13

Computational Intelligence

E. Aarts, H. ter Horst, J. Korst, and W. Verhaegh

"That's something I could not allow to happen"

HAL in Space Odyssey 2001

13.1 Introduction

Ambient intelligence is aimed at the realization of electronic environments that are sensitive and responsive to the presence of people (Aarts et al. 2002). The word "ambient" in ambient intelligence refers to our physical surrounding and reflects typical systems' requirements such as distribution, ubiquity, and transparency. The word "intelligence" reflects that the surroundings exhibit specific forms of social interaction, i.e., the ability to recognize the people who live and work in it, to grasp context, to learn and adapt to the users' behavior, and to show emotion.

As ambient intelligence is aimed at opening a world of unprecedented experiences, the interaction of people with ambient environments needs to become intelligent. Notions as media at your fingertips, enhanced-media experiences, and ambient atmospheres refer to novel and inspiring concepts that are aimed at realizing specific user needs and benefits such as *personal expression, social presence*, and *well-being*. These benefits seem quite obvious from a human perspective, but are quite hard to realize because of their intrinsic complexity and ambiguity. Obviously, the intelligence experienced from the interaction with ambient intelligent environments will be determined to a large extent by the computational intelligence exhibited by the computing platforms embedded in the environment, and consequently, by the algorithms that are executed by the platforms (Verhaegh et al. 2004).

The algorithmic techniques and methods that apply to *design for intelligence* in ambient intelligent environments are combined in the scientific and technological pursuit known as *computational intelligence*, which is aimed at designing and analyzing algorithms that upon execution give electronic systems intelligent behavior. For an introduction to the basic concepts in this field we refer to Engelbrecht (2002).

In this chapter we address the subject of computational intelligence in relation with ambient intelligence. The chapter is organized as follows. Since

computational intelligence is rooted in the classical field of machine intelligence, we start our exposition with a brief outline of the developments in that domain. Next, we explain what the major behavioral characteristics are that need to be addressed and implemented by intelligent algorithms. The main part of the chapter is devoted to a discussion of a number of computational paradigms that serve as a basis for the realization of the intelligent and social user interaction within ambient intelligence. Next we discuss some elements related to the computational complexity related to these computational paradigms. We conclude the chapter with a short discussion of some challenges related to the future development of computational intelligence within ambient intelligence and as a preliminary conclusion we argue that new computing methods are needed to bridge the gap between the class of existing algorithms and the ones that are needed to realize ambient intelligence.

13.2 Machine Intelligence

According to Merriam-Webster's Collegiate Dictionary, "intelligence" refers to "the ability to learn or understand or to deal with new trying situations." Other definitions are the ability to reason, apply knowledge, manipulate one's environment, or think abstractly. Over the years mankind has proved indefatigable in its attempts to build electromechanical machinery that exhibits some intelligent behavior. "Counting" is clearly one of the most important of these intelligent activities that inspired engineers and scientist to functionally incorporate into automatic machinery. In this respect, the calculating device constructed by the famous mathematician Blaise Pascal in 1642 is widely recognized as the first "digital" computing device, where digital refers to making use of a finite set of internal states. Also the automatic table calculator called the *Differencing Machine*, which was constructed by the eccentric but brilliant mathematician Charles Babbage in 1822, can be seen as a landmark development in machine intelligence.

13.2.1 Artificial Intelligence

Since the invention of the computer in the mid-1940s machine intelligence has become a significant scientific sub-domain of computer science, which is often denoted as artificial intelligence. According to Minsky (1986), artificial intelligence is the science of making machines to do things that require intelligence if done by man. Classical subjects of investigation are vision, natural language and speech processing, robotics, knowledge representation and reasoning, problem solving, machine learning, expert systems, man–machine interaction, and artificial life (Rich and Knight 1991; Winston 1992; Nilsson 1998; Brooks 2002; Russell and Norvig 2003). McCorduck (1979) provides a good account of the early developments in artificial intelligence including some of the controversies that arose among scientists about the limitations of computers in comparison with human beings.

Over the years the discussions about artificial intelligence and the question whether mankind would be able eventually to build intelligent machines have been plentiful. Alan Turing, an early pioneer in computing science, developed the so-called Turing test to discriminate between computation and mind (Turing 1950). The Turing test entails that a keen interrogator questions a machine and a human volunteer at the same time. The interrogator cannot see either of the two, and he can pose the questions to both of them by making use of a keyboard only. The machine is said to pass the test if the interrogator cannot tell the difference between the machine and the volunteer from the answers given by the two.

Turing's proposition became controversial and was questioned by scientists working in the field of natural languages and learning psychology. The philosopher John Searle (1980) criticized the Turing test with his well-known Chinese Room argument, which demonstrated that a machine could reply to the posed questions in a way that was indistinguishable from the volunteer, but that it developed no other attributes that are generally considered as expressions of human intelligence such as understanding and self-consciousness. This controversy gave rise to what is known as the *mind–body problem*, which addresses the question whether a perfectly reconstructed body would have a mind by itself.

Hawkins and Blakeslee (2004) go in their recent book beyond the classical approaches to machine intelligence and develop a general theory of the human brain. They argue that the brain is not a computer, but a memory system that stores experiences reflecting the structure of the world. The memory system can generate predictions based on the nested relations among the memories, thus giving rise to intelligence, perception, creativity, and consciousness. Furthermore, the authors argue how eventually intelligent machines can be built based on their theory of intelligence. Although Hawkins and Blakeslee provide new and inspiring ideas, their theory cannot provide solutions to some of the open problems related to intelligence in the human brain. For instance, the above mentioned mind–body problem remains unresolved to a large extent.

13.2.2 Movie Script Scenarios for Ambient Intelligence

One of the salient features of ambient intelligence is the massive integration of intelligent features into the electronic background of our environments. Clearly, these features should enhance the interaction of users with their environments, thus enabling a means for nonobtrusive and social interaction that may increase productivity and creativity. Evidently, these statements are just abstract wordings and to make them more concrete one often resorts to the description of use-cases or scenarios. Examples can be found in the ISTAG report published in 2002, which provides four scenarios that cover different aspects in the daily life of ordinary people (ISTAG 2001). A good scenario should position the different use-cases within a realistic context that is kept consistent and that is maintained all over the story covered by the scenario.

The ISTAG scenarios, for instance, use different persons for different settings: Maria in *Road Warrior*, Dimitrios in *Digital Me*, Carmen in *Trafic, Sustainability, and Commerce*, and Anette and Solomon in *Social Learning.*

As another approach to the issue of scenario building one may resort to the many movie scripts that have been developed in the science fiction movie genre. A classical example is *2001: A Space Odyssey* (1968, MGM) directed by Stanley Kubrick. In this movie script Kubrick features an intelligent computer named HAL that serves three cosmonauts in a journey through outerspace. During the journey HAL develops certain elements of cognitive and social intelligence, such as the ability to have natural conversation, to plan complex tasks, and to show emotion. Kubrick gives with HAL a convincing and realistic rendition of an intelligent machine that is capable of developing an affective relation with its user. More recently, Stork (1997) edited a collection of book chapters in which renowned specialists cover specific aspects of HAL's behavior in relation with machine intelligence, including supercomputer design, reliable computing and fault tolerance, gaming, speech processing, vision, man–machine interaction, planning, affective computing, and computer ethics. From Stork's collection of contributed chapters in machine intelligence, one may conclude that HAL could have been built by 2001 and that it would pass the Turing test. Remarkably, Turing himself predicted that it would be possible to build a machine that would pass the Turing test somewhere between 2000 and 2010.

2001: A Space Odyssey describes a scenario of a computer that interacts with users in a natural and intuitive way, but HAL is still a computer, a machine that acts as a stand-alone object. In ambient intelligence the aim is on integration of such intelligent functionality into distributed environments that surround human beings. To obtain scenarios that describe how this could become effective one again may resort to the wonderful world of movie scripts. A compelling definition of ambient intelligence is visualized in *Mathilda* (1996, Tristar Pictures after a book with the same title by Roald Dahl) directed by Danny DeVito in a scene where the little girl Mathilda discovers her ability to control objects in her physical environment by speech and gesture. The scene also illustrates profoundly the ultimate aim of ambient intelligence to put the user in the center of the ambient environment and to give him full control of it.

In *Total Recall* (1990, Tristar Pictures), Arnold Schwarzenegger spends a virtual vacation on Mars, but afterwards events force him to go there for real. In one of the many science fiction scenes Schwarzenegger communicates with an interactive display wall that can switch to a peace giving scenery of the Colorado mountains, which helps him unwind after he has received bad news from Mars.

In *Minority Report* (2002, 20th Century Fox), directed by Steven Spielberg, Tom Cruise is positioned in a digital world in which he is surrounded by interactive screens and displays everywhere, allowing direct interaction with the environment. The concept of a ubiquitous digital smart interactive environment is consistently maintained throughout the movie. Newspapers

have become interactive display foils that can play real-time video messages. Billboards have become public displays that provide urban annotation and support personal information access in the public domain. Even packaging material consists of interactive displays showing video commercials, and they can be controlled in a natural way, for instance by smashing it against the wall when it should stop advertising. Television viewing has become truly three-dimensional through the use of holographic display technologies that enable the viewer to actively participate in the scenes that are displayed.

The Harry Potter movies after the books by Joan Rowling provide other interesting examples of ambient intelligence scenarios in movie scripts. Especially, the third movie called *The Prisoner of Azkeban* (2004, Warner Bros. Pictures), directed by Alfonso Cuarón, depicts a world imbued with ambient intelligence features. The castle Hogwarts, school of witchcrafts, is flooded with pictures that "contain" living creatures and can act as interactive displays. They can even control access to rooms through the use of spoken passwords. The ceiling of the dining room is a huge display that can change the ambience of the room to match the occasion or the time of the day or season. Objects are smart and can be used to control the environment. They also can be used to access ambient information. For example, the *Rememberal* is an object that indicates through its color when the person who holds it has forgotten something. The ultimate example perhaps is the *Marauders Map*, which is an interactive two-dimensional map that reveals the geographical position of any person that is within range of the area displayed by the map.

All these scenes show examples of scenarios of distributed intelligent environments that support the people who interact with them, enabling them to perform daily tasks, and to become more productive, creative, and expressive. So, the notion of a computer as a physical device has disappeared. The computer has been integrated into the environment moving it to the background and bringing functionality to the foreground, leaving the user in total control. Another interesting observation is that technology is not going to make the difference in the end, but that the way technology is being applied and used will be the crucial factor. For instance, in both movies *Minority Report* and *The Prisoner of Azkeban* a world is depicted in which people are surrounded with interactive displays. Yet, the world in *Minority Report* is much more threatening than that in *The Prisoner of Azkeban* where the use of ubiquitous displays adds to the well-being of its inhabitants rather than providing a feeling of "big brother is watching you" as in *Minority Report.*

13.2.3 Social Versus Cognitive Intelligence

An important conclusion that can be drawn from the discussion presented in the previous sections is that intelligence in ambient intelligence is aimed at social intelligence rather than at cognitive intelligence. From the viewpoint of ambient intelligence, electronic doors that open automatically if a person is

approaching them are more socially intelligent than Deep Blue, IBM's powerful chess computer that beats Gary Kasparov, the world champion of chess, in a direct confrontation in May 1997. This means that ambient intelligence provides a new kind of challenge for artificial intelligence. In ambient intelligence the level of intelligence of an electronic system is judged by the way users perceive it from a social interaction viewpoint. Evidently, the types of tasks, as well as the context in which they are carried out, play an important role in the user perception.

Csikszentmihalyi (1990) has introduced the concept of *flow* to discuss the observation of intelligent behavior of human beings when performing tasks. Flow is an experience concept in psychology that is universally perceived in all cultures and ways of living. Flow refers to the feeling people experience when in contact with the world around them, providing a sense of reason and purpose of their activities. Flow can be very helpful in the description of social intelligence in everyday life. Dunne and Raby (2001) present in their compelling book a number of interesting design studies in which they place functionally modified electronic daily life objects, such as their *parasitic light, GSM table*, and *compass table*, in the homes of people and describe how they develop patterns of social interaction with these objects over time. It is astonishing to see how people develop affective relationships with these objects that are meaningless at first glance.

Reeves and Nass (1996) present another interesting study on social intelligence in the interaction of people with electronic equipment. They introduce the so-called *Media Equation*, which states that people should be able to interact with media in the same way as people interact with each other. This is an easy statement to express but hard to accomplish. Nevertheless, it directly implies that natural human interaction paradigms such as multi-modality, personalization, expression, and emotion should be key elements in social machine intelligence, and this implication has become generally accepted over the years.

Picard (1997) addresses the compelling issue of emotions in computing in a more scientific way by developing a theory on affectiveness in man–machine interaction. She not only discusses how computers might develop emotions but also why they must do so, and this may become quite useful in the design of ambient intelligent systems.

13.3 AmI Elements of Social Intelligence

From the discussion presented above one might conclude that the development of ambient intelligence is still in a conceptual phase and that we have to resort to a model of applications or scenarios to illustrate what we mean by the concept when we want to make it more concrete. Evidently, scenarios as presented above can be very helpful in the discussion on the realization of

ambient intelligence, but the proof of the concept evidently is in the realization of commercial products and services that are attractive to people. Over the years, a large variety of successful and less successful attempts have been made to develop such commercial products and services, and looking at the market there is already a lot that is available. Below, we present some of the most profound examples for the purpose of our discussion on machine intelligence and the corresponding computational methods.

To put the commercial products and services into perspective we distinguish between the following five classes of AmI (ambient intelligence) elements in social intelligence:

- See, hear, feel
- Understand, interpret, relate
- Look, find, remember
- Act, learn, adapt
- Create, express, emerge

This class division is rather arbitrary, and there is no theory that supports it. However, it combines the main social activities that human beings undertake in their daily lives into categories of well-defined and interrelated tasks that relate to the concept of flow as defined by Csikzentmihalyi (1990). Moreover, it provides a simple framework for the discussion of computational intelligence paradigms in ambient intelligence. Below, we present for each of these classes a number of examples of prototypes or commercially available products and services. Also this selection is chosen rather arbitrarily, but it may serve the purpose of providing an impression of what is already on the market, thus indicating where we stand in the development of ambient intelligence. Finally, we briefly sketch for each of the classes some of the basic computational paradigms that can be used to realize the socially intelligent interaction defined by the class. In the next section, we then discuss some of these paradigms in more detail with the purpose of presenting some of the mathematical models and computational details that are applied.

13.3.1 See, Hear, Feel

This class of AmI elements is concerned with the general aspects of visual, auditory, and other sensorial information processing. It refers to actions such as viewing screens, listening to spoken text or music, and sensing location and context. Examples of products and services in this domain are given in Table 13.1.

The computational paradigms that have been established in this domain are based on vision, speech processing, and sensor data fusion, which all heavily rely on signal processing techniques, often applied in the digital domain after an analog-to-digital conversion of the analog data. Most of the approaches follow three steps. Firstly, the environment is sensed through the

Table 13.1. See, hear, feel

OpticalSensors' "Talking Cane": warns blind or visually impaired persons through spoken language for obstacles in the vicinity.
Vivometrics's "LifeShirt": a lightweight, washable, sleeveless vest embedded with sensors for continuously collecting and classifying cardiopulmonary patient data.
Vos Systems's "IntelaVoice Voice Operated Dimmer": allows to dim your lamps with simple voice commands.
Logitech's "QuickCam Orbit" Desktop Video-Camera: automatically follows you when you move around.
EyeOn Trust's "Golfmate": allows to locate easily and immediately a golf ball on a golf course.
Singapore Technologies Electronics' Fever Screening System: shows whether passers-by are running a temperature.
Mitsubishi's ITS-ASV2 Car: provides audible and visual warnings when it detects that the driver is not alert enough.
Mercedes Benz' "Talking Alarm Kit": monitors the driving lane and alerts if pedestrians appear in front of the car.

collection of raw data from sensor networks, consisting of cameras, microphones, position detectors, motion trackers, chemical sensors, and others. Secondly, the data are processed and combined, and thirdly the data are classified.

Well-known techniques that are used in this domain are face and body recognition, geometric modeling of two- and three-dimensional objects, image segmentation, biometrical data processing, speech recognition, and synthesis. For the classification, one often resorts to *artificial neural networks* or *hidden Markov models*. The artificial neural networks are used as generalized class separators. They can deal with conflicting or incomplete data sets. The hidden Markov models typically use extremely large probabilistic networks of nodes that represent partial solutions. Final solutions are then obtained by computing the most likely feasible combination of partial solutions. This computation is often carried out using a mathematical programming approach called dynamic programming. This approach has regained much attention over the past years as a result of the huge increase in the capacity of present-day computing devices and the availability of large data sets.

13.3.2 Understand, Interpret, Relate

This class of AmI elements is concerned with aspects of giving meaning to certain events or activities that are sensed or recorded. In general terms this is the domain of reasoning, induction, and deduction. Again, we start the presentation with a short overview of some commercially available products and services as listed in Table 13.2.

Table 13.2. Understand, interpret, relate

Blissful Babies' "Why Cry": a calculator-sized battery-powered device that can analyze a baby's cry and give an indication of the cause.
Wow Wee Toys Ltd's "Speak2Click": a conversation robot that allows you to easily communicate with your PC.
Philips' ICat: user-interface robot that supports natural conversation and can be used for control and support tasks in the home.
Elite Care's "Smart House": assists occupants with diminished mental capability, by detecting their physical motions, movement of objects, and operation of appliances and lights.
K Laboratory's "FeliPo" Service: allows a mobile phone to detect information on a poster about advertisements that are customized for gender, location, time/date, and the weather conditions.
NewNow InetShop's "Sensor-Bin": a dustbin that will open its lid when you move directly in front of it.
Saab's "Alcokey": prevents a person from driving the car if not sober.

Most of the computational techniques that are applied in this domain are known as rule-based or expert systems. Upon input these systems argue and reason using data and relations among data. They typically apply heuristic rules to combine and transform data in order to reach valid conclusions or develop new pieces of information.

Expert systems constitute a classical field of research in machine intelligence, and substantial progress has been made over the years. Generally speaking, there are two elements that can be found back in most of the existing expert systems. Firstly, there is the *knowledge representation*, which determines not only the data structures that are used to specify the data items of the information that needs to be processed, but it also captures the relations among the data items in a data model that allows to manipulate the data items. Secondly, there is the element of *reasoning*, which refers to manipulating and transforming data items in order to reach conclusions that make sense within the context of use. The reasoning may also lead to a sequence of actions that need to be taken in order to accomplish a given goal and which is often referred to as a plan.

More recently, probabilistic methods have been introduced to reason about information. These methods apply the working hypothesis that different possible outcomes should be evaluated simultaneously to select eventually the most likely one based on the probability that certain initial conditions are met. *Bayesian methods* are a class of computational techniques that apply this approach with considerable success.

13.3.3 Look, Find, Remember

This is the classical domain of search and retrieval of information, browsing databases, and remembering specific pieces of information or events. This

class of AmI elements has become particularly interesting as a result of the ubiquitous availability of information throughout society. Having unlimited access to any kind of information is not a privilege if one is not supported by intelligent means to handle the information overload. So, this domain is central to information handling in ambient intelligence. Table 13.3 gives some examples of commercially available products and services in this domain.

There are many search paradigms available in this domain. A major class of approaches uses heuristic rules that construct a solution to the search problem. Other techniques use iterative methods that continually try to improve a given solution using certain criteria. *Local search* is a well-known example of a search paradigm that applies this approach. *Constraint satisfaction* is yet another method that uses a tree-search approach to find a solution to a search problem that satisfies a well-defined set of constraints. Associative memories based on artificial neural network models are used to complete partial information in retrieval problems or to find approximate matches. Other methods that allow for data mining with uncertainty apply approximate pattern matching techniques, which are quite similar to the computational approaches used in stochastic Markov models. More recently, one has developed the so-called Bayesian classifiers, which are statistical methods applying conditional and prior probability distributions to evaluate classification hypotheses.

Evidently, we need to mention many search techniques that have been developed to browse the Internet. Most of these methods apply heuristic search and pattern matching rules, some of which are based on graph theoretical models. The recent introduction of the Semantic Web (Berners-Lee et al. 2001), which enables not only the search for data items, but can also account for relations between data items, has largely stimulated the development of intentional search techniques.

Table 13.3. Look, find, remember

Siemens "Digital Graffiti": appears as an SMS message when a user with a phone walks by a given physical location.
Philips' "Easy Access": allows a user to find back a song in a music database by humming a tune of the song.
Hutchison's "3FriendFinder": allows location of other users of this service in a map on the mobile phone's display.
GPSTracks' "GlobalPetFinder": allows building a fence of any size, and will alert you if your pet wanders outside it.
Cirrus Healthcare Products' "Angel Alert": tracks children and warns when they stray too far from adult supervision.
Shazam Mobile-Phone Service: identifies a piece of music with a mobile phone.
SmartHome's "Memory Key": shows the status of the door (locked or unlocked) and how much time has passed since you locked it with that key.

13.3.4 Act, Adapt, Learn

This class deals with characteristics of personalization and adaptation over time, which can be obtained from the processing of time sequences of data that contain behavioral patterns. Table 13.4 gives some examples of products that exhibit certain of these characteristics.

Most computational methods in this domain apply some form of *machine learning* techniques. Many of these techniques build an internal model that tries to capture the input–output relation of a set of learning examples, which is often referred to as supervised learning. If such a set of learning examples is not available, similar techniques can be applied to the relation between observed data items to group them into classes, which can be used later to classify new data items. These implicit models can also be used to interpolate or extrapolate from existing data upon input of new data items. Examples of machine learning techniques that are frequently applied are stochastic learning models, such as *reinforcement learning* and *artificial neural networks*. In learning, the artificial neural networks are used as generalized function classifiers where the learning examples can be viewed as the data items in a multidimensional learning space. The computational models mentioned above can also be used to build models of user behavior, and they can be determined for different types of contexts, and combined within the same actor model. An example of such an approach can be found in the development of recommender systems that support in the selection of video items. Here user preferences are captured in multiple models that reflect different contexts of use.

Table 13.4. Act, adapt, learn

Noxa Med's Anti-Snore Pillow: eliminates snoring of a specific person by producing vibrations if it detects the onset of snoring.
Adidas' Intelligent Shoes: adapt automatically their characteristics to the surface on which the person who is wearing them runs.
OSIM's "iMedic 500 Advanced Massage Chair": measures automatically the length of a person's back and the shoulder position, to determine where it should apply pressure.
Toyota's "Prius" Car-Parking Assistance: allows a car to park itself without the driver having to touch the steering wheel.
Emfitech Oy's "SafeSeat" and "Safefloor": chairs and floors that can monitor the health and well-being of patients, and send warnings if needed.
Omron's "OKAO Vision": a user interface for an ATM or a ticketing machine that automatically adapts after observing the user's gender, age, and other attributes.
Tomy's Sleep Watch Doll: observes the user's sleeping habits, and bothers the owner when he or she is not going to bed or getting up at the usual time.

13.3.5 Create, Express, Emerge

This class of AmI elements is probably the most interesting of all since it deals with creativity in the most absolute form, i.e., creating something meaningful that is new. It has to do with new elements of forms and shapes that enhance the interaction of people with media in the broadest sense. Here, affectiveness and emotion come into play, and in the end, this class of AmI elements may drive very well the most challenging innovations introduced by the concept of ambient intelligence. First, let us have a look at some of the examples, shown in Table 13.5.

The domain "create, express, emerge" deals with revolutionary concepts and is aimed at creating an enhanced experience for its users. One speaks in this respect of poetic interfaces, intimate media, and affective computing, which are all different concepts but with the very same purpose of creating immersive sensorial experiences resulting from revolutionary shifts in interaction paradigms. Examples are the automatic generation of multimedia narratives or the creation of ambiences through automatic generation of lighting and sound effects. Also the stimulation of sensorial arousals through physical motion and vibration is part of the enhanced experience.

There are only a few general computational methods in this domain. Most approaches applied in the examples presented in the table use heuristic rules to create the experience, and there are no general guidelines on how to apply these rules successfully. *Evolutionary computing* techniques however constitute a class of generally applicable methods that may become of great

Table 13.5. Create, express, emerge

Philips' Ambilight: creates a halo around the television with dynamically changing colored light that enhances your viewing experience.
Digital Fashion's 3D Simulation System for Virtual Modeling: allows virtual modeling and coordination of clothes, cosmetics, and accessories in real time.
Citroen's "C-Airlounge" Concept Car: is equipped with projectors in the armrests and floor that create particular light effects and moods.
Andersen Windows' "Concept House": Has windows that can also function as Loudspeakers and displays.
Jabberwock's Chat Software: simulates an automated chat person on the Internet.
Sharp's LN-H1W "Lumiwall" Window Panel: provides light during night and day, by combining daylight transmission, solar power generation, and illumination.
D-BOX' "Odyssey": brings a new dimension to entertainment by moving the viewer's seat in synchronism with the action on screen.
Lofty's "Hotaru" Smart Pillow: helps the user fall asleep easily and comfortably by emitting light that varies in response to the breathing pattern of the user.

significance in this domain, because of their natural approach to the problem of creating new meaningful results from existing ones. These methods follow the basic laws of Darwinian evolution, which create new elements of a population through mutation and recombination of existing elements within the population. Fitness rules are applied to select the best ones and in this way, surprising new results can be generated.

Another approach is that of using agent technology. *Intelligent agents* are small software programs that act autonomously within a distributed environment. They can create new agents and modify existing ones based on the use of external information. They are responsive to their environments and can contribute to the purpose of evolution.

13.4 Computational Paradigms

In this Section, we address four paradigms for computational intelligence: *Search*,*Reasoning*, *Learning*, and *Evolution*. For each of the paradigms we present a number of basic algorithmic approaches that can be applied in the design of AmI applications.

13.4.1 Search

Search algorithms are applied in situations where a solution has to be found in a large set of alternatives, subject to a number of constraints, and often with an objective that needs to be minimized or maximized (Papadimitriou and Steiglitz 1982). For instance, speech recognition requires finding a text for a given utterance, such that the probability that the utterance corresponds to the text is maximized. Music playlist generation is about finding a sequence of songs that meets the constraints set by the user on, e.g., songs and transitions.

Dynamic Programming

The first method we elaborate on is called dynamic programming, and is used, for example, in approximate pattern matching, in solving hidden Markov models, and in planning problems.

The key idea of dynamic programming is that decisions are taken one by one, and that optimal sequences of decisions are built in a "recursive" way. More precisely, the problem at hand is modeled as that of searching an optimal sequence of decisions $d_1, d_2, \ldots, d_n$. For instance, in a planning problem where a choice has to be made which of a given list of items to include, the ith decision could be whether or not to include the ith item. By successively taking the decisions $d_1, d_2, \ldots, d_n$, a complete solution to the problem is constructed. In this planning example, the sequence of decisions determines the set of selected items.

For determining an optimal sequence of decisions $d_1, d_2, \ldots, d_n$, the *principle of optimality of successive decisions* is used. This principle states that an optimal sequence $d_1, d_2, \ldots, d_n$ consists of a first decision, d_1, followed by an optimal sequence of decisions $d_2, \ldots, d_n$ for the remainder of the problem. The idea is now that for every possible situation after the first decision, first an optimal sequence of decisions $d_2, \ldots, d_n$ is computed. Then, the first decision d_1 is chosen such that the combination of its direct effect and the effect of the optimal finish $d_2, \ldots, d_n$ is optimized. Note that in this way we have reduced the problem from n degrees of freedom to $n-1$ degrees of freedom.

The above step can be applied repeatedly, leading to the following dynamic programming algorithm. First, the optimal decision d_n is determined for every possible situation after the first $n-1$ decisions. Next, for every possible situation after the first $n-2$ decisions, the optimal decision d_{n-1} is determined, given the optimal finishing decision d_n for the situation after choosing d_{n-1}. This is repeated until we have finally determined the optimal decision d_1.

Key in the above recursive procedure is the way "every possible situation after k decisions" is represented. In dynamic programming, this is modeled by a set of *states*, where a state describes the effect of the previous decisions in such a way that the result of the remaining decisions can be determined. In the worst case, the set of states after k decisions is exponentially large in k. For many problems, however, the set may be significantly smaller. For instance, in the planning example, the state may be given by the total duration of the already selected items, assuming that the problem is about selecting items with a total duration not exceeding some limit D. Knowing the total duration of the previously selected items is all one needs to know about the previous decisions in order to determine the remaining decisions optimally. If all items have a duration, i.e., a multiple of 5 min, then the set of states is given by all multiples of 5 min between 0 and D, which is independent of k.

The final element in dynamic programming is a recurrence relation to determine the combined effect of a decision d_i and the optimal finish $d_{i+1}, \ldots, d_n$ after it. Let for each possible state t after decision d_i the value of the optimal finish be given by $f_{i+1}(t)$. Then, for each possible decision d_i taken in each possible state s, the recurrence relation expresses the value $v(s, d_i)$ of taking that decision in that state as function of the direct effect of decision d_i and the value $f_{i+1}(t)$ of the state t that is reached after this step. Then, the optimal decision in state s is the one that maximizes $v(s, d_i)$, yielding the optimal value $f_i(s)$ for state s. In the planning example, the recurrence relation may be that the value of taking a decision d_i in state s is the value of the ith item if it is selected and zero otherwise (i.e., the direct effect), plus the optimal value of future selections given the resulting state t, where the resulting state t is given by s plus the duration of the ith item if it is selected, and s otherwise. More formally,

$$v(s, d_i) = \begin{cases} v_i + f_{i+1}(s + l_i) & \text{if } d_i = \text{"select item } i\text{" and } s + l_i \leq D, \\ f_{i+1}(s) & \text{otherwise,} \end{cases}$$

where v_i is the value of item i, and l_i its duration.

As we can see above, optimal sequences of decisions are calculated "backwards," i.e., first optimal last decisions are computed, then optimal one-but-last decisions are computed, etc. Dynamic programming variants exist in which the decisions are calculated "forwards." In contrast to the above approach, where one typically uses the same state set for each decision, a forward approach typically only maintains the set of states that are actually reached after the first k decisions. If two decision sequences $d_1, d_2, \ldots, d_k$ end up in the same state, the one with the best value is kept, and the other one is discarded.

A final remark that we make about dynamic programming is that, whereas the above presented algorithm gives a guaranteed optimal solution, it may be adapted to save computation time, at the cost of losing the guarantee of optimality. Many examples are known in the literature to turn a dynamic programming algorithm into an approximation algorithm, by pruning the state set. For instance, in the planning example, one may round the states to multiples of 15 min, even though the items' durations are multiples of 5 min. As this reduces the state set, it hence reduces the computation time. Other known approaches in, e.g., speech recognition is a so-called beam search, where the state set per decision is restricted by discarding states that are not likely to give an optimal solution.

Heuristic Algorithms

Many problems cannot be solved exactly in a reasonable time, for instance because of their intrinsic complexity. In this situation one may resort to the use of heuristics, which drop the requirement of finding an exact solution at the benefit of substantially shorter running times (Osman and Kelly 1996). In this section, we discuss two such approaches, a constructive one and an iterative one.

Constraint satisfaction. The first approach we mention, which is a common method for feasibility problems, is given by constraint satisfaction (Tsang 1993). Constraint satisfaction can be seen as a constructive approach in the sense that it gradually builds up a solution.

Key in constraint satisfaction is the definition of *domains*, which give for each of the involved decision variables a set of values from which it can be chosen. These domains, which initially may be quite large, are reduced during the course of the algorithm, until they each contain only one value, and hence a solution has been found, or until one of them becomes empty, meaning that no solution exists.

Domain reduction techniques in constraint satisfaction build upon combining constraints with other variables' domains, as well as on combining several constraints. This process is called *constraint propagation*. For instance, if a decision variable x has domain [2,10] and variable y has domain [5,7], and there is a constraint implying that x has to be at least equal to y, then the domain of x can be reduced to [5,10], as other values cannot lead to a feasible solution. An example of combining constraints is that, in a planning problem,

if a task a is supposed to be executed on the same processor as another task b, and there is not enough time for a to execute before b has to be started (due to the domains), then one can draw the conclusion that a has to start after b has finished. The strength of constraint satisfaction is determined by the domain reduction power of combining constraints, and quite some research has been spent on the types of constraints that lend themselves best for constraint propagation.

If, at a certain moment, domains cannot further be reduced, while some domains contain multiple values, then a variable is chosen, and it is assigned a value from its remaining domain. Then again constraint propagation is applied as much as possible, to reduce the other variables' domains. If necessary, a new variable is assigned a value, etc. Obviously, the order in which the variables are considered for being assigned a value has a strong impact on the end result. Furthermore, value choices may turn out to lead to a situation where some domains reduce to empty sets. In that case, one may apply *backtracking* to undo (some of) the last decision(s) and to make other choices. Doing so, however, may increase the running time considerably, as one effectively is solving the problem exactly.

The field of constraint satisfaction can be positioned at the intersection of mathematical programming and artificial intelligence, and the fact that it contributed to the merger of these two fields is probably one of its major contributions in addition to the fact that it is quite a powerful method that can be applied to a large range of problems.

Local search. The second approach we mention is called local search (Aarts and Lenstra 1997), which works by starting with a rather arbitrary, yet complete solution, and iteratively making small changes to it. By defining the kind of alterations that can be made to a solution, e.g., replacing a song in a music playlist by another song, a so-called *neighborhood structure* is defined, which, for each solution, gives a set of solutions that can be obtained from it in one iteration. So, iteratively, a random solution is picked from the neighborhood of the current solution, and the objective function is evaluated for this new solution. If the effect is favorable, the new solution is directly accepted, and used for the next iteration. If the solution deteriorates, then in the simplest form of local search, called *iterative improvement*, the new solution is rejected. Doing so, local search may be trapped in a *local optimum*, meaning that no neighbor of the current solution improves the objective function. However, this local optimum may not be the overall (global) optimum.

To escape from a local optimum, *simulated annealing* uses a different way of treating deteriorating solutions, in the sense that they are accepted with a certain probability, which decreases with the amount of deterioration and over the course of the algorithm. The effect is that in the beginning, large deteriorations may be accepted, thereby ensuring a proper exploration of the solution space. As the algorithm continues, the chance of accepting

deteriorations decreases, until in the end (almost) only improvements are accepted. Although simulated annealing uses a simple acceptance mechanism, it has the nice theoretic property that it converges to globally optimal solutions with a high probability.

Although simulated annealing does accept deteriorating solutions with a certain probability, it may run the risk of jumping back directly in the next iteration, and it may be very difficult to reach a certain good solution if that takes several deteriorating steps to get there. To overcome the former problem, *tabu search* maintains a list of decisions that are not allowed to be undone in the next couple of iterations. The latter problem is tackled by *variable depth search*, in which a given number of steps are performed in a sequence, deteriorating or not, after which the best intermediate solution is taken for the next iteration.

Local search is easily applicable in the sense that it requires (nearly) no specific problem knowledge, and it has given good results for various well-known problems. Furthermore, if problem-specific knowledge is present, this may be used to further improve the performance. For instance, the neighborhood structure may be reduced such that only promising solutions are kept. Although this may affect the theoretical underpinning of the approach, it may speed up the algorithm drastically, by which it can obtain better solutions in practical running times.

13.4.2 Reasoning

Reasoning algorithms use knowledge for drawing conclusions. Motivation for using a separate knowledge component was originally derived from the "combinatorial explosion" encountered with many early artificial intelligence programs (Russell and Norvig 2003). An expert system, or knowledge-based system, consists of a reasoning program and a knowledge base (Stefik 1995). Knowledge bases can take the form of rule bases or ontologies, which provide machine-understandable definitions of terminology. Bayesian methods are widely used to handle uncertainty in reasoning algorithms.

Expert Systems

One of the earliest expert systems is Dendral, which inferred molecular structures from measurements done with a mass spectrometer. The system used rules which connect peaks in mass spectrometer graphs to specified subgroups of molecules. Another famous early expert system is MYCIN, which used rules to diagnose blood infections. MYCIN's rules connected symptoms to possible causes, while the likelihood of different causes was distinguished by means of numbers called certainty factors. In addition to the term expert system, the term decision support system has also come into use. Many decision support

systems contain a diagnostic reasoning component and advise people to perform a certain new observation or to execute a certain corrective action, for example. As an example, in the area of medicine, clinical decision support systems are attracting attention (Sim et al. 2001).

The area of expert systems and the related area of knowledge representation and reasoning face two central issues: knowledge acquisition and the tradeoff between expressive power and reasoning complexity. With respect to knowledge acquisition, the issue is to obtain declarative knowledge in such a way that it can be used by a system. The standard approach is to interview domain experts in order to obtain the required knowledge bases. However, in many cases, it has been hard or impossible to obtain and maintain suitable knowledge bases in this way; the problem to obtain suitable knowledge is also called the Feigenbaum bottleneck, after one of the originators of the field. Representatives from the application domain may develop the required knowledge bases, when a suitable metamodel of knowledge for the application domain is available. As an alternative to the involvement of domain experts, learning techniques may be useful to discover knowledge on the basis of evidence provided by data contained in databases or on the web, for example.

The second central issue that was mentioned, the tradeoff between expressive power and reasoning complexity, concerns the fact that if much freedom is allowed to express knowledge, then reasoning may become intractable. For the classical formal paradigm of reasoning, first-order logic, the problem to determine whether a conclusion is valid is undecidable in general; this was proved in the paper that introduced Turing machines as a general model of computation (Turing 1936). In order to enable reasoning, practical systems need to use restricted formalisms for expressing knowledge. The two central issues together lead to the challenge to develop a knowledge model for an application domain, enabling knowledge bases to be recorded and maintained in practice, in combination with a corresponding reasoning procedure realized by tractable algorithms.

Traditionally, a knowledge base typically consists of rules. In addition to rule bases, knowledge bases can also be *ontologies*, which provide machine-understandable definitions of the meaning of concepts. The combination of rule bases and ontologies leads to appealing possibilities for intelligent algorithms. For example, a system may get as input data from certain sensors, describing the context of use of the system in a low-level fashion. The sensor data are used in combination with an ontology to perform "context determination," making sense of the context of use by describing it in terms of high-level concepts defined in the ontology. Subsequently, a rule base, entirely phrased by means of high-level concepts on the level of the user, can be used to trigger actions performed by the system, for example, to realize preferences which a user has stated to apply to the current context of use.

The W3C is developing standard languages to support machine reasoning by means of knowledge on the web, to realize the vision of the semantic web (Berners-Lee et al. 2001). Two semantic web languages are already available

in standard form: RDF (Resource Description Framework) and OWL (Web Ontology Language). RDF plays a basic role by allowing the expression of statements. OWL allows the expression of ontologies, which define meaning of terms used in RDF statements. Simple ontologies can already be expressed with the RDF Schema (RDFS) vocabulary. The W3C is standardizing a rule language in the future. Even without rules, the languages RDF and OWL lead to the possibility of automatically drawing conclusions from information on the web. For RDF and OWL, the valid conclusions, commonly called entailments, are determined by a logic and its semantics. Although the standard syntax for RDF and OWL uses XML, the meaning of RDF and OWL knowledge bases is independent of XML, and abstracts from the XML serialization used. Here the notion of RDF graph plays a role. An RDF or OWL knowledge base is formalized as an RDF graph, which is a set of RDF statements, i.e., subject–predicate–object triples; subjects and objects of triples are viewed as nodes, linked by predicates. Predicates are usually called properties. RDF includes variables, which are called blank nodes, and which are, implicitly, existentially quantified. An RDFS or OWL ontology describes concepts (i.e., classes) and relationships (i.e., properties). The RDFS vocabulary enables in particular the definition of classes and subclass relationships, and the specification of domain classes and range classes for properties. OWL extends RDFS in various ways. For example, properties (i.e., binary relations) can be stated to be functional or transitive or symmetric, or to be each other's inverse; classes can be stated to be disjoint or to be the union or intersection of other classes; it is possible to use constraints to define classes, for example, to define the class of persons all of whose parents are American, or the class of persons with two children. There are two variants of OWL with different semantics: OWL Full and OWL DL. OWL Full entailment is undecidable. OWL DL imposes restrictions on the use of the language to ensure decidability. For example, classes cannot be used as instances in OWL DL. OWL DL is supported by techniques developed in the area of description logics (Baader et al. 2003). Although description logics form decidable fragments of first-order logic, for which optimized reasoners exist, verification of OWL DL entailment requires nondeterministic exponential time (Horrocks and Patel-Schneider 2003). In analogy to RDFS, a weakened variant of OWL has been described which does not impose restrictions on the use of the language and for which entailment is NP-complete, and in P when the target RDF graph does not contain blank nodes (ter Horst 2004).

Bayesian Methods

It has been common practice to handle uncertainty in expert systems by means of numbers. For example, we mentioned already the uncertainty factors of MYCIN. Bayesian methods use probabilities, and seem to form the most popular way of handling uncertainty in expert systems (Pearl 1988; Jensen 2001). A central role is played by Bayes' rule. For random variables V and W with

values v and w, respectively, Bayes' rule connects the conditional probability that $V = v$ given that $W = w$ to the converse conditional probability and the probabilities that $V = v$ and $W = w$:

$$P(V = v|W = w) = \frac{P(W = w|V = v)P(V = v)}{P(W = w)}.$$

This rule follows from the relationship between conditional probabilities and joint probabilities:

$$P(V = v|W = w) = \frac{P(V = v,\ W = w)}{P(W = w)}.$$

A *Bayesian network* is an acyclic, directed graph for which the nodes are labeled with random variables $V_1, \ldots, V_n$. A Bayesian network is essentially a compact representation of certain conditional independent relations: it is assumed that each variable V_i is conditionally independent of each set of variables $\boldsymbol{A}_i$ that are not descendants of V_i, given the set of parent nodes $\pi(V_i)$ of V_i: $P(V_i|\boldsymbol{A}_i, \pi(V_i)) = P(V_i|\pi(V_i))$. More precisely, this equation is assumed to hold for each value of each variable included. The conditional probabilities $P(V_i|\pi(V_i))$ are stored for each variable V_i, and include the probabilities $P(V_i)$ at the root nodes. Bayesian inference allows the computation of many probabilities and conditional probabilities by means of these probabilities $P(V_i|\pi(V_i))$. For example, the full joint probability distribution $P(V_1, \ldots, V_n)$ is the product of the conditional probabilities $P(V_i|\pi(V_i))$ for all $i = 1, \ldots, n$. This shows that if each variable has two values and if each node has at most k parents, then all the 2^n joint probabilities can be computed with at most $n2^k$ conditional probabilities. For values $w_1, \ldots, w_k$ of variables $W_1, \ldots, W_k$ and values $e_1, \ldots, e_m$ of evidence variables $E_1, \ldots, E_m$ appearing as descendants of the variables $W_1, \ldots, W_k$ in the network, there are standard, recursive procedures to compute the conditional probabilities $P(E_1 = e_1, \ldots, E_m = e_m|W_1 = w_1, \ldots, W_k = w_k)$ by means of the given conditional probabilities $P(V_i|\pi(V_i))$. Bayes' rule can be used in combination with such a procedure to go in the other direction, i.e., from "effect" to "cause":

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}.$$

The computation of conditional probabilities given a Bayesian network is NP-hard in general. When the Bayesian network has a relatively simple structure, for example, when there is at most one undirected path between any pair of nodes, there exist polynomial time algorithms. The most efficient general exact algorithms for probabilistic inference use the Bayesian network to form a kind of parallel computer which exchanges messages in both directions between neighboring nodes, for example, by performing lazy propagation in junction trees (Jensen 2001). There exist stochastic approximation techniques, which can handle much larger Bayesian networks than the exact algorithms.

Bayesian methods have been extended to decision graphs and dynamic Bayesian networks (Jensen 2001; Russell and Norvig 2003) using sensor data values, utilities, and actions. Using probabilistic reasoning, actions are selected that maximize expectation values of utilities. This forms a widely used way to realize the "utility-based agents" that will be mentioned in a later section. Robotic perception systems use variants of these methods that involve continuous random variables rather than discrete random variables (Russell and Norvig 2003).

Bayesian reasoning methods are powerful, even though knowledge acquisition often remains an issue. The structure of Bayesian networks can in many cases be obtained by means of causal knowledge that is available from domain experts, but the conditional probabilities $P(V_i|\pi(V_i))$ are typically more difficult to obtain; the question "where do the numbers come from?" continues to raise discussion. Learning methods for this problem have been widely investigated. We discuss two methods for obtaining the conditional probabilities when the structure of the network is known, and a database of cases (training data) is available, possibly with missing values of certain random variables. In the gradient ascent learning method (Russell and Norvig 2003), a maximum likelihood hypothesis for the conditional probabilities is found. Hypothesis h represents values of the conditional probabilities $P(V_i|\pi(V_i))$. The objective function to be maximized by a gradient descent procedure is the probability $P(D|h)$ of the observed training data D given a hypothesis h. This method leads to a local optimum. Another method that is widely used for obtaining conditional probabilities from a database of cases with missing data is the expectation maximization (EM) method, which can be described as follows (Nilsson 1998). The starting point is a database of cases which do not all have values for all the random variables $V_1, \ldots, V_n$. For example, in a situation where the variables are Boolean, there may be ten cases where the variables $V_1, \ldots, V_{n-1}$ are known to be one but where the value of V_n is unknown. These ten cases are then handled in terms of "weighted cases," with value $V_n = 1$ with weight $P(V_n = 1|V_1 = 1, \ldots, V_{n-1} = 1)$ and value $V_n = 0$ with weight $P(V_n = 0|V_1 = 1, \ldots, V_{n-1} = 1)$. In the following step the conditional probabilities $P(V_i|\pi(V_i))$ are given random values, and Bayesian inference is used to compute the conditional probabilities used as weights in the cases with missing values. The database of cases, with missing values thus interpreted in terms of weights, is then used to estimate the conditional probabilities $P(V_i|\pi(V_i))$ as fractions of frequencies $N(V_i, \pi(V_i))/N(\pi(V_i))$. The conditional probabilities $P(V_i|\pi(V_i))$ thus obtained are used to obtain new values for the conditional probabilities used as weights in the cases with missing values. This procedure is iterated, and again converges to a local optimum.

13.4.3 Learning

Learning is the ability to improve one's performance through experiences in the past. Learning algorithms are typically applied in situations where initially

only partial information is available to solve a given problem and gradually over time additional information becomes available and in situations where adaptation to changes in the environment is essential to obtain high-quality solutions. Examples of these types of applications are learning the preferences of a user for TV-programs on the basis of his/her viewing history, and adaptation to changes in voice and background noise in speech recognition.

Learning can range from simply memorizing past experiences to the creation of scientific theories (Russell and Norvig 2003). We focus on two specific computational paradigms, namely neural networks and reinforcement learning.

Neural Networks

An artificial neural network is a computational model that tries to follow the analogy with the human brain. It is built from artificial neurons or nodes, based on a simplified mathematical model of the biological neurons in the brain. A given node is connected to multiple input nodes, of which the states are given by $x_1, x_2, \ldots, x_n$. These connections have weights $w_1, w_2, \ldots, w_n$. The output or state y of the node may be discrete, say $y \in \{0, 1\}$, or real valued. For deterministic neurons, the output y is a function of the inputs

$$y = f(\Sigma_{i=1,\ldots,n} w_i x_i - b), \tag{13.1}$$

where b represents a threshold and f is some nonlinear function. If the neuron has a discrete output, then f will be a step function. For stochastic neurons with discrete output, the right-hand side of (13.1) gives the probability that the neuron has output 1.

Neural networks usually consist of many nodes that are connected in some way. A well-known example is a layered feed-forward network, where the nodes are arranged in multiple layers, from a first layer of input nodes via possibly multiple layers of intermediate nodes to a last layer of output nodes. In such a network, the nodes on layer i can only be input to the nodes in layer $i + 1$.

Such a feed-forward network can be used for classification purposes, where the states of the nodes in the input layer are directly determined by external input. They represent the features of the object that is to be classified. In successive steps, (13.1) is used to determine the states of the nodes in the successive intermediate layers, until in the last step the states of the output nodes are determined. These give an encoding of the class to which the object is supposed to belong. A simple encoding is given by using as many output nodes as we have classes that we want to distinguish, where a classification requires exactly one of the output nodes to receive state 1 and the others state 0. To realize a correct classification of objects, the connection weights w_i and the threshold b must be set appropriately for each of the neurons. If the relation between the states of input nodes and the required state of the output nodes is sufficiently understood, then these could be hard-coded. Usually, however, this is not the case.

Neural networks are especially interesting for applications where the relation between the states of input nodes and output nodes is unknown. By repeatedly providing it with examples of correct input–output combinations, a neural network will ideally be able to learn the underlying input–output relation by adapting the connection weights w_i and thresholds b of its nodes.

Research on feed-forward networks dates back to the 1950s and 1960s, when networks consisting only of input nodes and output nodes were extensively studied. These so-called perceptrons (Rosenblatt 1957; Minsky and Papert 1969) can represent only linearly separable concepts. The field has seen a strong revival in the 1980s and 1990s, when learning feed-forward networks with one or more intermediate layers using the back-propagation learning algorithm (Rumelhart et al. 1986) became common practice. The algorithm works in small iterative steps. In each step, observed errors at the output nodes are propagated back to the intermediate nodes to make small adjustments to the weights and thresholds. The success of learning algorithms in neural networks can be hindered by the possibility of getting stuck in local optima, by the slow speed of learning, and by the uncertainty on choosing an appropriate structure of the neural network. The structure of a neural network determines the number of layers and the number of nodes per layer.

Reinforcement Learning

Reinforcement learning is an example of unsupervised learning. Instead of learning from given input–output examples, reinforcement learning can best be characterized as learning from interaction. We next explain it in the context of sequential decision problems.

A sequential decision problem can be modeled as a Markov Decision Process (MDP) as follows. It models an agent whose environment can be in one of a finite set of states $s_1, s_2, \ldots, s_n$. In successive iterations, the agent has to choose one of a finite set of actions $a_1, a_2, \ldots, a_m$. The next state is determined by a probability distribution that depends on both the current state and the agent's action. Hence, the MDP is characterized by transition probabilities $P(s, a, s')$, which gives the probability of going from state s to state s' when action a is chosen. In addition, a reward or reinforcement function $r(s, a, s')$ gives after each action a reward that depends on the current state, the action, and the next state. Both the transition probabilities and the reward functions are assumed to be stationary, i.e., they do not change over time, and memoryless, i.e., they depend on the current state but not on previous states. The goal is to learn a policy that specifies (probabilistically) which action to choose in a given state such that the expected overall reward is maximized.

An optimal policy for a finite number of states and a finite number of iterations can be computed off-line using, for example, dynamic programming if transition probabilities and reward functions are given explicitly. In many applications, this is not the case. Instead, the agent starts with zero knowledge and it has to learn from interacting with the environment. A well-known

on-line reinforcement learning algorithm is Q-learning (Watkins 1989). The algorithm maintains state-action values $v(s, a)$ for each state s and action a, representing the expected benefit of choosing action a in state s. Initially, they are chosen randomly. Based on these estimates, the agent chooses a next action. Based on the resulting state and reward, the estimates are updated. A parameter balances between exploiting the current estimates and exploring new possibilities to improve the estimates. Many alternative learning algorithms have been proposed. For further details, we refer to Sutton and Barto (1998). Reinforcement learning has been applied in various settings, such as in dynamically adapting the quality of service in video processing in set-top boxes and digital TV sets, to adapt the highly fluctuating processing requirements to the available processing resources in programable hardware (Wüst and Verhaegh 2004).

13.4.4 Evolution

Evolutionary computing is inspired by the way species are thought to evolve over time in nature. Evolution can be viewed at the level of species and at the level of individuals. Intelligent agents are autonomous software systems that sense their environment and that can achieve a form of evolution by being responsive, goal-directed, and socially able. As a computational paradigm, evolution can be viewed as extending and including the previous paradigms (search, reasoning, learning).

Evolutionary Computing

Evolutionary computing covers various computational paradigms that draw their inspiration from nature, such as genetic algorithms, genetic programming, evolutionary programming, and genetic local search. The two key mechanisms in evolution are *selection* and *variation*.

In nature, selection is achieved by a process called survival of the fittest, meaning that individuals of a population that are better adjusted to their habitat have a higher chance of surviving. These individuals also have a higher chance of mating, which implies that their genotype is more probable to remain in successive generations of the population. As a result, the population as a whole is better fit to its habitat. Variation is established by *recombination* and *mutation*. Recombination is obtained by sexual reproduction, where the genotype of two parents is combined in a new genotype of their child. Mutation corresponds to random perturbations of the genotype of an individual. Variation creates individuals that are potentially better suited to their habitat.

The application of this computational paradigm to combinatorial optimization problems or, more general, to search problems is quite straightforward. If we want to find a high-quality solution from a large set of solutions, then this problem can be formulated in evolutionary terms as follows. A solution can be considered as an individual. Starting with a set of initial solutions

as starting population, the quality of the solutions in successive generations can be improved by preferably combining high-quality solutions to generate new solutions (recombination) and by realizing occasional random changes (mutation).

Genetic algorithms (Holland 1992) aim at faithfully mimicking this evolutionary process by representing each solution as a DNA-like string of characters and by realizing recombination through crossover operations. Other variants such as genetic local search aim at following the same basic principles more loosely, for example, by improving the individual solutions through iterative improvement until they are transformed into local optima, before the next recombination step is carried out. Genetic programming is closely related to genetic algorithms. Instead of recombining and mutating strings that represent solutions, genetic programming operates on computer programs (Koza 1994).

Intelligent Agents

Agents are computer systems that observe their environment by means of sensors and autonomously act in their environment by means of actuators; intelligent agents respond to changes in their environment and take initiatives in order to satisfy their objectives, and are moreover capable of interaction with other agents in order to satisfy their objectives (Wooldridge 1999). In other words, an intelligent agent is reactive, goal-directed, and socially able. Social ability implies negotiation and cooperation with other agents. Although there has been much work on reactivity and goal-directedness, there does not yet exist a widely accepted method of integrating goal-directed and reactive behavior. Investigations on intelligent agents often make significant use of other fields, such as game theory or modal logic. The semantic web languages already mentioned enable agents to understand the meaning of and to reason with information on the web.

The notion of intelligent agent has been used for the global organization of major recent textbooks on artificial intelligence (Nilsson 1998; Russell and Norvig 2003). Several kinds of agents are considered, with increasing capabilities, and with increasing use of search, reasoning, planning, and learning. Simple reflex agents work with simple condition–action rules, triggered by sensor observations. Model-based reflex agents use a model of the world and have an internal state, and react to new sensor input by updating the state of the world and by choosing an action that takes this state of the world into account. Goal-based systems use their continuously updated model of the world in combination with information about goals to determine their actions. Utility-based agents use numerical utility functions distinguishing the desirability of states, enabling more refined strategies for action than when only Boolean information about goals is available, for example, by using Bayesian methods as described above. Finally, learning agents are able to evaluate their actions and improve their performance.

We briefly discuss several approaches that have been investigated for developing intelligent agents, reflecting the diversity of the field (Wooldridge 1999). Layered architectures seem to have formed the most popular approach. There does not exist a widely accepted way of designing layered agent systems. There are systems where each layer connects to sensor data and action output, while other systems work with a kind of pipeline of layers from sensor input to output actions.

As another way of developing intelligent agents, the belief–desire–intention (BDI) approach (Bratman et al. 1988) has been widely used. The BDI model presents a kind of cognitive structure for agents, consisting of several parts. The information that an agent has about its environment is represented as a set of beliefs. The intentions of an agent represent the state that the agent is committed to reach. The desires of an agent consist of options available to the agent. A BDI agent continuously cycles through a sequence of steps. First, the sensor input is used to update the beliefs. Then, given the current beliefs and intentions, the desires (options) are updated, for example, in view of a plan to reach the intentions. Then, the set of intentions is updated, for example, to include new intentions to realize existing intentions. Finally, an action is chosen on the basis of the current intentions. The BDI architecture does not determine how to achieve a balance between two central issues already mentioned: reactivity and goal-directedness.

In another approach to develop intelligent agents, called reactive architectures, the idea is that complex intelligent behavior can emerge from combinations of simple behaviors, executed by simple reflex agents. One of the principal approaches in this direction is the subsumption architecture (Brooks 1986), which organizes situation–action rules in a subsumption hierarchy. An action is inhibited when there is an action lower in the hierarchy. In this way, an "avoid obstacle" action can be given the highest priority, for example. With this approach, agents can only react to local information. However, a form of cooperation between agents can be realized when agents perform actions such as "dropping" evidence leading to actions of other agents present at close distance. By using such mechanisms, it is envisaged that intelligent behavior can indeed "emerge" from the actions of a collection of simple agents. It is not clear how this approach, based on local information, can be combined with the use of relevant global knowledge.

We conclude this discussion of intelligent agents by briefly turning to an approach, which so far has remained primarily theoretical. If agents intend to coordinate their actions to achieve common goals, they might make use of models of the knowledge of other agents. In this connection, much work has been done using modal logics of knowledge and belief (Fagin et al. 1995). These logics are decidable extensions of propositional logic. We briefly discuss a logic that has been relatively popular, $S5_m$, which extends propositional logic with modal operators $K_1, \ldots, K_m$ for m agents. If P is a statement, then the statement $K_i P$ indicates that agent i knows P. The logic $S5_m$ has several axioms. First, the operators K_i distribute over implication:

$(K_i(P:Q) \wedge K_i P) \Rightarrow K_i Q$. Another axiom states that what each agent knows is true: $K_i P \Rightarrow P$. The last two axioms state that each agent knows that it knows something: $K_i P \Rightarrow K_i K_i P$, and also knows that it does not know something: $\neg K_i P \Rightarrow K_i \neg K_i P$. There are two inference rules. In addition to the familiar modus ponens rule, the logic $S5_m$ also has the so-called necessitation rule: given P, infer $K_i P$. Just as for propositional logic, various reasoning questions for these modal logics can be reduced to the satisfiability problem; for propositional logic, state-of-the-art SAT solvers can handle relatively large cases (Lynce and Marques-Silva 2002). The satisfiability problem for the logic S5 (i.e., only one agent) is NP-complete, just like the satisfiability problem for propositional logic; for $m > 1$, the satisfiability problem for $S5_m$ is PSPACE-complete (Halpern and Moses 1992). The logic $S5_m$ has been useful for reasoning about distributed systems, and has also been popular in connection with agents. However, for agents the logic is considered to be too strong, as it implies that an agent knows all the consequences of what it knows (this is called "logical omniscience"): if a statement P entails a statement Q, then in $S5_m$ we also have that $K_i P$ entails $K_i Q$. It is believed that a realistic approach would involve weaker logical capabilities on the part of agents.

13.5 Intrinsic Limitations

Since the introduction of electronic computing devices around 1950, scientists have been fascinated by the question whether there exists an intrinsic complexity of problems that makes them intractable. For a long time they have been speculating about the question of the intrinsic computational hardness of problems. In an early letter, Gödel (1906–1978) addresses questions to Von Neumann (1903–1957) about the existence of efficient primality tests and about the conjecture made by Gödel on the existence of a general procedure that reduces the N steps needed to exhaustively search all possible solutions to a problem – which Gödel calls *dem blossen Probieren* – to $O(\log N)$ or $O(\log N^2)$ steps. Unfortunately, Neumann's reply to this letter is not known, most probably because it was sent to him less than a year before he died of cancer at the age of 51. Gödels optimism about the complexity of combinatorial problems was not shared by a group of Russian scientists in cybernetics who were convinced that there existed a class of problems that could only be solved by complete enumeration, something they called *perebor*, and they were working on the proof of the conjecture that *perebor* could not be removed.

Theoretical computer science has provided a foundation for the analysis of the complexity of algorithms in computational intelligence, based on the original computational model of the Turing machine (Turing 1936). This has led to a distinction between *easy problems*, i.e., those that can be solved within polynomial running times, and *hard problems*, i.e., those for which it is generally believed that no algorithm exists that solves the problem within a time

that can be bounded by a polynomial in the instance size. Consequently, instances of hard problems may require running times that grow exponentially in their size, which implies that eventually certain tasks cannot be accomplished successfully within reasonable time. For some problems, running times can easily extend beyond a man's lifetime if the instance size is sufficiently large. Garey and Johnson (1979) provided an extensive list of intractable problems in their landmark book. The so-called *intractability* calls for workarounds. One frequently resorts to methods that do not carry out the task optimally but rather approximate the final result. These so-called approximation algorithms indeed can reduce computational effort, but if one imposes the requirement that the quality of the final result is within certain limits then again for many well-known problems exponential running times are inevitable. Therefore, one often resorts to the use of *heuristics* without performance guarantees, which improve running times considerably, but at the cost of final solutions that may be arbitrarily bad in theory.

The computational complexity theory of learning studies a number of problems that are all related to the question how closely a learning hypothesis that is built by a learning system resembles the target function if this function is not known. *Identification in the limit* refers to the capability of a learning system to converge to the target function for a given problem representation. Early work in computer science, building on Popper's theory of falsification, implies that this may not be possible, which implies that there are certain learning tasks that cannot be achieved. A general principle in computational learning is *Ockham's razor*, which states that the most likely hypothesis is the simplest one that is consistent with all training examples. The notion of *Kolmogorov complexity* (Li and Vitanyi 1993) provides a quantitative measure for this, given by the length of the shortest program accepted by a universal Turing machine, implementing the hypothesis. It thus provides a theoretical basis for the intuitive simplicity expressed by Ockham's razor. The theory of *Probably Almost Correct* (PAC) learning addresses the question how many training examples must be applied before a learning hypothesis is correct within certain limits with a high probability (Kearns and Vazirani 1994). The theory reveals that for certain nontrivial learning tasks this number may be exponentially large in the size of the input of the learning system, thus requiring exponentially many training examples and exponential running times.

13.6 Concluding Challenges

From the discussion of the computational paradigms presented above one can draw the conclusion that ambient intelligence may be very well served from a computational point of view by many paradigms that exist already. This indeed is correct, but it does not mean that there are no challenges in the design of algorithms for computational intelligence from an ambient intelligence point of view. Recently we have presented an in-depth discussion of this topic

(Aarts 2005) and below we summarize the most important conclusions from that discussion. In general terms there is a need for the following eight types of algorithms:

1. *Small-footprint algorithms* that allow for the execution of intelligent algorithms in low-power mobile systems such as personal digital assistants and personal communication devices.
2. *Real-time mathematical programming algorithms* that allow real-time execution in complex decision making processes based on mathematical programming techniques such as the dynamic programming algorithms mentioned in the previous section.
3. *Intentional search algorithms* that enable the effective and efficient handling of queries based on user intentions. This calls for approximate string search, pattern matching, and classification techniques that can deal with ultra-large databases.
4. *Feature extraction algorithms* that extract metadata from raw data, thus enhancing the classical functionality of storing and retrieving data items by adding dynamical data manipulation that allows for semantic information processing.
5. *Humanized algorithms* that account for human perceived values such as perceptive measures, inconsistency, and feedback. Consequently, rigid physical objective measures resulting from classical measures such as completion time, capacity, speed, and length need to be replaced by more psychological measures that reflect user experiences.
6. *Aware algorithms* that connect the physical world and the digital world and can process and fuse large amounts of physical data, thus providing awareness and contextual information.
7. *Life-long learning algorithms* that can develop dynamical user models that capture specific knowledge over time of users in relation to the tasks they want to perform.
8. *Collaborative algorithms* that support the distributed nature of ambient systems by allowing the use of autonomous components that run on local devices to perform collaborative tasks and generate immersive experiences.

These algorithmic challenges provide some main directions for future research into computational intelligence in ambient intelligence. Their accomplishment does not necessarily require the development of new computational paradigms. They require the development of ultra-fast implementations and the development of techniques that can handle huge amounts of information. Furthermore, the incorporation of human-centered evaluation techniques that can quantify human perceived values and can capture user behavior is a subject that requires further investigation. Evidently, it should be clear from the many new ideas emerging from the detailing and realization of the ambient intelligence vision that this new field poses great challenges on the design of intelligent algorithms.