2.3.2. For 8b) see the two paragraphs after the definition of the Most General Unifier. c) fails to unify because {fred/X} in the first argument, and then is passed to the X in the third argument. Eventually fred and george, different atoms, will fail to unify.

9.



Going left to right. The left-most deepest subtree gives a MAX of 8, which becomes the first β = 8 value at node A. After processing the second subtree (5,2,3), this β value changes to 5. The first α value at state B then becomes 5. The third deepest subtree (6,-2) gives a β at C of 6. The MAX of 2 from the (0,2) subtree would cause any further subtrees from C to be pruned (but there aren't any!). In the (5,8) subtree, 8 becomes a β at node D. When 9 is seen in the last deepest subtree, this causes the pruning of any further children of D. In this example the final state 2 (rightmost, deepest) is the only state not examined. A right to left search order would, of course, cause different pruning, but would result in the same eventual backed up value to B, namely 8.

10. p(good) = 0.25
    p(average) = 0.50
    p(bad) = 0.25
    p(accident|good) = 0.05
    p(accident|average) = 0.15
    p(accident|bad) = 0.25

$$\therefore p(\text{good}|\ \text{accident}) = \frac{p(\text{accident} \mid \text{good})}{p(\text{accident} \mid \text{good})\,p(\text{good}) + p(\text{accident} \mid \text{average})\,p(\text{average}) + p(\text{accident} \mid \text{bad})\,p(\text{bad})}$$

$$= (0.05)(0.25) / (\ (0.05)(0.25) + (0.15)(0.50) + (0.25)(0.25)\ )$$

$$= 0.083$$

11. We usually suggest the student begin this problem by working out the first several layers of the search space, on a very large piece of paper. This allows them to see what is going on and gradually find some heuristics that can prune down the search space. There are a number of these, only allowing movement of tiles towards the goal, preferring jumps to single moves. One interesting heuristic represents the board as a binary number, where 1s represent tiles out of place and 0s tiles in place. Thus 1110111 represents the start and 0000000 the goal; the heuristic reduces this number. Other heuristics simply calculate tiles on the "wrong side" of tiles of the opposite color.

    The final exam, is given at the end of the semester. In this particular semester we covered automated reasoning using resolution refutation systems from Chapter 14. We also discussed important aspects of programming in Prolog. We also have a general discussion on the summary material for the book, Chapter 16.

Name _____

Introduction to Artificial Intelligence Final Exam

Exam on final third of the course. No books or notes. Points (of 100) after each question number.

1. (20) Consider the following story:

   All people that are not poor and are smart are happy. Those people that read are smart. John is wealthy. Helen can read and is wealthy. Happy people have exciting lives. Wealthy people are not poor. Find someone with an exciting life.

   (a) Translate the story into predicate calculus expressions.

   (b) Translate the clauses into disjunctive normal form.

   (c) Show the resolution refutation solution process with an appropriate proof tree.

   (d) Use your example to demonstrate the answer extraction process for resolution refutations.

2. (12)

   (a) What is a multiple inheritance object system?

   (b) Name several computational tasks these representations are good for. Why are they good?

   (c) Why might a program designer opt for multiple inheritance over single inheritance?

3. (8)

   What is the role of data types in Prolog? How can the program designer address this issue, say if it was desired to have strong type constraints in a Prolog database?

4. (10)

   What is a hybrid expert system? Describe each of its components and how they are intended to interact together in problem solving.

5. (5 points test & 5 points extra credit)
   In what sense is the Prolog interpreter a theorem prover?

6. (9)

   (d) Describe conceptual dependencies.

   (e) Who first proposed this representational scheme and for what reason?

   (f) Describe two computational mechanisms for problem solving that might use this representation.

7. (12)

   Give three of the reasons why the MYCIN researchers created the Stanford Certainty factor algebra.

8. (8)

What is rapid prototyping and why is it an important piece of the expert system methodology? What in AI assists rapid prototyping?

9. (9)

(a) What is a meta-interpreter?

(b) Why is this problem solving tool important?

(c) Why are some languages better for designing meta-interpreters than others?

10. (7)

What do you see as the future of AI? Justify your answer.

Selected answers for the final exam:

1. This is the same question as on the mid-term, also question 1. A different solution process is required, see Section 13.1.3 of the text. The answer extraction process is presented in Section 13.2.5; compare solutions on Figures 13.6 and 13.9.

2. This is discussed for CLOS in Section 16.12.3. Issues of the expressive power (and dangers!) of using multiple inheritance systems are found throughout the text.

3. See discussion in Sections 15.6.1 and 15.6.2,

4. We briefly introduced this topic with the discussion in Section 8.3.4, balancing the strengths and weaknesses of different strong method systems. This simple overview was all the students were required to understand.

5. An answer may be found in Section 13.3. Here both a set of three axioms, including "unique names" and "domain closure" are presented that address the "negation as failure" issue. The use of "cut" and lack of an "occurs check" in PROLOG also limit PROLOG from being a sound and complete inferencing system.

6. Roger Schank created conceptual dependencies; answers in Section 7.1.3.

7. This discussion is in the paragraphs of Section 9.2.1.

8. See discussion in 8.1.2 and 8.1.3. Very high level languages or use of an expert system shell with rule-level modularity are important for rapid prototyping.

9. See discussion in Sections 8.2.1, 15.0, 15.6, and 16.7.

10. Anything goes here, as long as it is justified. One area we personally see in the future is very high level specification languages, with a declarative basis. The "limitations" presented in Chapter 16.3 also serve as important constraints.

Programming Assignments for Introduction to Artificial Intelligence

First assignment. Write search control algorithms in the Production system design pattern for:

1. Depth first search,

2. Breadth first search, and

3. Best first search.

These search control algorithms should be implemented in two languages, either Lisp or Prolog and some other language such as Java, and should be applied to two of the three problems:

1. The Missionaries and Cannibals problem (Three Missionaries and three Cannibals come to a river. There is a boat that will hold only two of them – either person can row the boat. If there are ever more Missionaries than Cannibals on either side of the river the Cannibals will get converted. Devise a series of crossings of the river so that all the characters get across and no one gets converted. This problem can also be stated in terms of eating Missionaries – enforcing both constraints makes the problem impossible.),

2. The 8-puzzle (p. 89), and

3. The Tile Problem (no. 5, p. 162).

Note that these three problems also give the students a nice opportunity to consider various representations and their appropriateness for each problem.

Second Assignment:

Use the software from EXSHELL (Prolog supplementary material) and add a set of rules to make a back-chaining rule based expert system. The application area is up to the student, but it should have at least 20 rules.

(Alternative software for the second assignment would be to use the Lisp expert system shell in the supplementary material or make a data-driven expert system using JESS or CLIPS. These expert system shells may be found on the web.)

SECTION I.4

Sample Course Description: Advanced Topics in AI

We now present a set of topics for an advanced AI course. These topics may vary according to instructor preference.

Advanced Topics in Artificial Intelligence

This will be a project based, discussion oriented focus on advanced issues in artificial intelligence. Topics will be presented from a research perspective, in that theoretical concerns and issues will be presented with each topic. Most topics will be centered on the design of a piece of software to exemplify the approach. About one of every four classes will be a journal-article-based discussion on a current or historically important topic in AI. These include: non-monotonic reasoning, truth maintenance systems, or alternative and synergistic reasoning strategies. There will be selected readings available to focus the discussions.

Text: Artificial Intelligence: Structures and Strategies for Complex Problem Solving, George Luger (GL), Addison-Wesley, 2009. Plus supplemental readings as necessary.

Sample supplementary readings: Computation and Intelligence: Collected Readings. George Luger, Editor, AAAI Press/MIT Press (1995).

I recommend this collection of readings for easy access to many important topics in AI.

An advanced course curriculum might include:

1. Natural Language Understanding, syntax, semantics, pragmatics. GL, chapters 5, 7, and 14. Project could be to write a recursive descent semantic net parser in PROLOG.

2. Building an expert system shell in LISP, C, or Java. Software includes ExShell or LISP-SHELL, chapters 15 and 16 (GL), or CLIPS, a C based data-driven shell from NASA. We also like JESS, a Java based rule based shell from Sandia National Labs (can be retrieved from the www).

3. An object-oriented design in Common LISP, CLOS, or C++, or Java, GL Chapter 16. The assignment is to build an object based simulation. See the thermostat example in Section 16.12.4.

4. A hybrid design, using C++, CLIPS, or Common LISP. See GL Sections 8.3.4 and 16.12.

5. The design of a synergistic reasoning system using the Blackboard technology. Design and integration of alternative search strategies.

6. Other research topics will be introduced throughout the semester.

Evaluation: No mid-term or final. Five assignments, each representing 20% of the course credit, selected from the topics above. Some examples:

(a) A one-page summary of each major research topic presented. There will be about eight of these, and they will be clearly identified during the course. (Required)

(b) A recursive descent, semantic net parser in Prolog. (chapter 15).

(c) An expert system designed in LISP, CLIPS, or JESS.

(d) An object system in CLOS, C++, or CLIPS.

(e) A hybrid design problem solver using CLIPS, CLOS, or C++.

(f) An application in the blackboard architecture (CLOS).

(g) The synergistic combination of multiple reasoning paradigms.

(h) A topic of your choice. This must be proposed to and approved by the instructor.

During an Advanced topics course in AI we feel it important that the students read and discuss papers. Some papers, classic in the field and important for seminal contributions, with references in L:

Computing Machinery and Intelligence, by Alan Turing, in (Feigenbaum and Feldman, 1963).

Newell and Simon (1976), The Turing Award Lecture, *Computer Science as Empirical Enquiry: Symbols and Search.*

Minsky (1975), *A Framework for Representing Knowledge*.

McCarthy and Hayes (1969), *Some Philosophical Problems from the Standpoint of Artificial Intelligence.*

Newell (1982), *The Knowledge Level*.

These are a sample of early papers in the AI field; we recommend more modern papers be added according to the tastes of the instructor and the direction the course takes. Many suggestions for papers may be found in the Epilogue and References at the end of each chapter of the AI book.

Section II

Introduction to the First Half of the Book

In Section II we introduce the first nine chapters of the book. These are the most taught materials in an introductory course, and thus this section of the Instructor Guide, as well as building AI algorithms in Java, Lisp, and Prolog, receives the greatest presentation space. We make a brief introduction to each chapter, as well as present a selection of worked-out exercises for that chapter.

SECTION II.1

Part I, Including Chapter 1

Chapter 1 AI: History and Applications

The intellectual and sociological roots of AI are exciting, and any serious practitioners of the discipline must be aware of these roots if they are not to be doomed to remake the mistakes of the past. We have found that most of our students are unaware not just of the origins of computing as a scientific discipline, but of the deeper intellectual, literary, and artistic foundations of the western intellectual tradition. Thus we were determined in designing our AI book to spend just a little time presenting the foundations of the discipline. In teaching a course in Artificial Intelligence, we usually spend a lecture and a half (1.5 hours) in introducing the concepts and applications that make up the practice of AI. If instructors do not feel comfortable discussing some of the philosophical issues such as Aristotle's matter-form distinction, Descartes' mind-body relationships, they can spend more time on the contributions of Babbage, Boole, Frege, Tarski, and Turing.

Turing's discussion on intelligence offers the instructor some important points for discussion. AI also presents important challenges to ethics and the accepted practice of society: Should a computer offer medical treatments? Should expert systems replace unskilled workers in societies' infrastructure? How about a computer psychiatrist? Under what circumstances are computer based problem solvers acceptable within our society?

Presenting the difficulties of AI application areas can also lead to fruitful discussions. What is so hard about understanding language? How can computers be taught to handle ambiguous language or other difficult problem solving situations? How can computer vision systems identify relevant features in a complex visual scene?

Exercises for Chapter 1

The exercises at the end of this chapter are intended to probe the deep philosophical, mathematical, and sociological issues introduced in Chapter 1, that underlie the study of Artificial Intelligence. Most of the exercises have no specific correct answers, but are rather presented to get the readers to begin to address difficult issues and to, in a small way, prepare themselves for the material in the rest of the book. We revisit many of these same questions again in Chapter 17.

SECTION II.2

Part II, Including Chapters 2 – 6

Introduction to Part II

Artificial Intelligence as Representation and Search

The most important concepts in introducing artificial intelligence are representation and search. These issues pervade our entire presentation. We created this small section to introduce the reader to the problems of knowledge representation and to the design of search strategies. The examples, such as predicate calculus, semantic nets, and graphs for search, are also specific pointers to material coming later in the book.

Perhaps the most important reason for placing an introduction to representation and search at this point in the book is to give a brief overview of the range of structures and strategies for problem solving before getting specific about particular representations (Chapters 2, 3, and 5) or instances of search algorithms (Chapters 3, 4, and 6).

About .75 hour is adequate for the instructor to present the material in this introduction. We suggest you use the figures to motivate the presentation and discussion. In the area of knowledge representation, the points brought up in the text can also guide discussion.

Chapter 2 The Predicate Calculus

From the beginnings of AI as a discipline, the Propositional and Predicate Calculi have been primary representation languages. Early 1950s work by Newell, Shaw, and Simon on the ***Logic Theorist*** was represented in a form of the propositional calculus. Even more sophisticated representations, such as semantic nets, frames, or objects, are often expressed as sets of and relations between predicate calculus expressions. The rule based expert system may be described and understood with a predicate calculus representation and simple inference mechanisms such as those found with the search strategies of Chapters 3 and 4 and the search architecture of the production system found in Chapter 6. Therefore we begin our presentation of AI tools with the propositional and predicate calculi.

Some groups of students may have had the propositional calculus as part of an earlier discrete mathematics course in the computer science or engineering curriculum. For these students, Section 2.1 may be treated as an optional review; if the material is presented, it requires about .75 hour.

We present the predicate calculus primarily as a language for representation, interpretation, and communication for humans and machines. Thus an alphabet of legitimate symbols is developed and combined to form terms. Terms are then combined expressions.

These expressions are further combined as descriptions of the world that may be interpreted and reasoned about.

We feel the most important theoretical concept developed in the chapter is the notion of a mathematics-based semantics. This is developed for both the propositional and predicate calculi. Inference rules and proofs are understood only in this context. Unification is seen as a tool enabling this process.

It usually requires three or four 1-hour lectures to finish the material of this chapter. We usually dwell extensively on the financial advice example at the end of the chapter, for a number of reasons:

1 It is developed with a predicate calculus representation.

2 It uses a practical interpretive environment.

3 It exemplifies use of inference rules and the unification algorithm.

4 The need for a search strategy points towards Chapters 3 and 4. In fact, we usually demonstrate this example with both a data-driven and a goal-driven search, and give a preliminary description of depth-first and breadth-first search.

Representation languages for the predicate calculus are arbitrary, with different texts often adopting different symbol systems. We selected the representational formalism that the PROLOG computer language requires in Chapter 15. This is done simply to cut down on the different syntactical forms the reader must master in going through our book.

Chapter 2 Selected Worked Exercises

1. Using truth tables, prove the identities of Section 2.1.2. For de Morgan's law: $\neg (P / Q) = (\neg P \wedge \neg Q)$

P	Q	¬P	¬Q	P ∨ Q	¬ (P ∨ Q)	¬ P ∧ ¬ Q	=
T	T	F	F	T	F	F	T
T	F	F	T	T	F	F	T
F	T	T	F	T	F	F	T
F	F	T	T	F	T	T	T

2. A new operator *exclusive-or* may be defined by the following truth table:

P	Q	P exor Q
T	T	F
T	F	T
F	T	T
F	F	F

Create a propositional calculus expression using only $\wedge, \vee$ and $\neg$ that is equivalent to P exor Q. Prove their equivalence using truth tables.

P	Q	$P \vee Q$	$P \wedge Q$	$\neg (P \wedge Q)$	$(P \vee Q) \wedge \neg(P \wedge Q)$
T	T	T	T	F	F
T	F	T	F	T	T
F	T	T	F	T	T
F	F	F	F	T	F

3. The logical operator $\Leftrightarrow$ is read if and only if. $P \Leftrightarrow Q$ is defined as being equivalent to $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$.

Based on this definition, show logical equivalence of $P \Leftrightarrow Q$ and $P \vee Q \Rightarrow P \wedge Q$:

(a) By using truth tables.

P	Q	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$	$P \vee Q$	$P \wedge Q$	$(P \vee Q) \Rightarrow (P \wedge Q)$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	F	F
F	F	T	F	F	T

5. (a) Prove that modus ponens is sound for propositional calculus. Hint: use truth tables to enumerate all possible interpretations, then show that wherever the premises are true, the first line in the truth table below, the conclusion is also true.

P	Q	$P \Rightarrow Q$	$P \wedge (P \Rightarrow Q)$	$(P \wedge (P \Rightarrow Q)) \Rightarrow Q$
T	T	T	T	T
T	F	F	F	F
F	T	T	F	F
F	F	T	F	F

6. Attempt to unify the following pairs of expressions. Either show their most general unifiers or explain why they will not unify.

(a) p(X,Y). and p(a,Z). Unifies with { a/X, Y/Z}.

(b) p(X,X). and p(a,b). Fails to unify since both a and b can not be substituted for X.

(c) ancestor(X,Y). and ancestor(bill, father(bill)). Unifies {bill/X, father(bill)/Y}.

(d) ancestor(X,father(X)). and ancestor(david, george). Unifies only if george is the father(david). Function evaluation should be added to the unification algorithm.

(e) q(X). and $\neg$ q(a). Will unify if provision is made for $\neg$ in the unification algorithm.

(f) p(X,a,Y). and p(Z,Z,b). Unifies with {a/X, a/Z, b/Y}.

10. Jane Doe has four dependents, a steady income of 30,000.00, and 15,000.00 in her savings account. Add the appropriate predicates describing her situation to the general

investment advisor of the example in Section 2.4 and perform the unifications and inferences needed to determine her suggested investment.

dependents(4) means **minincome(20000)** and by using rule 5, **savings(inadequate)**. Using rule 1, we can determine that **investment(savings)**.

11. Write a set of logical predicates that will perform simple automobile diagnostics (e.g., if the engine won't turn over and the lights won't come on, then the battery is bad). Don't try to be too elaborate, but cover the cases of bad battery, out of gas, bad spark plugs and bad starter motor.

    We have created several such knowledge bases in the supplementary programming materials.

12. The following story is quoted from N. Wirth's "Algorithms + data structures = programs" (Wirth 1976).

    I married a widow (let's call her W) who has a grown-up daughter (call her D). My father (F), who visited us quite often, fell in love with my step-daughter and married her. Hence my father became my son-in-law and my step-daughter became my mother. Some months later, my wife gave birth to a son (S1), who became the brother-in-law of my father, as well as my uncle. The wife of my father, that is, my step-daughter, also had a son (S2).

    Using predicate calculus, create a set of expressions that represent the situation in the above story. Add expressions defining basic family relationships such as the definition of father-in-law and use modus ponens on this system to prove the conclusion that "I am my own grandfather." This works fine with a little violation of the accepted semantics for family relationships!

Chapter 3 Structures and Strategies for State Space Search

Graph theory supports the algorithms of Chapter 3. We therefore place a review of graphs and the related terminology at the beginning of this chapter. If the reader has already covered this material, in a discrete mathematics course for instance, this section may be "optional" or serve as a quick review. If the material is covered, it requires about .5 hour.

We begin the chapter with the presentation of the finite state machine, including its use (the Moore Machine) as an "acceptor". This technology forms the basis for our later introduction of the probabilistic finite state machine as a tool for natural language processing.

All algorithms in Chapter 3 are based on exhaustive search. It is often helpful for the instructor to use forward pointers to heuristic search and the other more intelligent methods of later chapters. An example of this is to point out how the open and closed lists in depth-first and breadth-first search partition the graph. When the queue of breadth-first search is replaced by the priority queue, we get best-first search.

The roles of the open and closed lists are crucial for understanding all the search strategies. In fact the sizes of these lists can indicate when a particular search strategy is inappropriate. Depth-first search with iterative deepening is particularly important in this light.

The primary representational medium of Chapter 3 is the predicate calculus, whose inference rules lead us through an and or graph of possible choices. We continue to build with this representation until we see the rule based expert system in Chapter 8. Then we expand our tools with the representations of Chapter 7, widely used in the remaining chapters of the book.

This chapter offers many examples, and we encourage the reader to consider them carefully. Also the mathematical constraints on the size of the open and closed lists for the various search algorithms is important. The entire chapter, excluding the introduction to graphs, requires about 3 hours to cover.

Chapter 3 A Set of Worked Exercises

1. Define a Hamiltonian path as a path that uses every node of the graph exactly once. What conditions are necessary for such a path to exist? Is there such a path in the Konigsberg map?

   Yes, there are several possible Hamiltonian paths. For such a path to exist: 1. The graph must be connected, in that all states are reachable. 2. Define the degree of a node in the graph to be the number of paths entering the node. All nodes of the graph must have degree at least one (it is connected), and at most two nodes may have degree one (if these exist, they will be the beginning and ending states of the path).

2. Give the graph representation for the farmer, wolf, goat, and cabbage problem of figures in the Prolog supplementary programming material. Let the nodes represent states of the world; e.g., the farmer and the goat are on the west bank and the wolf and cabbage on the east. Discuss the advantages of each alternative representation at the beginning of Chapter 3 for searching this space.

   Figures such as those in the supplementary programming materials are much more intuitive and can help us explore the nature of the problem when we first encounter it. It can also support the relaxation of some of the constraints of the problem to begin to understand the search strategies and moves. Figures are more abstract and a generalization of the moves in the problem space, from which the legal moves will be selected. See traces of the program for actual solution paths through the graph.

4. Give an instance of the traveling salesperson problem for which the nearest-neighbor strategy fails to find an optimal path. Suggest another heuristic for this problem.

   The "nearest neighbor" heuristic is an example of a "greedy" search. A good way to generate such a situation is to make the cost of a neighbor near the end of the "nearest neighbor" path very high. At the end of the path there are often no alternatives but to take the high cost, and thus undo any advantage gained earlier in the search. Another algorithm would be to select the path just below the mean cost of all the paths from each state in the path.

5. "Hand run" the backtrack algorithm on the graph in Figure 3.29. Begin from state A. Keep track of the successive values of NSL, SL, CS, etc.

   We assume there is no goal state in the search space. Initialize:

   SL = [A]; NSL = [A]; DE = [ ]; CS = A;

LOOP	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F G B C D A]	[ ]
3	J	[J E B A]	[J K L E F G B C D A]	[ ]
4	K	[K E B A]	[K L E F G B C D A]	[J]
5	L	[L E B A]	[L E F G B C D A]	[K J]
6	F	[F B A]	[F G B C D A]	[E L K J]
7	G	[G B A]	[G B C D A]	[F E L K J]
8	M	[M G B A]	[M N H G B C D A]	[F E L K J]
9	N	[N G B A]	[N H G B C D A]	[M F E L K J]
10	H	[H G B A]	[H G B C D A]	[N M F E L K J]
11	O	[O H G B A]	[O P H G B C D A]	[N M F E L K J]
12	P	[P H G B A]	[M N H G B C D A]	[O N M F E L K J]
13	C	[C A]	[C D A]	[G H P O N M F E L K J]
14	D	[D A]	[D A]	[C G H P O N M F E L K J]
15	I	[I D A]	[I D A]	[C G H P O N M F E L K J]
16	R	[R I D A]	[R I D A]	[C G H P O N M F E L K J]
17	_	[ ]	[ ]	[A D I R C G H P O N M F E L K J]

   The algorithm returns FAIL

7. Determine whether goal- or data-driven search would be preferable for solving each of the following problems. Justify your answer.

   (a) Diagnosing mechanical problems in an automobile.

   Usually goal-driven. The search can be like MYCIN where the data is collected, suggesting possible goals (causes of the symptoms). Then each goal is checked.

   (b) You have met a person who claims to be your distant cousin, with a common ancestor named John Doe. You would like to verify her claim.

   A good approach here is a many frontier search: Develop several generations of John Doe's descendants. Develop several generations of your own and your (potential) cousin's ancestors. Look for people common to each graph.

   (c) Another person claims to be your distant cousin. He does not know the common ancestor's name but knows that it was no more than eight generations back. You would like to either find this ancestor or determine that she did not exist.

   In this case there is little alternative but to go back into the ancestor tree of each of you and to look for a common person.

   (d) A theorem prover for plane geometry.
   Theorem provers are almost universally goal-driven. The alternative of developing all possible theorems that follow from the axioms of geometry hoping to find the theorem

in question is both theoretically and computationally impossible for any interestin proof.

(e) A program for examining sonar readings and interpreting them, e.g., telling a large submarine from a small submarine from a whale from a school of fish.

This type problem is almost always data-driven, unless there are very few goals to investigate.

(f) An expert system that will help a human classify plants by species, genus, etc.

Usually data-driven. There are too many possible goals in most interesting cases.

10. Trace the good-dog problem of Example 3.3.4 in a data-driven fashion.

We assume a control regime that attempts all facts in their order in each rule. When a new result is concluded it is added to the facts and the process continues:

Fact 1 binds fred in rule 6.
Fact 2 binds fred and sam in rule 7.
Facts 3 and 4 add new fact location(sam, museum) in rule 9.
Facts 5 (and 1) add new fact gooddog(fred) in rule 6.
Rule 7 now fires to produce the answer.

11. Give another example of an and/or graph search problem. Develop part of the search space.

Prolog searches the and-or space of its rule or constraint specifications. Most rule based expert systems can be stated as an and-or graph search problem. Consider the automobile diagnostic program of the Prolog supplementary material.

13. Add rules defining adjectives and adverbs to the grammar of Example 3.3.6.

Np <-> adj n np <-> art adj n np <-> adj adj n
np <-> art adj adj n etc.
vp <-> v adv vp <-> v adv np etc.
adj <-> big adj <-> ugly adj <-> mean etc.
adv <-> easily adv <-> fiercely etc.

Several interesting problems from earlier editions of our book were left out of the 4th edition of Chapter 3. They are included here with their answers as possible extra exercises.

A. In depth-first search, the children of each node are generated and expanded in some order. What criteria should be considered in determining the best order in which to generate the child nodes?

A left-to-right or right-to-left order is assumed. This can be determined by rule order in the production set, c.f., Chapter 6. Heuristic information can be quite important on selecting states at any depth, however. You can expand the state of the cheapest cost, or

the state with the greatest confidence measure. We consider this in Chapter 4.

B. Would you use breadth-first or depth-first search for each of the following problems? What would you base your choice on?

(a) A chess playing program. Has to be depth-first, or at least guided (with heuristics, Chapter 4) depth-first. The branching factor is too large in chess to get to any interesting depth with exhaustive breadth-first search.

(b) A medical diagnostic program. Usually done depth-first. This allows the program to follow a line of reasoning and only change to a new goal or hypothesis when a previous one is confirmed or denied.

(c) A program to determine the best sequence of manufacturing steps to go from raw materials to a finished product. It should at least have a flavor of breadth-first, if not be totally breadth-first so that some process aren't satisfied to the detriment of later ones.

(d) A program that attempts to determine if two expressions in the propositional calculus are equivalent. Usually depth-first, as was the unify algorithm in chapter 2. This allows unification substitutions to be pushed forward through the remaining expression.

C. Answer questions 1 through 4 from Example 3.3.2.

1 Yes, h is true. a and c produce e. e and a produce h.

2 Yes, b is not part of the support of h.

3 Count search steps on the graph. Count each and premise.

4 There is the implicit assumption here that any fact not explicitly mentioned as true is in fact false. This, in AI work, is called the "closed world" assumption. We discuss it for reasoning schemes in Chapter 9.

Chapter 4 Heuristic Search

The search algorithms of Chapter 3 demonstrate the need of the "smarter" search presented in Chapter 4. The entire chapter requires about 4.5 hours to present: 4.1 about .75 hour, 4.2 about 1.25 hours, 4.3 about 1.25 hours, 4.4 about .75 hour, and 4.5 about .5 hours.

In the Fifth Edition we added a new section (4.1) on hill climbing and dynamic programming. These techniques are becoming more important in the stochastic analysis of natural language expressions, and thus we have inserted this material here for later use in the book.

The second part of the chapter, Section 4.2, is a natural follow-on from the material in Chapter 3. The open list is organized as a priority queue to develop the best-first search algorithm. This is then applied in a number of graph search examples.

Section 4.3 looks a little more deeply at the properties of heuristic measures. An "A Algorithm" is defined as a function of the depth of a state in the search plus the heuristic estimate of the distance it remains from a goal state. This algorithm is analyzed further into those heuristic measures that are bounded by the actual amount of work necessary to transform the considered state to a goal state. These search algorithms are called "A Star."

There are a number of properties, such as admissibility, that follow from the A Star property. The approach this book takes is to describe these properties without giving extensive proofs. These proofs are available in the literature, especially in Nilsson (1980) and Pearl (1984), for the interested reader. We feel the understanding and use of these results is a matter of engineering practice.

In Section 4.4 we apply heuristics to game playing, looking first at situations where we are able to exhaustively search the space of moves. Here we first present the MINIMAX procedure. Games usually require an heuristic that describes the competition in the play: not just how well an individual is doing, but accounting for the opponent's benefit and goals as well. MINIMAX can then propagate these constraints back up the search space to the present position.

The ALPHA-BETA procedure is next presented as an algorithm to limit search by using a form of the traditional "branch and bound" algorithm. ALPHA-BETA is able, on the average, to double the amount of graph searched with the same resource commitments of MINIMAX. A number of examples of both techniques are presented.

The final section of Chapter 4 discusses complexity issues in graph search, along with offering some general comments on the "information content VS compute cost" involved in applying the best-first search algorithm.

Chapter 4 A Set of Worked Exercises

1. Extend the "most wins" heuristic for tic-tac-toe two plies deeper in the search space of Figure 4.3. What is the total number of states examined using this heuristic? Would the traditional hill-climbing algorithm work in this situation? Why?

   To simplify things we will add two simple heuristics to the state evaluation.

2. Take a direct win or a "forking" win when possible. A "fork" move produces two possible next move victories where the opponent can only stop one of them.
3. Always select a move that avoids a direct loss.

   In this case the opponent will have only three possible responses to your selection in the left-most subtrree of Figure 4.3. Each of these has only one response. In the right-most

subtree of Figure 4.3, your opponent has six possible responses, but must select the one that avoids a direct loss. In this situation your next move is a "forking" win. Thus there are only four next states for your opponent to consider and each of these have a direct required response for a total of eight next-states to be considered to go two levels deeper in the search space. Only one of your opponents four choices will avoid a sure loss. This is in the left-subtree.

Yes, hill climbing works here. In fact the example given is an instance of hill climbing. In games, where backing out of an already selected move is usually not possible, a form of hill climbing is often appropriate.

4. Give two heuristics that a block-stacking program might use to solve problems of the form "stack block X on block Y." Is it admissible? Monotonic?

There is always the trivial heuristic of breadth-first search that is both admissible and monotonic. A better informed heuristic can look at the goal state desired, such as the stacking goal in the problem and set sub-goals that make the stacking conditions possible. In this case it would be to clear the two blocks, placing the cleared blocks on the table (and not on each other). Heuristics that focus on establishing preconditions for a top level goal to be possible are less work than accomplishing the full top level task, and are therefore admissible. Since these heuristic are organized by the top level goals they will most likely not be monotonic since other non-goal related subproblems may not be efficiently accomplished.

5. The sliding-tile puzzle consists of three black tiles, three white tiles, and an empty space in the configuration shown in Figure 4.29.

The puzzle has two legal moves with associated costs:

A tile may move into an adjacent empty location. This has a cost of 1.

A tile can hop over one or two other tiles into the empty position. This has a cost equal to the number of tiles jumped over.

The goal is to have all the white tiles to the left of all the black tiles. The position of the blank is not important.

(a) Analyze the state space with respect to complexity and looping.

(b) Propose a heuristic for solving this problem and analyze it with respect to admissibility, monotonicity, and informedness.

This is an excellent problem to use to gain understanding of the use of heuristics and the properties of Section 4.3. Breadth-first search is of course admissible, monotonic and very uninformed. Counting tiles out of the goal position is a simple useful heuristic that is admissible. A form of this heuristic would be to count all white tiles on the wrong side of each black tile. Counting tiles out of place is not monotonic, but is more informed than breadth-first search. An easy way to prove that a heuristic is non-monotonic is to watch its value change as it goes along a search path.

Other versions of this problem either limit the kinds of jumps possible or change the cost of jumps.

7. (a) As presented in the text, best-first search uses the closed list to implement loop detection. What would be the effect of eliminating this test and relying on the depth test, g(n), to detect loops? Compare the efficiencies of the two approaches.

   The depth test would detect loops, but would only eliminate them because the search had gone too deep without finding a solution and not because the search in fact cycled. This would only be a useful search technique if the cycling states remained at the head of the open list, and the depth cut off was less than the average size of the closed list. These parameters, including where the cycling states reoccurs on the open list could completely determine the very few times this "loop check" would be effective.

   (b) best-first search does not test a state to see if it is a goal until it is removed from the open list. This test could be performed when new states are generated. What effect would doing so have on the efficiency of the algorithm? Admissibility?

   It would improve the efficiency of the algorithm, in some cases radically. It would have no effect on admissibility, since admissibility guarantees the goal state will be found on the optimal path. Once on the open list, however, it may take a while to come to the front. Still this will eventually happen unless the branching in the space is really high.

8. Prove A^* is admissible. Hint: the proof should show that:

   (a) During its execution there is always a node on open that lies on an optimal path to the goal state.

   (b) If there is a path to a goal, A^* will terminate by finding the optimal path. A full proof may be found in (Nilsson 1980).

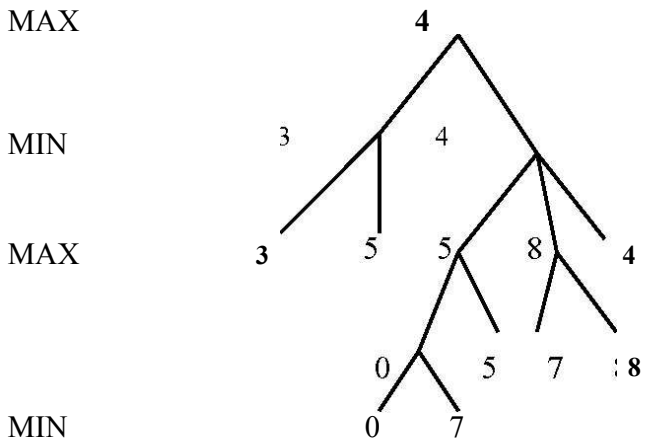
9. Does (or when does) admissibility imply monotonicity of a heuristic?

   Admissibility implies monotonicity only when the heuristic value of the goal is zero and any state in the search space may be used as a goal state, without revising the heuristic in the A^* algorithm. An example is the heuristic of "tiles out of place" in the 8-puzzle, that may be applied equally well to any state in the search space.

10 Prove that the set of states expanded by algorithm A^* is a subset of those examined by breadth-first search.

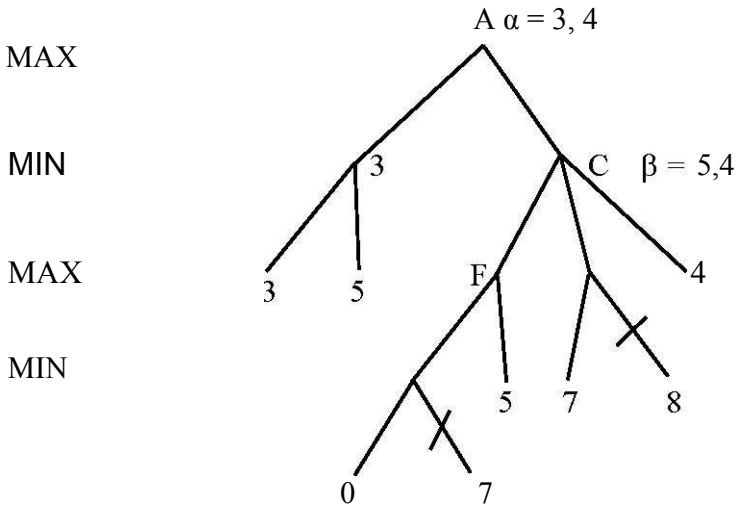
   Use the "more informed" proof of Section 4.2.3 with $h_1(n) = 0$, that is the heuristic measure of breadth-first search.

13. Perform MINIMAX on the tree shown in Figure 4.30.



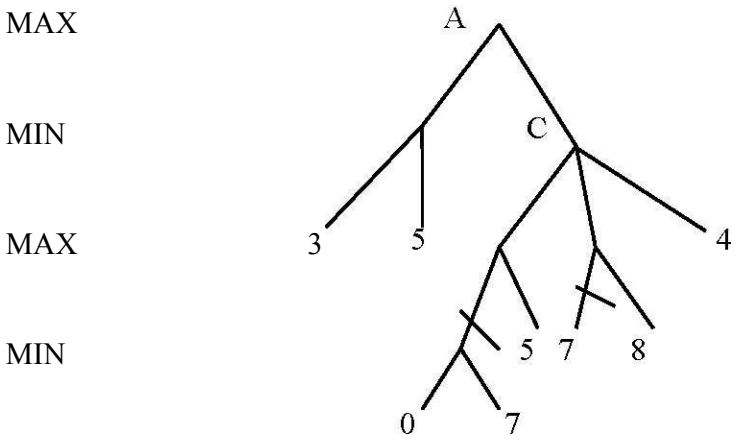
14. Perform a left-to-right alpha-beta prune on the tree of Exercise 13. Perform a right-to-left prune on the same tree. Discuss why different pruning occurs.

Left-to-right alpha-beta:



MIN on left-most deepest subtree (3,5) to get α at A = 3.
From next left-most deepest subtree (0,7), MIN of 0, prune.
F is now 5 and β at C = 5.
Subtree (7,8) is MAX, so with 7, 8 is β pruned.
Seeing the right-most 4, β at C = 4, and α at A = 4, the final backed up value.

Right-to-left alpha-beta:



Starting at the right-most deepest subtree, 4 gives β of C = 4.
Seeing 8 in subtree (7,8), 7 is β-pruned.
Seeing 5 next the entire subtree (7,8) is β-pruned.
A takes on the α value of 4.
5 and then 3 are examined (MIN) in the left-most subtree. The final value of A = 4.

16. Perform the alpha-beta pruning on the tic-tac-toe search of Figures 4.23 to 4.25. How many leaf nodes can be eliminated in each case?

    For Figure 4.23 we perform a left-to-right alpha-beta. All the states in the left-most deepest subtree are examined before selecting the MIN value of -1. This makes the α-value of the topmost node -1. In the middle subtree, when the first state is seen as -1, the remaining nodes (0,-1,0,-2) are all α-pruned. In the final subtree (1,2), the MIN determines that the final backed up value of the top node is 1.

    For Figure 4.24 we also perform a left-to-right alpha-beta. We look at all the six nodes of the left-most deepest subtree to get a MIN of 1 and an α value of the topmost (root) state of 1. The first node we see in the second to leftmost subtree (1,0,2,0,1,1) is 1, so we can α-prune the remaining five states. In the third subtree (2,2,3,1,2,2), we go as far as the fourth state, value of 1, before we can α-prune the remaining two states. Finally, the first state seen of the last subtree (1,1,0) is 1 so the final two states are α-pruned. The final value of the root node is 1.

    For Figure 4.25, we also go left-to-right. Looking at the left-most deepest subtree (1,2,1,2) we get a MIN value of 1 and an a value for the root node of 1. In all the remaining subtrees, the first state seen is a loss (- ∞), so all the (12) remaining states of each of these subtrees are α-pruned. The final value of the root node is 1.

17. A. Define an algorithm for heuristically searching and/or graphs. Note that all descendants of an and node must be solved in order to solve the parent. Thus, in computing heuristic estimates of costs to a goal, the estimate of the cost to solve an and node must be at least the sum of the estimates to solve the different branches.

    B. Use this algorithm to search the graph in Figure 4.31.

A complete algorithm for heuristically searching and/or graphs, along with a trace of its execution may be found in (Nilsson 1980).

Several interesting exercises from earlier editions of our book are omitted in the 4th edition. We include them (along with possible answers) as additional material for evaluation.

A. Why do game-playing programs work from a current state to a goal, rather than backward from a goal? What properties of a game might suggest using a backward strategy?

This is almost always because of the branching and depth measures of the search space. If the game tree is searchable, we would like to connect the present state to the goal, searching from either direction. Of course the A Algorithm, with all its properties, may be applied equally well in search from the goal that estimates the effort to get to the current state of the game!

B. Samuel's checker-playing program employed up to 16 different parameters in its evaluation polynomial. Suggest three evaluation functions for checkers that were not mentioned in the text.

We recommend the interested reader go to Samuel's original paper for a delightful discussion of the checker-playing program. He mentions the heuristics he employed and discusses how he was able to determine which of these were "effective." The reference is (Samuel 1959).

Chapter 5 Stochastic Methods

Stochastic methods are becoming increasingly important in the practice of AI, especially in the areas of natural language understanding (Chapter 15), and the creation of graphical models and other tools for reasoning in uncertain situations (Chapter 9). As a result we have created this new chapter for the fifth edition.

The foundation for a theory of probabilities is the principles of counting. These, including the addition and multiplication rules, permutations and combinations are developed in Section 5.1. If the students have a good background in discrete mathematics, this section (marked "optional") may be skipped.

The stochastic methodology is introduced in detail in Section 5.2, with definitions of probabilities, random variables, independence, and related concepts. Prior and posterior probabilities are also defined. Section 5.3 presents probabilistic finite state machines as well as a number of examples applying stochastic reasoning.

Section 5.4 is the culmination of the chapter where Bayes' theorem is presented. This equation links the prior expectations of a situation to new knowledge, given those expectations. The theorem is demonstrated with several examples.

The technology in this chapter points forward to the presentation of Bayesian Belief Networks and other graphical models in Chapter 9.

We estimate that it takes about 4.5 hours of lecture time to cover Chapter 5. Section 5.1, (optional), requires about 1.5 hours, Section 5.2 about 1 hour, Section 5.3 about .5 hour, and Section 5.4 about 1.5 hours.

Chapter 5 Selected Worked Exercises

1. A fair six-sided die is tossed five times and the numbers up are recorded in a sequence. How many different sequences are there?

$$_6P_5 == \frac{6!}{(6-5)!} = 720$$

2. Find the number of distinguishable permutations of the letters in the word MISSISSIPPI, of the letters of the word ASSOCIATIVE.

MISSISSIPPI

$$\frac{_{11}P_{11}}{_4P_4 * {}_4P_4 * {}_2P_2} = \frac{11!}{4!4!2!} == 34650$$

ASSOCIATIVE

$$\frac{_{11}P_{11}}{_2P_2 * {}_2P_2 * {}_2P_2} == \frac{11!}{2!2!2!} = 4989600$$

3. Suppose that an urn contains 15 balls, of which eight are red and seven are black. In how many ways can five balls be chosen so that:

   (a) all five are red? all five are black?

   (b) two are red and three are black?

   (c) at least two are black?

   (a) All five are red: $_8C_5 = 56$
   All five are black: $_7C_5 = 21$

   (b) $_8C_2 * {}_7C_3 = 56$

   (c) Total possible outcomes of choosing 5 balls from 15 are $_{15}C_5 = 3003$
   $|\{\text{at least 2 black}\}| = |\{\text{all possible outcomes}\}| - |\{\text{ 0 black }\}| - |\{1 \text{ black}\}|$
   $= {}_{15}C_5 - 56 - {}_7C_1 * {}_8C_4 = 3003 - 56 - 490$
   $= 2457$

4. How many ways can a committee of three faculty members and two students be selected from a group of five faculty and seven students?

$$_5C_3 *_7 C_2 = \frac{5!}{(5-3)!3!} * \frac{7!}{(7-2)!2!} = 10 * 21 = 210$$

5. In a survey of 250 television viewers, 88 like to watch news, 98 like to watch sports, and 94 like to watch comedy. 33 people like to watch news and sports, 31 like to watch sports and comedy, and 35 like to watch news and comedy. 10 people like to watch all three. Suppose a person from this group is picked at random:

   (a) What is the probability that they watch news but not sports?

   (b) What is the probability that they watch news or sports but not comedy?

   (c) What is the probability that they watch neither sports nor news?

   Let N be the set of people who watch news,

   S be the set of people who watch sports and

   C be the set of people who watch comedy.

   $|N| = 88, |S| = 98, |C| = 94$

   $|N \cap S| = 33$

   $|S \cap C| = 31$

   $|N \cap C| = 35$

   $|N \cap S \cap C| = 10$

p({watch news but not sports}) = $|N \cap S| / 250 = (88 - 33) / 250 = 0.22$

p({watch news or sports but not comedy}) = $|(N \cup S) \cap C| / 250 = ((|N|+|S| - |N \cap S|) - |N \cap S \cap C|) / 250 = (88 + 98 - 33 - 10) / 250 = 0.572$

p({watch neither news nor sports) = $| C| - |C \cup N| - |C \cup S| + | C \cap N \cap S| / 250 = (94 - 35 - 31 + 10) / 250 = 0.152$

Note: a Venn diagram can help to demonstrate these relationships.

6. What is the probability that a four-digit integer with no beginning zeros:

   (a) Has 3, 5, or 7 as a digit?

   (b) Begins with 3, ends with 5, or has 7 as a digit?

   Number of all possible four digit integers with no beginning 0s :
   9*10*10*10 = 9000

   (a) p({contains 3,5 or 7 as a digit}) = 1 – p({contains none of 3,5 or 7})
   = 1 – (6* 7* 7* 7 /9000)= 0.77

(b) p({begins with 3, or ends with 5 or has 7 as a digit}) = p({begins with 3}) + p({ends with 5}) – p({begins with 3 and ends with 5}) + = 10*10*10/9000 + 9*10*10/9000 – 10*10/9000 + .. = 1800/9000 + …

7. Two dice are tossed. Find the probability that their sum is:

(a) 4

(b) 7 or an even number

(c) 10 or greater

The sample space has total 6*6 = 36 possible outcomes.

(a) The combinations that give 4 are 1,3; 3,1; 2,2. So p({sum=4}) = 3/36 = 1/12

(b) The combinations that give 7 are 1,6; 6,1; 2,5; 5,2; 3,4; 4,3.
So p({sum=7}) = 6/36 = 1/6
p({Sum is a even number}) = p(both dice are even) + p(both dice are odd) = (3/6 * 3/6) + (3/6 * 3/6) = 1/2
Using the additive property of distinct outcomes
p({sum being 7 or an even number}) =1/6 + 1/2 = 2/3

(c) The combinations that give 10 are 4,6; 6,4;
The combinations that give 11 are 5,6; 6,5;
The combinations that give 12 are 6,6.
So p({sum is 10 or greater}) = 5*(1/6 * 1/6) = 5/36

8. A card is drawn from the usual fifty-two card deck. What is the probability of:

(a) drawing a face card (jack, queen, king or ace)

(b) drawing a queen or a spade

(c) drawing a face card or a club

(a) (4 + 4 + 4 + 4) / 52 = 4/13

(b) p(queen) + p(spade) – p(spade queen) = 4/52 + 13/52 – 1/52= 4/13

(c) p(face ∪ club) = p(face) + p(club) – p(club faces)
= 4/13 + 13/52 – 4/52 = 25/52

9. What is the probability of being dealt the following hands in a five card poker game (from the normal deck of fifty-two cards)?

(a) A "flush" or all cards from the same suit.

(b) A "full house" or two cards of the same value and three cards of another value.

(c) A "royal flush" or the ten, jack, queen, king, and ace all of the same suit.

(a) Number of combinations is the number of different suits, times the number of ways to pick all 5 cards from the same suit(combination of 13 cards taken 5 at a time). So

(b) p({5 cards from the same suit}) = $({}_4C_1 * {}_{13}C_5) / {}_5C_{52}$ = 99/4998 = 0.02

(c) **$({}_{13}C_1 * {}_4C_2 * ({}_{12}C_1 * {}_4C_3)) / {}_5C_{52}$???**

(d) 4 * (1/52 * 1/51 * 1/50 * 1/49) = 1/1624350 = 6.16E–7

11. Suppose that we are playing a game where we toss a die and then receive the amount of dollars equal to the value of the die. For example, if a 3 comes up we receive $3. If it costs us $4 to play this game, is this reasonable?

The chance of getting any face p(1), p(2), p(3) …, p(6) is equal, 1/6.

The expected value

Ex(E) = 2 p(6) + 1 p(5) + 0 p(4) + (–1) p(3) + (–2) p(2) + (–3) p(1) = (– 3) 1/6 = –

0.5 On average we lose $0.5 per play, so the game is not reasonable.

12. Consider the situation where bit strings of length four are randomly generated. Demonstrate whether or not the event of production of bit strings containing an even number of 1s is independent of the event of producing bit strings that end in a 1.

Number of all possible strings is 2*2*2*2 = 16

p({even number of 1s}) = 8/16 = 1/2

{1111, 1100, 0110, 0011, 1010, 0101, 1001, 0000}

p({end with 1}) = 8/16 = 1/2

{0001, 1001, 0101, 0111, 0011, 1101, 1011, 1111}

p({even number of 1s} ∩ {end with 1}) = 4/16 = ¼

{1001, 0101, 0011, 1111}

A and B are independent if and only if p(A ∩ B) = p(A) * p(B) , and

p({even number of 1s} ∩ {end with 1}) = p({even number of 1s}) × p({end with 1}),

∴ The two events are independent.

13. Show that the statement p(A,B|C) = p(A|C) p(B|C) is equivalent to both p(A|B,C) = p(A|C) and p(B|A,C) = p(B|C).

p(A,B|C) = p(A|C) p(B|C) shows A and B are conditionally independent given C. So given C, knowing B is true does not effect A's probability of being true, which is equivalent to p(A|B,C)=p(A|C); Given C, whether A is true or not does not effect B's probability of being true, that is p(B|A,C)=p(B|C).

14. In manufacturing a product, 85% of the products that are produced are not defective. Of the products inspected, 10% of the good ones are seen as defective and not shipped,

whereas only 5% of the defective products are approved and shipped. If a product is shipped, what is the probability that it is defective?

p(good) = 0.85 → p(defective) = 1 – 0.85 = 0.15 p(not shipped | good) = 0.10 → p(shipped |good) = 1 – 0.10 = 0.90 p(shipped | defective) = 0.05

$$\therefore p(\text{defective} \mid \text{shipped}) = \frac{p(\text{shipped} \mid \text{defective})\, p(\text{defective})}{p(\text{shipped} \mid \text{defective})\, p(\text{defective}) + p(\text{shipped} \mid \text{good})\, p(\text{good})}$$

$$= (0.05)(0.15) / ((0.05)(0.15) + (0.90)(0.85))$$

$$= 0.0097$$

15. A blood test is 90% effective in detecting a disease. It also falsely diagnoses that a healthy person has the disease 3% of the time. If 10% of those tested have the disease, what is the probability that a person who tests positive will actually have the disease?

p(disease) = 0.10 → p(healthy) = 0.90

p(positive | disease) = 0.90

p(positive | healthy) = 0.03

$$\therefore p(\text{disease} \mid \text{positive}) = \frac{p(\text{shipped} \mid \text{disease})\, p(\text{disease})}{p(\text{shipped} \mid \text{disease})\, p(\text{disease}) + p(\text{shipped} \mid \text{healthy})\, p(\text{healthy})}$$

$$= (0.90)(0.10) / ((0.90)(0.10) + (0.03)(0.90))$$

$$= 0.77$$

16. Suppose an automobile insurance company classifies a driver as good, average, or bad. Of all their insured drivers, 25% are classified good, 50% are average, and 25% are bad. Suppose for the coming year, a good driver has a 5% chance of having an accident, and average driver has 15% chance of having an accident, and a bad driver has a 25% chance. If you had an accident in the past year what is the probability that you are a good driver?

p(good) = 0.25 p(average) =

0.50 p(bad) = 0.25

p(accident|good) = 0.05

p(accident|average) = 0.15

p(accident|bad) = 0.25

$$\therefore p(\text{good}|\ \text{accident}) = \frac{p(\text{accident} \mid \text{good})}{p(\text{accident} \mid \text{good})\ p(\text{good}) + p(\text{accident} \mid \text{average})\ p(\text{average}) + p(\text{accident}|\text{bad}\ p(\text{bad})}$$

$$= (0.05)(0.25) / ((0.05)(0.25)+ (0.15)(0.50) +(0.25)(0.25))$$

$$= 0.083$$

17. Three prisoners, A, B, C are in their cells. They are told that one of them will be executed the next day and the others will be pardoned. Only the governor knows who will be executed. Prisoner A asks the guard a favor. "Please ask the governor who will be executed, and then tell either prisoner B or C that they will be pardoned." The guard does as was asked and then comes back and tells prisoner A that he has told prisoner B that he (B) will be pardoned. What are prisoner A's chances of being executed, given this message? Is there more information than before his request to the guard? This problem is adapted from Pearl (1988).

    Prior probability of each of A, B, C get executed is 1/3

A	B	C	p
t	f	f	1/3
f	t	f	1/3
f	f	t	1/3

    $\therefore$ p(A| not B) = 1/2

    Therefore, there is now more information than before his request to the guard.

Chapter 6 Control and Implementation of State Space Search

In Chapters 3 and 4 we considered search in some detail and it remains to create "software architectures" for implementation of that search. Let us briefly review what the state space model of problem solving has given us:

- Representation of a problem's solution as a path through a number of states of a graph.
- Search to systematically test different paths to a goal.
- A backtrack algorithm to systematically search this state space.
- Lists to keep explicit records of states under consideration:

  The **open** list to record the "frontier" of the search

  Closed to keep track of states already considered.
- Use of the stack, queue, and priority queue to implement depth-
- first, breadth-first, and best-first searches.

Chapter 6 presents the final material in Part II of the book. It requires about 4 lecture hours to present in full detail. Sections 6.1 and 6.2, recursion based and pattern-directed search

requires about 1 hour to present. We have found that many students from good computer science backgrounds already understand issues such as having the open list for depth-first search be the "stack" or set of current activation records plus related pointers found in the implementation of a recursive environment. If students do not have this knowledge, it may be worth spending more time on these concepts. This also helps clarify why the open list must be explicitly handled in breadth-first and best-first search. Section 6.3, the production system, requires about 1.5 hours to present; Blackboards, Section 6.5, requires .5 hour to present.

Thus, Chapter 6 builds these search algorithms in the context of more general programming paradigms. Section 6.1 introduces recursion as a high-level technique for implementing search algorithms. Recursive search, implements depth-first search with backtracking in a more concise, natural fashion than in Chapter 3 and forms the basis for many of the algorithms in this text. Recursive search is augmented through the use of unification to search the state space generated by predicate calculus assertions. This pattern-directed search algorithm (Section 6.2) is the basis of Prolog and many of the expert system shells discussed in Chapters 8, and the supplementary materials.

Next, in Section 6.3 we introduce production systems, a general architecture for pattern-directed problem solving that has been used extensively both to model human problem as well as to build expert systems and other AI applications.

Finally, in Section 6.4, we present another AI problem-solving technique, the blackboard, where multiple independent problem solvers, called "knowledge sources" interactively build solutions in a central global database, called the blackboard.

In the supplementary materials of the book, we implement many of these software architectures in Prolog, Lisp, and Java.

Chapter 6 A Subset of Worked Exercises

1. The member algorithm of Section 6.1.1 recursively determines whether a given element is a member of a list.

   (a) Write an algorithm to count the number of elements in a list.

   (b) Write an algorithm to count the number of atoms in a list.

   (The distinction between atoms and elements is that an element may itself be a list.)

   The approach question 1a. requires is to move recursively down the list counting each element in the list. That element may either be an atom or a structure, i.e., even if the structure has atoms within it, it is only counted once, as an element of the list. 1b. requires that each element of the list be broken down into its constituent atoms, and then these atoms counted. This is often called "car-cdr" recursion, because not only does the algorithm recurse down the list but also takes apart each structure that makes up the list. An example answer may be found in the length and count_atoms LISP functions in Figure 16.3, Section 16.1.7.

2. Write a recursive algorithm (using open and closed lists) to implement breadth-first search. Does recursion allow the omission of the open list when implementing breadth-first search? Explain.

The open list may not be omitted, since it is maintained as a queue and not a stack. The recursive environment mimics the action of a stack. A queue, on the other hand must be explicitly maintained, as the results of each recursive call are understood. This is discussed in detail for PROLOG, and an algorithm given, in Section 15.4.

3. Trace the execution of the recursive depth-first search algorithm (the version that does not use an open list) on the state space of Figure 6.5.

Of course, the result will be the same as the version that does use the open list! The recursive environment itself operates as a stack. To get the results required, assume loop detection in the algorithm, assume a depth bound, and evaluate the next state by taking the production rules in order. Note that the order of production rules generating the graph of Figure 3.14 differs from the order of the rules in Figure 6.5. Thus, your result will have a different search space.

5. Using the **move** and **path** definitions for the knight's tour of Section 6.2, trace the execution of **pattern_search** on the goals:

(a) path(1,9).

(b) path(1,5).

(c) path(7,6).

When the move predicates are attempted in order, c leads to looping in the search. It can be important to discuss loop detection and backtracking in this situation.

5a. Produce a trace like that of Figure 6.7:

Iteration	current state	goal state	conflict rules	rule fired
0	1	9	1,2	1
1	8	9	13,14	13
2	3	9	5,6	5
3	4	9	7,8	7
4	9	9		halt

5c. Again, as in Figure 6.6

1teration	current state	goal state	conflict rules	rule fired
0	7	6	11,12	11
1	2	6	3,4	3
2	9	6	15,16	15
3	2	6	3,4	3
4	9	6	15,16	15

etc.

Obviously we need loop detection in our algorithm for the production system; we have this, of course, when we implement our search algorithms as in Chapter 3. Note also that the original query had a one move direct solution that was missed by our algorithm. Best-first search can remedy this problem.

7. Using the rule in Example 5.3.3 as a model, write the eight move rules needed for the full 8 x 8 version of the knight's tour.

The forms of these rules will vary, of course but they will all add + or – 2 and 1 to the row and column measures of the present state to produce the new state. At some time in the production of the new state we much check that it will be on the board. This can be done as in the text by checking the current state to see if the present rule can be applied, or as in our example below assuming the current state is okay and then constraining the new state. The representation used here is a PROLOG-like version of the predicate calculus where each state or board position is called "state(Row, Col):"

CONDITION: state(Row, Col)

ACTION: state(NewRow, NewCol) ^ NewRow is Row – 2 ^ NewRow > 0
NewCol is Col + 1 ^ NewCol < 9.

The failure of any of the and constraints gets the system to consider the next rule or backtrack. The reader can create the seven other rules.

9. Consider the financial advisor problem discussed in Chapters 2, 3, and 4. Using predicate calculus as a representation language:

(a) Write the problem explicitly as a production system.

(b) Generate the state space and stages of working memory for the data-driven solution to the example in Chapter 3.

(c) Repeat b for a goal-driven solution.

For this answer we refer to the rules presented in Section 2.4.

9a. Put rules 1 through 8 in the production memory. Place the minsavings and minincome functions in the production memory also, to be used whenever a rule requires this mathematical calculation performed. Facts 9, 10, and 11 will be obtained from the used as needed. Now run the production as in either 9b or 9c.

9b. Go through the rules in order until you produce the result "savings(X)" for some X. When no premise is yet known, as in the first three rules in the first iteration, go on to the next rule. Finally, in rule 4 we are asked about savings and dependents. There are no rules that can conclude these results so the system goes to the user to obtain this information. When it is supplied, minsavings is calculated and then either rule 4 or rule 5 fires to conclude about the adequacy of the savings account. This new fact is added as a result that is now known by the system. At this point if it is breadth – first data driven

search we continue through the remaining 3 rules, obtaining information on earnings and calculating minincome. Otherwise, if we are doing depth-first search we take the result of rule 4 or 5 and go back to rule 1 again.

9c. Search should look something like Figure 3.26, depending only on the rule ordering. How this graph is generated depends on whether the goal driven search is breadth-first or depth-first.

10. Section 6.2.3 presented the general conflict resolution strategies of refraction, recency, and specificity. Propose and justify two more strategies.

    Two easy additional strategies would be to keep the response as general as possible, and thus not cut down on the set of possible solutions deducible by the production system, and similarity to previous firing conditions that might make it easier to perform repeated similar tasks, such as iteration or looping. These strategies would have to be spelled out in terms of the currently activated rules.

11. Suggest two applications appropriate for solution using the blackboard architecture. Briefly characterize the organization of the blackboard and knowledge sources for each implementation.

    The blackboard is now a very popular problem-solving paradigm. One important application would be to monitor all the processes that must come together in computer-assisted manufacturing. Another might be to describe all the diverse reasoning processes that can come together in a diagnosis problem (Skinner and Luger 1992).

SECTION II.3

Part III, Including Chapters 7, 8, and 9

Part III Representation and Intelligence: The AI Challenge

Knowledge representation is certainly one of the most important topics presented in our book. In Chapter 7 we begin our presentation of non predicate logic based representations. We do this with an historical focus, going back to the early history of AI and presenting the representation schemes in an "evolutionary order." This allows the reader to see how the strengths of one representation find their way into succeeding approaches while the weak aspects improve or tend to disappear in the next generation of representation.

Within our historical perspective we also break all representation AI representational schemes down into four rough categories. These categories are not intended to be definitive but rather to assist the reader in getting a general perspective. Over the past 25 years, numerous representational schemes have been proposed and implemented, each of them having its own strengths and weaknesses. Mylopoulos and Levesque (1984) have classified these into four categories:

1 **Logical representation schemes**. This class of representations uses expressions in formal logic to represent a knowledge base. Inference rules and proof procedures apply this knowledge to problem instances. These we saw in chapters 2 through 8.

2 **Procedural representation schemes**. Procedural schemes represent knowledge as a set of instructions for solving a problem. This contrasts with the declarative representations provided by logic and semantic networks. A production rule system is an example of this approach.

3 **Network representation schemes.** Network representations capture knowledge as a graph in which the nodes represent objects or concepts in the problem domain and the arcs represent relations or associations between them.

4 **Structured representation schemes**. Structured representation languages extend networks by allowing each node to be a complex data structure consisting of named slots with attached values.

We conclude Chapter 7 with the presentation of two novel architectures, the "Copycat" approach created by Hofstadter (1995) and his students, especially Melanie Mitchell (1993). We also present Rodney Brooks' (1991a) subsumption architecture, proposed in the context of robot exploration.

Chapter 8 presents strong-method problem solving, with a special focus on the rule-based expert system. We have sections in Chapter 8 on "knowledge engineering" and we emphasize the "quick prototyping" approach to program design. Chapter 8 also considers the case-based and model-based reasoning paradigms. We also present a small expert systems to demonstrate how search and explanation facilities work in data-driven and goal-driven expert systems.

present a small section (8.3.4) on hybrid systems that focus on the strengths and weaknesses of each approach.

We conclude Chapter 8 with a section on Planning. We begin with the traditional approaches of STRIPS (Fikes and Nilsson 1971) and consider issues such as the frame axioms (McCarthy 1980). We conclude with several examples of planning in NASA's recent space flights.

In Chapter 9, we consider reasoning in uncertain situations. We begin with the more traditional logic-based approaches, where truth-maintenance and nonmonotonic reasoning systems are presented. In Section 9.2, from a mathematically less restricted viewpoint, we present the Stanford Certainty Factor Algebra – the inference system for a number of early expert systems, including MYCIN – and fuzzy reasoning systems. We finish Chapter 9 with the stochastic approach to reasoning, including Bayes rule, Bayesian belief networks, and other graphical models.

Chapter 7 Knowledge Representation

In Chapter 7 we concentrate our efforts on network and structured representations. The chapter requires about 4.5 hours to present. Our introductory section, describes some of the inadequacies of logic-based representations. This material requires about .5 hour to present.

The remainder of 7.1 goes into the early network representations and requires about an hour to present. It begins with the semantic net, finds its precedents in even earlier research and discusses its foundation in psychological evidence. Some of the shortcomings of semantic nets are removed in the standardization found in conceptual dependencies. Here the notion of the existence of "semantic primitives" is discussed. Scripts – an extension of conceptual dependencies – and frames round out Section 7.1. These more structured approaches attempt to represent complex situations in the world, such as a child's birthday party or going to a restaurant. This section requires about 1.5 hours to present.

John Sowa's conceptual graphs are presented in 7.2. These are seen as a combination of network representation that has a direct translation into predicate calculus. It requires about 1 hour to present Section 7.2, including the graph operations such as join and restrict used to do inference in this scheme. This section requires about .75 hour to present.

In Section 7.3.1, we consider several more modern approaches to the issue of representation. Prof Hofstadter and his graduate students, especially Melanie Mitchell, created a hybrid architecture called "copycat". This approach was designed to capture analogical reasoning situations, for example to complete the sequence a, b, c is to c, b, a as x, y, z is to ???. Copycat had rudimentary knowledge of "order" in symbol systems such as the English alphabet, along with notions such as "precedes" and "follows". The Copycat approach has been recently extended to domains such as robotics, where a robot, by exploring, learns patterns in its world, including "walls" and "doors" (Lewis and Luger 2000).

Rodney Brooks, with his "subsumption" architecture, questions whether any explicit representation at all is required for problem solving. As Brooks' claims "let the world be its own representation (1991a). Finally, in Section 7.4, we present the "agents" approach to distributed proiblem solving. We define what we feel an agent system requires: that problem solving be situated, autonomous, flexible, and social. We describe in detail what these requirements entail, give a large number of references to agent based problem solving, and end the section with a short critique of the agent approach. It requires about 1.5 hours to cover Section 7.4.

The Epilogue and References, besides recommending further readings in the area, describes variants of the traditional logic based schemes that are also important in the task of representation (about .25 hour to present). In the supplementary materials for our book we build many of the representational schemes presented in Chapter 7 in Java, Lisp, and Prolog.

Chapter 7 A Selection of Worked Exercises

The exercises for Chapter 7 are fairly open ended. The reader has not yet been exposed to the language tools for building the representations described in this chapter. We encourage the interested reader to look ahead to the supplementary materials for this book for suggestions on building semantic nets, frames, parsers, and object systems in Prolog, Java, and Lisp.

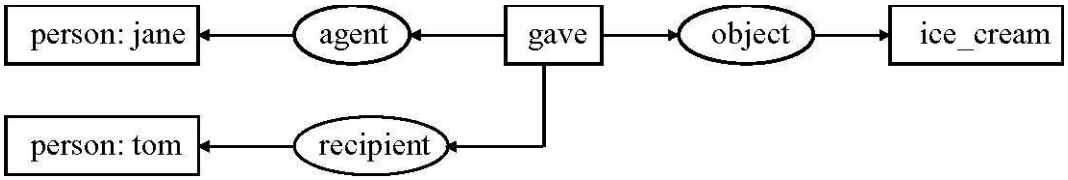
2. In Section 7.1 we presented some of the arguments against the use of logic for representing commonsense knowledge. Make an argument for the use of logic in representing this knowledge.

   There are two important papers the reader should consider in answering this question. From the "power of advanced AI representation" viewpoint is Minsky's "frame paper" (Minsky 1975). In this paper Minsky sets out an impressive set of arguments against the "logicist" representational viewpoint. On the other side of the issue McCarthy and Hayes (1968) feel that the various forms of logic already available are sufficient for all representational needs. Much of McCarthy's later research (1977; 1980) has been to build logical models for representation in nonmonotonic and default situations.

3. Translate each of the following sentences into predicate calculus, conceptual dependencies, and conceptual graphs:

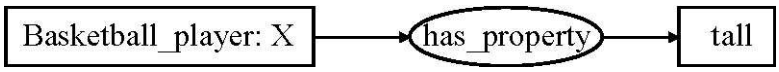
   "Jane gave Tom an ice cream cone."
   give(jane, ice_cream_cone, tom, past_time).

   For conceptual dependency, use form of bottom half of Figure 7.15.



"Basketball players are tall." has_property (basketball_players, tall).

For Conceptual dependency, see number 2, Figure 7.16.



All these sentences can have multiple renditions in most formalisms. Usually the verb becomes the key to building the representation, as shown above.

5. Translate the conceptual graphs of Figure 7.29 into English sentences.

One rendition of the two graphs:
John eats soup with his hand.
Kate believes it is false that John likes pizza.

6. The operations join and restrict define a generalization ordering on conceptual graphs. Show that the generalization relation is transitive.

There are two issues here. First all properties of a representation are true in its generalization. Secondly, any type extension of an entity will be found in its generalization, in fact is part of its generalization. Thus, if a generalization of a generalization is not a generalization, then one of the original generalizations is not a generalization.

7. Specialization of conceptual graphs using join and restrict is not a truth-preserving operation. Give an example that demonstrates that the restriction of a true graph is not necessarily true. However, the generalization of a true graph is always true; prove this.

Consider the example of g2 and g3 of Figure 7.22. Since all animals are not dogs, g3 need not be true whenever g2 is. If a CG is true then its generalization is still a true representation for that original situation. If the generalization was not true, then the original wasn't true.

8. Define a specialized representation language to describe the activities of a public library. This language will be a set of concepts and relations using conceptual graphs. Do the same thing for a retail business. What concepts and relations would these two languages have in common? Which would exist in both languages but have a different meaning?

They need have nothing in common. One hopes, in building representations for understanding languages, for example, there are some de facto invariances.

9. Translate the conceptual graphs of Figure 7.28 into predicate calculus.

One attempt:
eats(john, soup, hand).
believes(kate, hates(john, pizza)).

Note the problem with handling negatives in the predicate calculus.

13. Construct a hierarchy of subtypes for the concept vehicle; for example, subtypes of vehicle might be land-vehicle or ocean-vehicle. These would have further subtypes. Is this best represented as a tree, lattice, or general graph? Do the same for the concept move; for the concept angry.

    Multi-inheritance hierarchies are generally found to be more expressive. In fact any multiple inheritance graph can be extended to be a lattice, and can also be rebuilt as a single inheritance hierarchy.

14. Construct a type hierarchy in which some types do not have a common supertype. Add types to make this a lattice. Could this hierarchy be expressed using tree inheritance? What problems would arise in doing so?

    Often it is easier and more natural to design a multiple inheritance hierarchy. Single inheritance systems can often, even though equivalent for representation, be much more cumbersome, and require more extensive search. It is often the case that an "everything" (or "null" type is created to be a common supertype (subtype) for all entities, thus preserving the lattice structure of the representation.

15. Each of the following sequences of characters is generated according to some general rule. Describe a representation that could be used to represent the rules for:

    (a) 2,4,6,8, ...

    (b) 1,2,4,8,16, ...

    (c) 1,1,2,3,5,8, ...

    (d) 1,a,2,c,3,f,4, ...

    (e) o,t,t,f,f,s,s, ...

    Define a representation that could be used to represent any rules for sequences of numbers and letters.

    This type of problem is often handled by defining a language that can handle the regularities, for instance on a, b, and c, the language would be the integers, and the regularities handled by the order, for instance even integers or doubling. d requires two languages, integers and characters, and again a measure on their order. Copycat (Section 7.3.2) created such an architecture. Finally, e shows that even clever representations are often not good enough: the sequence represents the first letters of the names of the integers, one, two, three...

17. Describe a representation that could be used in a program to solve analogy problems like that in Figure 7.30. This class of problems was addressed by T. G. Evans (1968). The representation must be capable of representing the essential features of size, shape, and relative position.

    This type of problem is usually addresses with a set of predicate calculus descriptions for each situation and an analogical mapping going between sets of descriptions. We need descriptors for inside, outside, large, small, square circle,

triangle. Gentner's (1983) structure mapping is built on this approach. A description of the first two situations might include:

large(triangle).	large(circle).
small(circle).	small(triangle).
inside(triangle, circle).	inside(circle, triangle).

A mapping of the first state onto the second might indicate that a mapping of the third onto a potential answer would exchange the parameters of the inside predicate as well as exchanging the arguments of the large and small predicates.

18. Brooks' paper (1991a) offers an important discussion on the role of representation in traditional AI. Read this paper, and comment on the limitations of explicit, general-purpose representational schemes.

    To list but a few of the criticisms that Brooks made of the then "traditional" representational schemes:

    1 They were "pre-interpreted" in that the program designers often built into the representations the "meaning" expected to be required to operate in the world. This could often prove to be "incorrect" in the context where the representation was needed. Expert system designers often call this the "brittleness" problem.

    2 The traditional representations were "in the computer" (and thus had a highly rationalist flavor). The "embodied" tradition, which claim Brooks as one of its early advocates, suggest that the world itself be used as part of the representation.

    3 Read the article for a further critique of the then traditional approach to representation.

19. At the end of Section 7.3.1, there are five potential issues that Brooks' subsumption architecture (1991a) must address to offer a successful general-purpose approach to problem-solving. Pick one or more of these and comment on it (them).

    Some potential criticisms of Brooks include:

    1 With no global (or even distributed) memory how can the subsumption architecture keep track of what it has "learned" through its experiences with the world? Will it have to rediscover how to deal with things each time they are encountered? Hardly very intelligent.

    2 Will the subsumption architecture generalize? Brooks has constructed examples with relatively few layers (on the order of half a dozen). Can this be indefinitely expanded to capture a larger scope of intellectual activity?

    3 Read Brooks and the conclusion of Section 7.3.1 for further ideas.

21. There were a number of important issues presented near the end of Section 7.4 related to the creation of an agent-oriented solutions to problems. Pick one of these and discuss the issue.

Some thoughts:

1. Comment on the problems of distributed problem solvers in general. How can meaningful communication be set up in a distributed environment to support necessary cooperation among agents?

2. A related criticism is to ask how an agent can work at optimal intelligence levels with limited knowledge of a situation? Or is full knowledge of what's going on required? Can agents play their appropriate roles in a society without full knowledge of what they are part of? Many social scientists will claim that such agents can and in fact do.

3. Look for further ideas in the concluding paragraphs of Section 7.4.

22. Suppose you were designing an agent system to represent an American football or alternatively a soccer team. For agents to cooperate in a defensive or in a scoring maneuver, they must have some idea of each other's plans and possible responses to situations. How might you build a model of another cooperating agent's goals and plans?

The Robocup annual competitions as well as other cooperative robotics activities are now accessible on the WWW as well as the normal AI literature (Veloso et al. 2000).

Chapter 8 Strong Method Problem Solving

A very large amount of the material introduced so far in this AI book comes together in Chapter 8. There is representation with the predicate calculus (Chapter 2), graphs and the design of search algorithms (Chapter 3), heuristic search (Chapter 4), architectures for organizing search, especially the production system (Chapter 6), and finally, the introduction of alternative representational schemes (Chapter 7). In an important way, this chapter should bring a lot of theoretical issues together in the design of a practical product: the symbol based expert system problem solvers. It requires about 4.25 hours to present.

An expert system is a knowledge-based program that provides "expert quality" solutions to problems in a specific domain. Generally, its knowledge is extracted from human experts in the domain and it attempts to emulate their performance. As with skilled humans, expert systems tend to be specialists, focusing on a narrow set of problems. Also, like humans, their knowledge is both theoretical and practical, having been perfected through experience in the domain. Unlike a human, however, current programs have difficulty learning from their own experience; their knowledge must usually be extracted from humans and encoded in a formal language. This encoding is the major task facing expert system builders.

We begin the chapter by considering the overall design of a symbol-based expert system, discussing briefly each of its major components. We next present issues in problem selection and knowledge engineering. Exploratory programming is presented

as an important tool in coming to understand the problem domain. Likewise we show that conceptual models have a very important role in the knowledge acquisition process. This introduction takes 1.25 hours.

We next show, in Section 8.2, rule-based search with the production system, and how proof trees and rule stacks can provide explanations. We look at both data-driven and goal-driven inferencing systems. This is an important and exciting section as it demonstrates important tools for monitoring the search process as it moves through the graph. These techniques are built into Exshell in Chapter 15 and LISPshell in Chapter
16. This section requires about 1 hour to present.

In Section 8.3.2 we present model-based, case-based, and hybrid expert systems. With rule-based systems these tools share a commitment to an explicit symbol based representation. However the techniques supporting case-based and model-based approaches are quite different from each other as well as from the rule-based system. In this section we go into those differences, along with presenting numerous examples, including a propulsion system for a NASA space system (Williams and Nayak 1996a).

With the hybrid system, designers attempt to address the shortcomings and brittleness of one system with the power and strengths of another. For example, why should a rule-based system have to continually "think through" the alternatives of a common problem that it regularly sees? Rather a complementary case-based system should handle these situations by a simple lookup. Furthermore, when the causal relationships of a system are available, as in a set of electronic circuits or a NASA space vehicle propulsion system, why not just reason directly with these cause-effect relationships? The section ends with a discussion of the plusses and minuses of each approach to symbol-based expert systems. Section 8.3 requires about 1.25 hours to present.

Section 8.4 is an introduction to planning and requires about .75 hour to present. We present this material from an historical viewpoint beginning with the early STRIPS planner. We also introduce the early historical concerns of the "frame" problem. The frame problem asks the open philosophical question that if you model only the changes your robot makes in your world how can you keep track of those things that aren't changing? Or even worse, how can you describe things that are inadvertently side affected by the things that the robot does? And what about the things that aren't even mentioned by your specifications, for example that the robot's battery might be defective? These and other issues are addressed with the "frame axioms". We end this section with two more modern systems, Nilsson (1995) and his students at Stanford's "teleo-reactive" planner and an example of a NASA spaceflight planner (Williams and Nayak 1996a).

The material in this chapter points the way to using the PROLOG and LISP expert system software of Chapters 15 and 16. The software for a number of these application programs is available in the course software, and we often introduce it immediately after this chapter, with the assignment to build in LISP or PROLOG one or two expert system programs.

Chapter 8 Comments on Selected Exercises

1. In Section 8.2 we introduced a set of rules for diagnosing automobile problems. Identify possible knowledge engineers, domain experts, and potential end users for such an application. Discuss the expectations, abilities, and needs of each of these groups.

   It is important here to actually interview each of the domain experts and end users to find out typical problems and solutions offered. This interview process, as noted in the text, is quite an art, requiring patience and care to get results.

2. Take Exercise 1 above. Create in English or pseudocode 15 if ... then rules (other than those prescribed in Section 8.2) to describe relations within this domain. Create a graph to represent the relationships within these 15 rules.

   We have written a small expert system in PROLOG to accomplish exactly this task. It can be found in Section 15.7.2, with the graph searched presented as Figure 14.6. In this section a number of further "car rules" are presented. It is important to note, however, that these are still very simple systems and that a realistic system requires many days collaboration with the human expert.

3. Consider the graph of Exercise 2 above. Do you recommend data-driven or goal-driven search? breadth-first or depth-first search? In what ways could heuristics assist the search? Justify your answers to these questions.

   We used the Exshell software, provided with the course software to prune our auto mechanic advisor. Because of the problem, a goal-driven, depth-first search seemed to be appropriate. This is also the search that Exshell provides. Although each of our rules can take a confidence measure, and Exshell can propagate those confidences through the search space and prune whenever the confidence goes below a threshold, it does not perform a search prioritized by the confidence measures. Exshell also halts with its first recommended fix for the problem rather than exhaustively considering all solutions. The trace of Exshell running the automobile rules can also be found in Section 15.7.2.

5. Implement an expert system using a commercial shell program. These are widely available for personal computers as well as larger machines. We especially recommend CLIPS from NASA (Giarratano and Riley 1989).

   Many such pieces of software exist and we recommend going to the WWW or the AI literature to locate them. For an interesting modern tool we also recommend JESS, a Java based expert system shell. This tool is available through Sandia National Laboratories and details may be found on the WWW.

6. Critique the shell you used to do Exercise 5. What are its strengths and weaknesses? What would you do to improve it? Was it appropriate to your problem? What problems are best suited to that tool?

This is an excellent exercise, once the effort has been put into acquiring and using the software. Your critique should include whether or not the problem you selected was a good fit for the software you chose!

8. Read and comment on the paper "Diagnosis Based on Description of Structure and Function" (Davis et al., 1982.)

   This early paper laid much of the groundwork for the model-based reasoning approach to expert systems. One shortcoming of this early paper is that, besides diagnosis of electronic circuits, there were not a large number of domains suggested where this approach might be fruitful. Can you suggest several more application areas?

9. Read one of the early papers using model-based reasoning to teach children arithmetic (Brown and Burton 1978) or electronic skills (Brown and VanLehn 1980). Comment on this approach.

   There is an interesting connection between model-based systems and teaching. A group of cognitive scientists, including those referenced above, have conjectured that students learning an application domain, such as the diagnosis of electronic circuit boards, build mental models of the structure and functioning of these systems. If their mental model is identical to the actual system then it can be used for diagnosis. More importantly, when a human student fails to understand what is going on in the application domain it is conjectured that repair of their mental model is an appropriate remedy.

12. Read and comment on "Improving Human Decision Making through Case-Based Decision Aiding" (Kolodner 1991).

    Again, as in question 9, the assumption of a connection between a human's "mental model" of a situation and the actual functionality of that system is implied. In this sense the computer can prove a valuable assistant in human problem solving. Do you feel comfortable with this approach? Why or why not?

13. Create the remaining frame axioms necessary for the four operators pickup, putdown, stack, and unstack described in rules 4 through 7 of Section 8.4.

    We do the first operator: For pickup, the picked up block can create another clear block if it is true that that block was stacked on some other block. Also the picked up block is no longer on the other block.

    After creating these axioms (which are straightforward and similar to those already created), the reader should question the complexity cost of adding this number of support axioms to a system in order to maintain a sound inferencing scheme. Does this approach seem to be equivalent to the way humans work in their world?

15. Show how the add and delete lists can be used to replace the frame axioms in the generation of STATE 2 from STATE 1 in Section 5.4.

First look at all the frame axioms for the system. These tell how each operator affects the predicate descriptor of a state. Translate these constraints over into the add and delete lists for the operator. A frame axiom system is still implicit in the system, since we now assume that all and the only actions of the operator are contained in the add and delete constraints.

Are these then equivalent systems? Compare them for: a) getting the same job done, b) the mathematical "soundness" of their respective inferencing schemes, and c) the complexity costs of their inferencing schemes.

18. Show two more incompatible (precondition) subgoals in the blocks world operators of Figure 8.19.

Other incompatible subgoals in Figure 8.19:

(a) In the start state try to stack a on c. To do this, since c is already clear, b must be taken off a. Now if b is inadvertently stacked on c in this process, the stack condition for c is now violated.

(b) This same problem may be seen repeatedly in the overall goal of getting state 1, p 316 transformed into state 2, p 318. A partial solution would require all blocks to be unstacked and placed on the tabletop, then stacking up the goal situation could commence!

Of course, all that is required to answer this question is to find a task that requires (at least) two subgoals to accomplish. When the preconditions of these two subgoals can conflict, problems can arise. One simple task would be to place a block at a certain location. The preconditions of the hand being empty and the top of the block being clear can complicate each other.

19. Read the ABSTRIPS research (Sacerdotti 1974) and show how it handles the linearity (or incompatible subgoal) problem in planning.

ABSTRIPS came shortly after the STRIPS work and addressed some of the problems encountered with the STRIPS approach. There is now a huge research on planning. We have only scratched the surface!

20. In Section 8.4.3 we presented a planner created by Nilsson and his students at Stanford (Benson and Nilsson 1995). Teleo-reactive planning allows actions described as durative, i.e., that must continue to be true across time periods. Why might teleo-reactive planning be preferred over a STRIPS-like planner? Build a teleo-reactive planner in Prolog, Lisp, or Java.

Name several other tasks that might require "durative" procedures. Examples might include keeping a satellites solar panel focused on the sun or keeping a monitoring system focused on its target.

Chapter 9 Reasoning in Uncertain Situations

The important task of this chapter is the management of uncertainty. There are several paradigms presented for accomplishing this, including Bayes' formula, logic-based truth maintenance systems, and a most important tool for managing uncertainty in rule based expert systems, the Stanford Certainty Factor Algebra. It requires about 4 hours to teach this entire chapter, but what I usually do is only present a selection of the topics. This selection is often based on the types of programming assignments that I like to assign. We next briefly discuss the main topics presented in this chapter.

Section 9.1 presents logic-based tools for handling uncertainty. The most important of these tools represent methods for performing sound reasoning in domains where knowledge is both uncertain as well as can change its truth-value over time. These domains with changing truth values are often called "nonmonotonic". Since a nonmonotonic specification can change truth value over time, applying inference rules does not always "add" to the total knowledge base of true information. In the mathematical sense then, these knowledge bases are "nonmonotonic" in their size. Several predicates, including "unless" and "is consistent with" are presented to support this type reasoning. A "truth maintenance system" supports the growth and contraction of the knowledge base. We present several other mathematically sound approaches that also address this issue, including the use of minimum models, where only those (the minimum set of) expressions that satisfy a situation are assumed. This section requires about 1 hour to cover totally.

Section 9.2 covers unsound but heuristically motivated techniques for handling uncertainty. The two most important of these are the Stanford Certainty Factor Algebra and "fuzzy" reasoning. The Stanford Algebra was very influential in the design and building of early rule based expert systems including MYCIN. We show how the Stanford model of uncertainty is able to propagate constraints through the graph search, both for implementing best-first search as well as for pruning off low confidence branches of the search. We give a detailed example of a fuzzy control system solving the "inverted pendulum" problem. Both these approaches are meant to mimic how human expert's reason with imprecise information. The presentation Section 9.2 requires about 1.25 hours.

Finally, in Section 9.3, we consider stochastic approaches to uncertainty. The material in Section 9.3.1 introduces Bayes' rule. I find it useful to address this paradigm, and the complexity issues that the full Bayesian approach requires, in some detail. This complexity analysis, which is actually introduced and discussed in the text, helps motivate the use of other approaches to uncertainty, especially some of those introduced in previous sections. The use of d-separation and the assumptions of Bayesian belief networks also address the complexities of full Bayesian reasoning. Several examples of Bayesian belief networks as well as the clique tree propagation algorithm for belief net inference are presented at the end of the section, Finally, several other graphical models are presented including Markov models. Section 9.3 requires about 1.75 hours to present.

In the Epilogue and References section, several other variants of traditional logic, including modal, temporal, and higher-order logics are pointed to as possible areas for future readings and research.

Chapter 9 Comments on Selected Exercises

1. Identify three application domains where reasoning under conditions of uncertainty is necessary. Pick one of these areas and design six inference rules reflecting reasoning in that domain.

   We saw several examples of this in the symbol-based reasoning systems of the previous chapter. One interesting one is the area of diagnosing car problems. Expand and add certainty factors to the four diagnostic rules of Section 8.2 or else use the rules in Section 15.7.2 with certainty factors already present to solve this problem. Another area of uncertain reasoning is medical diagnosis, although a certain amount of domain knowledge is required to design a realistic system in this domain. A final example could be the diagnosis of hardware/software problems in a computer system, for example why a printer might not work.

2. Given the following rules in a "back-chaining" expert system application:

   $A \wedge \text{not}(B) \Rightarrow C$ (.9) $\qquad$ $C \vee D \Rightarrow E$ (.75)

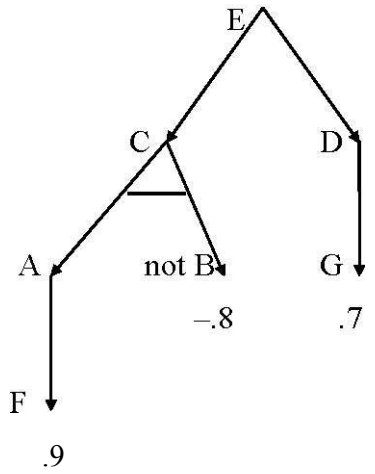
   $F \Rightarrow A$ (.6) $\qquad$ $G \Rightarrow D$ (.8)

   The system can conclude the following facts (with confidences):

   F(.9) $\qquad$ B(–.8) $\qquad$ G(.7)

   Use the Stanford certainty factor algebra to determine E and its confidence.

   We represent the given relationships in the following and/or search tree:



   Now, we try to establish E through C or D. So first we try to support C through A and not B. But F, which has .9 confidence supports A with confidence .6. Here we use the product rule to obtain the confidence of A given F as .54. Next, not B is true

with confidence – .8 so B is true .8. Since C is the "and" of A and B we take the MIN of their confidences, namely .54. Using the product rule again we have the confidence of C as .54 × .9 = .468, or rounded, .49. But to get E we need C or D. We have C at .49 confidence and D is true given G (using the product rule again) with confidence .56. Determining the confidence for E requires the MAX of its two "or" confidences, so we use .56. Finally, using the product rule again E is true with confidence .54 × .75 = .405, or rounded, .41.

4. Create a new example of diagnostic reasoning and use the Dempster–Shafer equations of Section 9.2.3 to obtain belief distributions as in Tables 9.1 and 9.2.

   A simple approach to solving this problem (that would go a long way to showing what is going on!) would be to take the same problem presented and change the belief measures. Then work the values of Tables 9.1 and 9.2 through again. A second approach would be to add one more parameter, say the possibility of allergies, to the set of four other hypotheses {cold, flu, migraine headaches, meningitis} that make up Q. Create some evidence that the patient has allergies and then work through the equations as did the example in the book.

6. Create another reasoning network similar to that of Figure 9.4 and show the dependency lattice for its premises, as was done in Figure 9.5.

   Elaborating the example of Figure 9.4, suppose we want to identify the sets of premises that support node n6. These will be all the sets of premises above (n4, n5) in the lattice of Figure 9.5, namely (n1, n4, n5), (n2, n4, n5), and (n1, n2, n4, n5).

7. Reasoning by assumption of a minimum model is important in human everyday life. Work out two more examples that assume minimum models.

   The assumption of a minimum model is important in most domains of (potentially) complex processing. For example, in driving a car, we assume properties of the car or the world are stable unless we have a strong indication otherwise. As we drive for instance, we believe the indications of our fuel, battery, and oil pressure gages are correct. We also believe the road doesn't drop into a huge hole over the next hill, where we can't yet see. (We don't attempt to pass on a two-lane road, however, unless we see that all is clear). In the Farmer, Wolf, Goat and Cabbage problem we assume that the boat is not slowly sinking, that the farmer is not so overweight that he can't row, that the river is narrow enough it is able to be rowed across, and so on.

8. Continue the inverted pendulum example of Section 9.2.2. Make two more iterations of the fuzzy controller where the output of one iteration makes up the input values for the next iteration.

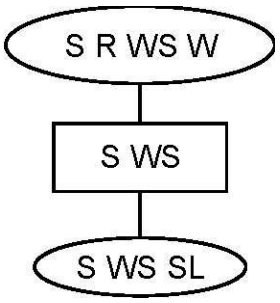
   Just repeat the steps taken in the book. One of the potential problems with fuzzy controllers is the fact that solutions can oscillate.

10. Go to the literature, for example Ross (1995), and describe two other areas where fuzzy control might be appropriate. Construct a set of fuzzy rules for those domains.

We suggest something simple, for example the amount of soap and rinsing to be used in a washing machine, depending on the weight and dirt level of the clothes to be washed. Another application area is for a vacuum cleaner to adjust to the fabric that it is attempting to clean, perhaps using different power for dirt levels and carpet type, such as for a kilim rug as compared to an attached wall-to-wall carpet.

11. Put another link in Figure 9.16, say connecting season directly to slick sidewalk and then create a clique tree to represent this situation. Compare the complexity issues with those of the clique tree of Figure 9.17.

Connecting state S to SL in Figure 9.16, requires that the clique tree have links both between states R and W, as in Figure 9.17b, but now also between S and WS, because they are both now parents of SL. The junction tree is:



As noted in the text, the complexity difference is a function of the sizes of the cliques.

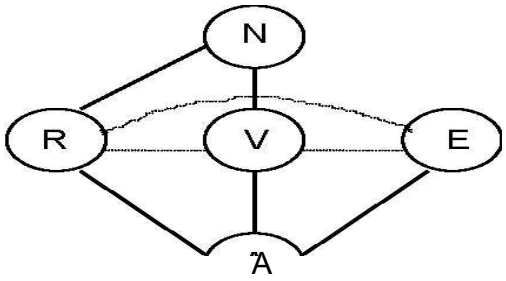
12. Complete the symbolic evaluations that are required to finish Table 9.4.
The table value for R = t and W = f is:

$$P(S = hot) * P(R = t \mid S = hot) * P(W = f \mid S = hot) +$$
$$P(S = cold) * P(R = t \mid S = cold) * P(W = f \mid S = cold)$$

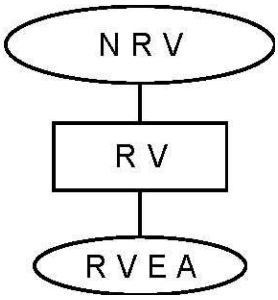
The final two values for the table are similarly produced.

15. Create cliques and a junction tree for the following situation (seen in Figure 9.23). Robbery, vandalism and an earthquake can all cause a house alarm. There is also a measure of the potential dangers in the neighborhood of the house.

Consider the graph of the Bayesian belief network of Figure 9.23. A set of cliques would be as created by the dotted lines:



R and E are connected because they are both parents of A. A junction tree for this graph is represented by:



SECTION II.4

Part VII, Including Chapter 16

Chapter 16 Artificial Intelligence as Empirical Inquiry

The material of Chapter 16 requires about 1.5 hours to present. When we wrote this chapter, we intended that it complement the introductory Chapter 1, covering the history and ideas that make up the foundation of artificial intelligence. Once readers have seen the techniques of AI, as they are presented during an AI course, it is important that they again consider its foundations and limitations. Even for students in an introductory AI class, perhaps seeing little of the advanced chapters, it is important to consider these issues. Our philosophical approach is very much that presented by Newell and Simon (1976) in their Turing Award lecture: Artificial intelligence, as is all of computer science, is an exercise in empirical inquiry.

Of course, a thorough presentation of many of the topics of this chapter, such as issues in cognitive science, would require a full college course. Our idea was to introduce these issues as important supplementary material for the AI student to consider. In fact, most AI researchers and designers continue to be fascinated by both the power and limitations of intelligence. The AI discipline attempts to reach a better understanding of both computation-based as well as human problem solving skills.

Rather than discuss these philosophical issues further here, we welcome our reader to our chapter. There are no exercises with this chapter, except to suggest that the reader go back and reconsider the exercises at the end of Chapter 1. Finally, the interested reader may continue the fascinating analysis of the power and limitations of our discipline by considering the material in the epilogue and references of Chapter 16.

Section III

Introduction to AI Advanced Topics

In Section III we present the "advanced topics" chapters of our book. These chapters are used selectively with introductory AI courses. These are also the materials most often used for an advanced course in AI, as discussed with the presentation of sample curricula in Section I of this instructor's guide. The advanced topics chapters fall into two parts. The first is Part IV where there are four chapters on machine learning. The second is Part V with chapters on automated reasoning and natural language understanding.

Based on the fact that most instructors use these materials selectively, we have attempted to make the chapters as modular as possible. This is especially true with machine learning. When teaching Part IV, we encourage the instructor to start with the general overviews at the beginning of each chapter. These sections will introduce the students to learning paradigm of that chapter; symbol-based, Chapter 10, connectionist, Chapter 11, social and emergent, Chapter 12 and probabilistic, chapter 13. After the introduction to the learning approach is presented then the class can visit any of the modular sections of those chapters.

In Part V it was not possible to preserve this overall modularity. This was because research in automated reasoning and natural language understanding are little more of a cohesive story. For example, the answer extraction process in binary resolution is based on understanding resolution, refutations, unification of variable bindings, standardizing variables apart, and so on. There are similar situations for understanding issues in natural language. Thus if you wish to present the materials of Part V, we suggest that you teach the chapters straight through.

In Section III of this instructor's manual, we have not worked out the exercises as we did in the introductory sections. There are two reasons for this. First, because the advanced nature of this material suggests that the student do fewer, and more complex, assignments. Some of these are suggested in the exercises. For example exercises 12, 14, and 15 of Chapter 10. The second reason is that the book author, in trying to get this IG on the WWW has not had time to complete more exercises. When these are available they will be added to the IG on the www.

Thus, except for Chapter 15, Understanding Natural Language, which does have some of the easier exercises worked through, Section III of this guide will simply comment on the material of the chapters themselves as well as offer guidelines for instruction.

SECTION III.1

Part IV, Including Chapters 10, 11, 12, and 13

Machine Learning

Computational learning has been an important component of research in AI from the very beginning. For example, we have Marvin Minsky's dissertation on neural systems, Nils Nilsson's early comments on machine learning, Samual's checker playing program, Gerlearnter's geometry theorem prover, and so on.

Even with all this activity over the years, however, there is no "unified theory" of learning. As a result the material in Part IV is loosely grouped into four approaches, symbol-based, connectionist, genetic, and probabilistic. But even within these general paradigms the individual research areas remain modular. Thus we recommend that the instructor begin with the general overviews for each chapter (Section X.1) and then cover the individual modular sections as desired.

Chapter 10 Machine Learning: Symbol-Based

This symbol-based machine learning chapter has about 65 pages. There are several related learning algorithms from this chapter written in Prolog and Lisp in the supplementary materials supplied with the book. Covering all this material can require about 8 hours of presentation. The material is highly modular, however, in that the instructor may pick several different sections of the machine-learning chapter and emphasize that material while omitting the rest.

Sections 10.0 and 10.1 introduce the general issues of machine learning and serve as an introduction to the remaining sections of the chapter; they require about .75 hours to present. The reader is encouraged to read Section 10.0 for a general introduction to the learning algorithms of the chapter. Figure 10.1 and the related discussion present a general model of the learning process. The first worked through example using this analysis is presented with Winston's concept learning program.

The first major presentation of this chapter is in Section 10.2, version space search. The idea is introduced in Section 10.2.1, algorithms are offered in Section 10.2.2, as well as several examples, and then, Section 10.2.3 shows a practical example of LEX, a search program that learns heuristics for solving symbolic integration problems. Finally, a PROLOG program that implements version space search may be found in Section 15.8. Version space search can be covered in about 1.5 hours.

The next major learning topic is the ID3 decision tree induction algorithm. This important algorithm has played an important role in a number of expert system shells for decision tree problem solving. The induction algorithm is simple and based on John von Neumann's infor-

mation equations. At each choice in the decision tree, the branch is taken that maximizes information gain. ID3 is a very general-purpose approach to learning and we see its use again in other situations, as in natural language. In Section 10.3 several examples of this algorithm are presented, and in the supplementary material, the decision tree induction algorithm is written in Lisp. These sections require about 1.25 hours to present.

Section 10.4 relates inductive bias to the ability to learn. Certainly the presence of a priori biases has an important influence on search-based learning. This short section, about a .5 hour presentation, delineates the nature and types of bias possible in search-based learning. The necessity of some bias in learning is discussed again in Section 16.2.

The next major section, 10.5, describes the role that knowledge can have in learning. There are a number of important examples of knowledge-based learning presented in this section, Meta-Dendral (Buchanan and Mitchell 1978), Explanation Based Learning, Analogical Reasoning, and Case Based Reasoning. (Case Based Reasoning is presented directly in Section 8.3). An algorithm for Explanation Based Learning is developed in Prolog in supplementary programming materials. This section requires about 1.5 hours to cover.

Section 10.6, unsupervised learning, requires about 1 hour to present. Unsupervised learning requires the program to form and evaluate concepts on its own, without the suggestions and criticisms of a teacher. Lenat's AM (Lenat and Brown 1984), conceptual clustering (Michalski and Stepp 1983), and COBWEB (Fisher 1987) make up the algorithms and examples of this section.

We present reinforcement learning in Section 10.7. Reinforcement learning is supervised in that an environment gives feedback as to the success of an action. The feedback, however, is rarely immediate, in that the learner must integrate reward measures across time, for example, in that some early move in a game may eventually be part of a win. In this section we take a simple example from tic-tac-toe and show how an eventual win generates a reward function. This section takes about .75 hour to cover.

We strongly recommend that the reader go to the original sources that may be found in Section 10.8, the Epilogue and References, to extend their knowledge in the domains of machine learning we have presented. Our presentation is, of necessity, only an introduction; there is a rich literature now available on each of the learning topics that we have presented.

Chapter 11 Machine Learning: Connectionist

Parallel distributed processing and genetic algorithms, Chapters 11 and 12, round out the material of Part IV. In contrast to the symbol manipulation models of the previous chapter, these last two approaches to machine learning view intelligence as arising from the collective behavior of large numbers of simple interacting components.

In Chapter 11 we present neurally-inspired models, also known as parallel distributed processing (PDP) or connectionist systems, which deemphasize the explicit use of symbols in problem solving. Instead, connectionists hold that intelligence arises in systems of simple, interacting components (biological or artificial neurons) through a process of learning or adaptation by which the connections between components are adjusted. Processing in these systems is distributed across collections or layers of neurons. Problem solving is parallel in the sense that all the neurons within the collection or layer process their inputs simultaneously and independently. These systems are also claimed to degrade gracefully because information and processing are not centrally controlled but distributed across the network's nodes and layers.

In connectionist models there is, however, a strong representational character, a strong inductive bias, both in the creation of input parameters as well as in the interpretation of output values. To build a neural network, for example, the designer must create a scheme for encoding patterns in the world into numerical quantities. The choice/bias of an encoding scheme can play a crucial role in the eventual success or failure of the network to learn.

In connectionist systems, processing is parallel and distributed with no manipulation of symbols as symbols. Patterns in a domain are encoded as numerical vectors. The connections between components, or neurons, are also represented by numerical values. Finally, the transformation of patterns is the result of a numerical operations, usually, matrix multiplications. These "designer's choices" for a connectionist architecture constitute the inductive bias of the system.

The algorithms and architectures that implement these techniques are usually trained or conditioned rather than explicitly programmed. Indeed, this is a major strength of the approach: an appropriately designed network architecture and learning algorithm can often capture invariances in the world, even in the form of strange attractors, without being explicitly programmed to recognize them. How this happens makes up the material of Chapter 11. The entire chapter requires about 6 hours to present.

The material in Chapter 11 is extremely modular, with very few cross-references between sections. This makes it possible for the instructor to pick and choose the components they wish to cover. We strongly recommend, however, that Sections 11.1 and 11.2 be covered. These sections, requiring about 2 hours to present, serve as an historical overview of work in connectionist systems. They also introduce the idea of the artificial neuron, McCullock-Pitts systems, and perceptron learning.

In Section 11.1, taking about 1.25 hours to cover, we introduce neurally inspired learning models from an historical viewpoint. We present the basic components of neural network learning, including the "mechanical" neuron, and describe some historically important early work, including the McCulloch-Pitts (1943) neuron. The evolution of the network training paradigms over the past 40 years offers important insights into the present state of the discipline.

In Section 11.2, requiring about .75 hour to cover, we continue the historical presentation with the introduction of perceptron learning, and the delta rule. We present a detailed example of a perceptron system used for classification.

In Section 11.3 we introduce nets with hidden layers, and the backpropagation-learning rule. These innovations were introduced in the evolution of artificial neural networks to overcome problems the early systems had in generalizing across data points that were not linearly separable. Backpropagation is an algorithm for apportioning "blame" for incorrect responses to the nodes of a multilayered system with continuous thresholding. Section 11.3 requires about 1 hour to present.

In Section 11.4 we present models for competitive learning developed by Kohonen (1984) and Hecht-Nielsen (1987). In these models, network weight vectors are used to represent patterns rather than connection strengths. The winner-take-all learning algorithm selects the node whose pattern of weights is most like the input vector and adjusts it to make it more like the input vector. It is unsupervised in that winning is simply identifying the node whose current weight vector most closely resembles the input vector. The combination of Kohonen with Grossberg (1982) layers in a single network offers an interesting model for stimulus-response learning called counter-propagation learning. Section 11.4 requires about 1 hour to present.

In Section 11.5 we present Hebb's (1949) model of reinforcement learning. Hebb conjectured that each time one neuron contributes to the firing of another neuron, the strength of the pathway between the neurons is increased. Hebbian learning is modeled by a simple algorithm for adjusting connection weights. We present both unsupervised and supervised versions of Hebbian learning. We also introduce the linear associator, a Hebbian based model for pattern retrieval from memory. This section requires about .75 hour to present.

Section 11.6, with presentation times of 1.25 hours, introduces a very important family of networks called attractor networks. These networks employ feedback connections to repeatedly cycle a signal within the network. The network output is considered to be the network state upon reaching equilibrium. Network weights are constructed so that a set of attractors is created. Input patterns within an attractor basin reach equilibrium at that attractor. The attractors can therefore be used to store patterns in a memory. Given an input pattern, we retrieve either the closest stored pattern in the network or a pattern associated with the closest stored pattern. The first type of memory is called autoassociative, the second type heteroassociative. John Hopfield (1982), a theoretical physicist, defined a class of attractor networks whose convergence can be represented by energy minimization. Hopfield networks can be used to solve constraint satisfaction problems, such as the traveling salesperson problem, by mapping the optimization function into an energy function (Section 11.6.4).

In Chapter 12, the final chapter of Part IV, we present evolutionary models of learning, such as genetic algorithms and artificial life. We discuss representational issues and bias in learning as well as the strengths of each learning paradigm again in Section 17.3.

Chapter 12 Machine Learning: Social and Emergent

This chapter considers learning algorithms patterned after the processes underlying evolution: shaping a population of individuals through the survival of its most fit members. The power of selection across a population of varying individuals has been demonstrated in the emergence of species in natural evolution, as well as through the social processes underlying cultural change. It has also been formalized through research in cellular automata, genetic algorithms, genetic programming, artificial life, and other forms of emergent computation.

Emergent models of learning simulate nature's most elegant and powerful form of adaptation: the evolution of plant and animal life forms. Charles Darwin saw "...no limit to this power of slowly and beautifully adapting each form to the most complex relations of life...". Through this simple process of introducing variations into successive generations and selectively eliminating less fit individuals, adaptations of increasing capability and diversity emerge in a population. Evolution and emergence occur in populations of embodied individuals, whose actions affect others and that, in turn, are affected by others. Thus, selective pressures come not only from the outside environment, but also from interactions between members of a population. An ecosystem has many members, each with roles and skills appropriate to their own survival, but more importantly, whose cumulative behavior shapes and is shaped by the rest of the population.

Because of their simplicity, the processes underlying evolution have proven remarkably general. Biological evolution produces species by selecting among changes in the genome. Similarly, cultural evolution produces knowledge by operating on socially transmitted and modified units of information. Genetic algorithms (GA) and other formal evolutionary analogs produce increasingly capable problem solutions by operating on populations of candidate problem solutions.

When the genetic algorithm is used for problem solving, it has three distinct stages: first, the individual potential solutions of the problem domain are encoded into representations that support the necessary variation and selection operations; often, these representations are as simple as bit strings. In the second stage, mating and mutation algorithms, analogous to the sexual activity of biological life forms, produce a new generation of individuals that recombine features of their parents. Finally, a fitness function judges which individuals are the "best" life forms, that is, most appropriate for the eventual solution of the problem. These individuals are favored in survival and reproduction, shaping the next generation of potential solutions. Eventually, a generation of individuals will be interpreted back to the original problem domain as solutions for the problem.

Genetic algorithms are also applied to more complex representations, including production rules, to evolve rule sets adapted to interacting with an environment. For example, genetic programming combines and mutates fragments of computer code to evolve a program for solving problems such as capturing the invariants in sets of data.

An example of learning as social interaction leading to survival can be found in games such as The Game of Life, originally created by the mathematician John Horton Conway and introduced to the larger community by Martin Gardner in Scientific American (1970, 1971). In this game, the birth, survival, or death of individuals is a function of their own state and that of their near neighbors. Typically, a small number of rules, usually three or four, are sufficient to define the game. In spite of this simplicity, experiments with the game of life have shown it to be capable of evolving structures of extraordinary complexity and ability, including self replicating, multi-cellular "organisms" (Poundstone 1985).

An important approach for artificial life, or a-life, is to simulate the conditions of biological evolution through the interactions of finite state machines, complete with sets of states and transition rules. These automata are able to accept information from outside themselves, in particular, from their closest neighbors. Their transition rules include instructions for birth, continuing in life, and dying. When a population of such automata is set loose in a domain and allowed to act as parallel asynchronous cooperating agents, we sometimes witness the evolution of seemingly independent "life forms."

In Section 12.1, requiring about 1.5 hours to present, we introduce evolutionary or biology-based models with genetic algorithms (Holland 1975), an approach to learning that exploits parallelism, mutual interactions, and often a bit-level representation. We first give pseudocode to implement a GA. We make this algorithm concrete through the presentation of two examples, first, the conjunctive normal form satisfiability problem, and second, the traveling salesperson problem. These two problems demonstrate the three key features of solutions through GAs. First, there is the encoding of the solutions for the problem into some forms of bit strings, Secondly, there is the creation of genetic operators that can alter in an acceptable fashion these bit strings to form the next generation of possible solutions. Finally, there is the critique of the new bit string as a possible solution with the decision whether it will survive for the next generation.

In Section 12.2, about 1 hour to present, we present classifier systems and genetic programming. A classifier system, often viewed as a set of production rules, takes feedback from an environment to remove ineffectual rules, reinforce good ones, as well as produce new behaviors (Holland et al. 1986). In genetic programming, pieces of program code are combined to create and adapt successful computer programs (Koza 1992).

In Section 12.3, requires about 1 hour to present. We describe concepts in artificial life (Langton 1995). We begin 12.3 with an introduction to "The Game of Life." This simple game captures some of the early research ideas of John von Neumann (Burks 1970) and his work with finite state machines, including combinations of machines that were Turing complete and could produce new machine, thus "artificial life". We close Chapter 12 with an example of emergent behavior from research at the Santa Fe Institute (Crutchfield and Mitchell 1995).

Chapter 13 Machine Learning: Probabilistic

In the past decade, and especially motivated by Pearl's introduction of the Bayesian belief net (BBN) technology (Pearl 1988), probabilistic models for machine learning have become ubiquitous across the field of AI. The applications of probabilistic learning include the areas of robotics and the design of control algorithms, real-time diagnostics and prognostics of complex mechanisms, as well as the data mining and interpretation of large information sources.

But actually AI's use of the stochastic methodology goes back much further, especially in the area of image recognition, natural language understanding, and speech understanding and generation. There tools such as Markov decision processes and hidden Markov models have proven valuable.

Chapter 13 is new to the Sixth Edition. In it we have gathered together the various pieces of probabilistic learning theory that were presented in Chapter 9 of the Fifth Edition as well as introduced a large set of important probabilistic modeling tools, totally new to the Sixth Edition.

Chapter 13 contains three major components, following the brief introduction to the material. The lecture time for the chapter is about 3 hours, depending on how great depth the instructor wishes to go into the applications of the chapter.

Section 13.1, requiring about 1.25 hours to present, introduces the hidden Markov model (HMM). Regular Markov models are presented in Section 9.3, where the value of any (observable) state has a probabilistic relation to those states that proceed it. Especially important is the first-order Markov model where the probability of the present is a function of the immediately previous state alone. In the HMM, each state of the world has an unobservable component that cannot be directly determined by the viewer. The goal of this type of modeling tool is to determine the best estimate of the content of the hidden state through the values of the components of the model that ARE observable. An example is this is the attempt to predict the health state of a complex system by observing values of heat and vibration sensors. See, for example, Section 13.1.2.

There are many forms that the HMM can have, and in Section 13.1 we catalog and motivate the use of several of these, including the auto-regressive, factorial, and hierarchical HMMs. We end Section 13.3 using n-gram HMMs to represent the interpretation of components of language. Dynamic programming, especially it probabilistic use with the Viterbi algorithm has been particularly valuable in this application.

Section 13.2 introduces dynamic Bayesian networks (DBNs) and requires about .75 hours to present. Like the variants of the HMM, the DBN is intended to capture invariants in a problem domain across time periods. An important task of DBNs is the task of structure induction, or the identification of the "best" network model to capture the structure of situations evolving across time. Although this is an exponentially hard problem to solve in general, there are situations where parameter induction through expectation-maximization, EM, (Dempster et al. 1977) and the use of greedy local search can create useful solutions.

An example of the use of the EM algorithm is given in Section 13.2.3. Our EM example, as well as other attempts to find best-fit models for complex situations, requires the presentation of DBN inference algorithms. We present a form of Pearl's loopy belief algorithm (1988) that uses a Markov random field to propagate constraints across models. We call our representation for these models and constraint propagation "Generalized Loopy Logic". Models are stated in a declarative fashion, as logic statements with probabilistic constraints. Loopy belief propagation is done through using the Markov random field (an undirected graphical form) to iteratively determine the best model, given its constraints and possible missing or newly acquired data. When the original model is directed and a-cyclic, this approach guarantees an optimal model when it converges (Pearl 1988, Sakhanenko et al 2006). Again, Section 13.2.3 presents a detailed example of the use generalized belief propagation across a dynamic Bayesian network.

In the final section of Chapter 13, Section 13.3 requiring about an hour to present, we define the Markov decision process (MDP) and the partially observable Markov decision process (POMDP). We demonstrate these tools in an extension of reinforcement learning, presented originally in Section 10.7, which shows a reward driven robot. This agent is required to exist (survive) in its world of rewards and punishments, and must update its decision processes (establish a policy) based on its experience in its world. An example of the use of the Markov decision process for this robot's control is presented in Section 13.3.3.

We conclude our discussion of Chapter 13 by presenting three of the exercises and possible answers.

Exercise 8. Jack has a car dealership and is looking for a way to maximize his profits. Every week ...

Possible answer:

The time scale of the problem is a week. The state is given by the number of cars s that Jack has in inventory at the beginning of the week. The action is how many cars to buy. For simplicity, let us assume that Jack has a limited parking lot, so he can have at most S cars at any point. Jack's actions are to buy a number **a** of cars, where **a** is an element of **[S - s, S]**. After this, Jack has no control over how many cars he sells, this is a stochastic transition performed by the environment. Let us assume that the number **k** of cars that he can sell is uniformly distributed between 0 and his inventory size **s** + **a**. Note that we could use other distributions here, e. g., Poisson. Given our assumption, the transition model is:

Pa(s,s+a-k)=1/(s+a), for each **k** in **[0, s+a]** and **Pa(s, s')=0**, for all other states **s'**

The rewards are easier expressed as a function of the current and next state:

Ra(s,s+a-k)=-ad+kc-u(s+a-k).

The long-term return, without any discounting, would be equal to the total amount of money earned (or lost) by Jack. If there is a discount factor **d < 1** then the return can be viewed like the long-term earnings estimated under an "inflation rate".

Exercise 9. Consider the general domain of grid-world navigation tasks, where there is a goal state, …

Possible answer:

The main idea here is that, if the penalty for slipping is very high, and outweighs the benefits of getting to the goal, then doing nothing can be the best thing to do. For instance, consider a grid with one obstacle in the middle, a reward of +1 at the goal and a penalty of -100 for hitting something. Suppose the slipping probability is 50%. Then trying to move causes an average reward of -50 per time step, so clearly the best thing to do is stay in place.

Exercise 10. When we go out to dinner we always like to park as close as possible to the restaurant …

Possible answer:

There are several possible solutions to this problem, and we discuss one particular possibility. We define an infinite state space, with states numbered according to their distance from the restaurant: -D, -(D – 1), ..., - 1, 0, 1, ... In each state, the corresponding parking spot may be empty or occupied. Hence, corresponding to any distance **s** from the restaurant, we have two corresponding possible states, **es** (**s** lot is empty) and **fs** (**s** slot is occupied). We also have a terminal state, **T**, corresponding to the car being parked. There are two available actions: **park** and **move**. The **move** action will just advance one slot to the right. Hence, in terms of transition probabilities, we have:

Pmove(es-1,es)=Pmove(fs-1,es)=ps

Pmove(es-1,fs)=Pmove(fs-1,fs)=1-ps

The reward for the **move** action is 0: **Rmove(s)=0** for all **s**. The **park** action in an empty slot should transition to the terminal state in which the car is parked: **Ppark(es,T)=1**

The reward for parking will be inversely proportional to the distance to the restaurant. This can be achieved through many different reward schemes, e. g.:

Rpark(es)=1/(|s|+1)

If the slot is not empty, the **park** action does nothing:

Ppark(fs,fs)=1

The reward in this case will be 0: **Rpark(fs)=0**. Since we have a continuing task, we will have a discount factor **d < 1**.

Note that a variation on this solution would be to allow the car to do a u-turn and jump all the way back to the initial state. This may be a more realistic model and would involve introducing a **loop-back** action.

This exercise is adapter from Puterman (1994).

SECTION III.2

Part V, Including Chapters 14 and 15

Advanced Topics for AI Problem Solving

Part V, Chapters 14 and 15 continues our presentation of important AI application areas. Theorem proving, often referred to as automated reasoning, is one of the oldest areas of AI research. In Chapter 14, we discuss the first programs in this area, including the Logic Theorist and the General Problem Solver. The primary focus of the chapter is binary resolution proof procedures, especially resolution refutations. More advanced inferencing with hyper-resolution and paramodulation is also presented. Finally, we describe the Prolog interpreter as a Horn clause and resolution-based inferencing system, and see Prolog computing to as an instance of the logic-programming paradigm.

Chapter 15 presents natural language understanding. Our traditional approach to language understanding, exemplified by many of the semantic structures presented in Chapter 7, is complemented with the stochastic approach. These include Markov models, CART trees, mutual information clustering, and statistics-based parsing. The chapter concludes with examples applying these natural language techniques to database query systems and also to a text summarization system for use on the www.

Chapter 14 Automated Reasoning

Theorem Proving, or as we prefer to call it, Automated Reasoning, is arguably the earliest application area of Artificial Intelligence. In fact the very first AI program was Newell, Shaw, and Simon's Logic Theorist (Newell and Simon 1963a). Throughout the history of AI, automated reasoning has played an important role. Its products include a large number of inferencing techniques and strategies. Perhaps its most important result is the Prolog computing language.

Three components make up an automated reasoning system: an unambiguous representation language, sound inference rules, and well defined search strategies. These three aspects are emphasized in several different automated reasoning systems presented in the material of Chapter 14. The entire chapter requires about 4.5 hours to present.

Section 14.1 requires about .75 hour of lecture to cover. In this section we present the earliest reasoning system, the Logic Theorist. It is important for the reader to see the simple propositional calculus representation system and the several inference and search schemes created by Newell, Shaw, and Simon. It is also interesting to see how these early search techniques matured into the important problem-solving model, the General Problem Solver. For those wishing to get further into this important early work in AI, we recommend Newell

and Simon's discussion in Human Problem Solving (Newell and Simon 1972) and the presentations in Computers and Thought (Feigenbaum and Feldman 1963).

The most important topic of this chapter is the schemes for binary resolution presented in Section 14.2. This 20-page section requires about 1.75 hours of lecture time. There are four major topics covered in this section: first the binary resolution inference scheme. This should be seen as a more general form of modus ponens. The task of proving binary resolution as sound, as well as the search strategies presented in Section 14.2.4 as complete is beyond the scope of our book, but details may be found in Chang and Lee (1973).

The second major topic covered in Section 14.2 is the nine-step algorithm for producing the Horn clause representation from unconstrained predicate calculus expressions. In Section 14.2.2 we work through a detailed example of this algorithm. The third topic covered is the search strategies that may be used with the Horn clause representation and the binary resolution inference scheme.

Finally, the most interesting part of 14.2 is the answer extraction process from resolution refutations. This algorithm is simple and elegant. With the use of pointers on partial solutions, and assumptions appropriate time and memory, all possible interpretations for a set of Horn clauses may be produced. Section 14.6 will build on this material to describe the action of the Prolog interpreter.

Section 14.3, requiring about 1 hour to present, addresses the issue of Prolog as an instance of logic programming. In this section we show how the inference engine of Prolog can be viewed as binary resolution using a left-to-right and depth-first search strategy and unit preference. Unification in this search scheme provides the variable bindings that extract the answers for the original query. Several issues are discussed, including the lack of an "occurs" check in unification, the use of "cut" to limit backtracking, and negation as failure. These three efficiency hacks in Prolog countermand the goal of creating a pure logic-programming environment. A set of axioms is presented, including a "unique names" axiom, which address the closed world problem of Prolog.

Section 14.4, the final section of this chapter, requires about 1 hour to present, and covers a potpourri of related inference schemes. First we cover general data and goal driven inference schemes based on propositional and predicate calculus representations. Secondly, we demonstrate two more powerful and general inference schemes, hyperresolution and paramodulation. Finally, we cover Bledsoe's (1971) natural deduction system. We end the chapter with ten principles that can help design automated reasoning schemes.

Chapter 15 Understanding Natural Language

In Chapter 15 we present issues and algorithms for computer based natural language understanding. The first major application presented in this book, Chapter 8, the design and

implementation of symbol based expert systems, needed only the representation of the predicate calculus and the search architecture of the production system to encode problem solving knowledge. In this chapter we use knowledge again, and several of the representation techniques of Chapter 7, in our effort to design computer algorithms to understand language.

Natural language understanding by computer is a very difficult problem that requires the full power of the advanced representations. In fact, many of the early representational techniques presented in Chapter 7 were created explicitly to address the difficult issues that are part of representing the syntax, semantics, and pragmatics of a human language.

It takes about 3 hours to present the material of Chapter 15. We begin the chapter with an introductory section with discussion of the difficulties of modeling language. We use 4 lines of a Shakespearian sonnet and make a brief analysis of the meaning intended in its creation Almost any lines of a modern song or aspects of popular culture can be used to make the same points. We also use an example job advertisement from the www to make similar points.

In Section 15.1 we break language understanding into a number of stages. Clearly this is a naive approach, as the levels of language analysis are usually intertwined, as when a speaker might use emphasis of a particular phoneme to change the total meaning of an expression. Nonetheless, computer based analysis of language has by tradition moved through these stages of analysis.

Most of the material in Chapter 15 focuses on parsing algorithms and the Chomsky language hierarchy. The beginning of a semantic analysis comes through the attachment of context sensitive information to the nodes of the parse tree as can be seen in the ATN parser of Section 15.3. A Prolog implementation of this type parse is in the supplementary material.

In Section 15.4, requiring about an hour, we present stochastic tools for language analysis. These tools, originally introduced with the ideas of Bayesian inference in Chapter 9, are here extended to the design of parsers and other language understanding tools. Statistical language techniques are methods that arise when we view natural language as a random process. In everyday parlance, randomness suggests lack of structure, definition, or understanding. However, viewing natural language as a random process generalizes the deterministic viewpoint. That is, statistical (or stochastic) techniques can accurately model both those parts of language that are well defined as well as those parts which indeed do have some degree of randomness.

Viewing language as a random process allows us to redefine many of the basic problems within natural language understanding in a rigorous, mathematical manner. It is an interesting exercise, for example, to take several sentences, say the previous paragraph including periods and parentheses, and to print out these same words ordered by a random number generator. The result will make very little sense. It is interesting to note (Jurafsky and Martin 2000) that these same pattern-based constraints operate on many levels of linguistic analysis, including acoustic patterns, phonemic combinations, the analysis of grammatical structure, and so on. As an example of the use of stochastic tools, we consider the problem of part-of-speech tagging.

The chapter ends, Section 15.5, with more applied examples: first, the design of a story understanding program and second, a conceptual graph representation for relational tuples in a database. This second application allows us to ask and interpret English language queries to the database. A final example shows how, with learning algorithms such as ID3 (Section 10.3), patterns can be found in the language of advertisements on the WWW. The references and epilogue for Chapter 15 suggest several readings for more information on the representations and algorithms for language understanding.

We usually take about four hours to cover the material in this chapter. In fact, we usually begin with a rather lengthy, .5-hour discussion of the difficulties of creating a computational model of language. Probably the most difficult issue is pragmatics: computers simply aren't "situated" in the world the way we humans are. A second important point is that with humans, a speech act represents a commitment, a responsibility. This is true whether the language statement is "Help!" "The dog bit the man," or "I love you."

During this first .5-hour presentation we also cover the problems implicit in splitting language analysis into syntax, semantics, pragmatics, etc. This is a fundamentally flawed approach to language, but the only one we have!! The language analysis of the mainline AI research community takes this direction, and like our text, emphasizes the importance of "knowledge" in understanding language. As this chapter progresses, it demonstrates methods for adding more and more knowledge to our understanding of language. In the final section we use explicit knowledge to understand stories and build the front end for a database.

The authors like the obvious pattern matching power of Prolog parsers and make programming assignments building on the material of Chapter 15, in particular the context free and context sensitive parsers, in Section 15.9. This same section, 15.9, also demonstrates a recursive descent semantic net parser. An interesting assignment is to extend the parser of Section 15.9 with adjectives and prepositional phrases, including the graph join algorithms enforcing semantic constraints, for example, humans can't bite dogs and only big dogs can bite humans.

A time line for presentation of the material in Chapter 15: Section 15.0-15.1, 5 hours; Section 15.2, .75 hour; Section 15.3, .75 hour; Section 15.4 and Section 15.5, .75 hour, for a total of about 4 hours. Most students find natural language understanding an exciting application area. There are a number of related follow on topic suggestions: work through some of the exercises with the students (see the comments on Chapter 15 exercises).

Finally, the philosophical issues presented in Chapter 16.3 offer important constraints on computational models of language, and we like to round out our presentation of natural language understanding with a 30-minute discussion of these issues.

Chapter 15 Selected Worked Exercises

1. Classify each of the following sentences as either syntactically incorrect, syntactically correct but meaningless, meaningful but untrue, or true. Where in the understanding process is each of these problems detected?

   Colorless green ideas sleep furiously. This example, attributed to Chomsky, is syntactically correct, and even has some context sensitivity with noun-verb agreement. Semantically, of course it is meaningless.

   Fruit flies like a banana. This sentence has two syntactically correct parses, one with flies as a verb and fruit the subject; the second has fruit as an adjective modifying flies, the noun and subject, and like the verb. Only the second parse has a meaningful semantic interpretation.

   George Washington was the fifth president of the USA. Syntactically correct, semantically meaningful, and false.

2. Discuss the representational structures and knowledge necessary to understand the following sentences.

   The brown dog ate the bone. The dog's world context sensitive syntax and semantic presented in 15.2.3 is sufficient to handle this problem. Adjectives must be added as specifiers of nouns in noun phrases. Specific pieces of knowledge that could be added are that the adjective describing the dog does not constrain the verb eat, and that when dogs eat bones, they do not usually consume them.

   Mary watered the plants. The original semantic net graph created by Quillian (1967) in section 7.2.2 could represent this situation. In fact an example of the knowledge represented in this situation can be found in Figure 7.3. Quillian used a graph join and constraining algorithm to find an interpretation of the ambiguous word plant.

   The spirit is willing but the flesh is weak. Almost any type of context sensitive parse is sufficient to indicate this is correct. The knowledge is another matter. This is a nice example of the new meanings words take on when used in a specific context. Spirit and flesh have unique meanings when contrasted as they are in the sentence. Note that the literal interpretations of these words are not much help here either: their meaning comes from their use in a particular English translation of a biblical text. There is a story, probably false, that this sentence was given to an early version of an AI Russian/English translation program. The sentence was translated first into Russian and then back to English. The result was "The meat is rotten but the vodka is great." How could this happen?

   3 & 4. Parse each of these sentences with the "dog's world" grammar of Section 15.2. Which of the sentences are illegal? Why? Extend the dog's world grammar of Section 15.2 to handle the illegal sentences of question 3.

The big dog bites the man. We must add adjectives to the context sensitive grammar of Section 15.2.3 to modify nouns to parse this sentence. Adding the rules could do this:

noun_phrase <-> article adjective number noun

adjective <-> big

Emma likes the boy. Proper nouns must be added to the context sensitive parse. This can be done by adding to the dictionary entry for nouns, even though this ignores knowledge implicit in a noun being a proper noun. A rule for this:

singular noun <-> emma singular

The man likes. This is a correct context sensitive parse, even though in English we usually expect likes to have an object. This constraint would be difficult to add to the context sensitive grammar of Section 15.2.3.

Bite the man. This does not presently parse. We must add a new rule where a sentence is a verb phrase followed by a noun phrase:

sentence <-> verb_phrase noun_phrase

5. Parse each of these sentences using the context-sensitive grammar of Section 15.2.3.

   To parse this sentence we must add rules that say a verb phrase can be a verb followed by a noun phrase *without* number constraint:

   verb_phrase <-> verb_phrase noun_phrase

   the dog bites the man

   the dog singular verb the man

   the singular noun singular verb_phrase the singular noun

   article singular noun verb_phrase noun_phrase

   article number noun verb article number noun

   noun_phrase verb_phrase sentence

7. Extend the dogs' world example to include adjectives in noun phrases. Be sure to allow an indeterminate number of adjectives. Map this grammar into transition networks.

   noun_phrase <-> adjective_list noun

   adjective_list <-> adjective

   adjective_list <-> adjective_list adjective

   where adjective is assigned to the various adjectives of the language.

   Augmenting the transition net parser includes three steps:

First, add a third and fourth path through the noun phrase transition net that requires either article -> adjective list -> noun or article list -> noun. Second, create an adjective transition net for all adjectives. Finally, add an adjective list transition net that is either an adjective list or an adjective.

8. Add the (following) context free grammar rules to the dog's world grammar of Section 15.2.1. Map the resulting grammar into transition networks.

   The transition networks must have new paths added:

   1 Sentence (Figure 15.4) will have a second path noun_phrase -> verb_phrase -> propositional phrase, with a second arc from verb_phrase to prepositional_phrase to final.

   2 We need to define a new transition network for prepositional_phrase: prepositional -> noun_phrase

   3 We must also define a preposition network for all acceptable prepositions.

9. Define an ATN parser for the dog's world example with adjectives (exercise 7) and prepositional phrases (exercise 8).

   Part a, the adjective list parser:

   The Noun phrase structure of Figure 15.6 must be augmented to contain an adjective description. Individual adjectives must then be described with structures similar to those of Figure 15.7. Finally, a new procedure noun_phrase-3 must be created, as in Figure 15.8, where:

   noun_phrase.adjective: = adjective

   in its then assignment code.

   Part b, the prepositional phrase parser:

   There are three steps. First, create two noun phrase descriptions similar to those of Figure 15.6, where the prepositional phrase is included as a field in the noun phrase structure and where a preposition followed by a noun phrase is a prepositional phrase. Second, make structures for prepositions similar to those of Figure 15.7. Finally, create two new noun phrase procedures in the algorithms of Figure 15.8 where the appropriate assignments are made to the structures of the first step of the preposition structures created in the second step.

11. Extend the context-sensitive grammar of Section 15.2.3 to test for semantic agreement between the subject and verb. Specifically, men should not bite dogs, although dogs can either like or bite men. Perform a similar modification to the ATN grammar.

    These constraints can most easily be handled through an inheritance hierarchy, where, for example John is an instance of a male is an instance of a human. We then create verb templates, for example for bites, where the subject of biting should be a non-human. We

have built exactly this representation and search strategy with the recursive descent semantic net parser presented in Section 15.3.

12. Expand the ATN grammar of Section 15.3 to include who and what questions.

    First, make a new structure called pronoun similar to the noun structures of Figure 15.7. Next, wherever there is a noun transition in the transition network, add also a pronoun transition. Of course pronouns will have number, just as nouns do. As an extension of the noun/pronoun description we could add the constraint of sex. Finally, add the number and sex assignment in the extended algorithm of Figure 15.8.