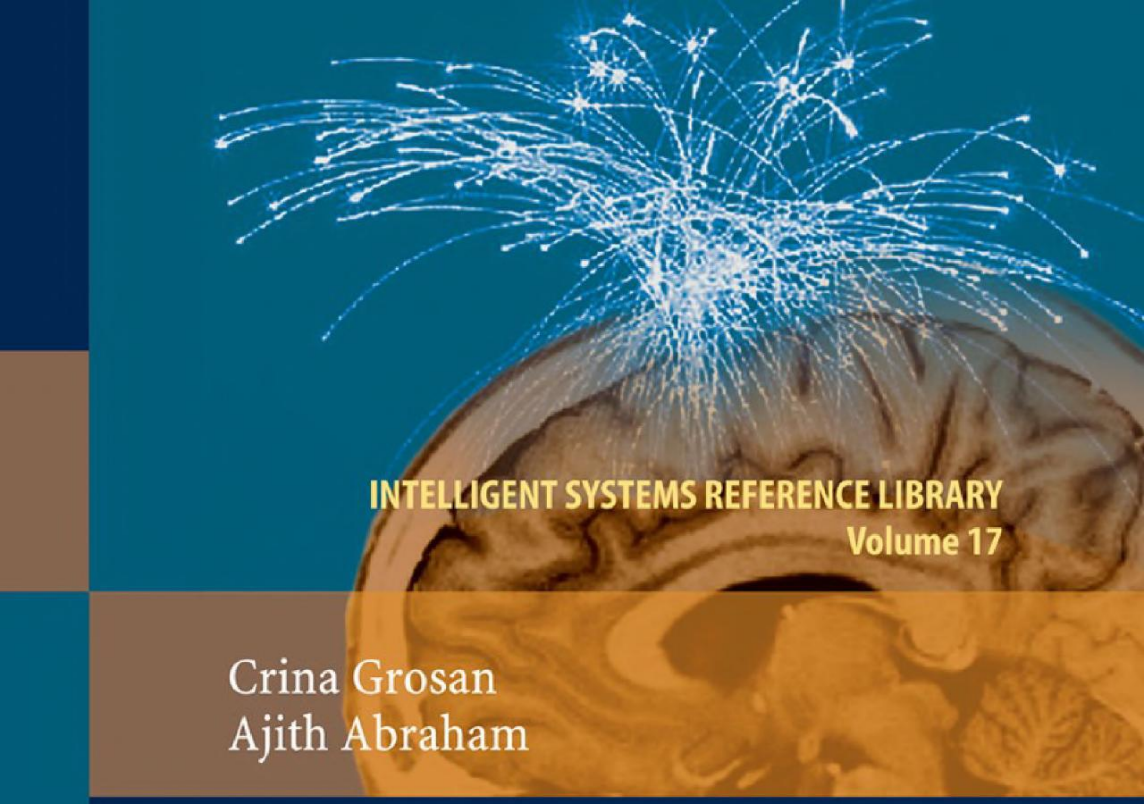

INTELLIGENT SYSTEMS REFERENCE LIBRARY
Volume 17

Crina Grosan
Ajith Abraham

Intelligent Systems

A Modern Approach

Springer

Crina Grosan and Ajith Abraham

Intelligent Systems

A Modern Approach

Dr. Crina Grosan
Department of Computer Science,
Faculty of Mathematics and Computer Science
Babes-Bolyai University, Cluj-Napoca,
Kogalniceanu 1, 400084 Cluj - Napoca
Romania
E-mail: cgrosan@cs.ubbcluj.ro

Prof. Ajith Abraham
Machine Intelligence Research Labs
(MIR Labs)
Scientific Network for Innovation and
Research Excellence
P.O. Box 2259
Auburn, Washington 98071-2259
USA
Email: ajith.abraham@ieee.org

ISBN 978-3-642-21003-7 e-ISBN 978-3-642-21004-4

DOI 10.1007/978-3-642-21004-4

Intelligent Systems Reference Library ISSN 1868-4394

Library of Congress Control Number: 2011928063

Typeset & *Cover Design:* Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Contents

Chapter 1
Evolution of Modern Computational Intelligence

1.1 Introduction

A conventional computational intelligence book introduction starts, many a times, with a history of Artificial Intelligence (AI) and what has been done up to date. This book introduction will try to start with what can be envisaged as future Computer Intelligence.

It is important to have a common definition of AI. Although it might be possible to find many (somehow similar) definitions for *artificial intelligence*, probably the most appropriate one can be the one stating: *creating machines which solve problems in a way which, done by humans, require intelligence*.

A question arises, which can be the interactive question open in any AI course: do we have artificial intelligence? The answer is not a simple one. There are at least two ways of seeing things.

If we look around at the existing *intelligent machines*, we can tell (just to enumerate a few examples) that we have machines, which can interpret handwriting better than humans, we have machines which take decision better than humans do, we have machines which make calculation millions of times faster than humans, we have machines that interpret data, huge amount of data, much faster and accurate than humans, machines which understand language and interpret and transcript it at least at the same level as humans and examples can continue. All these are just natural nowadays, but were hard to believe two decades ago.

On the other hand, if we look at the existing *machines* from a human level intelligence point of view, it is hard to admit that we have a *human level intelligent machine*. Intelligence, on its own, has a broad interpretation sense. If we look at a very intelligent man (usually Einstein is given as reference for an intelligent man) and we look at a person from a remote place, with less or no contact with the civilized world, we see a huge difference (in terms of intelligence) between the two. But if we look at the same person from the remote mountain and at a cockroach, we think that difference between Einstein and our mountain man is nothing compared to difference between mountain man and cockroach. From the evolution of human habilis (first human-like ancestors) 2 million years ago, to homo sapiens 100 000 years ago, to agricultural revolution 10 000 years ago and then to the

C. Grosan and A. Abraham: Intelligent Systems, ISRL 17, pp. 1–11.
springerlink.com

industrial revolution, the human intelligence undergoes significant improvements. But nowadays scientists clearly state that, in a natural way, no improvement can be further performed to the human intelligence; thus, the need of an artificial, more powerful intelligence.

Human level intelligence has not yet been reached if we measure this achievement as passing the Turing test. Turing test - proposed by Alan Turing, a British mathematician – also known as "imitation game" is a simulation in which a judge attempts to distinguish which of two agents, in two separate rooms, is a human and which a computer imitating human from their responses in a wide-ranging conversation of any topic. As a note to the Turing test, there is an annual competition known as Leobner Prize, which awards the best instantiation of the Turing Test.

Nick Bostrom (Future of Humanity Institute, Oxford University) noted that for achieving artificial intelligence, three things are required: hardware, software, and input/output mechanisms. The input/output mechanisms refer to the technology required by a machine to interact with its environment. This is already available in the form of cameras and sensors. We already see robots performing several human like tasks, etc. Thus, this is the simplest required part.

For hardware, we really need to have human level speed and high memory machines. In terms of memory, things are promising. For speed, we still have to wait. Human brain processing power ranges between 100 million MIPS to 100 billion MIPS. (1 MIPS = 1 Million Instructions Per Second). Fastest supercomputer today (as of 2010) is Jaguar, built by the Cray Company and housed at the Oak Ridge National Laboratory in Tennessee; it has a top speed of 1.75 petaflops per second. This means we don't have yet human level computer power even at the range of supercomputers.

The other remaining problem is software. Once we get human intelligence level machines, software will be required. In doing so, one has to understand how human brain works. This is part of the current research these days and at least two main directions follow from there: computational neuroscience and molecular nanotechnologies. Neuroscience is concerned with how the individual components of the human brain work. Research up to date reports good computational models of primary visual cortex. But simulating the whole brain requires enormous computing power. Molecular nanotechnologies work at nanoscale level which is 1 to 100 nanometers – from 1/1,000,000 to 1/10,000 of the thickness of an American dime. Many of the key structures of human nervous systems exist at nanoscale. The major challenge is to use nanomachines to disassemble a frozen or a vitrified human brain.

In parallel with getting the artificial intelligence or human level artificial intelligence, small steps have been performed in terms of algorithms and methodologies, which can be applied to solve simple or more challenging real-world problems. Much of the current research focuses on the principles, theoretical aspects, and design methodology of algorithms gleaned from nature. Examples are artificial neural networks inspired by mammalian neural systems, evolutionary computation inspired by natural selection in biology, simulated annealing inspired by thermodynamics principles and swarm intelligence inspired by collective behavior of insects or micro-organisms etc. interacting locally with their environment

causing coherent functional global patterns to emerge. These techniques have found their way in solving real world problems in science, business, technology and commerce. Computational intelligence is a well-established paradigm, where new theories with a sound biological understanding have been evolving. The current experimental systems have many of the characteristics of biological computers (brains in other words) and are beginning to be built to perform a variety of tasks that are difficult or impossible to do with conventional computers.

Although most of the AI related publications consider the birth of AI 15 years after the development of the first electronic computer (in 1941) and 7 years after the development of the invention of the first stored program computer (in 1949), evidences of artificial intelligence can be traced back in ancient Egypt and Greece. Most of AI scientists consider that the Dartmouth summer research project, organized in 1956 by John McCarthy (regarded as father of AI) at Dartmouth College in Hanover, New Hampshire, where the "artificial intelligence term has been coined" was the actual start of the AI as a science.

1.2 Roots of Artificial Intelligence

Logic is considered as being one of the main roots of AI. AI has been heavily influenced by logical ideas. Most members of the AI community would agree that logic has an important role to play in at least some central areas of AI research, and an influential minority considers logic to be the most important factor in developing strategic, fundamental advances. It started as long ago as in 5th century B.C. when Aristotle invented syllogistic logic, the first formal deductive reasoning system. The advances continued with small steps, with famous inventions of this millennium, examples like printing using movable type in the 15th century, invention of clocks as measuring machines in the 15th – 16th century, extension of this mechanism for the creation of other moving objects in the 16th century and so on. Pascal has invented the first mechanical digital calculating machine in 1642. This machine was an adding machine only, but later, in 1671, the German mathematician - philosopher Leibniz designed an improvement of the adding machine such as to incorporate multiplication and division. The machine – known as Step Reckoner – was built in 1973. The 19th century brings the ingenious project of the first computing machine. Looking for a method, which can overcome the high error rate in the calculation of mathematical tables, English mathematician Charles Babbage wished to find a way by which they could be calculated mechanically, removing human sources of error. He began to build Difference Engine, a mechanical device that can perform simple mathematical calculations in 1820 and then the Analytical Engine, which was designed to carry out more complicated calculations. Both devices finally remain just as prototype computing machines. Babbage's work has been later continued by Ada Augusta Lovelace, which remains as the world's first programmer. Babbage's Difference Engine was the first successful automatic calculator.

Another important contribution of 19th century is George Boole's logic theory, also known as Boolean logic or Boolean algebra. Even thought not much appreciated at the time it has been proposed, later after the publication of Boole's ideas,

an American logician Charles Sanders Peirce spent more than 20 years modifying and expanding them, realizing the potential for use in electronic circuitry and eventually designing a fundamental electrical logic circuit. Pierce never actually built his theoretical logic circuit, being himself more of a logician than an electrician, but he did introduce Boolean algebra into his university logic philosophy courses.

It was later when one of his students – Claude Shannon, one of the organizers of the Dartmouth conference and one of the pioneers of AI, a Nobel Prize winner among others – made full use of all these ideas.

Gottlob Frege, a German mathematician, essentially reconceived the discipline of logic by constructing a formal system, which in effect, constituted the first predicate calculus (1893-1903). Frege's logic calculus system consisted of a language and an apparatus for proving statements. Predicate calculus system consisted of a set of logical axioms (statements considered to be truths of logic) and a set of rules of inference that lay out the conditions under which certain statements of the language may be correctly inferred from others.

The 20th century brings the most significant contributions to the AI field. If the first half of the century is not that remarkable, starting with the second half results will come in an impressive rhythm. Bertrand Russell, the British logician who pointed out some of the contradictions of Frege's logic during their correspondence and who refined the predicate calculus, revolutionizes formal logic with his three-volume work he co-authored with Alfred North Whitehead, *Principia Mathematica* (1910, 1912, 1913). The mathematical logician Emil Post had his important contributions to computer science in the beginning of the 20th century. In his later work during the early 1920s, Post developed his notion of production systems, developed a unification algorithm, and anticipated the later findings of Gödel, Church, and Turing. Post developed a programming language without thinking of a machine on which it could be implemented. Another important logician of the 20th century is Kurt Gödel, who proved the incompleteness of axioms for arithmetic, as well as the relative consistency of the axiom of choice and continuum hypothesis with the other axioms of set theory.

One of the most significant figures in the development of mathematical logic is Alonzo Church, A Princeton professor and Alan's Turing's supervisor. His book – *Introduction to Mathematical Logic* – published in 1944 comprises some of his earlier remarkable results. The Church-Turing Thesis, a controversial work, came to solve one of the important problems for logicians formulated in the 1930s by David Hilbert: Entscheidungsproblem. The problem asks if there was a mechanical procedure for separating mathematical truths from mathematical falsehoods. Probably the most controversial figure among the mathematicians of the 20th century, the British mathematician Alan Turing is well known as the founder of some fundamental principles, which are required to prove the evidence of artificial intelligence. The famous Turing test remains until today the biggest challenge for the existence of artificial intelligence. His famous work Computing Machinery and Intelligence has been published in 1950, soon after the development of the first electronic digital computer and the first stored computer program.

Built in 1943-1945 at the Moore School of the University of Pennsylvania for the War effort by John Mauchly and J. Presper Eckert, the Electronic Numerical Integrator And Computer (ENIAC) was the first general-purpose electronic digital computer. It was 150 feet wide with 20 banks of flashing lights. Even thought it was meant to help in the WWII, ENIAC has not been delivered to the Army until just after the end of the war.

The ENIAC was not a stored-program computer; it is described by David Alan Grier as a collection of electronic adding machines and other arithmetic units, which were originally controlled by a web of large electrical cables. EDVAC (Electronic Discrete Variable Automatic Computer) was the earliest electronic computer. Unlike its predecessor the ENIAC, it was binary rather than decimal, and was a stored program machine.

The paper by Warren McCulloch, a neuroscientist, and Walter Pitts, a logician, "*A logical calculus of the ideas immanent in nervous activity*" published in 1943 is regarded as the start point of two fields of research: the theory of finite-state machines as a model of computation and the field of artificial neural networks. McCulloch and Pitts tried to understand how the brain could produce highly complex patterns by using many basic cells that are connected together. They gave a highly simplified model of a brain cell – a neuron – in their paper. The McCulloch and Pitts model of a neuron has made an important contribution to the development of artificial neural networks. But their neuron model had limitations. Additional features were added, which allowed the neuron to learn and one of the next major development in neural networks was the concept of a perceptron, which was introduced by Frank Rosenblatt in 1958. Another paper published in the same 1943 – "Behavior, Purpose and Teleology" – by Arturo Rosenblueth, Norbert Wiener and Julian Bigelow set the bases for the new science of Cybernetics.

The problem solving has been a central challenge for computer scientists and for the AI community too. AI scientists came with their own problems and with their own methods of solving them. George Polya, a Hungarian born American mathematician, suggests in his very famous book – *How to solve it* – four main steps to approach a problem: understand the problem, devise a plan, carry on with the plan and look back. Problem solving remains a central idea of AI and a *How to solve it* modern version using heuristics has been published in 2004 by Zbigniew Michalewicz and David Fogel.

A few important scientific results preceded the Dartmouth Conference. Among them are the following: Norbert Wiener's results in cybernetics (he is among the first scientists who coined the term cybernetics) and also in the feedback theory as if all intelligent behavior is the results of feedback mechanisms. This discovery had a huge influence on the initial development of AI. *The logic theorist* developed between 1955-1956 by Allen Newell (researcher in computer science and cognitive psychology at Carnegie Mellon University), J. Clifford Shaw (a system programmer who is considered the father of the JOSS language) and Herbert Simon (originally a political scientist who also won the Nobel Prize in economics in 1978 and has been awarded the Turing Award along with Allen Newel in 1975 for their basic contributions to artificial intelligence and the psychology of human cognition) is considered as being the first AI program. The

theorem proving can be reduced to search. The problem is represented as a tree model and the program will attempt to find a proof by searching the tree and by selecting the branch that will result in the correct proof. The program succeeded in proving thirty-eight of the first fifty-two theorems presented there, but much more importantly, the program found a proof for one theorem which was more elegant than the one provided by Russell and Whitehead (in *Principia Mathematica*). The impact of *The Logic Theorist* had in the development of AI made it a stepping-stone in the evolution of the AI field.

Although the enthusiasm of organizing the school at Dartmouth College was really huge and the expectations were great, as McCarthy he noted in the 1955 announcement of the conference:

"We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer."

the results of the meeting were not really spectacular. The conference was organized by John McCarthy and formally proposed by John McCarthy, Marvin Minsky, Nathaniel Rochester and Claude Shannon with the scope of bringing together American scientists working on artificial intelligence. There were a total of 10 participants at the Dartmouth Summer Research Conference on Artificial Intelligence. John McCarthy (who was teaching at Dartmouth at that time and after moved to Stanford University; also won Turing Award in 1971), Marvin Minsky (who also won the Turing award in 1969), Trenchard More (from Princeton), Ray Solomonoff (the inventor of algorithmic probability and an originator of the branch of artificial intelligence based on machine learning, prediction and probability), Oliver Selfridge (graduate student of Norbert Wiener's at MIT, (but did not write up his doctoral research and never earned a Ph.D.) and a supervisor of Marvin Minsky), Claude Shannon (known for his contributions in information theory and cryptography during the World War II while he was at Bell Labs; among other contributions he made a chess playing computer program and made a fortune by applying game theory in Las Vegas games and in stock market), Nathaniel Rochester (who designed the IBM 701 the first general purpose, mass produced computer and wrote the first symbolic assembler), Arthur Samuel (who developed the alpha-beta tree idea and proposed a Checkers-playing program (on IBM's first commercial computer, the IBM 701) that appears to be the world's first self-learning program; 1962 his program beat a state champion), Herbert Simon and Allen Newell.

1.3 Modern Artificial Intelligence

The Dartmouth Conference opened the era of new and most significant advances in the AI field. Advances continued in a much faster rhythm than before. Technology was also advancing and this gave more room to more difficult and ambitious projects. AI research centers began forming at MIT and Carnegie Mellon University. The challenges were to create systems that could efficiently solve problems by limiting the search such as The Logic Theorist, and making systems that could learn by themselves. Newel, Shaw and Simon, the authors of The Logic Theorist, wanted programs that solved problems in the same ways as humans do. They developed the General Problem Solver (GPS) in 1957, which is basically a computer program intended to work as a universal problem solver machine. Any formalized symbolic problem can be solved, in principle, by GPS, for instance theorems proof, geometric problems and chess playing.

Using a means-end-analysis approach, GPS would divide the overall goal into sub-goals and attempt to solve each of those. The program was implemented in the low-level IPL programming language. While GPS solved simple problems such as the Towers of Hanoi that could be sufficiently formalized, it could not solve any real-world problems because search was easily lost in the combinatorial explosion of intermediate states.

McCulloch and Pitts' neuron was further developed in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory. Rosenblatt's perceptron was able to recognize patterns of similarity between new data and data it has already seen in a feed-forward model that demonstrated a primitive type of learning or trainability. His work was highly influential in the development of later multi-layered neural networks. Soon after the development of the perceptron, many research groups in the United States were studying perceptrons. Essentially the perceptron is a McCulloch and Pitts neuron where the inputs are first passed through some "preprocessors," which are called association units. These association units detect the presence of certain specific features in the inputs. In fact, as the name suggests, a perceptron was intended to be a pattern recognition device, and the association units correspond to feature or pattern detectors.

In 1958, John McCarthy showed how, given a handful of simple operators and a notation for functions, someone can build a whole programming language. He called this language LISP, for "List Processing," because one of his key ideas was to use a simple data structure called a *list* for both code and data. LISP is the second-oldest high-level programming language in widespread use today; only Fortran is older. LISP was heavy on computer power and it became more useful in 1970s with the existing technology.

In the late 50's and early 60's Margaret Masterman and colleagues from Cambridge design semantic nets for machine translation. A semantic net is a graph, which represents semantic relations among concepts. Silvio Ceccato also developed in 1961 correlational nets, which were based on 56 different relations, including subtype, instance, part-whole, case relations, kinship relations, and various kinds of attributes. He used the correlations as patterns for guiding a parser and resolving syntactic ambiguities. Masterman and her team developed a list of 100

primitive concept types, such as Folk, Stuff, Thing, Do, and Be. In terms of those primitives, her group defined a conceptual dictionary of 15,000 entries. She organized the concept types into a lattice, which permits inheritance from multiple supertypes.

The first industrial robot was installed at General Motors in 1961. It has been developed at Unimation Inc., the first robotic company founded in 1956 by Joseph F. Engelberger (a physicist, engineer and entrepreneur who is referred to as the "Father of Robotics"). Over the next two decades, the Japanese took the lead by investing heavily in robots to replace people performing certain tasks.

In 1963, John Alan Robinson, philosopher, mathematician and computer scientist invented resolution, a single inference method for first order logic. Resolution is a refutation method operating on clauses containing function symbols, universally quantified variables and constants. The essence of the resolution method is that it searches for local evidence of unsatisfiability in the form of a pair of clauses, one containing a literal and the other its complement (negation). Resolution and unification have since been incorporated in many automated theorem-proving systems and are the basis for the inference mechanisms used in logic programming and the programming language Prolog.

In 1963, DARPA (Defense Advanced Research Project Agency) and MIT signed a 2.2 million dollar grant to be used in researching artificial intelligence (to ensure that the US will stay ahead of the Soviet Union in technological advancements).

In 1966, Joseph Weizenbaum form MIT described in *Communications of the ACM*, ELIZA, one of the first programs that attempted to communicate in natural language. In only about 200 lines of computer code, Eliza models the behavior of a psychiatrist (the Rogerian therapist). ELIZA has almost no intelligence; it uses tricks like string substitution and canned responses based on keywords. The illusion of intelligence works best, however, if you limit your conversation to talking about yourself and your life.

Some of the more well-known AI projects that followed the General Problem Solver in the late 60's included: STUDENT, by Daniel G. Bobrow, which could solve algebra word problems and reportedly did well on high school mach tests, ANALOGY, by Thomas G. Evans (written as part of his PhD work at MIT), which solved IQ-test geometric analogy problems, Bert Raphael's MIT dissertation on the SIT program that demonstrates the power of logical representation of knowledge for question-answering systems and Terry Winograd's SHRDLU, which demonstrated the ability of computers to understand English sentences in a restricted world of children's blocks (such as a limited number of geometric shapes).

Another advancement in the 1970's was the advent of the expert system. Expert systems predict the probability of a solution under set conditions. Due to the large storage capacity of computers at the time, expert systems had the potential to interpret statistics, to formulate rules. The applications for real practical problems were extensive, and over the course of ten years, expert systems had been introduced to forecast the stock market, medicine and pharmacy, aiding doctors with the ability to diagnose disease, and instruct miners to promising mineral locations.

This was made possible because of the systems ability to store conditional rules, and storage of information.

One of the earliest expert systems was DENDRAL, developed at Stanford University. DENDRAL was designed to analyze mass spectra. DENDRAL did contain rules and consists of two sub-programs, Heuristic Dendral and Meta-Dendral and its developers believed that it can compete and experienced chemist (was marketed commercially in the United States). The program was used both in industry and academia. MYCIN, another expert system developed at Stanford University too has been used to diagnose blood infections and recommend treatments, given lab data about tests on cultures taken from the patient. Although never put to practical use, MYCIN showed to the world the possibility of replacement of a medical professional by an expert system. PROSPECTOR has been developed by NASA. It takes geological information about rock formations, chemical content, etc, and advises on whether there were likely to be exploitable mineral deposits nearby. Popular accounts of AI say that Prospector (in 1978-ish) discovered a hundred-million-dollar deposit of molybdenum.

These are only some of the first expert systems. Many more have been proposed, including applications in all major domains such as medicine, agriculture, engineering, etc. Rule-based systems are a relatively simple model that can be adapted to any number of problems. A general form of expert systems is an expert system shell. An expert system shell is actually an expert system whose knowledge is removed. Thus, the user can just add its own knowledge in the form of rules and provide information to solve the problem. Expert system shells are commercial versions of the expert systems.

The programming language PROLOG was born of a project aimed not at producing a programming language but at processing natural languages; in this case, French. The project gave rise to a preliminary version of PROLOG at the end of 1971 and a more definitive version at the end of 1972 at Marseille by Alain Colmerauer and Philippe Roussel. The name Prolog stands for Progammation en Logique in French and was coined by Philippe Roussel. It can be said that Prolog was the result of a combination between natural language processing and automated theorem-proving.

It was in 1964 when the new theory of *fuzzy logic*, a different kind of logic, has been proposed by Lotfi Zadeh at University of California (Berkeley). The concept was not much used at that time in the United States, but in the 70's the Japanese started using fuzzy ideas incorporated in electronic devices. The fuzzy mechanisms were first developed for years in Japan before the rest of the world started using them. It took a long time until fuzzy logic got accepted even though it fascinated some people right from the beginning. Besides engineers, philosophers, psychologists, and sociologists soon became interested in applying fuzzy logic into their sciences. In the year 1987, the first subway system was built which worked with a fuzzy logic-based automatic train operation control system in Japan. It was a big success and resulted in a fuzzy boom. Universities as well as industries got interested in developing the new ideas. Today, almost every intelligent machine has fuzzy logic technology inside it.

Neural networks remained for years only at the stage of a single neuron (perceptron) since their discoveries in the 60's. Due to lack of machine power required for their computational tasks, neural network research didn't progress much until in the mid 80's.

In the early 1980's, researchers showed renewed interest in neural networks. Recent work includes Boltzmann machines, Hopfield nets, competitive learning models, multilayer networks, and adaptive resonance theory models. With the backpropagation learning algorithm (and later on with other learning algorithms) neural networks became widely used. Neural networks are adequately used for data classification and modeling through a learning process.

Another important milestone in the field of AI is the development of Evolutionary Computation. Under the name evolutionary computation, four major domains are covered: genetic and evolutionary algorithms, evolution strategies, evolutionary programming and genetic programming. Although some work in this field can be traced back to the late 1950's, the field remained relatively unknown to the broader scientific community for almost three decades. This was largely due to the lack of available powerful computer platforms at that time. The fundamental work of John Holland, Ingo Rechenberg, Hans-Paul Schwefel, Laurence Fogel and John Koza represents the base of the evolutionary computation, as we know it today. Holland introduced genetic algorithms, probably the most studied and further developed branch of evolutionary computation, with remarkable application in optimization and search problems. Ingo Rechenberg and Hans-Paul Schwefel contributed to the development of evolution strategies. Fogel proposed evolutionary programming and Koza is known for his contributions to the genetic programming methods. All these methods have been (and still continue to be) further developed and improved, with hundreds of thousands of publications related to this subject.

Swarm intelligence is a method, which allows decentralized, self-organized systems with relative simple single software agents to solve complex problems and tasks together, which neither agent could do alone. Examples include ants (from which the Ant Colony Optimization system has derived), which leave pheromone trails for others to follow, and go as far as swarm-robots being able to symbiotically share computing resources, birds and fish (from which the Particle Swarm Optimization algorithm developed), bacteria (which gave birth to Bacterial foraging optimization algorithm), Multi-Agent Systems are systems of similar, possibly specialized entities, which are able to collectively solve problems and so on. Swarm robotics is a comparative young field of science, focusing on the development of limited single robots which are able to perform direct and indirect communication with each other and to create dynamic horizontal systems with a collective behavior.

Apart from all these, some progress has been registered in computer games playing. For checkers game, there exist Chinook. After 40-year-reign of human world champion Marion Tinsley, Chinook defeated it in 1994. Chinook used a pre-computed end game database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions. For Chess game, there exists Deep Blue. Deep Blue defeated human world champion Garry

Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply. In Othello game, human champions refuse to compete against computers, who are too good. Various kind of robots have been developed in the last century (and many continue to be developed today) to help and replace human work in hard and improper conditions.

1.4 Metamodern AI

In the new millennium, the trends in AI remain almost the same but more courageous, with more enthusiasm and by far with more advanced technologies. Apart from the new developments in terms of concepts and methods, ensembles of existing paradigms and hybrid intelligent approaches play an important role. Interdisciplinary approaches towards problem solving are another key idea. But involving experts from multiple domains such as engineering, biology, computer science and cognitive sciences, the progress is much faster. For example, there is a specific interdisciplinary trend, NBIC, which stands for Nano-Bio-Info-Cogno, whose ideas and research plans sound very promising. Universal Artificial Intelligence, idea proposed by Juergen Schmidhuber, comes with universal reinforcement learners and decision makers.

A more general Idea is that of Singularity, a concept originally coined by Vernor Vinge and sustained by Ray Kurzweil and other researchers of the Singularity Institute for Artificial Intelligence. The Singularity is the technological creation of smarter-than-human intelligence and it is most likely to happen next the machine will reach human level artificial intelligence.

The book offers a gentle introduction to modern computational intelligence field starting with the first and most simple ways to approach problem solving (some standard search techniques) and then continues with other methods in a chronological order of their development. The contents of this book would be beneficial for various disciplines and is structured for a larger audience, from medical doctors, researchers / scientists / students / academicians and engineers from the industry.

Chapter 2
Problem Solving by Search

2.1 Introduction

An important aspect of intelligence is *goal-based* problem solving. Several problems can be formulated as finding a sequence of actions that lead to a desirable goal. Each action changes the *state* and the aim is to find the sequence of actions and states that lead from the initial state to a final (goal) state.

Searching through a state space involves the following:

- a set of states;
- operators;
- a start or initial state;
- a test to check for goal state.

A well-defined problem can be described by[1][2][3]:

- *Initial state*;
- Operator or *successor function* - for any state x returns $s(x)$, the set of states reachable from x with one action;
- *State space* - all states reachable from initial state by any sequence of actions;
- *Path* - sequence through state space;
- *Path cost* - function that assigns a cost to a path. Cost of a path is the sum of costs of individual actions along the path;
- *Goal test* - test to determine if at goal state.

2.2 What Is Search?

Search is the systematic examination of states to find a path from the start state to the goal state.

The *search space* consists of the set of possible states, together with operators defining their connectivity.

The solution provided by a search algorithm is a path from the initial state to a state that satisfies the goal test[4][6][7][8][9][11][12][18][20].

C. Grosan and A. Abraham: Intelligent Systems, ISRL 17, pp. 13–52.
springerlink.com

In real life situations search algorithms are usually employed when there is lack of knowledge and the problems cannot be solved in a better way.

Search techniques fall into three groups:

- methods which find *any* start - goal path;
- methods which find the *best* path;
- search methods in the face of adversaries.

The hardship in problem solving is to decide the states and the operator or successor function. Figures 1-3 illustrate some examples depicting the different modeling aspects of the search process.

Example 1: 8-puzzle

In the 8-puzzle example depicted in Figure 2.1 we have[10][26][33]:

States: location of blank and location of the 8 tiles
Operator (*successor*): blank moves left, right, up and down
Goal: match the state given by the Goal state
Path Cost: each step has the cost 1; total cost is considered as being the length of path.

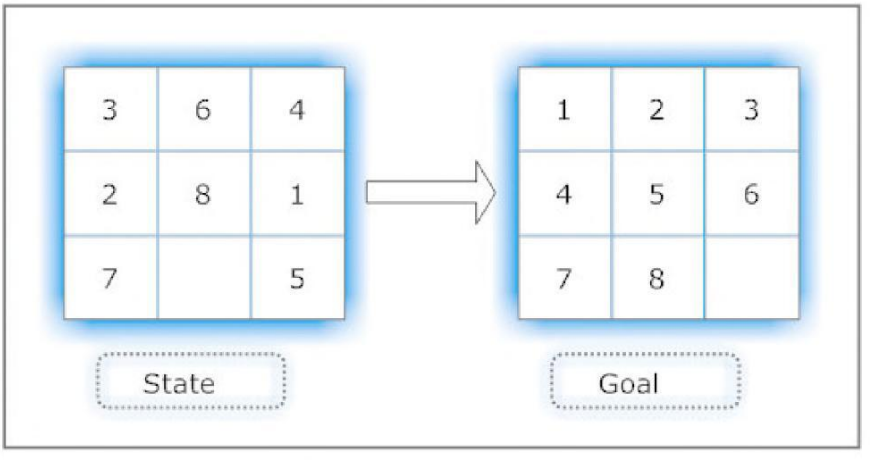

Fig. 2.1 8-puzzle example.

Example 2: N - Queens

The N-Queens problem requires arranging N queens on an N × N (chess) board such as the queens do not attack each other. This problem may be defined as:

States: 0 to N queens arranged on the chess board
Operator (*successor*): place a queen on an empty square

Goal: match a state with N queens on the chess board and no attacks among them (an example of a 5-queen goal state is given in Figure 2.2).
Path Cost: 0.

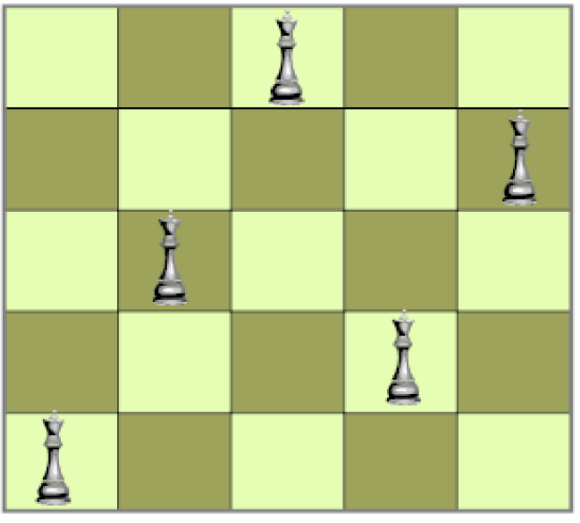

Fig. 2.2 N-queen (N=5) problem example.

Example 3 - Missionaries and Cannibals Problem

The problem can be stated as follows: three missionaries and three cannibals are on the left bank of a river. They have to cross over to the right bank using a boat that can only carry two at a time. The number of cannibals must never exceed the number of missionaries on any of the banks. The problem is to find a way to get all missionaries and cannibals to the other side, without leaving at any time and place a group of missionaries outnumbered by the cannibals.

For this problem we define:
State: The state consists of:

- the number of missionaries on the left bank,
- the number of cannibals on the left bank,
- the side of the bank the boat is on.

Operator: A move is represented by the number of missionaries and the number of cannibals taken in the boat at one time. Since the boat can carry no more than two people at once, there are 5 possible combinations:

```
(2 Missionaries, 0 Cannibals)
(1 Missionary, 0 Cannibals)
(1 Missionary, 1 Cannibal)
(0 Missionary, 1 Cannibal)
(0 Missionary, 2 Cannibals)
```

Goal: (0, 0, right)
Path cost: number of crossings.

2.3 Tree Based Search

The set of all paths within a state-space can be viewed as a graph of nodes, which are connected by links. If all possible paths are traced out through the graph, and the paths are terminated before they return to nodes already visited (cycles) on that path, a search tree is obtained. Like graphs, trees have nodes, but they are linked by branches. The start node is called the root and nodes at the other ends are leaves. Nodes have generations of descendents. The first generations are children. They have a single parent node, and the list of nodes back to the root is their ancestry. A node and its descendents form a subtree of the node's parent. If a node's subtrees are unexplored or only partially explored, the node is open, otherwise it is closed. If all nodes have the same number of children, this number is the branching factor[27].

2.3.1 Terminology

- *Root node*: represents the node the search starts from;
- *Leaf node*: a terminal node in the search tree having no children;
- *Ancestor/descendant*: node A is an ancestor of node B if either A is B's parent or A is an ancestor of the parent of B. If A is an ancestor of B, B is said to be a descendant of A;
- *Branching factor*: the maximum number of children of a non-leaf node in the search tree;
- *Path*: a path in the search tree represents complete path if it begins with the start node and ends with a goal node. Otherwise it is a partial path.

A node in the tree may be viewed as a data structure containing the following elements:

- a state description;
- a pointer to the parent of the node;
- depth of the node;
- the operator that generated this node;
- cost of the path (sum of operator costs) obtained from the initial (start) state.

It is advisable *not* to produce complete physical trees in memory, but rather explore as little of the virtual tree looking for root-goal paths [1][5].

State space is explored by generating successors of the already explored states. Every state is evaluated in order to see whether this is the goal state. A disadvantage of the tree search is that it can end up repeatedly visiting the same node. A solution to this is to store all the visited nodes but this will require a lot of memory resources. A more general approach is the graph search.

The nodes that the algorithm has generated so far during the search process are kept in a data structure called OPEN or *fringe*. Initially only the start node (the initial state) is in OPEN.

The search starts with the root node. The algorithm picks a node from OPEN for expanding and generates all the children of the node. Expanding a node from OPEN results in a closed node. Some search algorithms keep track of the closed nodes also in a data structure called CLOSED[29].

The search problem will return a solution or a path to a goal node. Finding a path is important in problems like path finding, *n*-puzzle problems, traveling salesman problem and other such problems. There are also problems like the N-queens and cryptarithmetic problem for which the path to the solution is not important. For such problems the search problem needs to return the goal state only.

An Example of search tree for the 8-puzzle problem is depicted in Figure 2.3.

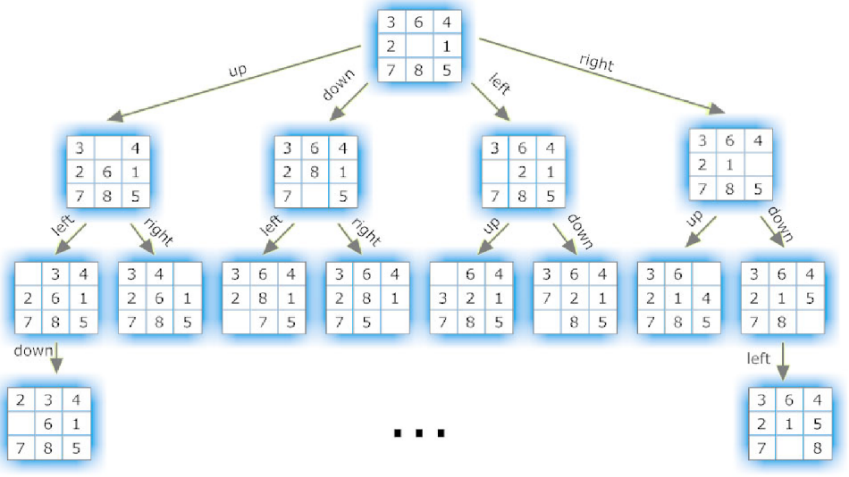

Fig. 2.3 Example of search tree for the 8-puzzle problem.

2.4 Graph Search

If the search space is not a tree, but a graph, the search tree may contain different nodes corresponding to the same state. The state space can be considered a graph G(V, E), where V is the set of nodes and E is a set of vertices, which are directed from a node to another node. Each node contains information including:

- a state description;
- node's parent;
- the operator that generated the node from that parent;
- other information.

Each vertex corresponds to an instance of one of the operators. When the operator is applied to the state associated with the arc's source node, then the resulting state is the state associated with the vertex's destination node. Each vertex has a positive cost associated with it corresponding to the cost of the operator.

Each node has a set of successor nodes corresponding to all of the operators that may be applied at the source node's state. Expanding a node means generating all the successor nodes of it and add them and their associated vertices to the state-space graph.

We have the following correspondence:

- *Initial state*: One or more nodes are designated as start nodes.
- *State space* – Initially, a starting node S is considered and V={S}. Then S is expanded and its generated successors (nodes and vertices) are added to V and E respectively. This process continues until a goal node is found;
- *Path* - each node represents a partial solution path from the start node to the given node. In general, from this node there are many possible paths (and therefore solutions) that have this partial path as a prefix;
- *Path cost*: the sum of the vertices costs on the solution path;
- *Goal test* – test applied to a state to determine if its associated node is a goal node and satisfies all goal conditions;
- *Solution*: a sequence of operators that is associated with a *path* in a state space from a start node to a goal node.

Remarks

(i) Search process constructs a search tree, where root is the initial state and all the leaf nodes are either nodes that have not yet been expanded or nodes that have no successors.

(ii) Because of loops, search tree may be infinite even for small search spaces.

The general search structure is given in Algorithm 2.1. *Problem* describes the start state, operators, goal test and costs. *Strategy* is what differentiates different search algorithms; based on it, several search methods exist. The result of the algorithm is either a valid solution or failure.

Algorithm 2.1

```
General_search (problem, strategy)
Use initial state of the problem to initialize the
search tree
Loop
        If there are no nodes to expand
        Then return failure;
        Based on strategy select a node for extension;
        Apply goal test;
```

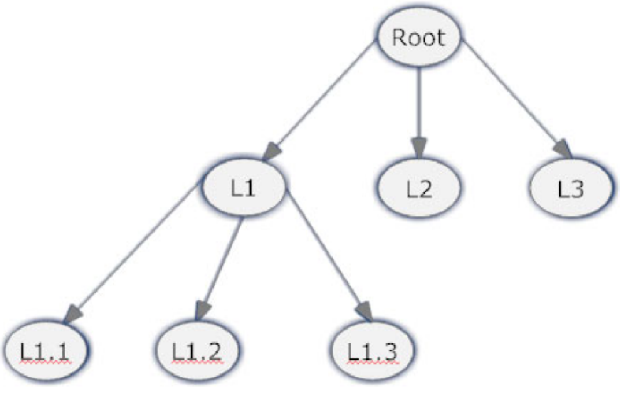

Fig. 2.4 Example of states layers in breadth first search.

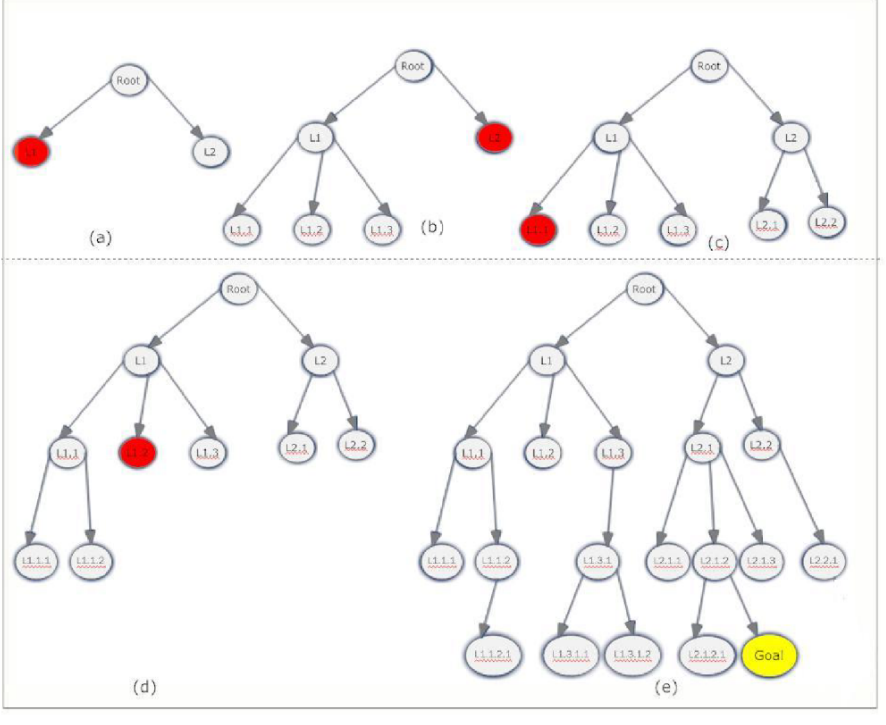

Fig. 2.5 Example of states expansion using breadth first search.

Hence, the conclusion is that the breadth first search algorithm cannot be effectively used unless the search space is quite small. The advantage of the breadth first search is that it finds the path of minimal length to the goal, but it has the disadvantage of requiring the generation and storage of a tree whose size is exponential to the depth of the shallowest goal node.

Example 1: Breadth First Search for 8-puzzle

A simple 8-puzzle example for which the goal state is reached in the third layer of expanded states is presented in Figure 2.6. The goal state is the one in which the blank is on the upper lest corner and the tails are arranged in ascending order.

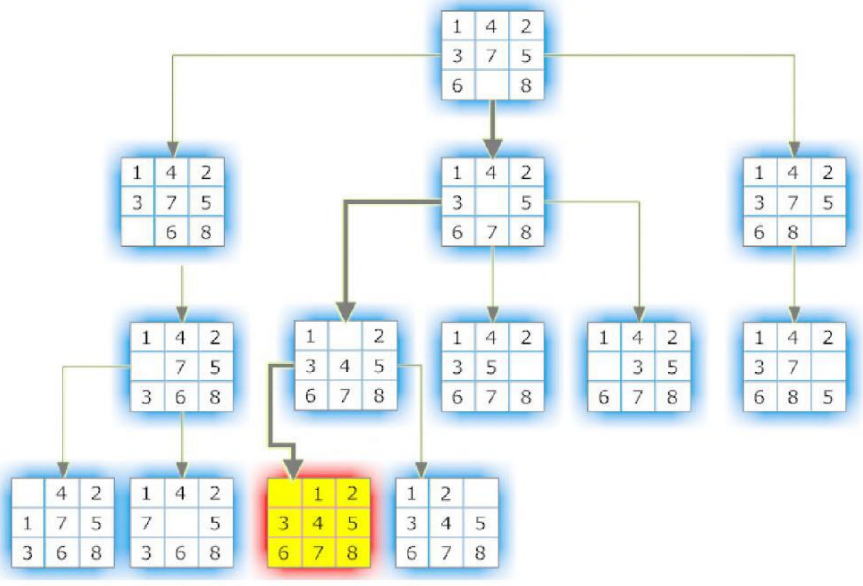

Fig. 2.6 Example of breadth first search for the 8-puzzle problem.

Example 2: Breadth First Search for Missionaries and Cannibals Problem

In order to simplify the explanations, the following notations are used: M for missionaries, C for cannibals and L and R representing the left or right side the boat is in. A graphical illustration of the problem is given in Figure 2.7.

A state can be represented in the following form:

(Left (#M, #C), Boat, Right(#M, #C),

which represents the number of missionaries and cannibals on the left side, the side the boat is, and the number of missionaries and cannibals on the right side respectively. Since the number of missionaries and cannibals should always be 3 on both river banks, we can simplify the notation of the state: (#M, #C, L/R). So,

Fig. 2.7 The missionaries and cannibals problem illustration.

the state represents how many people are on the left side of the river and whether the boat is on the left or right side.

There are five possible actions from any state:

- one missionary moves to the right bank;
- two missionaries move to the right bank;
- one cannibal moves to the right bank;
- two cannibals move to the right bank;
- one cannibal and one missionary move to the right bank.

The two important things to note are that each action results in a boat movement and there are at most five actions. Note that, starting from the initial state, 2 of the 5 actions violate the constraints of the problem (the cannibals outnumber the missionaries as in the case of the first two actions).

The search space for this problem consists on 32 states, which are represented in Figure 2.8. The shadowed states correspond to situations in which the problems constraints are violated.

LEFT BANK				RIGHT BANK			
0M 0C L	1M 0C L	2M 0C L	3M 0C L	0M 0C R	1M 0C R	2M 0C R	3M 0C R
0M 1C L	1M 1C L	2M 1C L	3M 1C L	0M 1C R	1M 1C R	2M 1C R	3M 1C R
0M 2C L	1M 2C L	2M 2C L	3M 2C L	0M 2C R	1M 2C R	2M 2C R	3M 2C R
0M 3C L	1M 3C L	2M 3C L	3M 3C L	0M 3C R	1M 3C R	2M 3C R	3M 3C R

Fig. 2.8 The State-space for the missionaries and cannibals problem.

An Example of a solution for this problem is presented in Figure 2.9. It is evident how the situation changes on both sides and also it may be also used to deduce what the boat will be carrying on both directions.

Solution

Left bank	Boat	Right bank
3M 3C	L	0M 0C
3M 1C	R	0M 2C
3M 2C	L	0M 1C
3M 0C	R	0M 3C
3M 1C	L	0M 2C
1M 1C	R	2M 2C
2M 2C	L	1M 1C
0M 2C	R	3M 1C
0M 3C	L	3M 0C
0M 1C	R	3M 2C
0M 2C	L	3M 1C
0M 0C	R	3M 3C

Fig. 2.9 A solution for the missionaries and cannibals problem.

2.6.2 Depth First Search

The depth first search algorithm is almost identical with the breadth first search algorithm with the main difference in Step 2.2.4 where the children is placed in the beginning of the queue compared to the end of the queue in the case of breadth first search (see Algorithm 2.3).

The queue here may be replaced with a stack. Nodes are popped from the front of the queue and new nodes are pushed to the front. The strategy always chooses to expand one of the nodes that is at the deepest level on the search tree. It only expands nodes on the queue that are at the shallower level if the search has reached a dead-end at the deepest level[14][16][19][35].

A path is expanded as much as possible until it reaches a goal node or can be expanded no more prior to expanding other paths.

Algorithm 2.3. Depth first search

```
Step 1. Form a queue Q and set it to the initial state
(for example, the Root).
Step 2. Until the Q is empty or the goal state is found
do:
            Step 2.1 Determine if the first element in
            the Q is the goal.
            Step 2.2 If it is not
                  Step 2.2.1 Remove the first element
                  in Q.
                  Step 2.2.2 Apply the rule to generate
                  new state(s) (successor states).
                  Step 2.2.3 If the new state is the
                  goal state quit and return this state
                  Step 2.2.4 Otherwise add the new
                  state to the beginning of the queue.
Step 3. If the goal is reached, success; else failure.
```

The difference between the way in which breadth first search and depth first search expansion can be observed by comparing Figures 2.4 and 2.10. The search performed by breadth first search in Figure 2.5 can be compared with the search performed for the same data by depth first search in Figure 2.11.

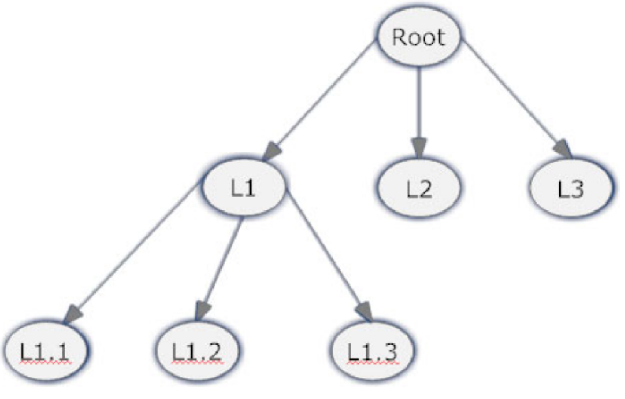

Fig. 2.10 Example of depth first search expansion.

Depth first search algorithm takes exponential time. If d is the maximum depth of a node in the search space, the worst case algorithm's time complexity is $O(b^d)$. However the space taken is linear for the depth of the search tree and is given by $O(bd)$.

The time taken by the algorithm is related to the maximum depth of the search tree. If the search tree has infinite depth, the algorithm may not terminate. This can happen in situations where the search space is infinite or if the search space

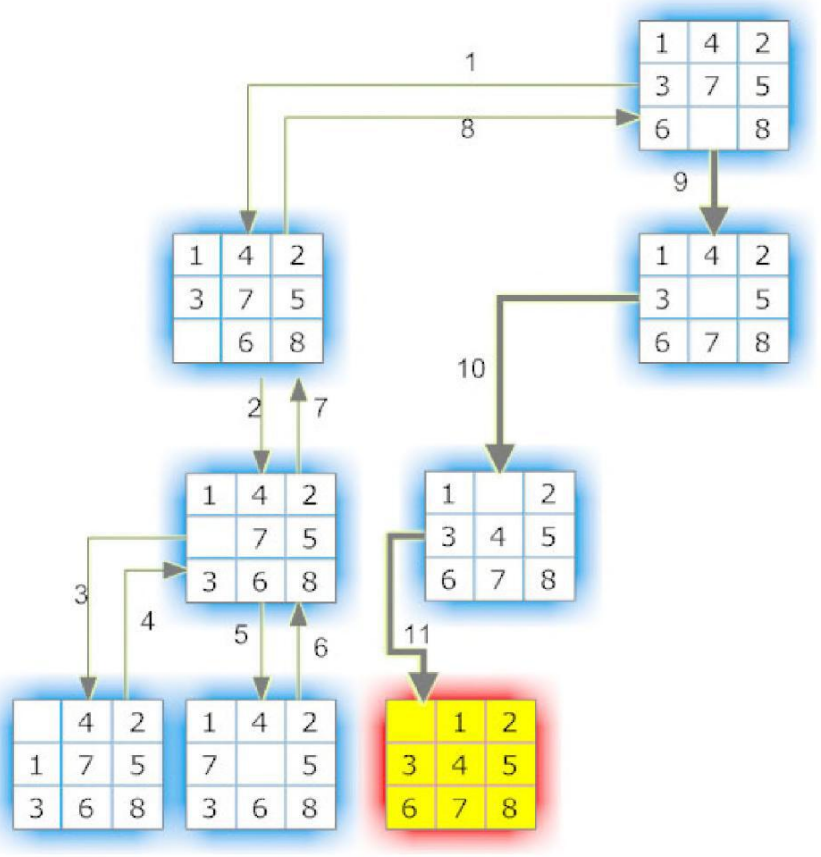

Fig. 2.13 Backtracking search applied for the 8-puzzle problem.

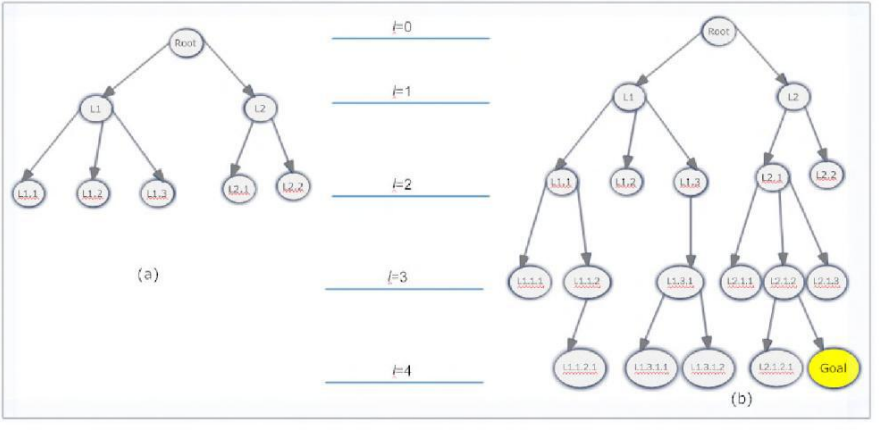

Fig. 2.14 Example of depth bounded search with l=2 – left (a) and l=4 – right (b).

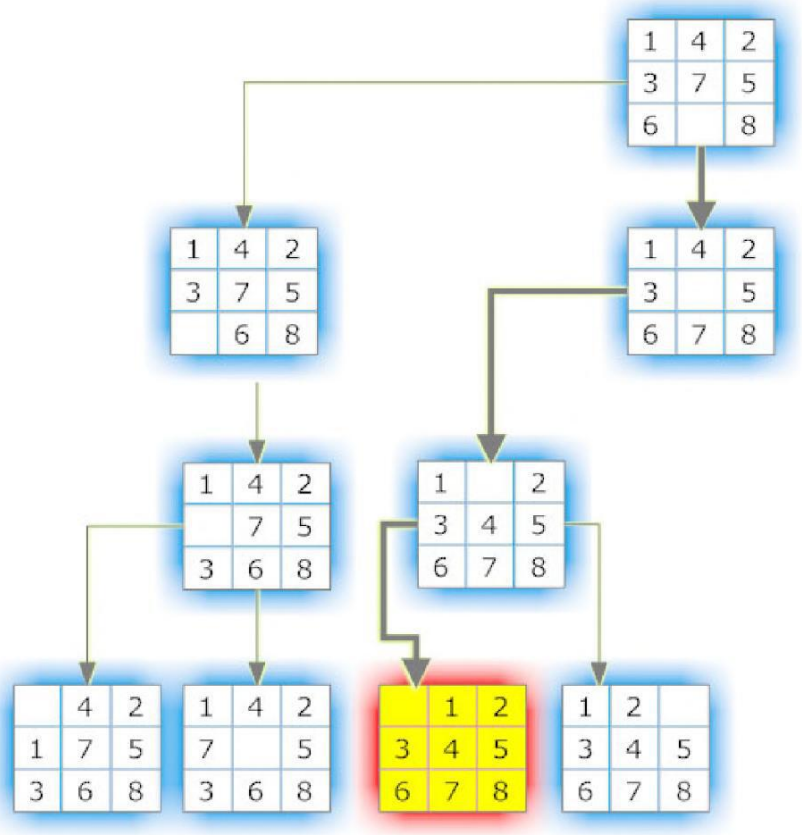

Fig. 2.15 Example of depth limited search for the 8 puzzle problem

2.6.5 Iterative Deepening Depth First Search

Iterative deepening dept bounded depth first search (also referred to as iterative deepening search) consists of repeated depth bounded searches using successive greater depth limits. The Algorithm first attempts a depth bounded search with a depth bound (or limit) 0, then it tries a dept bounded search with a depth limit of 1, then of 2 and so on. Since the search strategy is based on depth bounded search the implementation does not require anything new. The depth bounded searches are repeated until a solution is found[15][28][34][37][38][39].

An example of iterative deepening search with limits from 0 to 3 is depicted in Figures 2.15-2.18.

The iterative deepening search algorithm is simply described in Algorithm 2.4.

Algorithm 2.4. Iterative deepening search

```
Returns a solution or failure;
Input problem;
l = 0
While no solution, do
      Apply depth first search(problem, depth) from in-
      itial state with cutoff l
      If matched the goal
      Then stop and return solution,
      Else increment depth limit l=l+1
End.
```

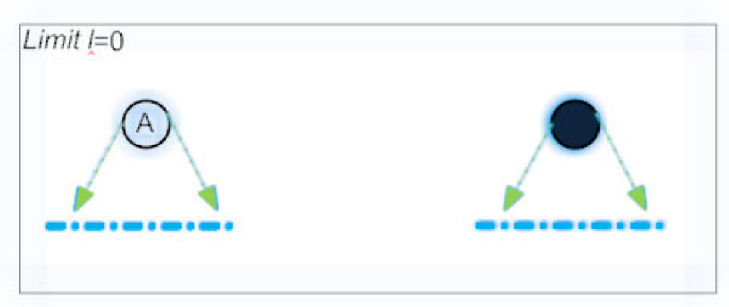

Fig. 2.15 Iterative deepening search for limit *l*=0.

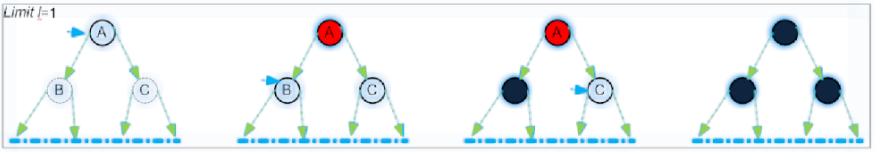

Fig. 2.16 Iterative deepening search for limit *l*=1.

The advantage of the iterative deepening search is that it requires linear memory and it guarantees for goal node of minimal depth. For large depth *d*, the ratio of the number of nodes expanded by iterative deepening search compared to that of depth first search or breadth first search is given by *b*/(*b*-*1*). This implies that for higher values of the branching factor the overhead of repeated expanded states will be smaller. For a branching factor of 10 and deep goals, there will be 11% (10/9) more nodes expanded in iterative deepening search than the breadth first search.

Iterative deepening search combines the advantage of completeness from breadth first search with that of limited space and ability to find longer paths more quickly of the depth first search. This algorithm is generally preferred for large state spaces where the solution depth is unknown. There is a related technique

called *iterative broadening*, which works by first constructing a search tree by expanding only one child per node. This algorithm is useful when there are many goal nodes.

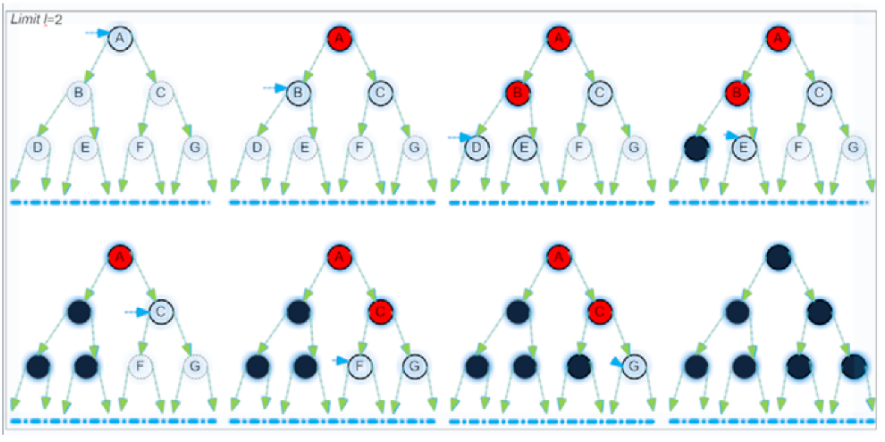

Fig. 2.17 Iterative deepening search for limit l=2.

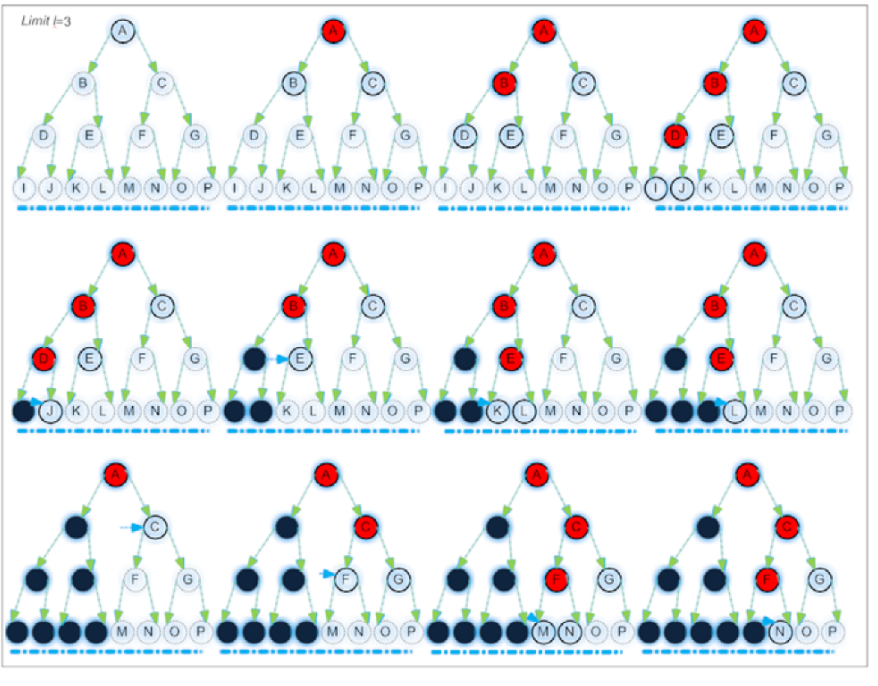

Fig. 2.18 Iterative deepening search for limit l=3.

2.6.6 Branch and Bound (or Uniform Cost Search)

With the branch and bound search, the node with the minimum cost is always expanded. Once a path to the goal is found, it is likely that this path is optimal. In order to guarantee this, it is important to continue generating partial paths until each of them has a cost greater than or equal to the path found to the goal. The branch and bound algorithm is presented in Algorithm 2.5.

Algorithm 2.5. Branch and bound (uniform cost) search

```
Return a solution or failure

Q is a priority queue sorted on the current cost from
the start to the goal
Step 1. Add the initial state (or root) to the queue.
Step 2. Until the goal is reached or the queue is empty
do
      Step 2.1 Remove the first path from the queue;
      Step 2.2. Create new paths by extending the first
      path to all the neighbors of the terminal node.
      Step 2.3. Remove all new paths with loops.
      Step 2.4. Add the remaining new paths, if any, to
      the queue.
      Step 2.5.  Sort the entire queue such as the
least-cost paths are in front.
End
```

Given that every step will cost more than 0, and assuming a finite branching factor, there is a finite number of expansions required before the total path cost is equal to the path cost of the goal state. Hence, the goal is reached within a finite number of steps.

The proof of optimality for the branch and bound search can be done by contradiction. If the solution found is not the optimal one, then there must be a goal state with path cost smaller than the goal state which was found which is actually impossible because branch and bound would have expanded that node first by definition.

Example

Consider the graph given in Figure 2.19 with the initial node S and the goal node G and the cost associated to each edge. Te problem is to find the shortest path (or the path with the lowest cost) from S to G.

The way in which uniform cost search is applied to obtain the optimal solution for this problem is presented in Figure 2.20 and described as follows.

Consider S as the initial state and S is expanded into A and C (Figure 2.20 (a)).

Since the path S – C has the lowest cost until now, C is the next expanded node. C is expanded into B and D.

11. Kosko, B.: Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence. Prentice Hall, Englewood Cliffs (1992)
12. Li, X.H., Chen, C.L.P.: The Equivalence Between Fuzzy Logic Systems and Feedforward Neural Networks. IEEE Transactions on Neural Networks 11(2), 356–365 (2000)
13. Nomura, H., Hayashi, I., Wakami, N.: A Learning Method of Fuzzy Inference Systems by Descent Method. In: Proceedings of the First IEEE International conference on Fuzzy Systems, San Diego, USA, pp. 203–210 (1992)
14. Pedrycz, W., Card, H.C.: Linguistic Interpretation of Self Organizing Maps. In: Proceedings of the IEEE International Conference on Fuzzy Systems, San Diego, pp. 371–378 (1992)
15. Grosan, C., Abraham, A.: Hybrid Evolutionary Algorithms: Methodologies, Architectures and Reviews, Hybrid Evolutionary Algorithms. In: Grosan, C., et al. (eds.) Studies in Computational Intelligence, vol. 75, pp. 1–17. Springer, Germany (2007)
16. Hart, W.E., Krasnogor, N., Smith, J.E. (eds.): Recent Advances in Memetic Algorithms. Studies in Fuzziness and Soft Computing, vol. 166 (2005)
17. Moscato, P.: Memetic algorithms: A short introduction. In: Corne, D., et al. (eds.) New Ideas in Optimisation, pp. 219–234 (1999)
18. Swain, A.K., Morris, A.S.: A novel hybrid evolutionary programming method for function optimization. In: Proceedings of the Congress on Evolutionary Computation (CEC2000), pp. 1369–1376 (2000)
19. Jang, R.: Neuro-Fuzzy Modeling: Architectures, Analyses and Applications, PhD Thesis, University of California, Berkeley (July 1992)
20. Feng, J.C., Teng, L.C.: An Online Self Constructing Neural Fuzzy Inference Network and its Applications. IEEE Transactions on Fuzzy Systems 6(1), 12–32 (1998)
21. Sulzberger, S.M., Tschicholg-Gurman, N.N., Vestli, S.J.: FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks. In: Proceedings of IEEE Conference on Neural Networks, San Francisco, pp. 312–316 (March 1993)
22. Tano, S., Oyama, T., Arnould, T.: Deep combination of Fuzzy Inference and Neural Network in Fuzzy Inference. Fuzzy Sets and Systems 82(2), 151–160 (1996)
23. Kasabov, N., Song, Q.: Dynamic Evolving Fuzzy Neural Networks with 'm-out-of-n' Activation Nodes for On-line Adaptive Systems, Technical Report TR99/04, Department of information science, University of Otago (1999)
24. Mackey, M.C., Glass, L.: Oscillation and Chaos in Physiological Control Systems. Science 197, 287–289 (1977)
25. Nauck, D., Kruse, R.: Neuro-Fuzzy Systems for Function Approximation. In: 4th International Workshop Fuzzy-Neuro Systems (1997)
26. Lin, C.T., Lee, C.S.G.: Neural Network based Fuzzy Logic Control and Decision System. IEEE Transactions on Comput. 40(12), 1320–1336 (1991)
27. Bherenji, H.R., Khedkar, P.: Learning and Tuning Fuzzy Logic Controllers through Reinforcements. IEEE Transactions on Neural Networks (3), 724–740 (1992)
28. Abraham, A.: Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques. In: Mira, J., Prieto, A. (eds.) Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence, Granada, Spain, pp. 269–276. Springer, Germany (2001)
29. Abraham, A.: Optimization of Evolutionary Neural Networks Using Hybrid Learning Algorithms. In: IEEE 2002 Joint International Conference on Neural Networks, World Congress on Computational Intelligence (WCCI 2002), Hawaii, May 12-17 (2002)
30. Abraham, A.: ALEC -An Adaptive Learning Framework for Optimizing Artificial Neural Networks. In: Alexandrov, V.N., et al. (eds.) Computational Science, San Francisco, USA, pp. 171–180. Springer, Germany (2001)