CHAPTER 1  Introduction
1: Introduction

1.1 The weak method theory of problem solving

Since the birth of cognitive science in the fifties, human problem solving has been one of its central topics. The marriage between psychology and computer science proved to be especially fruitful, since simulation of cognitive processing allowed deeper insights into the empirical data from human participants than was possible with the now old-fashioned techniques offered by behaviorists. A landmark in problem solving was Newell and Simon’s 1972 book Human Problem Solving. Newell and Simon show detailed analyses of data collected from human participants, along with results from computer simulation. The main conclusion of the book is that human problem solving can be characterized by a small set of methods. These methods require very little knowledge about a particular problem, and are therefore sometimes called weak methods. The tie between psychology and computer science was very strong in this enterprise, since most of the weak methods were algorithms used in artificial intelligence, the sub-discipline of computer science most involved with cognitive science.

The weak-method theory pictures problem solving as search in a problem space. This problem space is a directed graph that has problem states as its nodes, and problem operators as its vertices. A state represents the current configuration of the problem, and operators manipulate these configurations. In problem-solving terms, an operator transforms a current state into a new state. Figure 1.1 shows a simple example of a problem space, the example of the blocks world. This world consists of a table and three blocks, and the only possible action is to move one uncovered block from its current spot to a new spot, either on another block or on the table. Each of the possible configurations of blocks is a state, and is represented in the figure by a rounded rectangle. There is one possible operator: moving a block. This operator can be instantiated in multiple ways, as depicted in the figure by arrows. Suppose the problem starts with the configuration depicted in the upper-left corner of the figure, and the goal is to build the pile of blocks depicted in the lower-right corner. Solving the problem involves selecting a sequence of instantiated operators that transform the start state into the goal state, in this case moving block A to the table, moving block B onto block C, and finally moving block A onto block B.

The problem-space view of problem solving transforms the abstract idea of problem solving into a concrete, easily depictable problem, the problem of deriving the right sequence of operators to transform the start state of a problem into a goal state. To actually find this sequence, one of the weak methods can be applied. Which method is most appropriate depends on the amount and type of knowledge the problem solver has about the problem. The most simple methods are blind-search methods, like generate-and-test, depth-first search and breadth-first search. These methods only assume knowledge about the set of possible states, allowed operators, and the consequences of these operators. Each method systematically searches the problem space until it stumbles over a goal state, in which case the problem has been solved.
The weak method theory of problem solving

Blind-search methods assume that the problem solver has no way of knowing whether a certain state is close to the goal or which operator can bring it closer to the goal. This kind of knowledge is called heuristic knowledge, and methods that use heuristic knowledge are called heuristic methods. The most simple heuristic method is hill-climbing. Hill-climbing assumes a heuristic function that can estimate the distance between a state and the goal state. Using this function, the operator that leads to the most promising new state can be selected. For example, in the blocks-world problem of figure 1.1 the heuristic function might be the number of blocks that are in the right place with respect to the goal state.

A more complex method is means-ends analysis. Means-ends analysis involves a comparison between the goal state and the current state, and the selection of an operator that reduces the difference. If the selected operator is not applicable in the current state, a subgoal is created to reach a state in which the desired operator is applicable. Figure 1.2 shows an example of means-ends analysis: planning a trip from Groningen to Edinburgh. The most notable difference between Groningen and Edinburgh is that they are situated in different countries. So an operator is sought that reduces this difference, in this case flying from Amsterdam to London. This operator is, however, not applicable in Groningen. So getting from Groningen to Amsterdam becomes a subgoal, and is solved by taking the train to Amersfoort and then to Amsterdam. The difference between London and Edinburgh can be found in the same way. An important advantage of means-ends analysis is its divide-and-conquer strategy. This aspect is especially important if the problem space is large or infinite, which is often the case in practice. The disadvantage of means-ends analysis
is its requirement of additional knowledge. It must be possible to find differences between states, differences must be ranked in some way (in the example: a difference in country is more important than a difference in city), and operators must be keyed to these differences.

To summarize: for each of the weak methods there is a parallel between the knowledge needed and efficiency. One would expect that as participants gain more knowledge in a certain problem domain, they will tend to use more efficient methods. Jongman (1997) has found some evidence for this hypothesis. In her study, participants have to find information on the Internet. While a majority of the participants start using a hill-climbing strategy, many of them switch to means-ends analysis as they gain experience.

**Problems of the weak-method theory**

Despite the fact that the weak-method theory offers a systematic framework for studying problem solving and provides explanations for many aspects of human problem solving, it leaves a number of questions unanswered. A first problem of the weak-method theory is that it assumes precise and unambiguous knowledge about problem states, operators and goals, even for the most simple blind-search methods. This assumption is correct for many problems used in problem-solving research, like the towers-of-hanoi, the eight puzzle and blocks-world puzzles.
The weak method theory of problem solving

Research that stresses the importance of insight in problem solving on the other hand, uses problems for which this assumption does not hold. A well-known example is the nine-dots problem (figure 1.3), in which the problem is to connect all nine dots using four connected lines. The difficult aspect of this problem is the fact that a solution is only possible if lines are used that extend beyond the borders of the 3x3 grid of points. In problem-space terms, the problem basically has an infinite number of possible operators, since there are infinitely many ways to draw a line. Participants tend to reduce the set of possible operators to operators that just draw lines between two points of the 3x3 grid. The crucial step in solving the problem is the realization that this reduction is too severe. So problem solving not only involves selecting the right sequence of operators, but also finding out what the operators are, and what they do. The example also shows that re-evaluating the operators currently used may be part of the problem-solving process.

A second problem is the fact that in many cases not all the activities of a participant can be explained in terms of clear problem-solving methods. Participants use multiple strategies for a single problem, skipping between them and inventing new ones on the fly. People tend to forget results if they can not be used immediately, or have to use memorization techniques to prevent forgetting things. Finally, and that is a criticism often quoted, people have the ability to “step out of a problem”, to reason about their own reasoning (see, for example, Hofstadter, 1979, for an extensive discussion of this point). Evidence for this kind of meta-reasoning are exclamations like “This doesn’t work at all”, and “Let’s try something different”. Although it is not at all clear how extensive meta-reasoning can be, people evidently use some sort of self-monitoring to prevent them from doing the wrong thing for too long.

The third problem is that the weak-method theory does not explain how people gain a higher level of understanding in a certain problem domain. An example of this is mathematics. In order to be able to solve simple algebraic equations like $2x + 3 = 7$, one must master simple arithmetic first. Composite concepts from arithmetic form
1: Introduction

the basic building blocks of simple algebra. Solving \( 2x + 3 = 7 \), for example, takes at least four simple arithmetic operators. Experience allows people to collapse these operators into higher-level operators, so they can solve the equation in just one step. Mastering simple equations is a prerequisite for more complex mathematics like differential equations. The idea of several levels of understanding is quite common in developmental psychology, and stems from the stage theories of Piaget (1952).

The three problems discussed above, although somewhat different in nature, boil down to the same issue: learning. The problem solving process is not a pure search process but also includes exploration. Exploration is necessary to learn what the possible operators are and what they do or to question the operators if they fail to perform well. Exploration can also derive and refine heuristic knowledge, and find out what methods and strategies are most suitable for the current problem. To be able to do this several strategies must be tried and compared. Learning can also result in higher-level operators and an increase the level of abstraction of the problem-solving process. Exploration can also attempt to use knowledge from other domains for solving the current problem.

Problem solving from the viewpoint of skill learning

The main topic of this thesis is to study the learning aspect of problem solving. While complex problem solving will be the starting and the end point, several tasks will be discussed that are not strictly problem-solving tasks, unless one adopts Newell’s claim that any task is a problem-solving task. So the topic is actually broader and extends to skill-learning in general, with complex problem solving as the main skill to be studied.

An important theme throughout the thesis will be the distinction between implicit and explicit learning (Reber, 1967; Berry, 1997). Implicit learning is often defined as unconscious learning; the learner is unaware of the fact that he or she is learning, and is unable to recall what is learned afterwards. Increased task performance is the only indication something is learned. Explicit learning, on the other hand, supposes a more active role of the problem solver. An example of this type of learning is when the participant sets explicit exploration goals, or explicitly decides to memorize aspects of a certain problem because they may be useful for another problem. Both types of learning are important for problem solving. During search the problem solver gains information in an implicit fashion, since learning is not the goal but only a by-product. Search for the solution may be alternated by setting explicit learning goals that try to combine earlier experiences, perform generalizations, explore other problem domains, or, on a more mundane level, try to keep partial results active in memory.

One of the core problems of search as a problem solving method is the fact that problem spaces are often very large or infinite. The reason for this is that in each state
The weak method theory of problem solving

there are several possible operators leading to new states. In general, the size of the
problem space grows exponentially with the maximum length of the sequence of
operators. For human purposes, blind, systematic search in an exponential problem
space will only be successful if the sequence of operators is relatively short. If longer
sequences are required, knowledge is needed to offer guidance in the choice of
operators, to retrieve partial sequences used for other problems, or to collapse
several operators into one composite operator. Therefore, the maximum capacity for
solving problems in a certain domain is determined by the knowledge for this
domain extended by a limited amount of search. Actually solving a problem using
search, possibly enhanced by explicit learning, may extend the space of solvable
problems.

Figure 1.4 shows an impression of this idea. The top figure represents the set of all
possible problems, loosely ordered in the sense that more complex problems are at
the top of the rectangle, and less complex problems at the bottom. The horizontal
dimension is used to indicate that problems are related to each other. Some of these
problems can be solved by a particular individual by a relatively simple procedure.
This portion of the set is indicated by the black area at the bottom of the set. Problems
in the grey area require more effort, and need some combinatorial search. Problems
in the white area require so much search that the problem becomes practically
unsolvable.

Problems in the black area take relatively little time. As soon as the grey area is
entered, combinatorial search is needed, which increases the time requirements
exponentially. At some point these time requirements become unpractically high,
marking the beginning of the white area. Learning increases the black area in the set,
sometimes by a single item, sometimes, after generalization, by a substantial area. As
a consequence the border between the grey and the white area also moves outwards,
as indicated by the small arrows in the graph. Take for example the left-most peak in
the figure. This might represent the algebra skill of a certain individual. This
individual is, for example, capable of solving equations without much effort (black
area), able to solve simple problems of integration by trying out several different
methods (grey area), but not proficient in doing double integrations yet (white area).

The time requirements are shown in the graphs at the bottom of the figure. Problems
that can be solved in a direct fashion usually do not require much time. But once the
expertise runs out and combinatorial search is needed, the grey area is entered and
the time requirements increase exponentially with the amount of search needed. Due
to this increase, the time requirements soon exceed practical limitations (white area).
This discussion is of course still very informal. A more formal approach will be
discussed later in this chapter.
1: Introduction

Figure 1.4. Impression of the set possible problems. Some can be solved easily (black area), some need combinatorial search to find the solution (grey), and others cannot be solved at all. The top figure outlines the expertise of a certain arbitrary individual who has three areas of expertise. The small arrows in the top figure indicate the effects of learning. The "peaks" in the figure indicate areas in which this particular individual is an expert. The two graphs at the bottom indicate the time to find the solution given the type of search needed and can be seen as a vertical cross-section of the top figure. The left graph represents a novice, who has to use search for almost everything, and the right graph represents an expert, who can solve many problems in a direct way.
1.2 How to study learning in complex problem solving?

Within cognitive science there are a number of research paradigms to study learning. The main paradigm to study learning is the experimental paradigm used in cognitive psychology. A common approach is to present participants with a sequence of similar problems, and see how their performance improves with respect to reaction time (latency) and rate of errors. One fundamental law found in this fashion is the power law of practice, a law that states that regardless what the task is, the reaction time can be described by the function:

\[ T_n = b n^{-\alpha} \]  

In this equation \( T_n \) is the reaction time for trial \( n \), and \( b \) and \( \alpha \) are constants.

Another method often employed in experimental learning research is the search for dissociation effects. Typical experiments first expose participants to some information, which is tested at a later time using different types of tests. Typical examples of dissociations are:

- If a participant is tested directly after learning, he or she performs equally on test A and B. If he or she is tested again after a week, performance on test A is the same, but performance on test B has decreased severely (Tulving, Schacter & Stark, 1982)
- Performance of a participant suffering from amnesia is equal to a healthy participant on test A, but much worse on test B (e.g., Morris & Gruneberg, 1994).

Dissociations are often used as evidence for the existence of different memory systems, for example a separate implicit and explicit memory.

Although experimental work offers many insights in the nature of learning and memory, the standard experimental paradigm is limited to phenomena that can be quantified easily in, for example, the power law of practice, or the hypothesis that implicit and explicit information is stored in separate memory systems. Take, for example, the power law of practice. The smooth form of the curve suggests learning is a continuous process. Although this may well be the case, this is not necessarily so. As noted by, amongst others, Siegler (1996), the smooth curve may have resulted from averaging several step-functions. Also, a hypothesis about the existence of two separate memory systems is rather crude, and offers little insight into the necessity of separate memory systems. As we will see later on in chapter 4, dissociations can sometimes also be explained using a single memory system.

Because the pure experimental paradigm can only state rather global hypotheses, it often limits itself to experiments where all participants behave roughly the same. Participants only tend to behave the same if there is only one way to do things. In terms of figure 1.4, only problems in the black area are investigated. The grey area,
however, is the area where interesting learning phenomena with respect to problem solving can be found. In that area almost all participants will behave differently due to the exponential number of choices. So it will be much more difficult to state hypotheses in the usual fashion. As a consequence, participants can no longer be studied as a group, but must be studied individually. The challenge is to still be able to make generalizations about the population, despite individual differences.

The paradigm that machine learning offers for the study of learning radically differs from what is used in experimental psychology. Complexity is the main challenge. Although many types of algorithms are used, some of which will be reviewed in chapter 2, the common goal in machine learning is to derive generalized knowledge from examples, sometimes guided by domain knowledge. The goal is to arrive at an accurate generalization using the most efficient algorithm. In a typical machine learning study to judge the quality of a new learning algorithm, a set of examples is used. For example, in a medical setting, an example contains a number of symptoms and a diagnosis. The set of examples is split in two parts, a training set and a test set. The training set is first given to the learning algorithm, which tries to generalize rules or other representations that can predict a diagnosis from the symptoms. The test set is then used to judge the correctness of these representations. A new algorithm is judged to be promising, if its performance on the test set exceeds the performance of a number of established learning algorithms. Performance is measured by the number of correct classifications the algorithm makes on the test set, and by the time it needs to learn the training set.

Machine learning algorithms are quite powerful when judged with respect to efficiency and quality of classifications. Whether or not the learning of such algorithms has any similarity to human learning is not considered important. This does not necessarily mean algorithms from machine learning are useless for studying human learning, since evolution may well have optimized human learning in the same way computer scientists try to optimize machine learning. Nevertheless, machine learning algorithms often make computational assumptions that are not easy realizable for humans. People can, for example, not learn large databases of examples easily.

A third domain of cognitive science in which learning is studied is developmental psychology. Developmental psychology studies changes in behavioral capacities in children over time. According to some theories these changes can be characterized by transitions between stages, meaning there are periods with little change and periods with large changes in capacities. Developmental psychologists are mainly interested in these changes and their characteristics, and less in the processes that cause these changes. Studying how a complex skill is learned in several steps can offer important clues about the nature of the learning processes that cause the change in skill. Possibly the most cited example is the learning of past tenses (Rumelhart & McClelland, 1986; Pinker & Prince, 1988; Elman, Bates, Johnson,
How to study learning in complex problem solving?

Karmiloff-Smith, Parisi & Plunkett, 1996). The literature often distinguishes three stages in this particular skill. In the first stage, all past tenses are learned as separate facts. The second stage is characterized by the discovery of a rule for regular verbs. This rule is, however, overregularized so that irregular verbs that were used correctly in the first stage are now put in the past tense using the regular rule. Only in the third stage the irregular words are recognized and used correctly. Although this description tells us little about the processes that cause change, it reveals nevertheless that an interplay between rules and examples is important. We will come back to this issue in a later chapter.

Since one of the goals of this thesis is to approach learning in problem solving from an experimental perspective, we have to deal with the problems mentioned earlier. Alan Newell already noted the limitations of the classical experimental paradigm in 1973, when he wrote his famous paper titled “You can’t play twenty questions with nature and win”. According to Newell, psychologists investigate cognitive phenomena. Examples of these phenomena are:

1. recency effect in free recall
2. reversal learning
3. rehearsal
4. imagery and recall

Although these are just four items from Newell’s list of 59, they will discussed more extensively later in this thesis. All four of them will turn out to be important for problem solving. Newell’s criticism focuses on the fact that despite the fact that all these phenomena are researched thoroughly, no clear theory of cognition emerges. The main type of structures psychology attempts to establish are binary oppositions. Among these oppositions are the following:

1. Continuous versus all-or-none learning
2. Single memory versus dual memory
3. Existence or non-existence of latent learning
4. Stages versus continuous development
5. Conscious versus unconscious

Again these examples are picked from a list of 24, and will become important at some point in the discussion later on. The point Newell tries to make is that resolution of all these binary oppositions (“20 questions”) will not bring us any closer to a grand theory of cognition. Fortunately, Newell also proposes three solutions to the problem, two of which we will discuss here.

A first solution is to create a single system, a model of the human information processor that can carry out any task. He also proposed a candidate for such a
system, namely a production system. If a production system would be given the right set of rules, it should, in principle, be able to perform any experimental task. It turned out this solution became the main paradigm dominating the rest of Newell’s work. In 1990, he wrote “Unified Theories of Cognition”, in which he presented his final proposal for a grand theory of psychology. At that time, the single system idea had already spread, and other people had been thinking about unification as well. Anderson’s 1983 book “The Architecture of Cognition” is an example, in which Anderson presents his ACT* system. The Rumelhart and McClelland 1986 books “Parallel Distributed Processing” also attempt to bring all types of cognitive phenomena together in a single paradigm. The single-system approach has two important aspects: it constrains the researcher in the type of theories he can state, in the sense that the theory has to fit in the system, and it forces the researcher to be very precise: the theory has to be simulated within the system. In this thesis I will also conform to this single-system approach. The system is the ACT-R 4.0 system, a descendant of ACT*, as described in Anderson & Lebiere (1998). ACT-R and its competitors will be discussed in detail in chapter 2.

A second solution Newell offers is to analyze a single complex task. This addresses the problem that psychology often designs its experiments according to the phenomenon studied, resulting in simple tasks. The choice for a complex task is less common, because it is very hard to relate results of a complex task to a single phenomenon. Experiments using complex problems do however offer sufficient samples of all human cognitive capacities. A possible complex problem is chess. Chess involves planning, means-ends analysis, all types of learning, mental imagery, etc. If we were able to know all there is to know how people play chess, would this not be a big step towards understanding cognition in general? I will also adopt this second recommendation in this thesis. But in stead of focussing on a single task, I will focus on a single class of problems: NP-complete problems.

1.3 NP-complete problems

What is a complex problem? There are many ways to give a subjective judgement of how difficult a problem is. Chess is difficult and tic-tac-toe is easy. Fortunately, there are more formal ways to categorize problems. A formal approach also requires us to be more precise on what a problem is. First we will examine how to formally look at problems and problem solving. Then we will look at what complexity is, and by the end of the section the class of NP-complete problems will be discussed.

In informal speech, the term problem has two different meanings. We can talk about a problem as a general category, for instance the problem of deciding the next move in chess. It is not possible to give an answer to this question, because it depends on the position on the chessboard. The term problem can also be used in a more specific
sense: what move should I make on a chess board with the black king at e1, the white king at e3, and a white rook at a8? In this case a specific answer is possible: move the white rook to a1, checkmate. A problem in the general sense is a set of problems in the specific sense. To avoid confusion, the formal term problem refers to a problem in the general sense, and a specific problem is called an instance. This distinction can roughly be compared to the terms “task” and “trial” in experimental psychology: a task is a general description of what a participant must do, a trial is a specific instance of the task.

A formal definition of a problem defines it as a set of instances and a criterion. “Solving a problem” means that we decide for a particular instance whether or not it satisfies the criterion. For example, a formal description of the informal problem of deciding whether there is a forced checkmate for white specifies the set of instances as the set of all possible configurations of chess pieces on the board, and the criterion is the yes/no-question of whether a forced checkmate is possible for white. This last characterization of the criterion is of course still informal: the formal definition involves all rules of chess. “Solving a problem” in formal terms means we have a solution for all instances in the set. If the set is finite, the solution may be an enumeration of all solutions, but usually a solution for a problem is some algorithm that can decide whether the criterion holds or not. In order to formalize an informally stated problem, like “what is the best next move in a certain chess position” it must be stated as a yes/no-question, for example “Is move X in position Y the best move?”. A solution to this problem is an algorithm that computes this answer for any possible move and possible position in chess.

To be able to define the complexity of a problem in a meaningful way, it has to have an infinite set of instances and there must be some way to measure the “size” of an instance. Unfortunately, the example of chess is not infinite: although the number of positions is huge, it is nevertheless finite. The game of checkers, which is played both on an 8 x 8 board and a 10 x 10 board, can be generalized to a problem with an infinite number of instances by allowing n x n boards.

Very simple problems, however, can have infinite sets of instances. For example, the problem to decide whether a list is sorted or not has an infinite set of instances. The size can be defined by the length of the list. Summarizing, a problem can be defined in the following terms:

- A set of instances
- A criterion (a yes/no-question about instances)
- A size function on each of the instances
- A solution, i.e. an algorithm that can decide whether the criterion holds for a certain instance.
1: Introduction

Now suppose we have some way to find the “best” algorithm to solve a problem. This “best” algorithm will use computational resources. The amount of resources the algorithm consumes is an indication of its efficiency. But since it is the best algorithm for a certain problem, the efficiency of the best possible algorithm defines the complexity of the problem. So what do we mean by “use of computational resources”? There are two computational resources, time and (memory) space. Since the use of these resources is related, time is often the resource an analysis of complexity focuses on.

Complexity theory uses relative time instead of absolute time. The time it takes a certain algorithm to solve a problem is expressed in a complexity function, which maps the size of the instance on the amount of time it takes to solve the problem. This complexity function gives a much clearer indication of the efficiency of the algorithm than absolute time can. If a small increase in the size of the instance causes a large increase in time, the algorithm is inefficient. So, an algorithm with a linear complexity function is more efficient than an algorithm with a square or exponential complexity function. Complexity functions can be calculated, if the algorithm is known, or approximated empirically when the algorithm is too messy or complicated to analyze.

If we want to know the complexity of a problem, we are looking for an algorithm that solves this problem and has the best complexity function. So the complexity of a problem is the lower bound of the complexity of all the algorithms that solve it. Some problems, like deciding whether an item is in an unsorted list, have only a linear complexity. The most efficient algorithm is to examine the items in the list one by one and compare them to the item we seek. The average number of items that has to be examined is $n/2$ if the item is in the list, and $n$ if it is not in the list ($n$ is the length of the list). Other problems have a higher complexity. Problems that have an exponential time complexity are called intractable. The source of complexity is often combinatorial: if, for example, $n$ elements must be ordered, the number of possible ordenings is $n!$. If there is no systematic way to weed out the major part of these ordenings, the problem is intractable. In the case of checkers on arbitrarily large boards (I will not use chess, because it is finite), the number of board positions to be examined increases exponentially with the number of moves you want to look ahead. The question if white can win from the first move is decidable in principle, but not in practice, because there are more possible checkers games than atoms in the universe.

Why is exponential time complexity intractable, and polynomial complexity tractable? Because exponential functions grow so much faster than polynomial functions. This can be illustrated using part of a figure from Garey & Johnson (1979) that shows the time it takes to solve an instance of a problem of size $n$, given the fact that a single operation can be carried out in a microsecond (figure 1.5). One might
argue that some problems with a polynomial complexity, especially with a high exponent (e.g. \(n^{200}\)), are also intractable, but in practice these types of complexities never occur (only in contrived problems).

The consequences of intractability

Intractable problems are interesting candidates for Newell’s idea of a complex problem that exposes many aspects of human cognition. Since they have an exponential time complexity, it is impossible to use an efficient procedure that solves all instances of a problem. It is, however, not always necessary to be able to solve all instances of a problem, it may be enough to be able to solve a relevant subset of them. Relevant in this case means that the system somehow has a use for them. So for any particular intractable problem, we may have a situation similar to figure 1.4: some instances of the problem, particularly instances with a small size, can be solved efficiently, some instances need additional search that may require exponential exploration of cases, and some cases are unsolvable within a reasonable amount of time. So intractable problems may serve as a miniature but faithful representative of the case of learning problem solving.

To further improve on the representativeness of example problems, we will narrow down the set of intractable problems to the set of NP-complete problems, which is in itself a subset of NP. “NP” is an abbreviation for Non-deterministic Polynomial. A problem in NP can be solved by a non-deterministic Turing Machine in polynomial time. Less technically, given an instance of an NP-problem and a path to its solution, (so not only the yes/no-answer, but also the choices that are made to reach it) it is possible to check this solution using a tractable algorithm. In summary: finding the solution may be intractable, but checking it is tractable.
Although it has technically not yet been proven that NP-complete problems really are intractable, the general consensus is that they are for all practical purposes (Garey & Johnson, 1979). In the next section some examples of NP-complete problems will be examined, showing the broad range of domains they appear in. Nevertheless they form a tight class due to their completeness-property. This completeness property means that any NP problem can be transformed into a particular NP-complete problem by an algorithm of polynomial complexity. So, take for example the travelling salesman problem, a well-known NP-complete problem. Due to its completeness property, it is possible to take an instance of another NP-complete problem, for example resolving a particular ambiguity in a sentence, and transform this instance into an instance of the travelling-salesman problem. So if you find an efficient solution for one particular NP-complete problem, you have automatically found an efficient solution for all of them. Regrettably, this doesn’t mean that a partial solution (in terms of figure 1.4) will be at all helpful in this matter. Nevertheless, if it is possible to gain insight into how people partially overcome the problems of combinatorial explosion with respect to one particular NP-complete problem by learning, it carries the promise that this learning scheme may also work for other hard problems.

1.4 Examples of NP-complete problems

NP-complete problems may be very interesting problems to study, but this endeavor is purely academical if these problems have little to do with real-life situations. In this section a number of examples of NP-complete problems will be examined to show that NP-complete problems are part of everyday life. For some of these problems, for example language, almost everyone is an expert. For other problems, for example scheduling problems, extensive skill is normally thought of as the competence of an expert.

Most of the problems discussed here have been catalogued by Garey and Johnson (1979), together with their basic reference. Most examples explained here require some answer, instead of just “yes” or “no”. A problem that requires an answer can almost always be converted to a yes/no question, as I have shown in the case of chess.

Examples in Planning

A plan is a sequence of actions that achieves a certain goal. Sometimes reaching the goal is enough, but in other cases additional requirements must be satisfied, like finding the most efficient sequence. Planning nearly always involves time and optimizing time. People plan every day, for example how to make coffee, a plan that requires no search. Other types of planning do require some search, for example to plan a route through town to go through a shopping list (Hayes-Roth & Hayes-
Examples of NP-complete problems

![Figure 1.6. Example of the travelling-salesman problem (left)](image)

Roth, 1979), or to plan a meal (Byrne, 1977; van den Berg, 1990). Other planning tasks involve scheduling, for example school and hospital rosters, or planning symposia (Numan, Pakes, Schuurman & Taatgen, 1990). Computer science has invested much effort in programs for planning, resulting in different approaches: hierarchical, non-hierarchical, opportunistic and script-based planners (See Akyürek, 1992 for an overview).

Most planning problems are intractable unless heavily restricted. We will look at two intractable problems that are closely related to planning. In the travelling-salesman problem the task is to find the shortest closed route connecting a set of cities. More precisely, a number of cities is given and a matrix stating the distance between each pair of cities. A route is a sequence of cities, and the length of the route is the sum of the distances between successive cities. Figure 1.6 shows a case of the travelling-salesman problem with four cities. The thick line indicates the shortest route, which has a length of 15.

The general problem is NP-complete, but we can imagine a particular salesman, who always visits a subset of, say, 25 cities, and who has developed his own private strategy for solving the problem. When this salesman is transferred to another part of the country, he has only limited use for his experience: he can use some of his old knowledge, but must devise some new procedures for his new environment.

The travelling-salesman problem obviously is a planning task, and shows much resemblance to other planning tasks, for example the shopping-task from Hayes-Roth & Hayes-Roth (1979). It is often easy to prove that a certain planning task is intractable, using the fact that the travelling-salesman problem is intractable.

A second planning problem is scheduling. In this problem each instance consists of a set of tasks, each of which has a certain length, a number of workers, a partial order on the tasks, and an overall deadline. The task is to create a schedule for all the tasks,
obeying the precedence constraints as specified in the partial order and the deadline. Figure 1.7 shows an example of an instance of this problem.

Again the general problem is intractable, but particular sub-problems may be attainable. For example, the timetable of a certain school is always made by a particular deputy headmaster. Although it takes him two full weeks every year, he is the only one in the school who can do it at all. Previous experience is the key to successful problem solving in this case, another indication of the importance of learning.

Language
Understanding natural language is generally not considered to be problem solving. However, formal theories of language, especially with respect to grammar or syntax, use the same terminology as the formal theory of problem solving. For example, part of the natural language understanding process is concerned with the question whether a sentence is grammatically correct. In problem-solving terms, the set of instances is the set of all (finite) sequences of words. The criterion is the question whether a particular sequence of words is grammatically correct or not.

Part of research in linguistics concerns the construction of grammars and grammar systems that describe language. The goal of a grammar of a certain natural language is to be able to produce every grammatical sentence in that language, but no other, ungrammatical, sentences. A grammar system aims to provide a framework within which all grammars of natural languages can be fitted. Chomsky (Chomsky & Miller, 1963) has defined the basic types of grammars: finite-state, context-free, context-sensitive and unrestricted grammars, called the Chomsky hierarchy.
Examples of NP-complete problems

Grammars can produce language, but to parse natural language, to decide whether a certain sentence belongs to the language, an automaton is needed. It can be shown that each of the four grammar systems from the Chomsky hierarchy corresponds to a certain type of automaton: finite-state grammars to finite-state automata, context-free grammars to push-down automata, context-sensitive grammars to linear-bounded automata, and unrestricted grammars to Turing machines. Chomsky has shown that finite-state grammars are too restricted to be able to generate a complete natural language. Unrestricted grammars, due to their connection with Turing machines, are undecidable. This leaves context-free and context-sensitive as possible formalisms, of which context-free is always considered a more desirable alternative, because parsing a context-free grammar is tractable.

The important question is whether the generative power of context-free grammars is enough to generate natural languages.

Barton, Berwick & Ristad (1987) argue this discussion has outlived its usefulness, and more modern methods must be used. They show that the fact that a grammar is context-free is no guarantee for efficiency. The generalized phrase structure grammar system (GPSC), for example, has the seemingly attractive property that any GPSC can be converted to an equivalent context-free grammar. This suggests that since context-free grammars can be parsed efficiently, a GPSC can also be recognized easily. Barton et al. show this argument is misleading, because for a GPSC G of size \( m \) the equivalent context-free grammar has in the order of \( 3^{m!m^{2m+1}} \) rules.

Barton et al. propose complexity theory as a replacement for the equivalence-to-context-free-grammar criterion. It is a much more precise and reliable instrument to measure the efficiency of a grammar system. They also argue efficiency is an important criterion for natural language systems: if we have a formal system of a natural language that uses combinatorial search (an intractable algorithm) where it is not really necessary, there obviously is some systematic property in the language that the formal system fails to account for. For nearly all grammar systems used in linguistics, parsing turns out to be an intractable problem. According to Barton et al., this is partly due to intractable properties of language itself, but can often also be attributed to the formalism: it simply fails to account for certain features of the language. The unnatural sources of complexity must of course be expelled from the formalism, but the natural intractable properties can not. They must be accounted for by what Barton et al. call a performance theory, in which they hint at least some combinatorial search takes place.

An example of an intractable property of natural language understanding is the combination of agreement and lexical ambiguity. Agreement refers to two or more words in a sentence having the same number, gender or other feature, like in subject/verb agreement. Lexical ambiguity refers to the fact that a single word can
have different functions, as with homonyms. For example, the word ‘walk’ can be either a noun or a verb. In the case of a verb, it can be either first or second person singular or plural. Agreement grammars are simple context-free grammars that can account for both agreement and ambiguity. However, Barton et al. prove that the problem of parsing an agreement grammar is NP-complete with respect to the length of the sentence.

The conclusion is that although care must be taken to avoid unnecessary intractability in language, it cannot be avoided altogether, and what remains must be accounted for by a so-called performance theory. This performance aspect is of course rather problematic. In Chomsky’s theory the performance part of language is just a degraded version of the “ideal” competence counterpart due to human limitations. In the theory of Barton et al. performance has a function that can not be formalized but is nevertheless crucial.

So, even understanding everyday language is in itself already an intractable problem. Therefore language performance can not be explained purely by a static syntactic framework. The learning component, as is the case with other intractable problems, has to be part of the explanation of the human capacity of understanding language.

Puzzles and games
Research on problem solving is often done on toy problems. Puzzles in which letters must be replaced by numbers, missionaries and cannibals must be shipped over a river, problems where blocks must be rearranged by a robot arm, or puzzles where numbered tiles must be pushed around to get them in sequence. The problem with each of these problems is to what extent results, either empirical or by simulation, can be generalized to other domains. Especially in the case of computer simulation, the fact that a simulation solves a certain problem has no significance, because a conventional algorithm can do the same job. Even when a convincing simulation can be made, it is difficult to generalize the results.

Some games are different, however. They go beyond the toy-realm, because they keep eluding final solutions. Chess, checkers and Go are examples of games that have a long history of gradual improvement, never reaching perfection. The games of checkers and Go are intractable when generalized to an n x n board. Although chess is highly complex, it is not intractable because it can not easily be generalized to an n x n board, and standard chess games are always finite. Complexity theory needs some kind of infinity to work with. Other kind of puzzles are also intractable, for examples fitting words into an n x n crossword puzzle.

So, studying intractable problems is a far greater challenge than working with toy-problems. They pose a real challenge to problem solving, but with a larger pay-off. Since no conventional algorithms exist, the fact alone that a system simulating
The limits of task analysis, or: why is learning necessary for problem solving?

human problem solving on an intractable problem can solve certain cases is significant.

Mathematics
The main and original source of intractable problems is mathematics. Many problems involving graphs, partitioning, matching, storage, representation, sequencing, algebra and number theory are intractable (Garey & Johnson, 1979).

One of the most well-known NP-complete problems stems from logic: the satisfiability problem (SAT) (Cook, 1971). The problem is to find, for a propositional logic formula, values for the variables so that the formula evaluates to true. A straight-forward algorithm used to solve SAT is called truth-table checking, which amounts to checking every possible combination of values for the variables. Since in propositional logic a variable can have two values, the number of combinations to be checked is $2^n$, where $n$ is the number of variables. This is obviously an exponential function, leading to an intractable algorithm.

Another nice property of the problems mentioned here is the fact that they are (with the possible exception of the language problems) knowledge-lean. That is, they are already highly complex without needing huge data banks of knowledge to work on. This makes simulation a lot easier, and the results easier to interpret.

1.5 The limits of task analysis, or: why is learning necessary for problem solving?

The picture sketched in figure 1.4 is one of gradual change in mastery of a problem due to learning. But how important is this learning aspect? Suppose we want to make a task analysis of scheduling. Wouldn't it be useful to constrain the total set of instantiations of scheduling to a manageable subset, and derive a set of rules and methods that can account for that subset? More specifically, is it possible to create an account of how an expert scheduler works, assuming an expert is someone with a set of methods that is broad enough to render learning superfluous?

Suppose we have a scheduling expert. This expert can solve some instances of scheduling, but has problems with other instances: these instances take too much time to solve. For each expert, we can divide the total set of scheduling instances into two subsets: the instances he can solve and the instances he can not solve. This boundary is not entirely clear-cut, since the amount of time the expert is willing to invest in a solution plays a role, but due to the exponential increase in solution time this willingness for extra effort pushes the boundary only very slightly. There are many experts of scheduling, each of whom has his own expertise and knowledge of
scheduling, so each has his own subset of instances he can do and subset of instances he cannot do. Now suppose we want to find the ultimate scheduling expert. If the normal expert can solve something, the ultimate expert can do it too, so the set of instances that the ultimate expert can solve is the union of all sets of solvable instances of all possible experts.

In order to find the ultimate expert, we now examine a subset of all possible experts, the experts that can only solve a single instance. If this expert is presented with its instance of expertise, it gives its memorized answer, but if another instance is presented, it says it doesn’t know. So, each of these experts has a set of instances it can solve of just one member. Now, if we take the union of the knowledge of all these dumb experts, we get the ultimate dumb expert, who happens to know the answer to any instance of the problem. This is clearly in contradiction with the fact that the problem is intractable, so we must conclude that the assumption that an ultimate expert exists must be false.

The conclusion of this formal exercise is that there are no ultimate experts for intractable problems. There is always something left to learn, always a new member, or preferably, a set of members that can be added to the set of items that can be solved. But, the reply might be, suppose we incorporate this “learning” in the algorithm. Shouldn’t this algorithm be capable of solving any instance of the problem, clearly contradicting the fact that it is intractable? The answer is that a learning algorithm is not an algorithm in the normal sense. A learning algorithm changes after each instance it has or hasn’t solved, so it defies the usual analyses of algorithms. A learning algorithm is not a solution to the problem of intractability. However, it can offer explanations for the fact why intractable problems are only mildly problematic for people.

The fact that learning is an essential part of problem solving also shows that the traditional art of task analysis has its limitations. For many problems a task analysis is impossible, because even experts still learn, and use learning to solve problems. The usual idea that at some point an expert knows all there is to learn is not true in general. The same point can be made with respect to linguistics. Viewing language as a static formal structure that must be discovered by linguistic research is like trying to make a task analysis of an intractable problem, so it cannot expose the full extent of language processing.

One of the research approaches to task performance is to get a full account of performance first, and worry about learning later. The previous analysis shows this approach will not work for complex tasks. As models discussed later in this thesis show, task performance is an intricate interplay between learning and performance. Just focussing on performance will only give a very limited insight into what is going on.
If traditional task analysis is an insufficient formal theory of task performance, what should replace it? Architectures of Cognition have the capability. They are formal enough to allow general analyses and making predictions, and they incorporate learning. Instead of focussing on the knowledge of an expert, the focus will be on the learning mechanisms that allow one to become an expert and that allow experts to maintain and adapt their knowledge.

1.6 Overview of the rest of the thesis

The goal of this thesis is to gain more insight into skill-learning, in particular learning of complex problem solving. The way to accomplish this goal is to use a single theory in the form a cognitive architecture, and to start with a single complex problem, the scheduling problem. In chapter 2, the discussion is centered around the topic of the architecture. There are currently four influential architectures of cognition, Soar, ACT-R, EPIC and 3CAPS. I will first establish some general criteria to compare these architectures, after which all four architectures will be discussed.

Human problem solving on the scheduling task, discussed in chapter 3, will turn out to be a puzzle with many pieces. People tend to rehearse and forget things during problem solving. People discover new strategies if old strategies don’t work. Some global statistical analysis using multi-level statistics will chart the outlines of the learning process. A detailed protocol analysis will shed some more light on what is going on in the reasoning process.

The approach for chapter 4 to 6 will be to study each of the pieces of the puzzle offered by the experiment using well-known experiments from cognitive psychology. These tasks will be modeled in ACT-R to gain insight into how the particular phenomena relate to the cognitive system as a whole. Chapter 4 will pick up the issue of implicit and explicit learning in general, and rehearsal in particular. ACT-R offers a new type of explanation for the implicit-explicit distinction by removing its Newellian binary status and offering a unifying explanation of an apparent distinction. The bottom line will be that explicit learning can be explained by learning strategies, general knowledge specifically aimed at the acquisition of new knowledge.

Chapter 5 further investigates these learning strategies. It tries to offer a rationale for using a learning strategy, and investigates the representation of learning strategies in terms of ACT-R. The best domain to study learning strategies is developmental psychology. The idea is that learning strategies themselves have to be learned, so the best way to find out more about them is to compare children of different ages. The chapter ends in modeling two particular learning strategies, and seeing whether
they are applicable to multiple problems, and whether any evidence can be found for the fact that the strategies themselves are learned.

Chapter 6 focuses on another discussion with respect to skill learning, whether skills are learned by generalizing examples into rules, or by just storing and retrieving examples. The answer will turn out to be that both methods are used, and that the impact of these methods on performance depends on how useful they are.

In chapter 7, I return to the primary goal of modeling scheduling. Using all of the insights gained in the smaller projects of chapter 4 to 6, a model will be presented that is able to solve small scheduling problems and learn from this process in a human-like fashion. This model can be used to generate verbal protocols of problem solving, and is able to make some predictions with respect to individual differences.

Chapter 8, finally, is used to draw some conclusions. An overview will be given of the skill-learning theory developed during the thesis, and some applications of this theory are discussed. The usefulness and shortcomings of ACT-R will be discussed. In a sense, the approach used in this thesis will turn out to show close resemblance to the final theory we will arrive at. But this is as it should be, since figuring out how learning in complex problem solving works, is in itself also a form of complex problem solving.