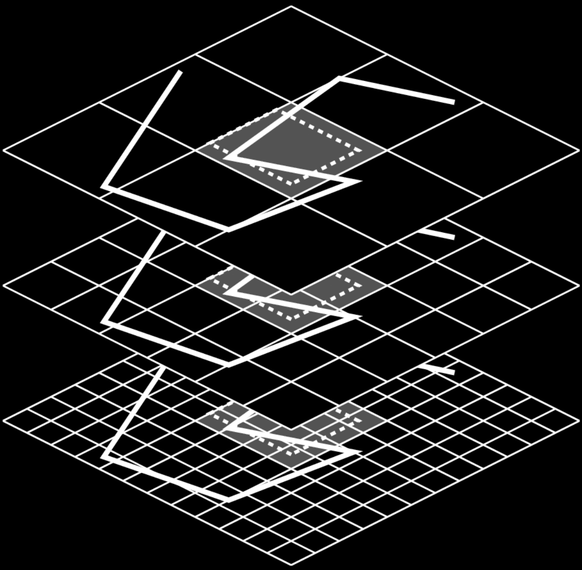
Cognitive Spacetime

A Contribution to
Human-Centered Adaptivity in E-Learning



Kevin Fuchs

Cognitive Spacetime

A Contribution to Human-Centered Adaptivity in E-Learning

Dissertation

Approved by the University of Education Karlsruhe,
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy (Dr. phil.)

by Kevin Fuchs

from Göppingen, Germany

2019

1st reviewer:	Prof. Dr. Mutfried Hartmann
2nd reviewer:	Prof. Dr. Peter A. Henning
3rd reviewer:	Prof. Dr. Gerd Gidion
subject:	Computer Science
date of submission:	August 1, 2018

Acknowledgements

This work was hard. But unimagined things become possible when we meet the right and warm people at the right time.

Let me express my great gratitude to my supervisor Prof. Peter A. Henning who substantially promoted and encouraged me professionally as well as personally. At his institute I found the best working conditions one could wish for, providing space for passion and creativity.

I am also deeply grateful to my supervisor Prof. Mutfried Hartmann who was an inspirational and motivating patron to me. Our conversations broadened my horizon beyond my field and were a vital enrichment for me and this work.

I also thank the other members of my thesis committee: Prof. Gerd Gidion and Prof. Petra Lindemann-Matthies.

Last but not least, I sincerely thank the Friedrich Naumann Foundation for Freedom and the generous scholarship they gave me. This research would not have been possible without this support.

Contents

1 Human Machine Symbiosis

The raise of so-called artificial intelligence has made people believe that computers may some day be congenial with human beings. In the past computers were regarded as effective but soulless and unintelligent assistants to free humans from routine tasks. Computers were supposed to perform time-consuming but mechanical calculations. Today's computers are universal machines that can execute an almost unlimited variety of software. The increase of processing speed allows us to implement complex software which does not seem to have much in common with past computing machinery. In the field of education this awakened the desire to build algorithms which didactically support learners or even emulate human-like tutors.

However, despite the apparent complexity of today's software, algorithms are step-by-step procedures which in their core are purely mechanical. So before introducing just another approach for technology-enhanced learning let me reconsider a seemingly naive but fundamental question. Given the nature of how computers work on the machine-level, can we emulate human-like tutors with computers? I believe that we can not because human beings are in possession of abilities which can not be implemented with algorithms due to their mechanical kernel and the formal systems on which algorithms are built. However, there exists a concept with which we can implement a mutual human-machine interaction that enables computers to at least adapt themselves to a learner. The result of this is what we call "adaptive systems". Suppose an algorithm owns a knowledge representation of learning material and the learner. Then it can deduce on that knowledge base, modify it and adapt its own procedures. The key questions are: how do we construct such a knowledge base, how do we infer on it and in what way does today's technology provide better methods than in the past?

This introductory chapter is supposed to provide a theoretical understanding of what we can implement with computers and how we can implement it. This will sketch the possibilities and limitations of technology enhanced processes. I will show that on the one hand even an adaptive algorithm — which looks almost intelligent from outside — is based on only mechanical procedures. But on the other hand such an algorithm can be powerful enough to implement an adaptive human-machine interaction which can be utilized for technology-enhanced learning. In the subsequent chapters I will elaborate in detail on the question how a respective knowledge base and deduction process can be implemented. In this context I will introduce the "Cognitive Spacetime" model which represents a novel method to model learner's behavior in arbitrary learning environments. Within the Cognitive Spacetime similar behaviors among learners can be found and analyzed. In contrast to other approaches the Cognitive Spacetime allows for taking social aspects of learning into account. For example we can identify potential learning mates and bring learners socially together by analyzing their behaviors on the grounds of similarities. Another aspect highlighting the Cognitive Spacetime is its facility to focus on comparisons between learners which makes it a more learner-centric method. Moreover, it is based on the observation of real learners which means that its knowledge base increases and improves over time.

I should note that the term "Cognitive Spacetime" also occurs in the context of psychology and was published by Stocker in

2014 [105]. However, the Cognitive Spacetime model which I introduce in this work is an entirely different concept. It is derived from the "Cognitive Space" model which was developed in the INTUITEL research project [56, 57]. The "Cognitive Space" of INTUITEL was coincidentally also published in 2014 and there is no coherence between Stocker's and my work.

1.1 Teaching Machines

The algorithmic emulation of a human teacher and its implementation with machines has been firing the imagination of both computer scientists and philosophers. We remember the first attempts of Sidney Pressey [83, 87] and the research of Burrhus F. Skinner [10, 103]. In the first half of the 20th century Sidney Pressey introduced what he called teaching machines. These were mechanical devices which could serve specific tasks like multiple choice tests or the completion of gap texts. Followed by Pressey's attempts, the psychololgist Burrhus. H. Skinner elaborated on teaching machines claiming that they were a promising technology for the improvement of mass education [103].

At that time, educational psychologists focused on the concepts of behaviorism. Pressey's teaching machines therefore in particular emphasized the technique of reinforcement learning. The idea behind Pressey's machines was to keep the learner engaged and modify her behavior by a constant flow of instant stimuli. When the learner entered her answers into the device she got an immediate feedback by receiveing a "correct" or "wrong". In fact today's world of e-learning still contains many testing and training procedures that still follow this straight and simple pattern.

Although Pressey's first attempts took place almost a century ago, the motivation behind it and the hopes people like him and Skinner associated with machinery are not really different from today. Skinner's paper on teaching machines in 1958 [103] is filled with arguments that could also be replicated by contemporary publications. He claims that a student should not only be a passive receiver of information. Instead she should be an active part in the instructional process which would be supported by teaching machines in so far as they keep the learner in permanent interaction. Interactivity has been considered a significant argument until today. Contemporary research stresses the impact of interactivity for learner's motivation and success in e-learning scenarios [12, 28, 71, 74, 99]. Interactivity may be considered as interaction of learners with learners, interaction of a learner with the instructor or interaction of the learner with the system [6].

Skinner considered the utilization of technology as a tool to make education serve more people more efficiently. His imagination followed the idea of different machines and programs to serve learners of different types and learning speeds. In other words: technology was supposed to make education more scalable and adaptive in terms of learners' diversity. Skinner thought of teaching machines as a "private tutor" not replacing teachers but supporting individual learners where a human teacher was not available.

These arguments are remarkable as they have not changed too much as far as e-learning in the 21^{st} century is concerned [95, 96, 97]. Even the idea of combining multiple media is not new. Skinner also described how teaching machines could be used in combination with additional learning material like maps, charts, diagrams and auditory content. He also sketched a universal paradigm for technology enhanced learning that is still present and valid in today's e-learning systems: he stated that the process of teaching and learning has to be transformed into a linear sequence of small atomic steps to be implementable with a machine.

Actually, Pressey's and Skinner's ideas and questions have remained contemporary. And the ahistoric ductus on it may give us the impression that the digitization of education has been stuck through decades. However, we should consider two things. First, steady technological progress has been increasing our expectations by new technologies. Therefore, the question how learning can be supported by technology can never be answered definitely. Instead it was and will be lifted to a new level each time some new technology is available. Second, classical educational institutions have remained places where many people meet at the same place to study the same things at the same time for the same goals. Or in other words: they have remained places where diversity is flattened and where many desires for forms of digital learning never emerged.

However, beyond the classroom, digital technologies and the world wide web have allowed access to alternative sources of knowledge. Outside the classroom students became more independent from time and local restrictions. Students use digital media in addition or even in opposition to what teachers present them. Additionally, the sheer variety of digital content enables students to explore things from multiple perspectives. Therefore, the digitization of education has actually taken place. But it seems as if this happened somewhere outside the classical educational institutions. In particular, this has changed the role of teachers. They are no longer the only point of reference in the learning process. This has advantages but it also bears some risks. Students learning with material on their own may be more free but they also run into the risk of losing their way. It would be a desirable thing if we could use technology to help teachers to understand and supervise their students outside the classroom. In this scenario algorithms are supposed to be the teacher's tools and not her replacement. The wide field of learning analytics has been motivated by this desire. The guiding theme of the work which I present here, is therefore best understood as a new concept for human machine symbiosis including the student, the teacher and the computer in a threesome mutual relationship.

1.2 Computability

Technological progress, especially in the field of artificial intelligence, has fed the imagination of intelligent algorithms that can communicate and empathize with human beings. In fact, there has been a significant increase considering the level of connectivity and complexity we can build with computers. But in an impressive way the underlying models of computers or any other form of automaton have been unaltered since these days. In 1936 Alan Turing introduced the model of a universal state-based machine [111, 112] which — together with the λ calculus by Alonzo Church [22, 23, 24] — has remained the strongest model for any form of algorithm — no matter if it is realized with wires and semiconductors or with mechanical gadgets made of wood and metal.

In fact, Pressey's teaching machines were nothing else than state-based automatons that can also be expressed as instances of Turing's universal machine. Actually there are only two factors that have leveraged the capacity of computers to perform complex algorithms: Firstly this is the principle of nesting and reusing algorithms in the context of high level programming languages. One line of code in a high level programming language is translated into maybe hundreds or thousands of instructions on a lower machine level. The lowest of all those levels is the central processing unit (CPU) of a computer. Any program in any programming language is finally compiled to a small and lucid set of very basic operations the CPU can understand. The second factor is nothing more than processing speed, meaning how many of those atomic operations a CPU can perform within a second. I will explain this in detail and we will see that — although this may sound magic or irritating in the first place — these two factors are the major difference between Pressey's simple machines and today's technology.

Around 1672 Gottfried Wilhelm Leibniz developed his stepped reckoner — a mechanical device that could perform all four arithmetic operations. Leibniz formed the idea that if an appropriate formal language was provided, there should be a step-by-step procedure that can calculate the truth values for any given mathematical statement. This relates to David Hilbert who in 1928 formulated a problem that became known as the "Entscheidungsproblem" (decision problem) [58, 59] which is summarized as follows: Suppose we have a formal, consistent system based on axioms and rules of inference that allow us to build new theorems. Is there a mechanical procedure that can determine for any statement that is formulated from that system if it is true or not and does this procedure answer this question within a finite number of steps? In other words this asks for a formal system to be complete, meaning that for any statement we can say if it is true or not. It is obvious that if we want to use such a system for computation it has to be consistent, which means that a statement must always be said to be exclusively true or false no matter which way of inference is applied.

In 1931 Kurt Gödel gave a partly negative answer to the Entscheidungsproblem [43]. He referred to the system of the Principia Mathematica [120] proving that there are statements that can not be decided to be true or false. He concludes that if a formal system is expressive enough and if it is consistent we can formulate paradoxes that are not decidable within the system itself. This does not mean that these statements are not decidable at all — they may be explainable from outside the system but not inside of it. Gödel managed this proof by building self-referring statements the provability of which would follow from their non-provability which leads to a contradiction. Gödel's mathematical argument can be illustrated metaphorically by the following example: Imagine a person who has written a sentence on his chest saying "The words on my back are false". His back shows the sentence "The words on my chest are true". These are self-referring contradicting statements leading into an infinite loop. We may discuss this paradox from a meta-perspective, identify it as an insane construction and reject it. However, this is only possible if the observer is located outside in some higher, more expressive system. A computing procedure that is based on a particular formal system can not look at itself from such a position. It is captured inside that system and can only infer with its axioms and rules. Therefore, for a computing procedure such paradoxes are undecidable and this is the reason why decidability has been such a big question in the field of computing machinery. The question of decidability constitutes the limits of anything that is and will be computable with machines.

Kurt Gödel gave a negative answer to the Entscheidungsproblem for the formal system founded by the Principia Mathematica. But still a universal answer was missing. In the thirties of the 20th century Alonzo Church and Alan Turing found that universal answer which was negative, too. Both Church and Turing invented a model each defining very clearly which functions are computable and which are not. Church's model became known as the λ calculus [22, 23, 24] whereas Turing's work became famous as the Turing Machine [111, 112]. The λ calculus and the Turing a Machine are formulated in entirely different ways but in fact they are equivalent. This equivalence is known as the Church-Turing thesis.

Alan Turing's respective paper had the title "On Computable Numbers, with an Application to the Entscheidungsproblem" [111]. As the title suggests, the invention of the Turing Machine served a greater objective: namely finding an answer to Hilbert's Entscheidungsproblem: Is there a mechanical procedure with which we can say for any statement in a formal system if it is true or not? The term "mechanical procedure” is a synonym for ”machine”. Before one can answer this question a definition of ”machine” is needed. Actually this is where Turing’s way of proof begins. In the mentioned paper he first excessively described his definition of a universal machine. Then — given that definition — he worked out that and why there are statements that cannot be decided with such a machine. His arguments also formed the so-called ”Halting Problem”. In plain words, the halting problem formulates the simple question if an algorithm stops or runs eternally. Remember Gödel’s way of proof for incompleteness: he built a mathematical statement based on self-reference the provability of which would conclude from its non-provability. Turing did something very similar by sketching the following problem: Imagine a computer running a program P_1 which may halt. Now imagine a computer running another program P_2 that never halts if the program P_1 halts. Now let P_1 be P_2, meaning that P_2 runs depending on itself but doing the opposite. The question if the program halts is undecidable.

1.3 Turing Machine

Alan Turing did not only answer the Entscheidungsproblem, telling us what can not be computed by a machine. The Turing machine also tells us what on the contrary is computable. It is a universal model for any computationally complete system. I will now elaborate in more detail on the Turing Machine. Such a machine, as illustrated by figure 1.3 consists of an endless input/output tape, a read/write head working on that tape and a control unit that contains the program to be executed. The tape is subdivided into cells. Each cell may contain one symbol from a finite alphabet. Such a symbol embodies the smallest piece of information. The machine can move the read-/write head either left or right but always only by one cell for each iteration. Depending on the program, the machine has

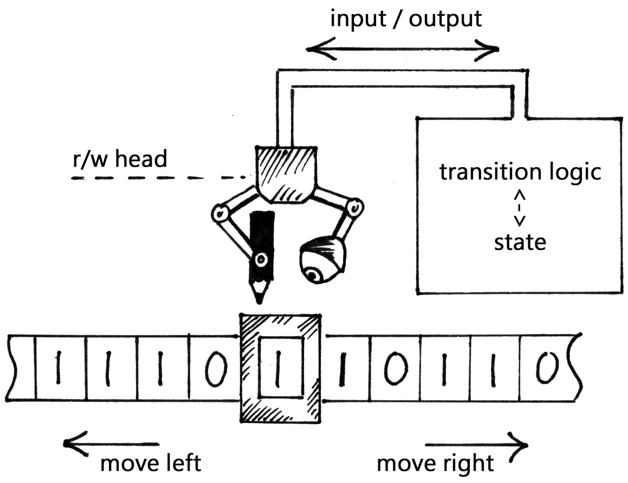


Figure 1.1: Turing machine

a finite set of inner states. The combination of a read symbol and the inner state causes a transition into the next state of the machine. This set of transformations is defined by a set of rules incorporated by the program.

In his publication Alan Turing used tables to define these transitions. I now give an example of a simple Turing machine that checks if two given binary sequences of same length are equal. I call this Turing machine the "Letter Comparator". We suppose that the two binary numbers are represented on the tape like this:

_	_	1	0	0	x	1	0	0	_	_

Underscores "_" indicate blank symbols and "x" separates the two binary numbers to be compared. The Turing machine is defined by table 1.1 which contains for each line the current state, the symbol that is read, the executed operation and the transition to the next state. The operations are:

- print a symbol on the tape
- move the head by one cell to the right
- move the head by one cell to the left

Table 1.1 describes a machine that starts from the initial state ”$\mathfrak{i}$” with the read/write head positioned on the first binary digit on the left. The machine first deletes this digit printing a blank symbol in its place. Remembering the found digit, the machine moves to the right until it finds the first binary digit following the delimiter symbol ”x”. If that digit is not equal to the remembered one the machine stops in the state ”$\mathfrak{f}$”. If the digit is equal it is replaced by ”x” and the machine moves back to the first remaining digit on the left. From there the process repeats. If all digits are equal the machine stops in the state ”$\mathfrak{t}$”.

In his original paper Alan Turing also explains how a pair of current state and input symbol can also refer to another table of a different Turing machine. This is equivalent to a program calling a sub-program. In our example the binary sequences could represent ASCII codes for single letters. A routine for string comparison of whole words could therefore be a superior Turing machine calling the Turing machine of table 1.1 for each letter of the string. This simple construction with a machine calling another machine enables us to build very complex algorithms being built from smaller ones.

Remember Pressey's teaching machines. Those were mechanical constructions that can also be replicated with Turing machines. Take for example gap texts for which students had to type specific words into the machine. This is exactly described by the mentioned string comparison algorithm. Even more simple is the implementation of multiple choice questions. If each possible answer is labeled with a single symbol we only

Table 1.1: Letter Comparator

state	*symbol*	*print*	*move*	*next state*
$\mathfrak{i}$	x			$\mathfrak{t}$
$\mathfrak{i}$	0	_	Right	$\mathfrak{z}$
$\mathfrak{i}$	1	_	Right	$\mathfrak{o}$
$\mathfrak{o}$	x		Right	$\mathfrak{e}$
$\mathfrak{o}$	0		Right	$\mathfrak{o}$
$\mathfrak{o}$	1		Right	$\mathfrak{o}$
$\mathfrak{z}$	x		Right	$\mathfrak{r}$
$\mathfrak{z}$	0		Right	$\mathfrak{z}$
$\mathfrak{z}$	1		Right	$\mathfrak{z}$
$\mathfrak{e}$	x		Right	$\mathfrak{e}$
$\mathfrak{e}$	0			$\mathfrak{f}$
$\mathfrak{e}$	1	x	Left	$\mathfrak{a}$
$\mathfrak{r}$	x		Right	$\mathfrak{r}$
$\mathfrak{r}$	0	x	Left	$\mathfrak{a}$
$\mathfrak{r}$	1			$\mathfrak{f}$
$\mathfrak{a}$	x		Left	$\mathfrak{a}$
$\mathfrak{a}$	0		Left	$\mathfrak{a}$
$\mathfrak{a}$	1		Left	$\mathfrak{a}$
$\mathfrak{a}$	_		Right	$\mathfrak{i}$

have to compare the student's answer with the symbol associated with the correct answer using our Letter Comparator from table 1.1.

1.4 Complex Machines

The Turing Machine and the λ-calculus are more than just models about how computers and algorithms work. They provide an irrevocable answer to the question what is computable and what is not, no matter if we build a machine from gears and arms or from semiconductors and wires. With all of this in mind, the question arises what leveraged the enormous performance of computers in the last decades.

Ultimately, the answer is given by two simple factors: firstly the nesting and reuse of functions and secondly the processing speed. On the machine level a central processing unit (CPU) uses a relatively small set of basic operations. These are for example arithmetic operations like *Add*, *Subtract*, *Multiply*, *Divide* or logical operations like *And*, *Or* and so on. Each operation consists of the operator and its operands. Both the operator and the operands are encoded by binary numbers. To illustrate this, the arithmetic operation 7+3 might be encoded by three numbers each of which is eight bits long:

$$\underbrace{10101110}_{\text{Add}}\,\underbrace{00000111}_{7}\,\underbrace{00000011}_{3}$$

We can easily imagine a CPU as a Turing machine that processes a tape containing an arbitrary number of such operations, each of them encoded by binary numbers of the above form. The machine reads bit by bit and each time it recognizes a particular operator the machine is transferred into the respective state — in this example this would be the "Add" state. Then the Turing machine calls another Turing machine in the sense of a sub routine. This sub machine processes the consecutive bits representing the operands, performs the requested operation on them and prints the result to some space on the tape. Finally, the child routine ends returning to the parent machine which awaits the next operator on the tape. This is how computation works on the lowest level.

Actually hardly anybody will write computer programs directly for a CPU bit by bit, although it is possible. Instead we use assembly languages to replace binary numbers by human-readable mnemonics like "ADD 7, 3" or high-level programming languages like C, C++, C#, Scala, Haskell or Java. Any of these languages is translated into binary machine code to be processed on a CPU.

The mightiness of high-level programming languages is induced by the simple fact that one line of code may be translated into a large number of single machine operations. Moreover, multiple lines of code can be encapsulated by reusable functions which again can be nested by other functions. On an even higher level large sets of functionality are provided by libraries and frameworks (see figure 1.2).

Remember that this paradigm of reusable, nested code is already expressed by the Turing machine and its capability of calling other Turing machines. However complex a computer program may be, at the very bottom it is translated into binary machine code that is composed by only a relatively small set of basic operations. After all, the growing capacity of today's and future devices is determined by the number of such basic operations a CPU can process per time unit. Therefore, it is processing speed coupled with function nesting that makes all the difference between today's technology and Pressey's first teaching machines.

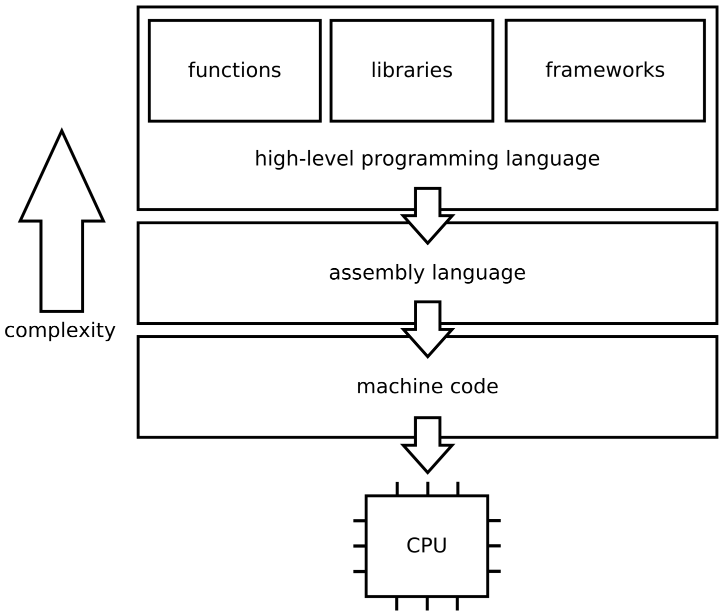


Figure 1.2: From high-level languages to machine code

1.5 The Empathetic Algorithm

In the previous writing I have sketched a brief and condensed history of what constitutes the foundations of computer technology. I also summarized the early efforts to construct teaching machines and I located them within the theoretical structure of computer science.

The Turing machine and the λ-calculus universally answer the question of computability. Whatever we intent to implement with algorithms, these models tell us if and how it can be implemented. This is not a merely theoretical assertion. For example the Haskell programming language is built up from the λ-calculus [30]. When it is compiled to machine code a Haskell program is translated intermediately into a representation of the λ calculus.

The motivation of this work is carried by the intention to implement a kind of private tutor with computers. Having Church's and Turing's models of computability in mind, can we frame rules on the implementation of such a system? In other words: is there a generic design pattern for human-like tutorial guidance that can be derived from these models? This question leads to a great venture inasmuch as we are lacking a clear imagination of what essentially a "human-like" tutor is supposed to be. What are the significant traits of such an entity?

The main trait of a human teacher is empathy in the sense of being able to take over the perspective of her student. This is not only a requirement for teachers. Any form of successful human interaction is conditioned by the imagination of another person's mind, but it plays a particularly important role for the field of education and instructional design. Taking over another person's perspective is the basic requirement to explain things to somebody else. This is also true and especially challenging in the case of an instructional designer who creates online courses without being confronted personally with her target group. During the design process she has to imagine the perspectives and situations of those students who potentially use her learning material. A good and sensitive instructional designer will try to imagine her target group as a diverse spectrum of different learning habits, goals and preferences. She therefore will provide multiple ways for individual students to work on the respective learning material. Generally this is expressed by the concept of user models. In the field of e-learning different flavors of such models exist, commonly known as learner models, learner personas or digital twins.

We should bring to our mind that the original process of thinking, imagination and perspective taking is achieved by the human teacher who then transforms it into a representation that can be processed within the formal system of a machine. Consequently, the machine reproduces the very result of human thinking and empathy. It does not replicate the process of thinking and empathizing by itself. Obviously the human being is the constructive authority whereas the computer only represents the executing instance following human-made rules. In the scenario of technology-enhanced learning, the human's role is to instruct a computer to instruct another human. Algorithms may create learning recommendations and instructions but they can only do this on the basis of human knowledge which in the end is the teacher's knowledge. This clarifies the role of such algorithms but the question remains how to build them.

An algorithm that is able to create didactically meaningful instructions for an individual learner would require the emulation of perspective taking skills. In order to explain something to somebody else or instruct him, one needs to imagine things from the other person's view. Human beings start learning this form the age of four. This has been researched by Jean Piaget [85, 118]. A well-known experimental-setup to illustrate per-

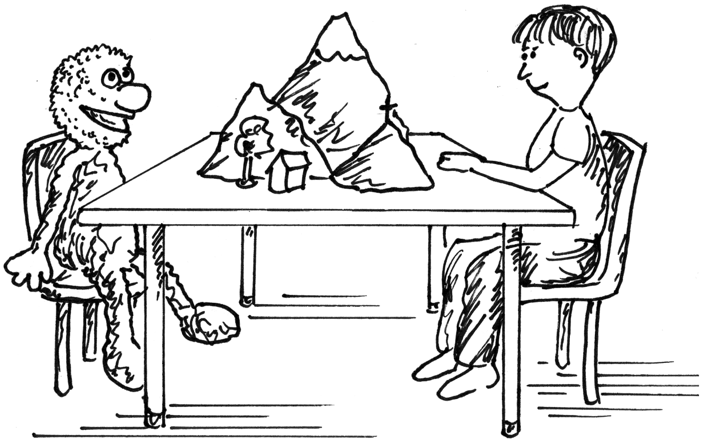


Figure 1.3: Three mountains task

spective taking skills is Piaget's Three-Mountains task which is shown by figure 1.3. In this arrangement a child is situated in front of a three-dimensional landscape. When viewed from different perspectives the diorama shows different details. A doll is seated down at the table with the child sitting at the opposite side. The child is asked to imagine what the scenery looks like from the doll's perspective. From a collection of photographs, the child has to select the one which represents best the doll's view. Children who are younger than four usually fail and select the photograph that represents their own view. At the age of eight, children successfully complete the task. Young children are obviously not able to imagine another person's perspective, unable to recognize the difference between themselves and others. This is what Piaget called the egocentrism of the child [29, 64].

A machine can easily be programmed to perform this very specific task. Pattern recognition can be used to recognize aspects, parts and perspectives of the landscape. However, this would require humans to separate the entire task into clearly defined sub-procedures and have them solved by human-programmed algorithms. Deep learning algorithms from the field of artificial intelligence are capable of developing new repertoires of behavior based on training with sample data. Such a machine can be trained for a set of different landscapes. But if the machine is confronted with a new and totally unknown landscape it is doomed to fail. Furthermore, the Three-Mountains task mainly considers the visual-spatial perception. But taking the perspective of another person also includes inner processes that are not observable from outside. Imagining what is going on in another person's mind therefore remains a human privilege and there is a reasonable explanation why computers are not capable of it.

Let me illustrate this with an example by Robert L. Selman who wrote on the developmental psychology by Piaget. In Selman's example a child is forbidden to climb on trees as she may hurt herself. Now there is a kitten stuck on a tree. What is the child supposed to do if on the one hand the kitten has to be saved but on the other hand the child is forbidden to climb on the tree [100]. If we want to understand the meaning and relevance of rules, it is essential to take over the perspective of the other person who made the rules. Moreover, understanding the meaning of rules is required for rejecting or altering rules, considering other conflicting rules or circumstances.

Instead of a human child, let me generalize these thoughts to any form of a "system". What happens if a system works on a set of rules but is incapable of understanding the context and meaning of them? Taking the example above, we can formulate two simple rules: save the kitten and never climb on trees. These two rules work well as long as there is no situation triggering both rules. But if the kitten has to be saved from a tree, these two rules create a situation which is plainly not decidable because one rule implicates the opposite of the other rule. It may be decidable for a conscious human being. But it is undecidable for a system which is unable to look at its own rules from a higher perspective, and this is especially the case for any form of machine or algorithm. Actually, this can be understood as another variation of Hilbert's Entscheidungsproblem. Alan Turing's and Alonzo Church's answers accordingly restrict everything which is implementable with algorithms. Taking a Turing-complete programming language it is impossible to program the above described dilemma of the kitten one-to-one. This is because such programming languages are exactly designed this way. Any system which is supposed to be computationally complete must prohibit the formulation of such paradoxes.

Being able to look at your own rules from a meta-perspective or to take over another person's perspective is related to what we call consciousness. If we want to take over another person's perspective we first have to be aware of ourselves by dissociation to other people and our environment. Consequently, replicating a human teacher with a computer would therefore demand from us to create a machine that is conscious of itself.

Can machines be conscious? Alan Turing once suggested the Turing test to falsify if a machine can be considered "intelligent" [113]. Besides intelligence, the Turing test is often also related to the question of consciousness. The experiment, as shown by figure 1.4 focuses exclusively on the way a machine communicates with a human being. A human test person talks to both a computer and a human over a terminal. The computer and the human counterpart are hidden from the test person. This way the terminal-based conversation is the only way for the test person to find out if he is talking to a computer or a human. The computer passes the test successfully if the test person can not distinguish it from the human.

This definition is remarkable as it does not make any assumptions on the way human intelligence or consciousness works on

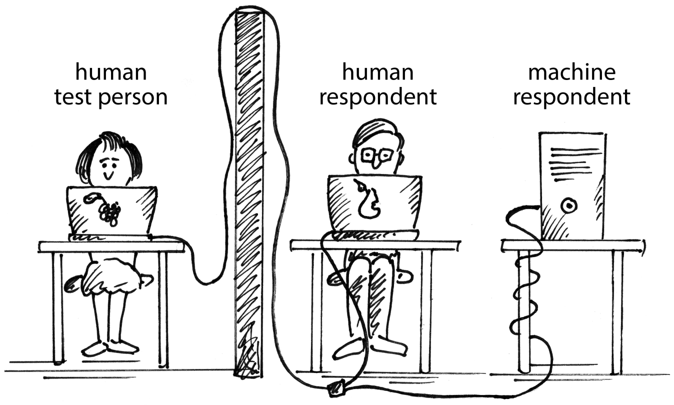


Figure 1.4: Turing test setup

the neural, biological or psychological level. It declares human beings as the only authority to judge, using human-machine interaction as the basis of elevation. Moreover, this approach follows the simple principle of falsification: we gracefully assume that a machine is intelligent as long as a human being is not able to distinguish it from a human counterpart.

The Turing Machine and the λ calculus provide strong models telling us what is implementable with machines. I opened this section with the question if these models can provide us guidance for the implementation of human-like tutoring systems and it lead to the unsatisfactory problem of consciousness. Can machines be conscious? I choose an easy escape and leave the discussion on conscious machines to philosophers, contenting myself with a more modest concept for which I certainly can say that it is implementable with algorithms. This concept is called "adaptivity" and it is also a core pattern in human communication. Whenever we try to understand another person's perspective and situation we commonly start from suggestions that are not exact and may be built from prejudices. We use mutual communication to converge and make our image more accurate. Human intelligence therefore is closely related to communication. This is why Alan Turing's idea to make human-machine interaction a basic criterion for intelligent machines directs us in the right way. However, communication should not be understood only in the sense of verbal language. It should rather be seen as a wide range of interactions that constitute an exchange of information and the creation of knowledge from this information. We can consider this as a form of an adaptive process: We start from a fuzzy image of another person. Information exchange and mutual feedback makes this image become crisper and we constantly refine and alter our image.

This principle can seamlessly be transformed into adaptive human-machine interaction. In the context of computer-based instruction and teaching, an adaptive system is able to observe a user's behavior and progress, match it with its knowledge base and modify its own procedures. The system may then generate recommendations for the user. The user may follow the recommendations or she may deviate from it. The system may then react on this deviation by reconsidering the user's new situation and generate new recommendations. This describes a mutual form of adaptivity with the user influencing the procedures of the machine and vice versa.

Mutual adaptivity is a superior alternative to the aforementioned user models. I claim that the idea behind user models is partly based on a misconception. This is in particular the idea that we can somehow replicate a learner's behavior if only we know her "configuration" well enough. The misconception is formed by the unspoken premise that it is at all possible — at least to a certain degree — to have enough information. This is heavily challenged by the sheer complexity and overabundance of possible influencing factors. This is especially meaningful if we consider learning not only as a cognitive but also as a social process which takes place in institutions and their social settings. There are simply too many variables, which are hard to measure and even hard to identify [97].

On the contrary, the idea of mutual adaptivity immanently appreciates incomplete and uncertain knowledge. We can start from a vague set of assumptions about a learner. But in the long term the adaptive process will result in a converging image. I will now introduce this concept of adaptivity on the basis of the Turing Machine. Doing so I will present a universal design pattern for adaptive systems that is independent from any technology it is built upon.

1.6 Adaptivity with Turing Machines

How do we create adaptive algorithms? In the previous sections we learned that the Turing Machine reads symbols from a virtually endless tape. The combination of an input symbol and the current inner state causes a transition into the next state. Additionally, with a particular transition the machine may print a symbol back onto the tape. A stream of output symbols may represent information for a human or another machine. But it can also be read by the same machine which printed them. In other words: the Turing machine may recycle its own output stream as its input stream. This allows an interesting application: within a transition the machine can print a representation of its internal state on the tape which then is read again as input by the same machine causing another transition. I claim that this simple mechanical back-coupling mechanism can be interpreted as a transient manifestation of primitive machine consciousness. It is transient in so far as the back-coupling process only lasts for a single instance of a state transition. However, when implemented in the form of a cycle this is the only but sufficient way to implement an adaptive algorithm to perform human machine interaction.

First, imagine that the inner state of the machine — which is incorporated by the program and its data — includes a knowledge base about the user and the semantics of the content she is working with. Second, assume that the machine gets information from outside about the user and the environment. This information may be provided by the user's actions or by sensory data. Any input from outside triggers a state transition that also modifies the knowledge base according to the retrieved user input or sensory data. Within such a transition the system may feed its own input with a representation of its new inner state, generating instructional output for the user. This process describes an adaptivity cycle that is geared by user actions and sensory data and it provides a universal definition of an adaptive system based on the model of the Turing machine.

Let my apply this to a "teaching machine" or — more generally — to an adaptive learning environment. An adaptive Turing machine implementing some kind of tutoring functionality should host a knowledge base to embody a representation of learning content and learning behaviors. The knowledge base should be both human-understandable and machine-processable. By being also human-understandable, the knowledge base can be built and understood by a teacher without the need of special technical expertise.

Furthermore, the system must observe the learner. The observation may include the learning material as well as arbitrary parameters measured by the system or observed by the teacher. If relevant actions of the learner or changes of the parameters take place, this has to trigger an update of the internal knowledge base. The machine then has to back-couple itself to its own altered inner knowledge base, perform an inference process on it and finally generate recommendations for the learner. The learner's reaction again becomes an input for the next transition. This is what finally constitutes a mutual interplay between human and machine. Two key questions arise: first, how can we model knowledge about learning content and learners? Second, how can a machine infer on it?

2 Adaptivity in E-Learning

In the previous chapter I have introduced a universal design principle for adaptive systems to build a mutual human-machine interaction cycle. The described method utilizes the capability of a Turing Machine to back-couple on itself within a state transition. To be feasible in a learning environment there must be a knowledge base representing a learner's state and the semantics of the learning content. This knowledge base has to be altered with every action of the learner and according learning recommendations have to be generated for the learner. In this chapter I elaborate in greater detail how such a knowledge base can be constructed and how we can algorithmically draw conclusions from it in order to generate learning recommendations for a specific learner.

2.1 Ontologies

My leading principle is the idea that pedagogical and didactic expertise should be left with human experts like teachers, tutors and creators of learning material. They should define the semantics of the learning material whereas the only responsibility of the computer is inference and the generation of recommendations. This defines a human-machine symbiosis with humans providing the knowledge base and algorithms doing the calculation. This requires a kind of language that is shared and understood by both humans and the machine.

Ontology languages offer excellent possibilities to transform human knowledge into both a human- and machine-readable form. Moreover, formal methods exist to perform logical inference on ontology languages. We can therefore use ontologies for the encoding of human knowledge as well as for deriving verifiable statements from them. In section 2.1.1 (Knowledge Representation) I discuss ontology languages and their application for adaptive e-learning systems. I will refer to the INTUITEL research project (Intelligent Tutorial Interface for technology Enhanced Learning) which demonstrates the utilization of ontology-based descriptions. In section 2.4.3 (Inference Process) I will also explain how an inference procedure on an ontology works on the machine level.

Inferring on ontologies is closely related to the problem of decidability which I discussed in the introductory sections. Actually, an ontology language is a formal system built from axioms and rules. Anything that counts for formal systems and their computability therefore also holds for ontology languages. An ontology language must be carefully defined to be decidable. The INTUITEL project provides a clear example how ontologies can serve to select those objects from an arbitrary set of learning material that is most appropriate for a specific learner and her particular situation. I will show that the result of an inference process is a simple step-by-step set reduction of pieces of learning material.

2.1.1 Knowledge Representation

Semantic web ontology languages like OWL [5, 9] allow us to formulate human knowledge in a way which can be understood by both humans and computers. The INTUITEL system (Intelligent Tutorial Interface for Technology Enhanced Learning) [39, 56] is the first prototype of an adaptive learning environment which uses ontologies as a knowledge and inference basis. Using the OWL language, INTUITEL can infer on didactic and pedagogical meta knowledge. On the one hand INTUITEL is a universal design pattern for adaptive e-learning systems. It provides a sample architecture and network protocols from which producers of e-learning system may derive their own implementations. On the other hand the developers of INTUITEL also implemented a prototype system. They also built plugins for multiple Learning Management Systems like Ilias, Moodle, Crayons, eXact or Clix. These plugins implement the respective protocols and lift the mentioned learning management platforms to adaptive systems [39, 57, 114, 115]. INTUITEL sends textual recommendations of learning units the learner is supposed to process next. The according plugin receives these messages over the network protocols and presents them to the learner within the user interface of the e-learning platform. Figure 2.1 provides an overview of the INTUITEL components and how they interact.

Let me now explain how, in principle, an ontology language is used to represent didactic and pedagogical knowledge and how a reasoning process infers on it. The following example ontology, as illustrated by figure 2.2 describes knowledge about both learning material and learners. It contains the following information:

1. Maria is a student and she has difficulty with the topic "Lorentz Transformation"
2. Exercises A to D deal with “Lorentz Transformation”
3. Exercises A and B are didactically required before C and D
4. Maria prefers video-based learning material
5. Exercise B is video-based

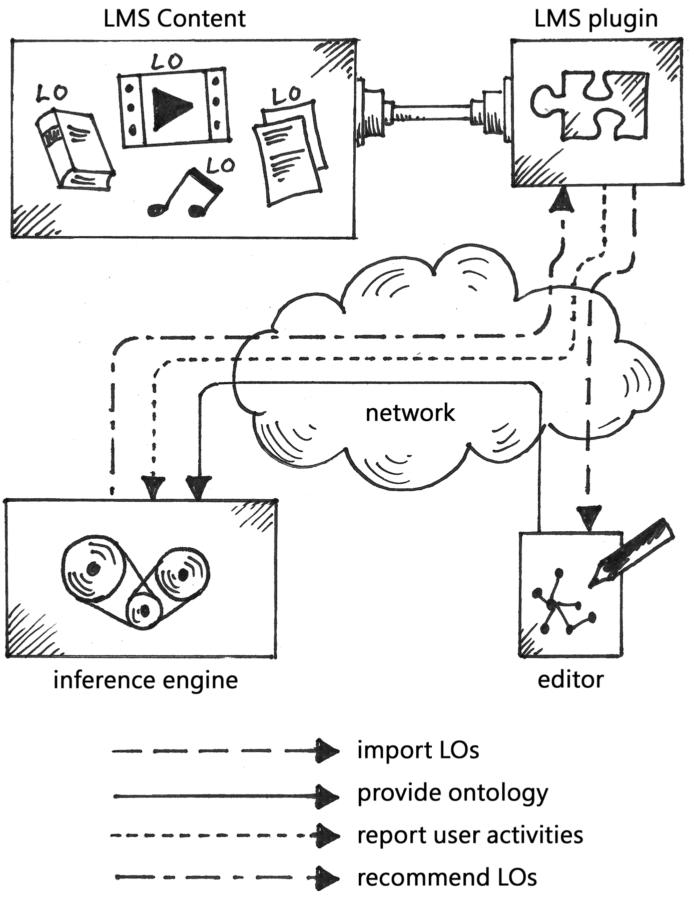


Figure 2.1: Components of the INTUITEL system

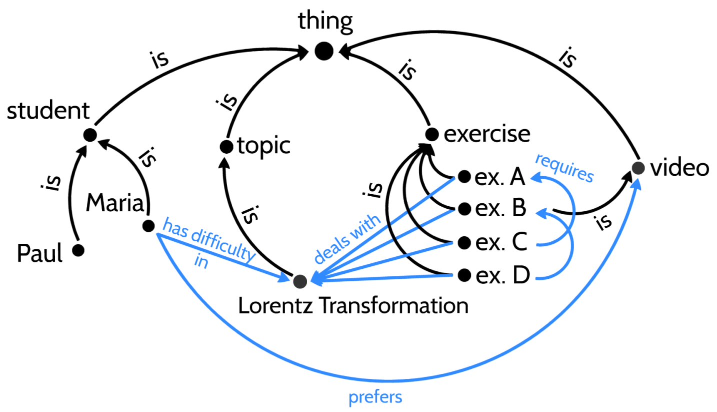


Figure 2.2: example ontology

Rule 1 tells us that for the student Maria all objects may be of interest if they deal with "Lorentz Transformation". Rule 2 tells us that exercises A,B,C an D satisfy this condition. We therefore can reduce the set of learning material to these four exercises. Rule 3 provides another restriction saying that Maria should first proceed with exercise A or B because C and D expand on them. Consequently, in a further step we can reduce the set to exercises A and B. Rule 4 tells us that Maria prefers video-based content. In the last step this rule can be used to even perform a further reduction of the learning content. If video-based content exists among the learning objects we can recommend them with higher priority. Rule 4 finally says that exercise B is video-based. Consequently we can recommend exercise B to Maria.

This procedure describes a step-by step set reduction by following simple axioms and logical rules. With each step the application of a rule reduces the set of learning material by one step further until all rules are applied and a minimal set of appropriate learning objects remains. The process includes the application of knowledge referring to both the learning material and the learner. However, if the set reduction process is supposed to be executed by a computer the according axioms and rules have to be defined very clearly. What we actually need is some kind of vocabulary and grammar specifying which semantic annotations are allowed. There must be a basic set of strictly defined axioms and rules as well as clear instructions how we can build new theorems from them. Otherwise a machine can not work on it. Taking the example ontology above we would have to define entities like “student”, “exercise” or “video” and relations like “deals with”, “has difficulty with” and “prefers”. If this rule set is defined properly a machine can infer on any ontology which uses these annotations.

Formulating such a clear system of machine-processable axioms and rules is exactly the purpose of ontology languages. The aforementioned INTUITEL system is based on a pedagogical ontology which is defined with the Web Ontology Language (OWL) [5, 9]. This ontology is derived from the "Web Didactics" approach which was introduced by Norbert Meder [70, 75, 108, 109]. The Web Didactics approach is the result of a meta analysis of learning material and common patterns of structure and organization. The vocabulary of this ontology organizes learning material by learning units which by scope are comparable to lessons. These units comprise learning objects which are semantically interconnected by their didactic roles.

In such an ontology, the term "learning object" (LO) has a central meaning. A learning object represents an atomic piece of knowledge which may be a book, a single page of a document, a video sequence or a hyperlink to a source from the world wide web. But learning objects may also be composed by smaller ones. For example a book can be regarded as a learning object which again consists of single chapters. In a learning management system a composed learning object may also be represented by a folder or similar container objects. The granularity finally depends on the content editors and the underlying platform. The pedagogical ontology of INTUITEL also provides annotations to define hierarchical and chronological relations between both learning objects and higher-level learning units. This allows for the definition of predefined, recommended learning pathways [55, 124]. Figure 2.3 shows an exemplary structure of an INTUITEL-enhanced course. The diagram shows learning objects and how they are organized into lessons and courses. The arrows indicate semantic relations forming learning pathways.

From the pedagogical ontology one can derive subject-specific ontologies for any knowledge domain. This is what separates INTUITEL from common learning and training software which is often restricted to specific subjects and disciplines. The ontology-based approach allows for encoding human meta-knowledge in a plug-and-play manner. As long as the subject-specific ontology follows the rules of its parent ontology — the pedagogical ontology of INTUITEL — the inference engine of INTUITEL can process it.

In addition to the pedagogical ontology, INTUITEL also includes a learner state ontology which contains knowledge about the learner's characteristics and progress measures. Although this is a separate ontology INTUITEL merges both the pedagogical and the learner state ontology into one. The result of this is an ontology that contains the pedagogical and the learner related knowledge in one combined representation. Remember our example ontology with the student Maria. This ontology contained pedagogical knowledge like exercises dealing with specific topics or being didactically dependent on other exercises. This is analogue to the pedagogical content of the ontology. On the other side the example also describes learner-related knowledge like Maria having difficulty in "Lorentz Transformation" or preferring video-based material. This is the learner-related content which is ana-

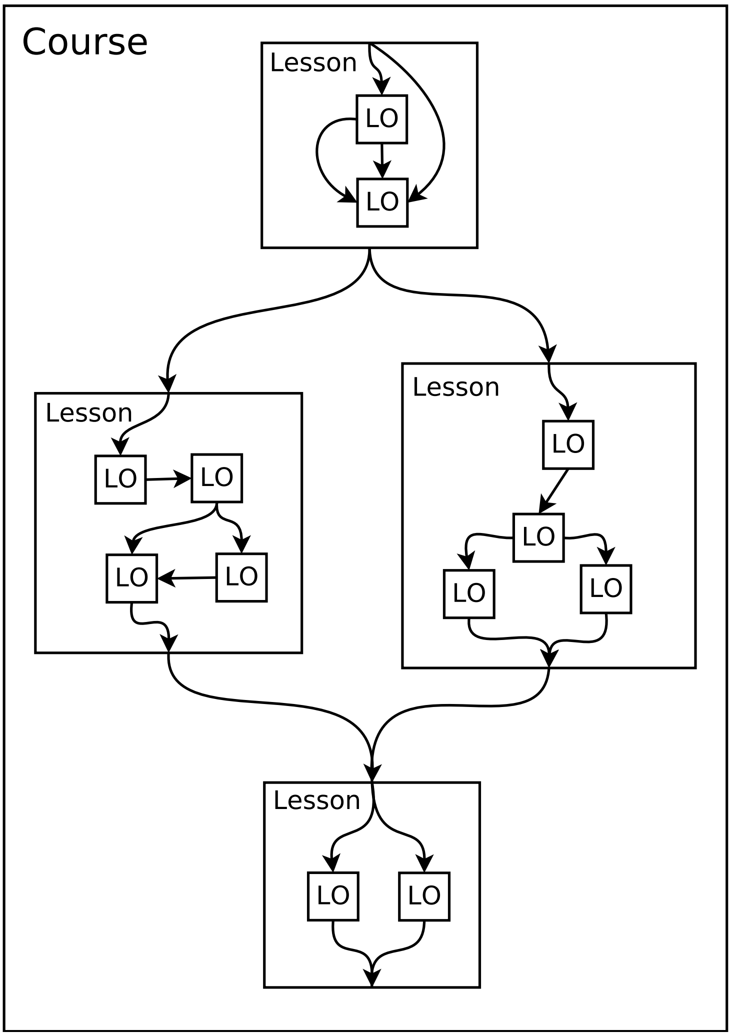


Figure 2.3: Structure of an INTUITEL-enhanced course

logue to the learner state ontology. Note that the example ontology with Maria is just an illustrative example to sketch how in principle an ontology description of learning material and learners works. The original pedagogical ontology and its is explained in detail by the authors of INTUITEL in [39].

2.1.2 Inference Procedures

If an ontology is clearly defined we can have algorithms infer on it. Remember my elaborations on formal systems and decidability in section 1.2 (Computability). An ontology, basically, is a formal system built on axioms and rules from which we can create new theorems. These theorems again can be verified with the same rules the ontology provides. However, in order to have algorithms infer on an ontology it has to be decidable. Remember, that decidability is linked with the expressiveness of a system. If a formal system is too expressive it can not be guaranteed that a theorem is decidable. In other words: a formal system which is too expressive, can not be computationally complete. For this reason, ontology languages have to be restricted by their level of expressiveness. The Web Ontology Language OWL provides different levels of expressiveness: OWL Full, OWL DL and OWL Lite [9]. OWL Full is the most expressive version but it does not provide computational completeness. There can never exist any reasoning procedure for full computation of such an ontology. OWL DL is a restricted version of OWL Full. OWL Lite, finally, is an even more reduced variant. In contrast to OWL Full, both OWL DL and Lite satisfy computational completeness.

Inference on ontologies can be performed by universal semantic reasoners like Pellet [102] and HermiT [60, 101] or by self-developed software modules as it is the case with INTUITEL. At first glance it seems hard to imagine how software can generate logical conclusions from an ontology. Remember that

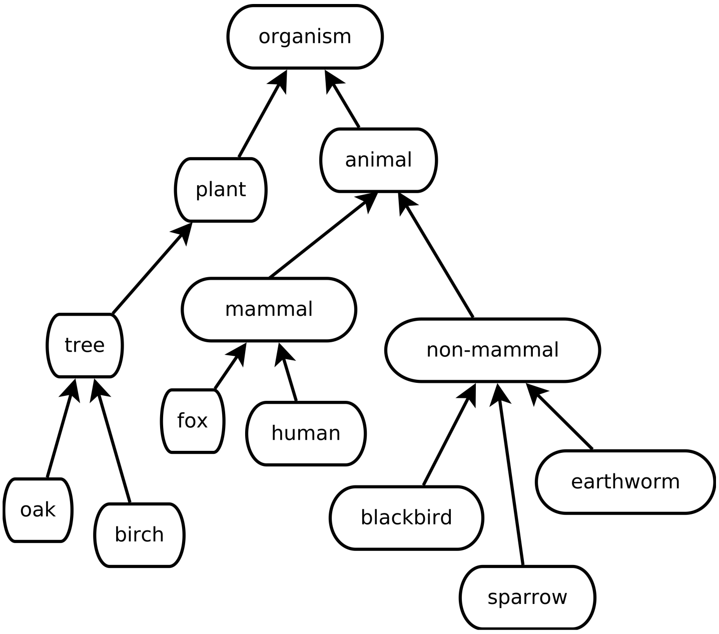


Figure 2.4: example ontology with organisms

finally any algorithm can be represented in terms of the Turing machine. I now provide an extremely simple algorithm for verifying relations as they are present in ontologies. Assume that there is an ontology as shown in figure 2.4. This ontology contains subclass relationships of organisms. For example we can tell from the ontology that a fox is a mammal and a mammal is an animal and not a plant. If a sequence of relations is provided, we can determine if a respective transitive relation is true. Take for example the following sequence of relations:

$$fox \rightarrow mammal \rightarrow animal$$

From this sequence, the transitive conclusion that a fox is an animal can be determined as true:

$$(fox \rightarrow mammal \rightarrow animal) \Longrightarrow (fox \rightarrow animal)$$

Conversely, the conclusion that a fox is a plant can not be verified:

$$(fox \rightarrow mammal \rightarrow animal) \not\Longrightarrow (fox \rightarrow plant)$$

Provided a sequence of relations in the form "something is something else", we may answer questions like "is something something else?". If we know that a fox is a mammal and a mammal is an animal we can positively answer the question "is fox an animal?". I now define a simple algorithm for this task:

First: Assign each element of the ontology a unique symbol in the form of a binary number of fixed length. For example we may write "fox" as the binary number 0001, "mammal" as 0010, "animal" as 0011, "plant" as 0100 and so on.

Second: Formulate the according relations in the ontology as a simple ordered list of symbols. Again take the relations of the fox example:

$$fox\ is\ a\ mammal,\ mammal\ is\ an\ animal$$

This accords to the following simple ordered list of symbols which internally are represented as binary numbers:

$$fox, mammal, mammal, animal$$

Third: Formulate your question "is something something else?". Take the symbol for "something" and place it at the beginning of the ordered list. Take the symbol for "something else" and put it at the end of the list. for example the question "is fox an animal?" results in the expanded list below.

$$fox, fox, mammal, mammal, animal, animal$$

Fourth: any list of this sort has an even number of symbols. Start from the beginning of the list pairing every two consecutive symbols:

$$(fox, fox), (mammal, mammal), (animal, animal)$$

If and only if all paired symbols are equal, the question "is something something else" will be answered with "true".

Remember that the first rule claimed that symbols are encoded as binary symbols of equal length. In section 1.3 (Turing Machine) I defined the "Letter Comparator" — a Turing machine to compare two sequences of binary numbers. I also explained that a Turing machine may call another Turing machine in the sense of a subroutine. We can easily imagine a Turing machine that successively uses the "Letter Comparator" as a subroutine to validate if the pairs of symbols are equal. As soon as any of the subroutines yields "false" the question "is something something else?" is answered with "false". If all subroutines yield true, the overall answer is true.

Of course reasoners for ontology languages are much more complex. But this extremely simple example gives you an idea how we can use simple mechanical procedures to perform logical reasoning. It also clarifies an immanent characteristic of machine-based reasoning: at its core, any reasoner is just a dumb machine comparing numbers. Anything in an ontology — classes, instances and relations — is represented by a number and the machine can only determine if two things are the same or not. The machine is never aware of any meaning. What things mean can only be determined by the human being who defines and interprets the ontology.

Note, that all rules which hold for formal systems in general are also true for ontologies: the ontology must be computationally complete. Moreover, the reasoner can only work on things it knows from the definitions of the ontology. If a new entity or relation occurs which is not part of the definitions, the reasoning process will fail.

2.2 Cognitive Spacetime

In his paper on teaching machines from 1958 [103] Burrhus Skinner already anticipated a basic design principle for the implementation of instructional designs with mechanical procedures in machines and algorithms: He claimed that learning content has to be transformed into some form of linear sequence of atomic learning items. The concept of learning objects in INTUITEL and their alignment along learning pathways closely reflects this principle.

Semantic relations between pieces of learning material — as they can be described by ontologies — create a semantic network with non-linear and ambiguous pathways. In this sense, any instructional design or learning behavior results in transforming these relations into a linear sequence along the time dimension. Taking the picture of a semantic network, the result of an instructional design corresponds to the projection of this network into a linear trajectory. I therefore consider time as a crucial dimension for the characterization of learning behavior. Ontology languages can be used to describe predefined learning pathways that are based on didactic recommendations of a didactic expert. We also can determine a learner's current position and state within such an ontology.

However, ontologies are hardly appropriate for capturing and describing the historicity of learning behavior. While learning analytics has been a popular field, handy models for the temporal description of learning behavior have been missing. A major problem is constituted by the enormous variety of variables and the richness of heterogeneous data sources that are supposed to be taken into consideration for learning analytics tasks.

In section 2.2.1 (Worldlines of Learning) I discuss data that I retrieved from en experimental setup to illustrate how spatial and temporal patterns give us an insight into learning behavior. I then introduce a feature space which is built from sets of meta-data dimensions. Those meta-data dimensions describe features of learning content and learners in an arbitrary learning environment and locate both learning content and learners inside the feature space.

In addition to its ontology approach the INTUITEL project already provided a basic predecessor model of a so-called "Cognitive Space". My feature space is an extensive enhancement of the INTUITEL model. Because time is a crucial dimension in that feature space it enhances the "Cognitive Space" to what I call the "Cognitive Spacetime". Both the Cognitive Space and the Cognitive Spacetime share some similarities. However, the spatial dimensions of the two models are defined entirely different. This is explained in detail in section 2.2.2 (Learning Content Feature Space).

Learners who progress on specific learning content are located at corresponding positions in the Cognitive Spacetime model. Being observed over time, learners' behaviors draw trajectories in that space. Section 2.2.3 (Learning Histories as Spatio-Temporal Trajectories) gives a profound explanation of this whereby section 2.3 (Spatio-Temporal Databases) discusses how data structures and algorithms from the field of spatio-temporal databases can be exploited to analyze, compare and cluster trajectories of different learners.

2.2.1 Worldlines of Learning

For temporal analysis of learning behavior I introduce two types of information: spatial and temporal information. Spatial information refers to classical variables that can be measured in a learning environment or about an individual. These may for example be variables indicating location, device, skill- and progress-levels, frequency of chat and forum usages, grades, scores or meta-data items which describe the learning material. Temporal information refers to time-related patterns and correlations among the spatial data. Together, spatial and temporal data become spatio-temporal information. Suppose a learner's cognitive position is described by n spatial and one temporal dimension. Together, this constitutes a $(n+1)$-dimensional spacetime. When learners progress within that spacetime they draw trajectories which — following Hermann Minkowski's concept of spacetime — we may also call learners worldlines [76, 77].

The principal focus of this work is on the development of a generic spatio-temporal model that can abstract any numeric form of such data to purely geometric trajectories. Imagine learners who progress on similar learning content in a similar way. If their behavior is reduced to only geometric information we can describe similar learning histories only on the basis of spatio-temporal nearness in an abstract space. The spatio-temporal model which I discuss here provides an elegant means to join multiple heterogeneous data sources. Moreover, it reduces the complexity of such data to a plainly geometric problem. This section illustrates by some examples how spatio-temporal information can be manifested in a data set. It provides a basic understanding for the concept of spatio-temporal nearness which is then generalized in section 2.2.2 (Learning Content Feature Space).

The following provides a special visualization of experimental data that was captured from real learning environments. The data was visualized with a special coloring technique. For each data item its global minimum and maximum value was determined. Within that range, the values were associated with colors from the HSL color spectrum running from purple as the lowest to red as the highest value which resulted in a heat map. Similar values could then be identified visually by the similarity of their colors. To achieve a maximum of visual discrimination, for each variable the whole color spectrum was exploited which means that the minimum and maximum values always were assigned the according boundaries of the color spectrum with intermediate values distributed evenly.

The data was stored and visualized with the Hypercube Database which is a prototype of a spatio-temporal database. The database stores such data as geometric objects in the form of multidimensional trajectories which have the shape of hyper-polylines. Accordingly the aforementioned visualization actually represents such trajectories. However, as it becomes difficult to print trajectories with more than three dimensions I decided to use this coloring technique. The Hypercube Database is introduced in chapter 3. To introduce the concept of spatio-temporal nearness, a further understanding of the database is not needed yet.

The first sample includes only a small number of 15 students and two spatial dimensions, but it is sufficient to demonstrate how in principle we can discover cooperation among students using a concept of spatio-temporal nearness. The students edited 47 articles in a Semantic MediaWiki. The logging data of the MediaWiki included only two data records indicating whether a student viewed or edited an article. The articles were grouped by topics and they were enumerated whereby each topic group was given a continuous interval of numbers. These numbers were associated the respective values of the color spectrum. As articles of the same topic group fall into a continuous and narrow interval of numbers they correspond to a subarea in the color spectrum. We can therefore identify students working on articles of the same topic group at a glance.

Figure 2.5 presents an excerpt of the data including seven students. It lists one trajectory for each student covering a time period of three months. The first row of each trajectory contains the normalized histogram of the student's activities which indicates when and how frequently he worked on the MediaWiki platform. The second row shows that the student has viewed an article and the third row shows articles that were edited. Illustration 2.5 reveals three topic groups: bluish, greenish and orange-colored. Trajectory 2 and 4 represent students that joined the course later. Note that the visualizations hold a value for a variable until it is updated by a new event. This means that if a student worked on a specific article and if she is inactive for the following two weeks, the visualization will show the respective color for a period of two weeks.

I stated that the coloring technique is supposed to visualize spatio-temporal nearness for high-dimensional vectors. In this example, there are only two dimensions which is the view and edit variable. Therefore, we can also illustrate the students' activities by real trajectories. Figure 2.6 shows the same data in the form of spatio-temporal poly-lines. These trajectories include the time dimension equally to the spatial dimensions. Following Hermann Minkowski, we may also call them the "worldlines" [76, 77] of learners in Cognitive Spacetime. The view and edit variable represent the spatial dimensions whereas time serves as the third dimension. In this illustration we can identify the same clusters by spatial and temporal nearness. Trajectories 1 and 2 correspond to

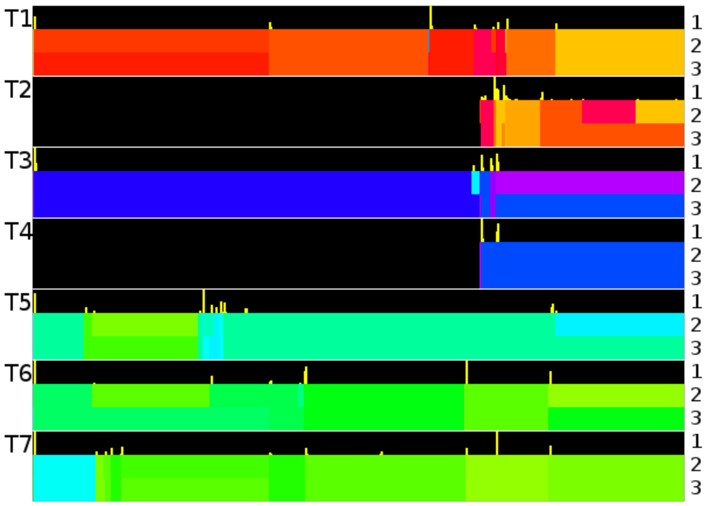


Figure 2.5: cooperating users

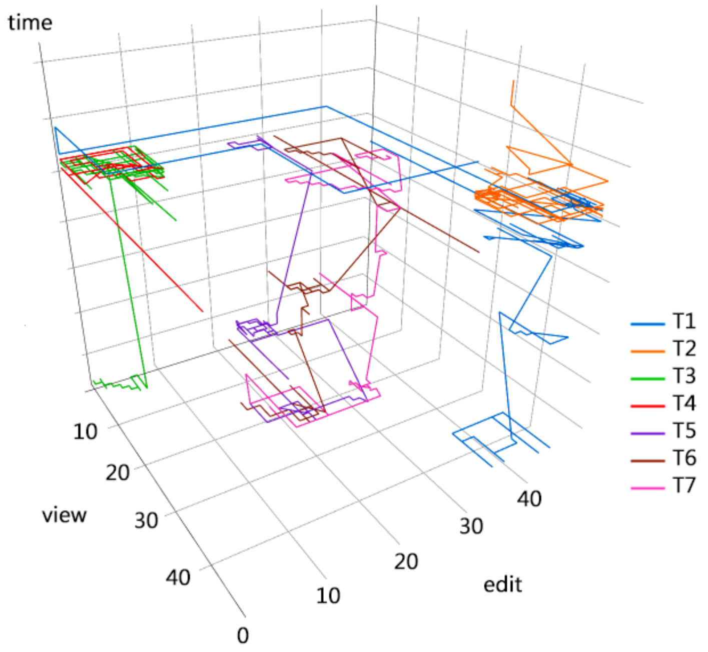


Figure 2.6: cooperating users' trajectories

the orange-colored group in figure 2.5. Trajectories 3 and 4 correspond to the bluish group and trajectories 5, 6 and 7 correspond to the greenish group.

We can also see that trajectories 2 and 4 represent students that joined the course later which is indicated by the fact that their trajectories start lately along the time axis. We furthermore observe in figure 2.6 that the students' activities increase around a certain time point which is also visible in the histograms in figure 2.5. This time point was shortly before the exam period. While some of the students had been lazy during the lecture period they nervously started editing

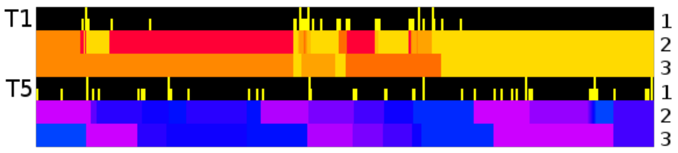


Figure 2.7: cooperating students, 90 minutes period

their articles at the end of the semester. However, both figures reveal activities after the exam period because students were given the chance to correct and improve their articles afterwards to achieve better grades.

The recorded data can be visualized at per-second granularity. Figure 2.7 shows a zoomed excerpt for two students of a lecture with a time period of 90 minutes. Students often edited articles by following the enumerated lists of Wiki pages swapping between viewing and editing mode. This is expresssed by the consecutive gradient color patterns especially in the data of trajectory 5. In the trajectory plot (figure 2.6) this results in stair patterns which can be seen best in trajectory 6 and 7. Both visualizations — the coloring technique and the trajectory plot — show another commonly observed behavior: students did not complete one article by another. instead they edited them partwise returning to articles again and again. With the coloring technique this is obvious because of repeating colors. In the trajectory plot we can identify this behavior by circular patterns in trajectories 1–4.

As this example includes only a small number of students and dimensions it is not representative. But it gives us a clear hint how in principle the concept of spatio-temporal nearness can be used to make common learning behavior visible. Another – more representative example – is data that was retrieved from students' behavior on a video platform: The platform contained lecture recordings for students and served as an additional offer for students supplementing the common lectures. The platform content included 55 different video files and 200 users. It provided logging data over a period of four semesters. The logging data was extracted and processed from the internal log files of the platform and it resulted in the following data items for each user:

- ***system:*** the operating system of the user (Windows, Mac OS, Linux, Android, etc.) — 10 different systems were identified
- ***subnet ip:*** the user's subnet ip according to the net mask 255.0.0.0 — 33 different subnets were recorded
- ***title:*** a unique title identifying a single video file — 55 different titles were included
- ***seen:*** the percentage value of how much of a single video file a user has watched ranging from 0% to 100%

As in the example before, each variable is visualized as a time line representing its values in the form of a color map. The student's activity is again represented as a normalized histogram in the first row of each track. According to the heat map visualization each operating system is represented by one in 10 different colors, each subnet IP by one in 33 colors and each video title by one in 55 colors. The seen-percentage value is represented by a color scale that equates to the spectrum of light running from purple (0%) to red (100%). We can visually characterize student's behavior on the basis of their activity frequency, fluctuations in their variables and time-dependent correlations between variables. In this concrete example, five groups could be determined.

Regular users repeatedly and continuously used the video service during periods of several months. Figure 2.8 visualizes three representative individuals of that group.

Occasional users watched the lecture recordings only occasionally. A visualization example of three occasional learners is illustrated by figure 2.9.

Dense users were inactive during longer time periods but used the video platform intensively during short time periods which indicates that they used the lecture recordings for specific purposes like the preparation for an exam. See figure 2.10 where the magnified area represents a time period of 10 days.

Mobile users were indicated by a fluctuation of the "system" and "subnetip" variable. Mobile learners were mostly present among the group of regular users. A visualization of three representative mobile users is shown in figure 2.11.

Stationary users rarely or never changed their location or operating system. For example some dense users showed stationary behavior (see the first dense user in figure 2.10). One reason might be that when students prepare for exams they often do this in a ritualized way to discipline themselves which involves learning at particular times in a particular place like in a library or at home. A visualization of three representative stationary users is shown in figure 2.12.

Note that the visualizations hold a value for a variable until it is updated by a new event. This means that if a student watches for example 25% of a video and if she is inactive for the following two weeks, the visualization will show the respective color for a period of two weeks.

The visualizations reveal two types of patterns: spatial and temporal ones. Temporal patterns are constituted by periods and frequencies of activity and the change rate of variables. Spatial patterns refer to measured variables which in these examples are subnet ip, seen percentage, operating system, video title and activities like viewing or editing a document. Spatial and temporal information together form spatio-temporal patterns.

In the above examples spatio-temporal nearness might be understood as similar wiki articles, the same subnet ip or the same operating system of a user within the same time period. The visualizations encode spatial nearness with similar colors which is most obvious in the case of cooperating students on the MediaWiki platform. In terms of space the MediaWiki example constitutes a two-dimensional vector with the "view" and "edit" variable as its components. In the video platform example this vector has four dimensions composed by the variables "system", "subnet ip", "title" and "seen". We can imagine an abstract space in which these vectors draw trajectories as they evolve over time just the same way as they do in figure 2.6. This provides a first insight about how arbitrary variables can be transformed into a spatio-temporal model. But how can we utilize such nearness for learning environments and what kind of analysis can be useful?

In the case of the MediaWiki platform we obtained a spatial model by assigning same-topic articles to corresponding intervals of numbers which located same-topic articles near to each other in the space of the database. However, the variables of the video platform are hardly useful to operate with spatio-temporal nearness. For example the subnet ip variable alone is only a number and two subnet IPs can not tell us much about "nearness". However, using subnet ips we could approximately resolve geographic locations which would be a more useful application. Equally, the video title is useless as long as we do not have any further information. As long as we only use the raw data items we can only say if two learners watch the same video or not or if they are in the same subnet or not. Consequently, we first need a more sophisticated concept to define similar objects in a learning environment. Second we need to define how these similarities have to be transformed into a geometric model so we can utilize it for trajectory analysis.

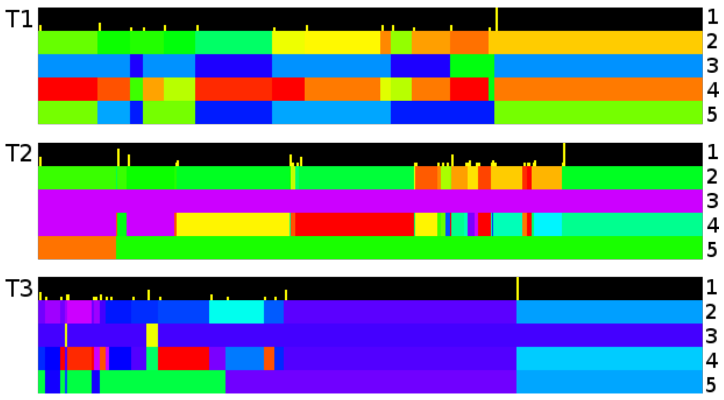


Figure 2.8: regular users with histogram (1), video title (2), system (3), seen percentage (4), subnet ip (5)

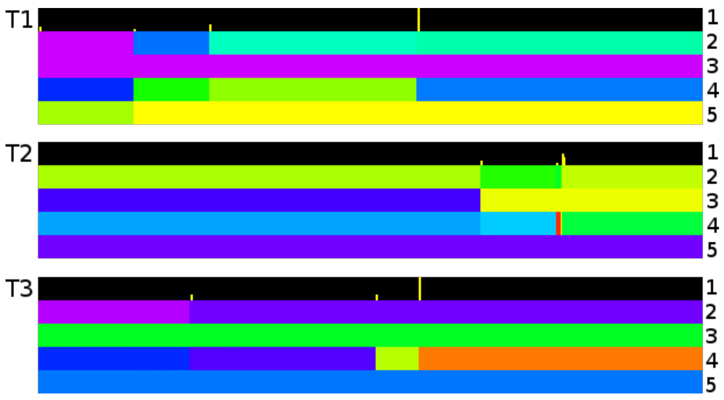


Figure 2.9: occasional users with histogram (1), video title (2), system (3), seen percentage (4), subnet ip (5)

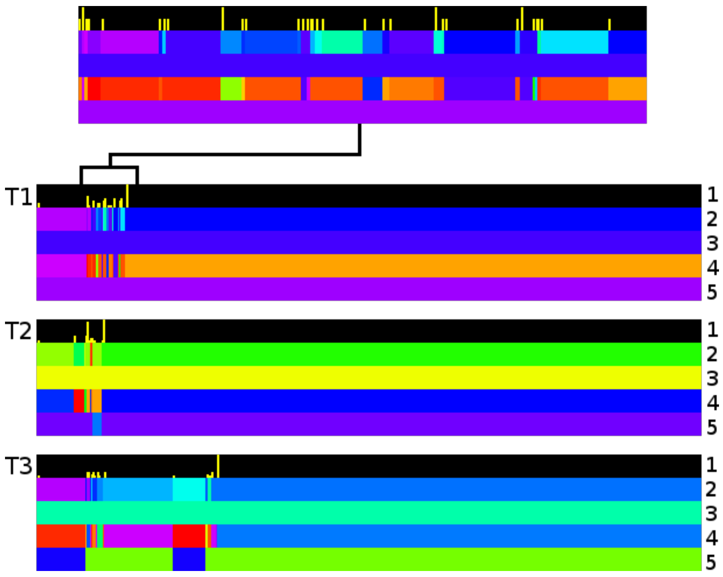


Figure 2.10: dense users with histogram (1), video title (2), system (3), seen percentage (4), subnet ip (5)

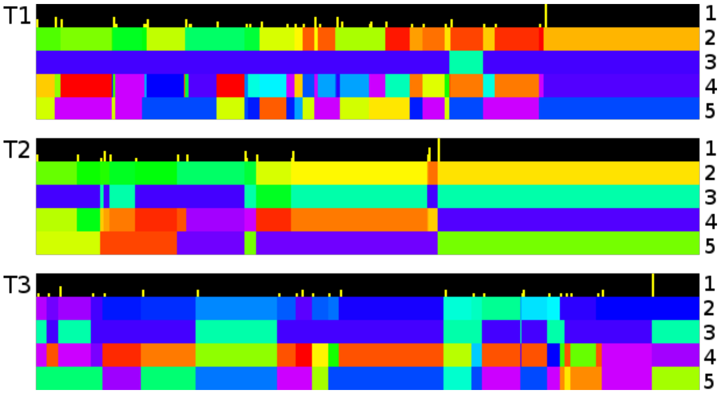


Figure 2.11: mobile users with histogram (1), video title (2), system (3), seen percentage (4), subnet ip (5)

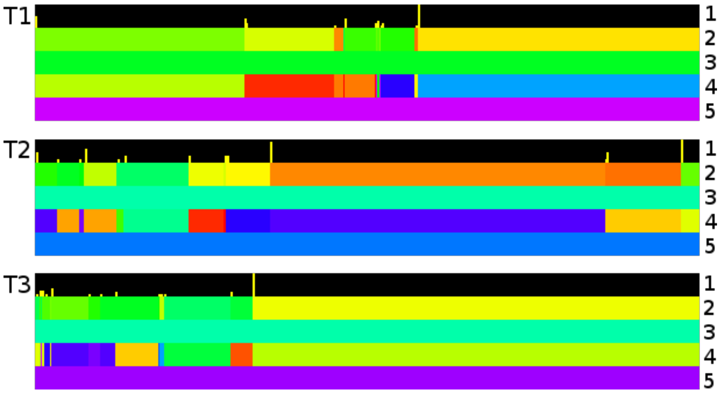


Figure 2.12: stationary users with histogram (1), video title (2), system (3), seen percentage (4), subnet ip (5)

2.2.2 Learning Content Feature Space

In section 2.1.1 (Knowledge Representation) I introduced the INTUITEL project and its ontology-based approach to represent didactic and pedagogical meta knowledge. One aspect about INTUITEL has not been mentioned so far: the INTUITEL system also implemented a basic spatial model of individual learning pathways. this concept is called the Hypercube Model [41] and it is the preliminary version of my enhanced "Cognitive Spacetime" model. It is based on a k-dimensional feature space that describes atomic learning objects (LO) by k meta-data items. Before elaborating on that feature space I will describe the original Hypercube Model of the INTUITEL project. Then I will explain in detail how the Hypercube Model is altered and enhanced to form the feature space of the Cognitive Spacetime.

Hypercube Model

The Hypercube model of INTUITEL describes a n-dimensional space with n denoting the number of atomic learning objects in a learning environment. The term "learning environment" may refer to any form of digital or even analogue collection of learning material that is related to a specific topic or knowledge domain. A Learning Management System or a Wiki platform are just two concrete examples of learning environments. The concept is not limited to specific technologies.

Imagine a collection of n atomic learning objects. At any time we may determine how much a learner has progressed on specific objects in that collection. In the Hypercube Model this is expressed for each learning object by a numeric value from the interval $[0, 1]$. At the very beginning all objects will be valued 0 for a particular learner. The more the learner proceeds in the learning environment, the more learning objects will be assigned higher values. Taking all n learning objects together we obtain n intervals. These intervals are applied as coordinate axises in a n-dimensional hyperspace. As these dimensions are restrained to the interval $[0,1]$ we obtain a hypercube. A learner's position in this space is expressed by a vector $L = (l_1, ..., l_n)$. This vector is also called the "cognitive position" of the learner and the entirety of the space inside the hypercube is called the "cognitive space" [40, 56].

Just as an aside, let me mention that the INTUITEL hypercube in principle can be regarded as a fuzzy hypercube for which the vectors inside are fuzzy sets [41, 68, 72, 98]. Figure 2.13 illustrates a hypercube with $n = 3$ learning objects whereas figure 2.14 illustrates a hypercube with $n = 4$. Each edge represents the interval $[0,1]$ for a knowledge object a learner may attend to.

Suppose that a learner proceeds and progresses within the learning environment. Then the according cognitive position vector evolves over time. The result of this movement is a trajectory in the n-dimensional space of the hypercube. If a learner works on the knowledge objects in quite a disciplined manner — finishing every learning object before starting the next object — this results in straight-line pathways that are strictly aligned along the edges of the hypercube. In contrast, The learning pathway of a less disciplined learner would result in a more volatile pathway which is located inside the hypercube [106]. This is illustrated by the 3D-hypercube in figure 2.15 where the bold edges represent the disciplined learner and the dotted trajectory represents a volatile pathway.

In section 2.1.1 (Knowledge Representation) I explained how ontology descriptions are used to define recommended learning pathways. However, the pathways which I discuss here are user-specific pathways that are defined by the behavior of an individual user. In order to recommend learning objects to a learner INTUITEL infers on its pedagogical ontol-

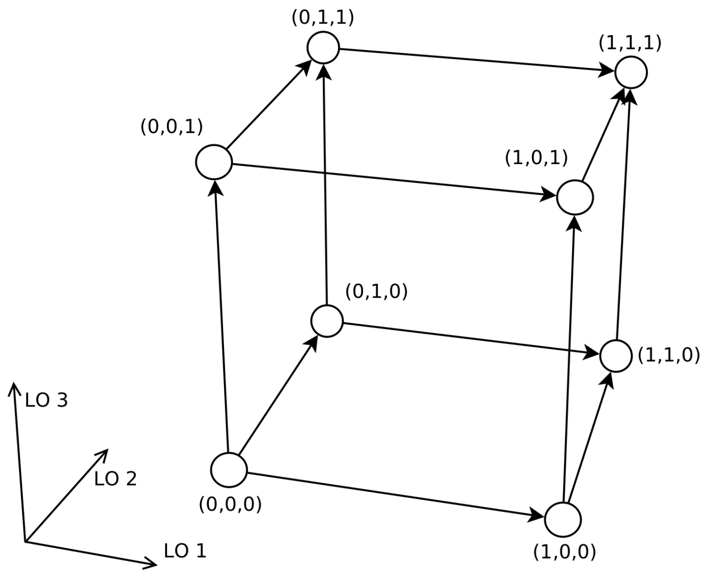


Figure 2.13: 3d hypercube for $n = 3$ learning objects

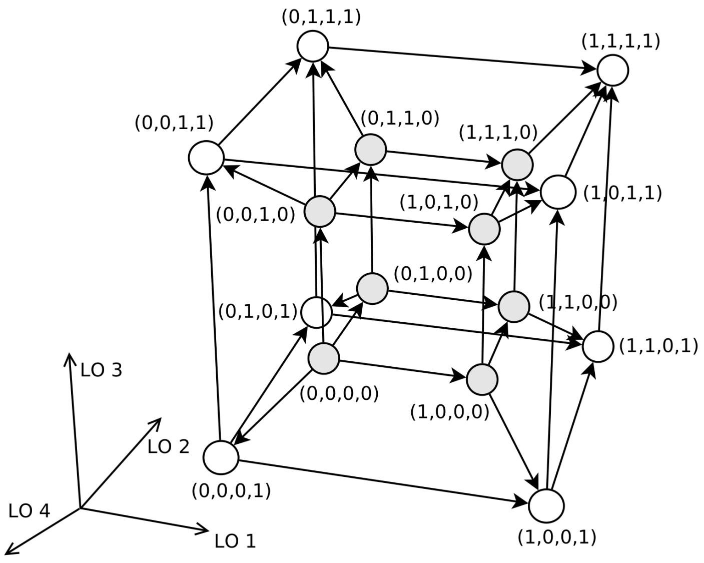


Figure 2.14: 4d hypercube for $n = 4$ learning objects

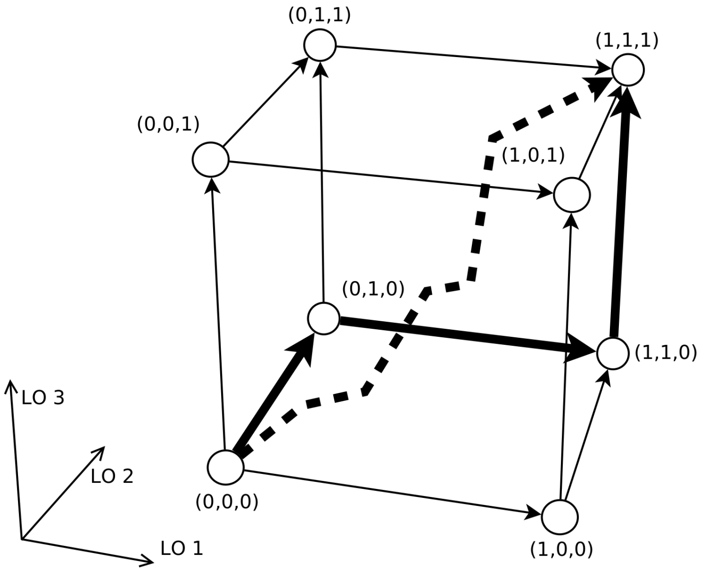


Figure 2.15: 3d hypercube for $n = 3$ learning objects with pathways

ogy. This inference procedure may provide multiple predefined learning learning pathways. INTUITEL uses the Hypercube paradigm to compare these predefined pathways to the user-specific pathway. Then the INTUITEL system tries to find the predefined pathway that is most similar to the user-specific pathway.

The INTUITEL algorithm for pathway selection is based on a simple vector comparison which includes the learning objects a learner has worked on and their chronological order on the pathway. Again, let n be the number of learning objects in a course. Then the learner's cognitive position is represented by the n-dimensional vector $P = \{x_i\}$ with $i = 1, \ldots, N$ and $x_i \in [0, 1]$. x_i represents the degree of progress with respect to the i^{th} learning object.

The value of x_i may be determined in different ways: It may denote the learner's degree of progress for the respective learning object as percentage value. It may be concluded from a test item to review the learner's progress. Given the most simple case, x_i only contains the information if a learner has attended to a LO or not. The latter case would have x_i degenerate to $x_i = \{0; 1\}$. At the start, when a learner has made only little progress, the vector P will point close to the origin of the hypercube. The more the learner progresses, the closer P will converge to the theoretically maximum position which equates to the vector $P_e = \{x_i | \forall x_i = 1\}$. P_e implies that the learner has progressed every single learning object in the course with 100% success.

To compare a learner's individual cognitive position with a predefined learning pathway the developers of INTUITEL introduce two vectors: LP represents the learner's progress values of all learning objects in the course reflecting the chronological order on the predefined learning pathway. In other words: LP takes the progress values of the learner and arranges them in the order he was supposed to learn. The second vector is denoted by ULP which is an abbreviation for *User Learning Pathway.* ULP contains exactly the same progress values as the vector LP except for one difference: the sequence of the values reflects the chronological order in which the learner actually did progress the learning objects. In this sense the user specific cognitive position vector ULP is just a permutation of the vector LP. The latter one represents the cognitive position as it should be when the learner follows exactly the predefined pathway.

Let me give an explanation which is slight variation of an example in [41]: Suppose that there is a small course with $n = 10$ learning objects $\{LO_1, LO_2, \dots LO_{10}\}$. A predefined learning pathway includes a subset of six learning objects in a particular order:

$$LP \Rightarrow (LO_3, LO_1, LO_5, LO_4, LO_8, LO_6)$$

If the learner first processes LO_1 completely, second LO_4 completely and third LO_5 to a degree of 60% his vectors LP and ULP correspond to the following:

$$ULP = \{x_1, x_4, x_5, x_8, x_3, x_6\} = \{1, 1, 0.6, 0, 0, 0\}$$

$$LP = \{x_3, x_1, x_5, x_4, x_8, x_6\} = \{0, 1, 0.6, 1, 0, 0\}$$

Through the computation of a distance measure between these two vectors, INTUITEL selects the LP that fits most the current cognitive position of the learner. If the inference process on the pedagogical ontology retrieves multiple learning pathways this measure is used to select the learning pathway that fits most the current cognitive position of the learner [106].

It must be stressed that the vector LP and its permutation ULP are snapshots of the learner's current cognitive position.

Indeed it reflects the chronological ordering of attended learning objects. But it does not yet consider temporal aspects and historicity as I have sketched in section 2.2.1 (Worldlines of Learning) because it does not explicitly include a temporal dimension. Furthermore, although the Hypercube model does already anticipate a preliminary form of learning trajectories, it does not yet include a concept describing similarities of learning objects. It is therefore still far away from a real feature space with which we may implement spatio-temporal relations.

Advanced Hypercube Model

In the following I describe how the previously explained Hypercube is enhanced to a real spatio-temporal feature space. The original Hypercube Model, as it was developed by the INTUITEL researchers, maps a progress measure from the interval $[0, 1]$ to a dedicated dimension for each learning object. This hypercube does not contain information about the similarity of learning objects and it does not explicitly include the time dimension. Therefore, the Hypercube Model is extended as follows: The dimensions do no longer map only progress measures for learning objects. Instead, they represent values of arbitrary meta-data items. These values do not need to be restricted to the interval $[0, 1]$, they may have any value that is numeric. Consequently, the hypercube becomes a hyper-rectangular, which in case of optional normalization may degenerate again to a hypercube. Finally time is added as a special dimension.

In this feature space learning objects are located on the basis of their meta-data descriptions. This allows us to find similarities by spatio-temporal nearness. Learners draw trajectories in this feature space as they progress along the learning content. In addition to LO-specific meta-data the feature space may also contain learner-related information. A learner's trajectory therefore is composed by three types of meta-data:

- ***LO properties:*** a number of n meta data items which describe only properties of the learning objects
- ***learner properties:***a number of o meta data items describing only properties of the learner
- ***learner/LO properties:***a number of p meta data items describing learner-specific properties with regard to learning objects

In this form the feature space contains $(n+o+p)$-dimensional data which do not necessarily describe learning content alone. Instead the feature space describes learners and their position within that $(n+o+p)$-dimensional space. Accordingly a learner can be understood as a time-evolving vector:

$$V = (\underbrace{v_1, \ldots, v_n}_{\text{LO prop.}}, \underbrace{v_{n+1}, \ldots, v_{n+o}}_{\text{learner prop.}}, \underbrace{v_{n+o+1}, \ldots, v_{n+o+p}}_{\text{learner/LO prop.}})$$

A preliminary question arises: what kind of meta-data is most appropriate to model spatio-temporal similarity measures? In the course of previous considerations I mentioned that there are data items which are less suitable and other items that fit better into a spatio-temporal model. There are some basic requirements to identify appropriate meta-data descriptions. Firstly, such data items should be representable in a numeric form that allows for discrimination and ordering of values. A negative example was given by the "subnet ip" and the "title" variable in the video platform data which I discussed in section 2.2.1 (Worldlines of Learning).

The dimensions of the feature space should be formed by items which have an ordinal scale or higher. Nominal scales like a subnet ip or a video title can represent spatial information only in the sense of “is the same” or “is not the same”. On the contrary all items which have an ordinal scale or higher may be potential candidates. This does not mean that data with nominal scales can not be used. Indeed, they may provide useful information. But if the feature space is largely or even only constructed with nominal data a spatio-temporal model like the Cognitive Spacetime will not make much sense. For such cases, classical methods may be a better choice. The listing below provides an example of appropriate meta-data items:

LO properties

1. suitability for beginners on a scale from 0 to 5 points
2. suitability for advanced on a scale from 0 to 5 points
3. suitability for profession group X on a scale from 0 to 5 points
4. accessibility for people with handicaps on a scale from 0 to 7 points
5. usage of simple language on a scale from 0 to 5 points
6. portion of video content from 0% to 100%
7. portion of text content from 0% to 100%

learner properties

1. usage of social plugins on the platform (chats, forums, etc.) in activities per week
2. bandwidth of the learner’s internet connection in MBit/s
3. biometric sensor data like eye tracking to determine the learner’s state of arousal and concentration

4. overall knowledge about the course evaluated regularly as a score value from a test
5. geographic location
    - longitute of GPS coordinates
    - latitute of GPS coordinates
6. Big Five personality traits
    - score on the "openness" dimension
    - score on the "extraversion" dimension
    - score on the "agreeableness" dimension
    - score on the "conscientiousness" dimension
    - score on the "neuroticism" dimension
7. Kolb Learning Style Inventory
    - score on the "Concrete Experience" dimension
    - score on the "Reflective Observation" dimension
    - score on the "Abstract Conceptualization" dimension
    - score on the "Active Experimentation" dimension

learner/LO properties

1. completion level from 0% to 100%
2. test score from 1 to 10 points if the learning object includes a test
3. grade from 1 to 6 if the learning object is associated with a grade

These examples also include variables like Kolb learning styles [66, 67, 73] or personality traits like the "Big Five" model [44] which will probably be static over long periods. Such data can be included in the Cognitive Spacetime. But — similar to nominal data — the great advantages of the Cognitive Spacetime will not pay off if the model largely or only consists of static dimensions. But they can provide useful applications if they are used in combination with time-changing data.

The learning styles and the personality traits illustrate another interesting aspect of the Cognitive Spacetime: many instruments like personality tests perform their classifications on the basis of test batteries which first provide their results in the form of scores along specific dimensions. The classifications of these tests are then derived from these scores. The Cognitive Spacetime allows for integration of the original score dimensions instead of using the classification output.

Let us suppose that a learning environment like a Wiki or a Learning Management System contains a set of learning objects. The learning objects may even be distributed over multiple heterogeneous systems. Such platforms usually provide functionality to annotate learning content with arbitrary meta-data. We can annotate the respective learning material according to the examples above. Provided that the concerned platform lets us record the learning objects that are accessed by learners, we can associate learners' access records with the according meta-data. Essentially, this also works if the platform does not provide meta-data annotation functionality. The meta-data may also be kept separately from the platform and it may even be associated with the learning content retrospectively. The only important information that is needed is what learning objects were accessed and when they were accessed. Together with an appropriate meta-data description we can transfer this conflated information into the advanced Hypercube Model which then constitutes the feature space.

2.2.3 Learning Histories as Spatio-Temporal Trajectories

In the feature space learning objects are spatially closer to each other the more similar meta-data items they share. Being observed over time, learners leave traces in the feature space as they attend to different learning objects over time. These traces can be modeled as trajectories which are hyper-polylines. If two learners work on familiar learning objects and if they progress on them in a similar manner their trajectories will be spatially and temporally close to each other. This is how the aspect of historicity is modeled by the feature space. In the original Hypercube Model of the INTUITEL approach this space was called the "Cognitive Space" and the learner's current position in it was called her "Cognitive Position". In the advanced version of the hypercube, time is integrated as a crucial dimension. I hence call it the "Cognitive Spacetime".

Let me now picture a simple example for a learner drawing a trajectory in a feature space. Imagine a three-dimensional feature space built from the following meta-data items:

1. suitability for profession group X on a scale from 0 to 5 points (LO property)
2. bandwidth of the learner's internet connection in MBit/s (learner property)
3. completion level from 0% to 100% (learner/LO property)

Let us denote the first item with x_1, the second item with x_2 and the third one with x_3. A learner's position in this feature space is defined by a time-dependent three-dimensional vector $v(t) = (x_1(t), x_2(t), x_3(t))$. The learning environment shall contain N learning objects each denoted by LO_n. Let us assume that the learning environment records six consecutive events of the learner at six different points in time:

At time t_1 the learner starts working on the learning object LO_1. LO_1 is rated as suitable for the learner's profession group by 4 points and the learner has not yet made any progress on the learning object. The learner is working on a desktop system with a bandwidth of 3MBit/s.

At time t_2 the learner is still working on LO_1 but she has completed it to a degree of 60%. Her bandwidth has dropped to 2MBit/s.

At time t_3 the learner starts working on LO_3, bandwidth is at 3MBit/s and LO_3 is suitable for her profession group with 1 point.

At time t_4 she stops working on LO_3, completion level of LO_3 is only 10%, bandwidth has not changed.

At time t_5 she has returned to LO_1 and completes it to 100%. She is now working on a mobile device and bandwidth has dropped to 0.5MBit/s

At time t_6 the learner proceeds with LO_2, completion level starts at 0% and bandwidth is still at 0.5MBit/s. LO_2 is suitable for the learner's profession group with 3 points.

This activity sequence results in the following vectors:

$$v(t_1) = \begin{pmatrix} 4.0 \\ 3.0 \\ 0.0 \end{pmatrix} \quad v(t_2) = \begin{pmatrix} 4.0 \\ 2.0 \\ 0.6 \end{pmatrix} \quad v(t_3) = \begin{pmatrix} 1.0 \\ 3.0 \\ 0.0 \end{pmatrix}$$

$$v(t_4) = \begin{pmatrix} 1.0 \\ 3.0 \\ 0.1 \end{pmatrix} \quad v(t_5) = \begin{pmatrix} 4.0 \\ 0.5 \\ 1.0 \end{pmatrix} \quad v(t_6) = \begin{pmatrix} 3.0 \\ 0.5 \\ 0.0 \end{pmatrix}$$

This basic and very hypothetical example illustrates how — in principle — learner activities are transformed into spatio-temporal trajectories and how they are represented in the Cognitive Spacetime model. Figure 2.16 illustrates two exemplary

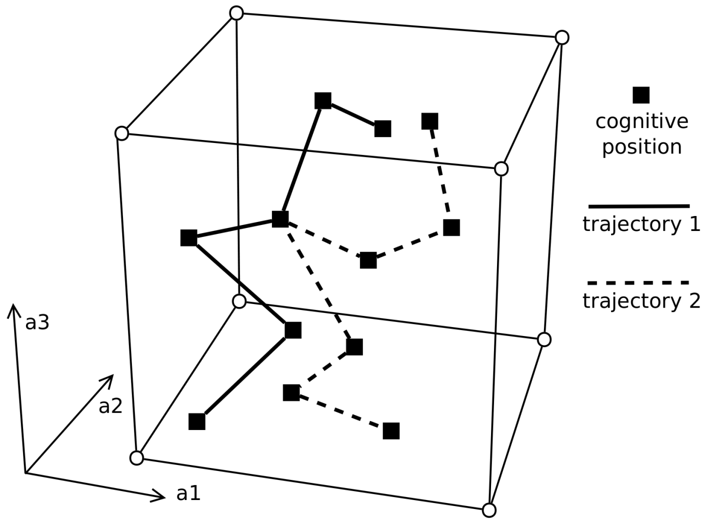


Figure 2.16: learning trajectories in a space of $k = 3$ meta-data dimensions (a_1, a_2, a_3)

trajectories in a Cognitive Spacetime which consists of three spatial dimensions. Similar learning trajectories can be identified by calculating a distance measure for them. In the most intuitive case this is the average euclidean distance between two trajectories. For example we can search for a number of trajectories that are closest to a particular trajectory. Moreover, we can partition the Cognitive Spacetime into logical subspaces which we can utilize for certain analysis strategies. A good method in this sense is to look for correlations between subspaces. In other words: we can determine if spatial nearness in a subspace correlates to spatial nearness in another subspace.

Subspace Partitions

I have outlined that there are three different types of meta-data: those which only describe properties of the learning objects (LO properties), those which describe characteristics of learners (learner properties) and those describing learners' attributes with respect to specific learning objects (learner/LO properties). These three types can be considered as subspaces in the feature space. We can therefore look at the Cognitive Spacetime in different ways obtaining different meanings. We can either look at the Cognitive Spacetime as a whole including all dimensions or we can work with projections reflecting only the LO properties, the learner properties or the learner/LO properties.

The subspace of LO properties primarily describes properties related to learning content. However, in a mediate way they provide information about a learner. Let us think of an advanced learner belonging to a particular profession group who prefers text-based content. If the learner is free to choose from the learning material her preferences will probably converge to content that is annotated with the according meta-data. In the previous paragraphs I explained the concept of a learner's cognitive position. Although LO properties refer to learning objects they should also be considered as being related to the learner because it represents her position in Cognitive Spacetime.

In contrast to LO properties the subspace of learner properties clearly describes traits and characteristics that are related only to the learner. Finally, The subspace of learner/LO properties bridges both learner and LO related properties.

The outlined division into subspaces is actually a question of definition and interpretation. Technically all dimensions exist in one space whereas subspaces are only obtained by partial projection. We can of course create any form of individual subspaces with any meaning. For example the big five personality traits and the Kolb learning styles may be regarded as a "personality" or "cognition" subspace. Information about the bandwidth of a learner's internet connection could be subsumed under a "technical" subspace. It depends on the analysis strategy which subspaces are useful. Anyway, correlations of subspaces can yield interesting information: if phenomena in one subspace are related to phenomena in another subspace this results in simultaneously occurring spatial nearness in both subspaces. This impacts directly the results of nearest neighbor searches. If two subspaces strongly correlate, a nearest neighbor search will provide similar results in both subspaces.

Learning Pathway Modeling

We can utilize cognitive position vectors to model hypothetical learners and their equally hypothetical movements through the Cognitive Spacetime. We can either use this to generate learning recommendations or to validate ideas of learning behavior. For this purpose we model a hypothetical learner and her behavior by a time-related sequence of cognitive position vectors with hypothetical values on their Cognitive Spacetime dimensions.

The cognitive positions of the hypothetical learner may be defined by real representative learning objects or by hypothetical learning objects with some particular properties. Such a trajectory can be used for two purposes: We could use it as a reference trajectory in order to recommend learning objects to a learner who is close to this trajectory. This may be useful for teachers if they want to recommend default learning pathways to learners. Alternatively we can understand the trajectory as an assumption about a specific type of learner and we can find out if learners who are similar to that type really exist in a data set of real learners.

Learning Recommendation Scenarios

With the help of the Cognitive Spacetime we could consider some interesting questions. A possible scenario may be the following: suppose, we track learners locations. We could do this by resolving their IP to geographic locations and networks. Furthermore, let us observe how frequently they use chats and forums. We may now determine if learners with certain traits do more often socialize with other learners either by meeting in the same location or using chat and forum functionality more frequently. We could also determine if such behavior correlates with certain types of learning objects and achievements. Such perceptions may induce individual recommendations for learners.

For example an intelligent learning environment can predict a learner's behavior by finding similar trajectories of other learners and recommend appropriate learning material. We could also identify potential learning partners and groups who have similar traits and trajectories in the Cognitive Spacetime. Such recommendations can also consider a learner's current physical location preferring those potential partners who are most similar in the subspace of geographic coordinates. Potential partners could then be recommended to meet either physically or online. This example shows that we can use the Cognitive Spacetime model to improve learning with regard to not only cognitive but also social processes and features.

Respective recommendations may either be generated by a software system or they may be made by a human tutor analyzing the data. The INTUITEL system which I described in section provides a telling example how textual recommendations can be integrated seamlessly into existing learning environments (see section 2.1.1). The developers of INTUITEL implemented plugins for common Learning Management Systems like Ilias, Moodle, Crayons and Clix [39, 41, 114, 115]. These plugins use a network-based protocol to communicate with an external engine that generates the recommendations. Interestingly, it is not overly important if the recommendations are generated algorithmically or if they are conceived by a human. In the INTUITEL system, recommendations are calculated by an inference engine in real-time. Instead we can also think of a system that allows teachers to first analyze learning trajectories with a software implementation of the Cognitive Spacetime. Then the teacher may formulate recommendations for individual learners in a software-based recommendation system or simply by mail or — even more simple — in a face-to-face situation. If the teacher uses a software-based recommendation approach, the system may store these recommendations and present them to the learner as soon as she is online again. Let me give two examples for such recommendations:

> Dear learner, I have found that other learners proceeded with the following course material. . . This learning material may help you.

> Dear learner, I have found that two other learners of your course have been working on the same topic and exercises during the last three weeks. They also are often at the same locations. Would you like to meet them, so you can learn together?

Both recommendations can be made on the basis of nearest neighbor searches in the Cognitive Spacetime model. The first example searches for nearest neighbor trajectories considering similar learning content. The second example is also based on a nearest neighbor search. But additionally, it considers dimensions of the Cognitive Spacetime which are associated with the learner's location. In both cases the recommendation offers learning objects to the learner which are contained by the trajectories of the other learners. This may also consider the chronological order in which learners proceed with learning material. This way we can predict learning pathways of learners by looking at similar pathways of other learners in the past and the way they proceeded from the current cognitive position of the learner in question.

As already outlined, recommendations of this type can be made by humans but we can imagine that they can also be transformed into algorithms. We can even think of a hybrid system providing both automatically generated and human-given recommendations. An even more powerful hybrid solution may be a platform that allows human teachers to define their own procedures for automated recommendations. Figure 2.17 summarizes possible architectures.

In the end this facilitates a form of human machine symbiosis which is shaped by a triad. The triad consists of the learner, the teacher and the machine. The machine serves as an analyzing and executing instance. The teacher uses it to find similarities between learners and instructs the machine to instruct the learner. In case of the Cognitive Spacetime it is also the teacher who defines the dimensions of the model. The teacher owns the didactic expertise and he determines both the similarity measures and the recommendations that result from such similarities. Finally, the machine acts as an intermediate instance between the teacher and the learner.

2.3 Spatio-Temporal Databases

In the previous section I described how learners and their personal traits, characteristics, individual parameters as well as their learning contents are transformed into spatio-temporal trajectories in the Cognitive Spacetime model. The key idea

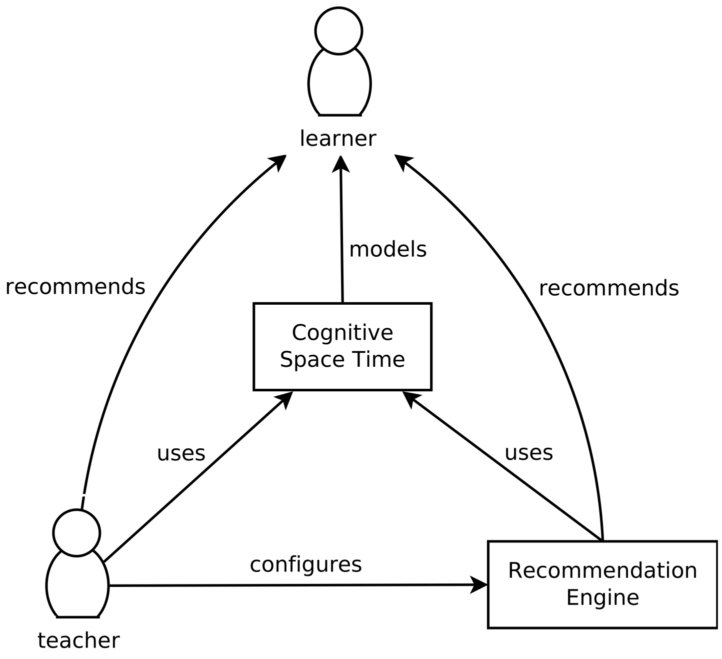


Figure 2.17: Application scenarios for the Cognitive Space-time with teachers and learners

behind the Cognitive Spacetime is born from the following assumption: once we obtain trajectories as purely geometric objects in a spatio-temporal representation, we can make use of a large pool of algorithms and data structures that have been developed in the field of spatio-temporal databases (STDB). A major achievement of the work which I present here is based on the fact that problems related to learning analytics can now be accessed with methods of the STDB domain. Moreover, this addresses a central problem in learning analytics: namely the complexity and heterogeneity of variables which was often seen as an obstacle that is difficult to overcome. Because all data is abstracted to geometric structures and relations, the Cognitive Spacetime allows us to combine data from various kinds and sources. Similarity measures are based only on spatial and temporal relations.

Let me stress that existing data structures and algorithms for STDB are mostly motivated from the area of geo-informational Systems (GIS) which by nature deal with real-world moving objects like vehicles, humans, or weather phenomena. These objects are located in a 2D or 3D space while the Cognitive Spacetime model constitutes a potentially high-dimensional space. We can therefore not simply apply existing methods. Instead we have to identify those techniques which are applicable for high-dimensional problems and optimize them.

Despite the dimensionality problem — which is indeed solvable — learners' movements through Cognitive Spacetime result in trajectories that are predestined to be modeled with a spatio-temporal database (STDB). Database systems are especially designed for the management of large data sets. They use special indexing techniques in order to have large data sets searched and analyzed very efficiently in time. In principle, this is also the case for spatio-temporal databases. But their indexing techniques and search algorithms are optimized for geometric objects end temporal information about these objects.

Research and implementations of STDBs mainly origin from the field of geo-informational systems [1]. A possible query might be the following (based on examples in [52]):

Dear database, give me the names and routes of all helicopters that will pass the area of storm 'Cathrina' within the next two hours.

The result would be a table containing some helicopter names and their routes in the form of trajectories.

Applying this to learning histories we could analogously process the following query:

Dear database, take the latest 10 LOs processed by learner 'Maria'. Give me the names and the learning trajectories of three other learners who are closest to these 10 LOs and who have achieved the highest grades in the LO identified by 'final exam'.

This query could be the basis for learning instructions that recommend the learning pathways of other learners who have worked on similar learning objects and who were especially successful in a final exam. The exam itself may also be a learning object in the learning environment for which personalized meta-data exist telling us about a learner's score on that test.

The above query is formulated with natural language. STDBs provide query languages often as a variant or super set of SQL [21, 34, 35, 51]. These query languages include spatial and temporal logic as well as according operators and predicates. In contrast to standard databases, STDBs implement indexing structures and querying algorithms that are optimized for spatio-temporal objects and logic.

There exist various research projects and implementations of spatio-temporal databases. STDBs can be subdivided into temporal databases which focus on the time dimension, spatial databases only considering spatial dimensions and spatio-temporal databases dealing with both spatial and temporal dimensions.

A summary of temporal database implementations is provided by Böhlen in [14]. Purely temporal databases are for example the ARCADIA database for clinical applications [25], Calanda for time series with financial data [31, 94], ChronoLog running on top of a standard Oracle database [13], HDBMS [92, 93], TDBMS [110] and finally TimeDB for general purpose which is based on the ATSQL query language [15, 16, 48, 62, 104].

The field of spatio-temporal databases is mostly dominated by geographical information systems (GIS), network and facility management, land information systems (LIS) and image processing [1]. For example GRASS GIS [81] and GeoToolKit [8] are geographical information systems while the CONCERT database focuses on management of raster images [88, 89]. The SECONDO database is a multi-purpose system for spatio-temporal data [26, 49].

Depending on the meta-data descriptions of the learning environment, the spatio-temporal model underlying the Cognitive Spacetime may consist of a high number of spatial dimensions. Due to their scope of application most STDBs consider only two or three spatial dimensions. An exception is the DEDALE database which is based on a constraint database technique. This constraint-based approach describes spatio-temporal objects as point sets which are defined by logical constraints [45, 90]. With this concept a trajectory can be defined as an infinite point set which is described by a hyper-polyline. Each segment of this hyper-polyline is defined as a piece of a straight bounded by a particular interval. For example the following constraints define a trajectory as illustrated by figure 2.18:

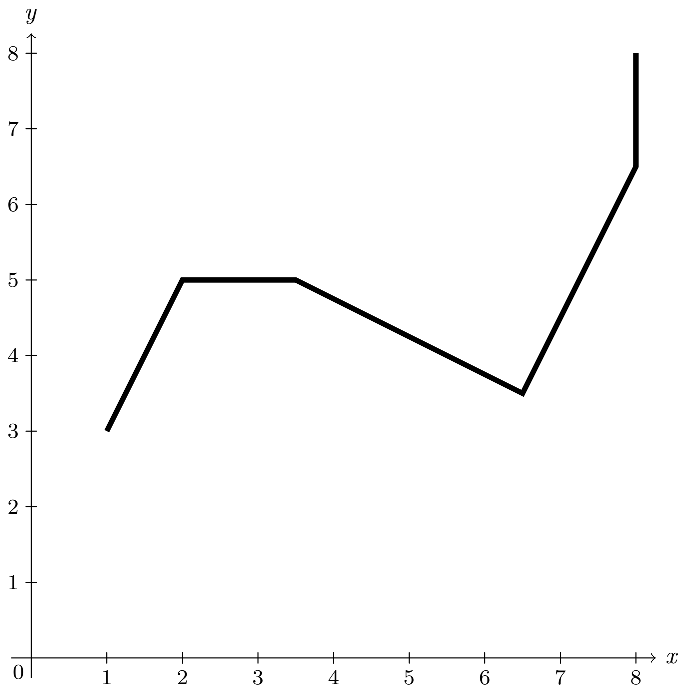


Figure 2.18: 2D trajectory defined by constraints

$$
\begin{aligned}
& (y = 2x + 1 \wedge 1 \leq x < 2) \\
\vee\ & (y = 5 \wedge 2 \leq x < 3.5) \\
\vee\ & (y = -0.5x + 6.75 \wedge 3.5 \leq x < 6.5) \\
\vee\ & (y = 2x - 9.5 \wedge 6.5 \leq x < 8) \\
\vee\ & (x = 8 \wedge 7.5 \leq y < 8)
\end{aligned}
$$

Indeed, the DEDALE system can be used for a high number of dimensions [46, 47]. The constraint-based approach is very useful for spatial objects in the form of infinite point sets. Database queries that work on such constraint-based representations can be implemented with relational algebra. The result of these queries are also infinite point sets in the form of constraints. However, in the Cognitive Spacetime the objects in question are trajectories for which particular characteristics are more important. Trajectories are interpolated sequences of ordered points and have to be treated as connected objects. In order to operate on trajectories efficiently, we should be able to address trajectories as coherent objects. This implies that there must be special preservation mechanisms which identify a single segment with the trajectory it belongs to. Especially for spatio-temporal indexing this kind of trajectory preservation becomes important as segments should not only be indexed with respect to their spatio-temporal location but also considering the trajectories they belong to. therefore, the DEDALE database is not the optimal candidate.

2.3.1 Spatio-Temporal Indexing

In order to search and analyze large data sets, databases implement indexing techniques to enable fast and efficient search. The overall strategy of indexing is to pre-structure the search space and to subdivide the search into a rough preliminary step and a final refinement process. A desired goal is that the duration of a database search does not suffer to much from the amount of data that is stored in it. This is true for classical databases and it is also true for spatio-temporal databases (STDB). Good indexing techniques provide logarithmic time complexity or even better. Especially in the case of high-dimensional data a good indexing-technique is a challenging goal.

Existing research and implementations of spatio-temporal databases provide various approaches for indexing spatio-temporal objects. Most of these techniques are based on the R-Tree family [7, 32, 53, 84, 125]. R-trees can index arbitrary geometric objects like points, point sets, line segments and polygons by simply defining the minimal bounding boxes of these objects. A minimal bounding box for an n-dimensional object is a n-dimensional rectangular of minimal extent containing the object completely. With R-trees the search space is divided into a hierarchical organization of bounding boxes in multiple levels. On the leave level it contains the smallest bounding boxes which include spatially close geometric objects. On the levels above bounding boxes are grouped to larger bounding boxes.

This tree structure allows for an easy implementation of range searches. For a range search the tree is traversed from the top to the leaves by selecting those bounding boxes on each level which are contained or intersected by the range. This may yield objects that are not or partly contained by the range if the minimal bounding boxes at the leave level are not completely in the range of the search. Therefore, a refinement step is performed at the end to calculate the final result. Figure 2.19 shows an example of an R-tree structure for two dimensions and figure 2.20 shows a partition of bounding boxes in a 3d space.

There are lots of variations and advancements of the R-tree. For example Gueting et al. list the 3D R-tree, the HR-tree, the RT-tree and the MR-tree. For moving objects with respect to the current time and the near future they refer to TPR-trees, multilevel partition trees, kinetic B-trees and kinetic external range trees [52].

What indexing techniques are appropriate, strongly depends on the kind of data and the kind of queries to be performed. As far as the Cognitive Spacetime is concerned, the queries of interest refer to past movements. Therefore, indexing methods which focus on only current movements or the prediction of

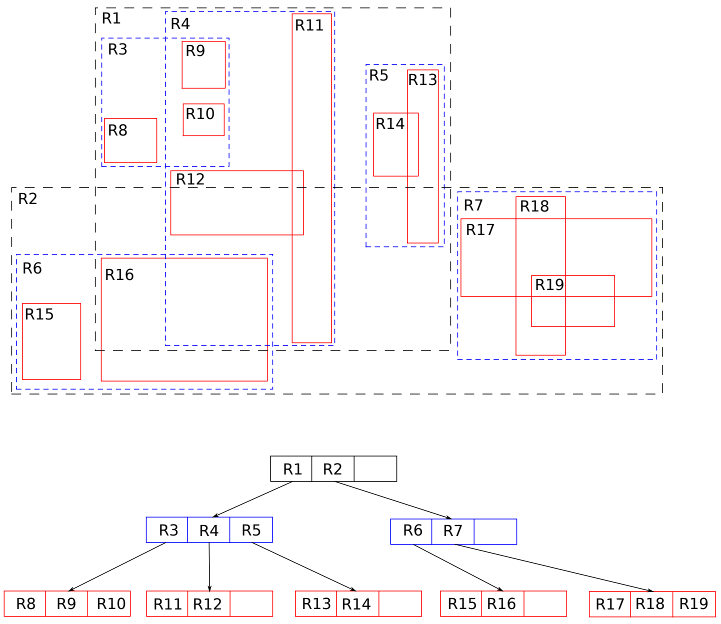


Figure 2.19: Example of a 2d R-tree (Public Domain, Wikipedia)

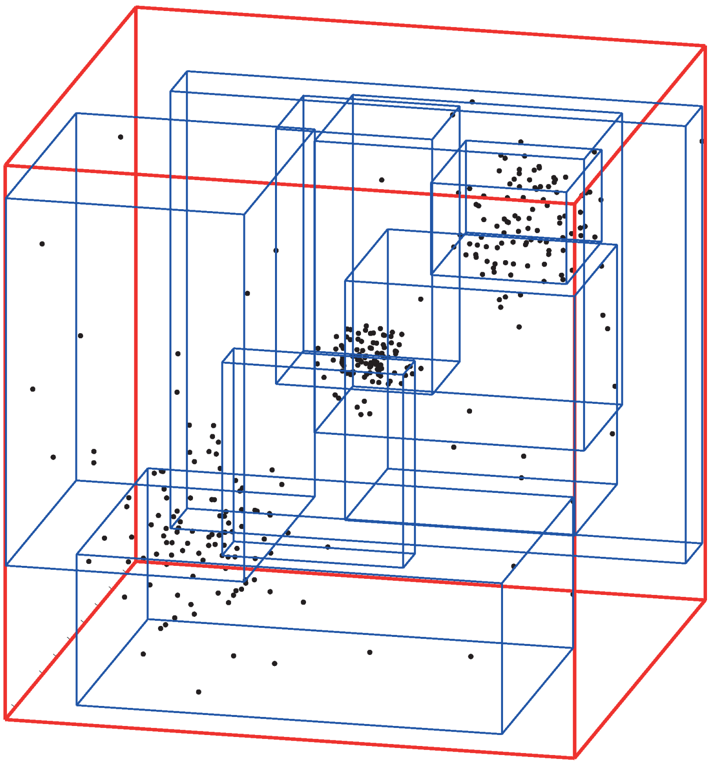


Figure 2.20: Example of a spatial partition for a 3d R-tree (Public Domain, Wikipedia)

them in the future are not relevant. Also we are less interested in querying geometric objects representing areas or point sets as it would be useful in Geographical Information Systems. Instead the Cognitive Spacetime has to provide queries which refer to entire trajectories. For the special field of trajectory indexing there is the Spatio-Temporal R-tree (STR-tree) and the Trajectory Bundle Tree (TB-tree) [84]. Both are designed for performing point, range and nearest-neighbor queries as well as trajectory-based queries [52].

However, there is a significant problem about indexing high-dimensional trajectories. I have already outlined that search trees for spatio-temporal data commonly partition their search space by bounding boxes. The more dimensions are involved, the more these bounding boxes tend to overlap. As a consequence more sub-branches of the search tree have to be traversed in order to find a specific object. This makes R-trees become inefficient in high dimensions. The X-tree which is derived from the R-tree seems to be optimized better for multiple dimensions [11]. But actually, the Cognitive Spacetime focuses on spatio-temporal trajectories which own some particular characteristics separating them from other spatio-temporal objects. The trajectories in question use to grow in a monotone way along the time dimension. Trajectory segments are usually inserted chronologically meaning that an index structure does not have to be reorganized for past insertions, changes or deletions. This is a feature which especially characterizes the dynamic structure of R-trees. But in the case of Cognitive Spacetime this feature is dispensable. We can exploit this peculiarity to implement an indexing structure that does not suffer under high dimensions.

A good alternative in this sense is a grid-based index structure as it was implemented by SETI (Scalable and Efficient Trajectory Index) [17]. SETI uses a two-level structure treating the spatial and the temporal indexing separately. The spatial search space is partitioned by static, non-overlapping, spatial cells instead of bounding boxes. The grid index approach can be considered as especially interesting for two reasons: First, as the temporal dimension is treated separately, we can use an existing temporal database to index the time dimension. Second, the developers of SETI show that their technique "‘does not suffer from the curse of dimensionality"’ [17]. Sidlauskas et al. provide a comparison of tree-based and grid indexes in [117]. They also studied different parameterizations for the best performance of both tree and grid index structures.

Figure 2.21 shows an example for a two-dimensional grid index. The space may be partitioned by different interval lengths for each dimension. The interval length should be chosen with respect to the distribution of the data along the according dimension. The trajectories are inserted into the index structure segment by segment. Each segment has to be fit entirely into one cell. If a segment intersects the boundary of two cells it is split at the point of intersection.

2.3.2 Spatio-Temporal Queries

Once an index structure is established it provides the foundation for efficient queries. With such queries we can find similar trajectories and even perform trajectory clustering to find common learning pathways, behavioral patterns and correlations. The field of spatio-temporal databases provides a large amount of algorithms. I decided for a carefully selected set of queries which build on one another. Together they form a relatively small but a pleasingly powerful hierarchy of queries with which we can perform both nearest neighbor search and clustering.

Range Query The simplest query which can be performed is a range query. Range queries are defined by intervals on each dimension such that the query conforms to a window in the

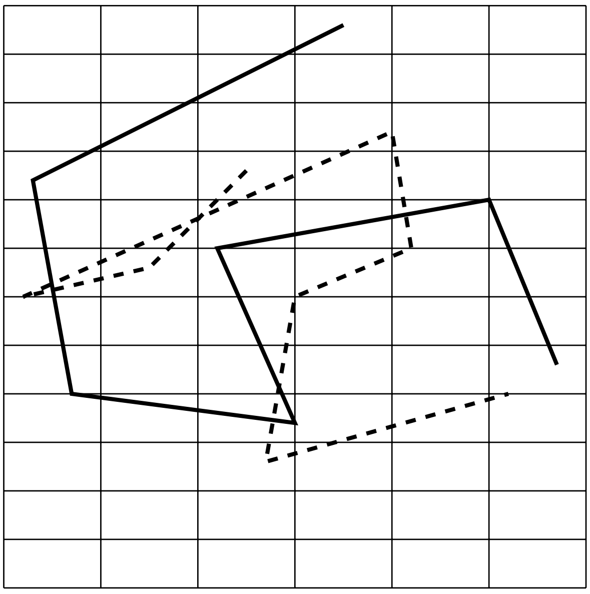


Figure 2.21: Example of a 2D grid index with two trajectories

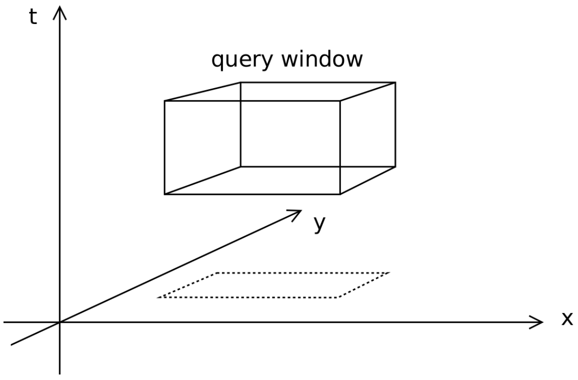


Figure 2.22: 3D range query with two spatial and one temporal dimension

form of a k-dimensional hyper-rectangle (see figure 2.22). A range query asks for all objects that lie inside or intersected by the query window.

Circular Query Additionally, I define the circular query which is a special form of a range query. It asks for all points which are at a maximum distance from a point of interest. Its shape depends on the distance metric we use. If euclidean distance is used the query equates to a radius r with a respective circle in 2D and a hypersphere in n-dimensional space. Using the Manhattan distance or the Chebyshev distance we obtain hypercubes for n dimensions and squares for two dimensions. Figure 2.23 shows the shape of a circular query for Manhattan, Euclidean and Chebyshev metric. If the Chebyshev distance is applied the circular query can be directly implemented as a range query with equal interval length on each dimension.

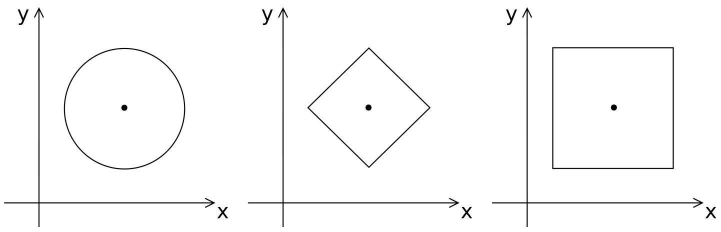


Figure 2.23: From left to right: 2D circular query for Euclidean, Manhattan and Chebyshev distance

K Nearest Neighbor Query As the name suggests, a K nearest neighbor query searches for a number of K objects that are closest to a point of interest. Basically, a nearest neighbor query can be understood as a sequence of circular queries with iteratively growing radius. The iterations stop as soon as K objects are found within the current radius. Again, the shape of the query is determined by the distance metric. Prasath et al. provide an overview of different distance metrics and their efficiency for nearest neighbor queries [86]. Figure 2.24 shows a nearest neighbor query for $K = 2$ with three iterations for the Euclidean distance and two intersecting trajectories.

Incremental K Nearest Neighbor Query In Cognitive Spacetime we are mostly interested in distances and nearest neighbor relationships between trajectories. Trajectories can be regarded as interpolated point sets. Therefore, nearest neighbor relationships between trajectories can be approximated by an incremental sequence of nearest neighbor queries. A trajectory can be approximated by a sequence of points along the trajectory: $T = p_1, p_2, \ldots, p_n$. We can implement queries that apply a $K + \delta$ nearest neighbor query for each p_i whereby δ is an estimated parameter to ensure that the accumulated result contains all K nearest neighbors. Figure 2.25 illustrates such

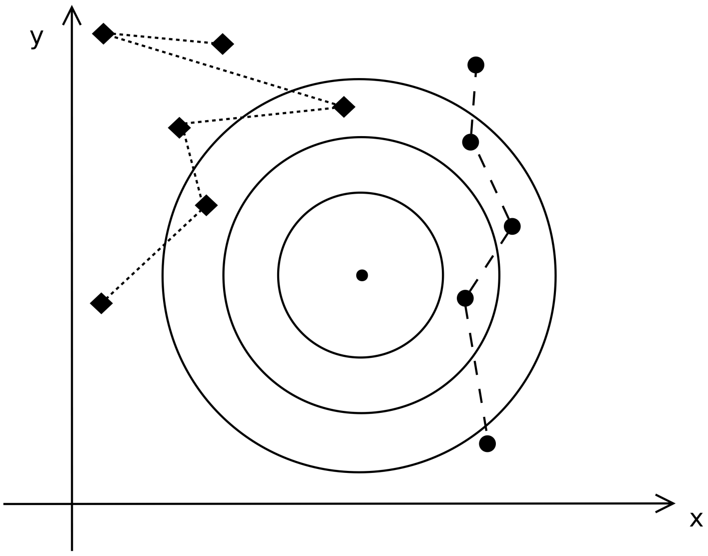


Figure 2.24: 2D circular K nearest neighbor query for $K = 2$ and Euclidean distance

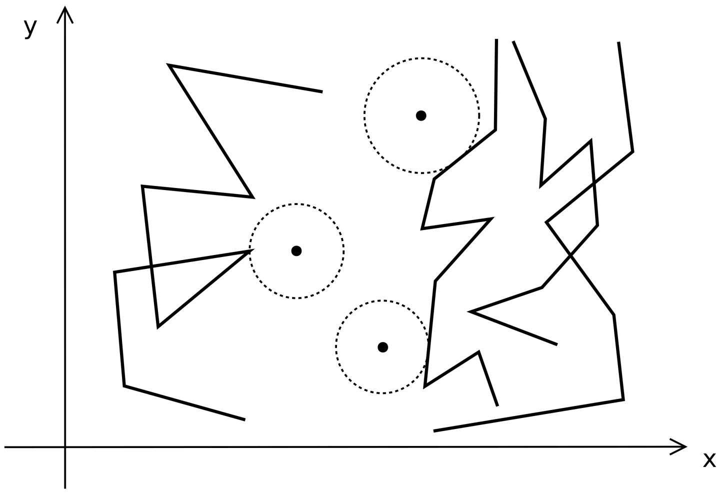


Figure 2.25: incremental nearest neighbor query with three sample points

an incremental query with three sample points. The algorithm was introduced by Chen et al. in [20]. They also provide an estimation strategy for δ. Moreover, Chen et al. enhanced their method to also consider the ordering of the query points along the trajectory. They achieve this by applying the single nearest neighbor queries in a recursive strategy.

Besides Chen et al. there are various methods to perform nearest neighbor queries on trajectories. Many of them are designed to work on R-trees or on variations of them [91]. Some of them are developed for grid index structures [54, 121, 123]. It also makes a difference if near trajectories are supposed to be found with respect to a single query point (point-to-trajectory relation) or with respect to an entire trajectory (trajectory-to-trajectory relation). The algorithm by Chen et al. can be used for both types of queries. The query object, to which nearest trajectories are supposed to be found, may be a single point. This is just a special case of the algorithm for which the list of query points contains only one object. In general, algorithms for nearest neighbor relations are not trivial when both the query object and the neighbor objects are trajectories. This is because the computational effort increases massively. Choosing some representative points from the query trajectory and applying an incremental nearest neighbor search may provide a good approximation. It is also a good compromise regarding computational complexity and precision.

Let me give a brief impression of the work of other researchers. Gao et al. propose two algorithms based on an R-Tree which can find nearest neighbor trajectories with respect to a static query point [42]. Güting et al. describe a method based on a 3D-R-tree in [50]. Frentzos et al. propose depth-first and best-first algorithms which are also based on R-Trees [37, 38]. Their algorithms work either for stationary or moving query points. Xia et al. introduce a technique which uses a grid index [121]. It is built on the MapReduce Framework [27] which is a programming paradigm for parallel computing on computer clusters. So, Xia et al. solve the problem of computational complexity by parallelization. In [54] Hasan et al. discuss continuous nearest neighbor queries. They do not deal with query objects in the form of historical trajectories but in the form of continuously updated queries which is relevant for the observation of movements in real time. Their publication has some relevance to the work presented here, because they use a hierarchical grid index which they call "grid-tree". This grid tree uses multiple levels of successively refining grids for better query performance. In a similar way Zheng et al. also use a hierarchical grid index structure for trajectory similarity queries [123].

2.3.3 Clustering

There exist various approaches for the clustering of trajectories. For example Byoung-Kee Yi et al. deal with similarity patterns in time sequences [122] considering time warping techniques. In addition to this, the work of Vlachos et al. can be mentioned, who propose similarity functions to find longest common subsequences (LCSS) among trajectories [116]. They show that their technique is especially appropriate for data with noise.

In [82] Pelekis et al. provide distance operators to find similar routes within the same time period or without the time dimension. The latter case considers only the spatial projection of trajectories. Additionally, they introduce derived similarity metrics based on speed and directional patterns. To calculate the similarity of trajectories, they use the area which is enclosed by the projection of two trajectories in a 2D plane.

Kalnis et al. subdivide moving objects by time slices [63]. Each time slice is a snapshot of frozen points. These points are clustered applying the DBSCAN algorithm [36]. Kalnis et al. also adopt several optimization techniques which includes a method they derive from MPEG-2 video encoding. They treat time slices like video frames and approximate intermediate time slices to reduce the computational effort.

Nanni and Pedreschi address the idea that not all time intervals may have the same importance for clustering [80]. Their work is based on the OPTICS algorithm (Ordering Points To Identify the Clustering Structure) [4] which is an enhanced version of the DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise) [36]. In [80] Nanni and Pedreschi adopt clustering with respect to multiple possible time intervals choosing the most significant result. They apply this to traffic and transportation patterns. But we can also transfer this to learners. Time periods shortly before exams will have another meaning than other time periods and students will probably show different behaviors and clusters. Another approach in this sense is provided by the work of Jae-Gil Lee et al. who focused on similar sub-trajectories [69].

An interesting approach uses discrete Fourier transformation to map trajectories into a feature space of lower dimension. Agrawal et al. introduce this method stressing that the first few frequencies are sufficient to obtain good results of similarity queries [2].

A general survey of clustering techniques is included in a publication by Kisilevich et al. [65]. Amongst other methods they summarize distance-based and density-based clustering, visual-aided and pattern-based techniques.

However, in a rather straight forward way clustering can be performed simply on the basis of nearest neighbor queries. Imagine some clusters of spatial objects. If the parameter K of a K nearest neighbor query is sufficiently small it will mostly or even only provide nearest neighbor points from inside a cluster, no matter which point of reference is chosen. This indicates that we can obtain information about clustered objects by applying multiple nearest neighbor queries in some intelligent way. This works for point sets as well as for trajectories if the functionality of a nearest neighbor query is available.

I briefly explain two clustering methods which are based on nearest neighbor queries. The first technique is the nearest neighbor chain algorithm [78, 79]. It starts from an arbitrary point and successively constructs a nearest neighbor graph from it. This is a directed graph connecting consecutive nearest neighbors. When two points are reciprocal nearest neighbors they are clustered and replaced by the center of the cluster. The procedure is then repeated starting from the last point before the two reciprocal points. This way clusters develop step by step.

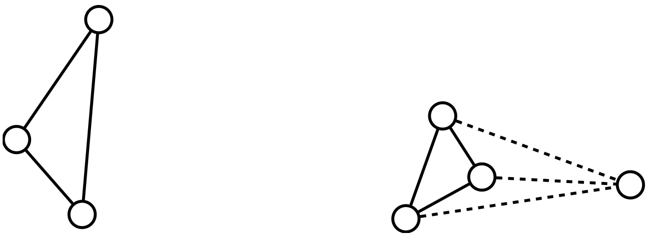


Figure 2.26: Shared nearest neighbor graph for $K = 2$

The nearest neighbor chain algorithm works for points. In principle we could think of building it also from nearest neighbor queries with trajectories. However, the algorithm replaces reciprocal nearest neighbor points by their center. Applying this to trajectories is not trivial because some kind of center trajectory would have to be constructed and it is not clear how this can be implemented. Instead I suggest another algorithm which is called the shared nearest neighbor algorithm [33, 61]. It performs an agglomerative hierarchical clustering which is based on a very particular similarity measure. Instead of using a classical distance measure the algorithm looks for the number of nearest neighbors two points have in common. Based on this, the algorithm builds a shared nearest neighbor graph. This is an undirected graph the edges of which are weighted by the number of shared nearest neighbors.

Figure 2.26 shows an example for a graph which is built from K nearest neighbor queries with $K = 2$. Dotted lines represent two shared neighbors and straight lines represent three shared neighbors whereby this also includes the respective point as its own nearest nearest neighbor. This graph can be represented by a similarity matrix assigning each pair of points the according weight. This matrix can then be used to perform clustering. the values of this matrix can directly be obtained from a K nearest neighbor query. It therefore can be applied not only for points but also for trajectories or any other form of object as long as the functionality of a nearest neighbor query is available.

Another conceivable approach is to use a grid index structure to find clusters of objects. For each cell in the grid index we can easily store statistical information like the number of trajectory segments which are located in that cell. This leads directly to a density measure for each cell. We can use this information to find connected regions of high density which is equivalent to finding spatio-temporal clusters. Two representative algorithms for this approach are the CLIQUE (Clustering In QUEst) [3] and the STING (Statistical Information Grid) [119] algorithm. A comparison of the two algorithms was given by Suman and Rani [107].

Both the CLIQUE and STING algorithm are originally designed for clustering point sets. However, it does not make a difference if the objects of interest are points or single trajectory segments. In any case, such a grid-based clustering algorithm can be used to find regions in the Cognitive Spacetime which are traversed by learners frequently and with high density. Figure 2.27 illustrates a possible result of grid-based clustering in a two-dimensional grid of trajectories. As figure 2.27 shows, this technique is especially useful if we want to find trajectories that share common sub-trajectories.

2.4 Adaptivity Cycle

In section 1.6 (Adaptivity with Turing Machines) I introduced a universal model for an adaptive algorithm based on the Turing machine. Let us remember that the Turing machine is more than just an abstract imagination of a machine. It is a strong concept telling us what we can implement with computers. Any statement which is based on the Turing machine

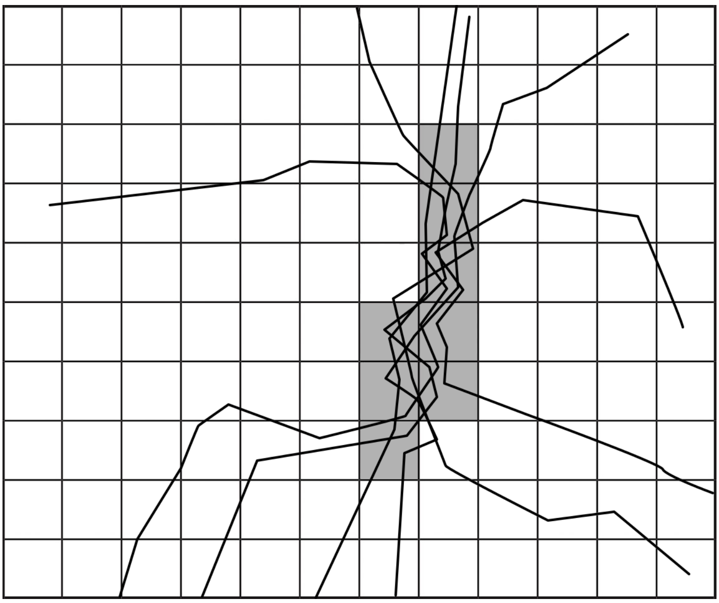


Figure 2.27: Grid-based clustering

is also true for any form of algorithm, no matter from what kind of technology it is built. Therefore, the model of the adaptive Turing machine provides us a clear design pattern for adaptive learning environments. Such a system must include a knowledge base reflecting both knowledge about the learning content and the learner. This knowledge base is continuously updated by outer events. Outer events usually are actions by the learner or updates of data which are processed about her. Each update of the knowledge base triggers a new process of inference thereof.

The result of this inference process may be a recommendation for the learner. The learner's reaction on this recommendation will again trigger the next update of the knowledge base. This leads to an adaption loop which results in a mutual interaction between the learner and the computer: the learner modifies the knowledge base of the algorithm and the algorithm modifies the behavior of the learner which again modifies the knowledge base and so on.

The learning recommendations can be handled in more or less rigid ways. They can be implemented as strict instructions which a learner has to follow or they can be handled as non-binding recommendations. In the latter case the adaptivity cycle will react in a very accommodating way if the learner deviates from the recommendation. With each iteration of the adaptivity cycle, the learners state is reconsidered and new recommendations are generated.

The all-important question is: what must a knowledge base and a respective inference procedure look like? Essentially there is an infinite number of answers. However, if we claim the system to shape a triangular relationship between learner, teacher and machine, there is a fundamental criterion such a system should satisfy. The knowledge base should be comprised in a form that is understandable by both humans and computers.

In section 2.1 (Ontologies) I have outlined how ontology languages can be used to encode semantics according to pedagogical and didactic meta-knowledge. I also showed how in principle machines can infer on such knowledge representations. While ontologies work well for semantic relations they are hardly suitable for the representation and analysis of the historicity of learning behavior. For this purpose I introduced the model of Cognitive Spacetime.

I furthermore described how spatio-temporal queries can be used with the Cognitive Spacetime to analyze learning trajectories and find similar pathways among learners. In this section I will explain how these two processes can be integrated into an adaptivity cycle. Let me stress that this idea is not limited to ontology-based reasoning or the Cognitive Spacetime model. The adaptivity cycle is a more universal concept within which the Cognitive Spacetime and the ontology approach may be replaced or enhanced by any other process.

The adaptivity cycle is subdivided into three parts: the knowledge base update process, the inference process and the generation of learning recommendations. In the following the respective parts and how they combine Cognitive Spacetime and the ontology approach are explained.

2.4.1 Knowledge Base

The knowledge base contains a representation of learning objects and learning histories which is divided into three partial representations but not limited to them:

- A pedagogical ontology which is built upon semantic annotations about learning objects and their didactic interrelations.

- A learner state ontology which contains meta-data about learners and their relations to learning objects in the pedagogical ontology.

- The Cognitive Spacetime embodying a spatio-temporal model of the learning objects in the learning environment and learners' time-dependent cognitive positions within that model.

The first two items in this list — the pedagogical and the learner state ontology — were formulated and implemented in the INTUITEL project. Refer to section 2.1.1 (Knowledge Representation) to get more information about the structure of the ontology and the inference process on it. The third item — Cognitive Spacetime — was implemented in the form of a experimental prototype system which is called the "Hypercube Database". I have already outlined the data structures and the spatio-temporal query algorithms on a theoretical basis. The concrete implementation of them is explained in chapter 3 (Hypercube Database).

It is evident that the ontology on the one hand and Cognitive Spacetime on the other hand are based on data sharing similar meanings. Both the ontology and Cognitive Spacetime describe learning material and learners with meta data. However, the way this is done is fundamentally different in the two cases. The ontology describes only semantic relations in the form of "object X is an instance of..." or "object X is related to object Y". On the contrary, the dimensions of the Cognitive Spacetime are meta-data in the form of time-dependent variables which are assigned numeric values. Each of these variables rates a specific property of learning objects and learners. The aggregate of all these properties finally constitutes a learner's specific cognitive position in the Cognitive Spacetime.

2.4.2 Learner-Triggered Knowledge Base Update

Together the Cognitive Spacetime and the learner state ontology contain information about learners. This may include static or fluid information. Fluid data may be provided by a learner's current or latest learning object as well as by measured parameters like bandwidth, biometric sensor data or activity and performance measures of any kind. Static data may include information about age, educational level, socioeconomic background, personality traits or learning styles. While static features will not change over time, fluid information has to be updated continuously. Whenever a learner progresses on a learning object this should trigger an update of the information that is stored in the Cognitive Spacetime and the learner state ontology.

Within a digital learning environment one can easily track learners' activities and send according data to a recording and recommendation engine. But this is not restricted to a single learning environment. We can also think of multiple learning environments feeding a central engine or multiple engines sharing the same knowledge base. This allows us to support learners over multiple learning environments. How the tracking of learners and the generation of recommendations can work over different platforms was in principle anticipated by the INTUITEL system. Initially the INTUITEL prototype was designed to support one recommendation engine for one learning management system. But the developers also formulated a network protocol for the information exchange between the learning management platform and the engine (see the publication in [39] pp. 62–74). Such a protocol-based approach appears to be suitable to be extended to a distributed system landscape.

2.4.3 Inference Process

At its core, an inference process can be seen as a stepwise set reduction of learning objects. The result is a specific subset containing learning objects which are most appropriate according to a learner's state and history. This becomes particularly clear if we look at the inference process as it was implemented in the INTUITEL system. This was explained in section 2.1.1 (Knowledge Representation). At the beginning there is a set which contains all learning objects that are available in a learning environment. By following the semantic relations in the pedagogical and the learner state ontology this set is successively reduced by simply excluding those learning objects which do not fit the required relations.

Applying spatio-temporal queries on learning trajectories in Cognitive Spacetime can also be understood as such a set reduction process. In this case, the result set simply consists of the learning objects that are part of those trajectories returned by the respective queries. For all parts of the knowledge base, including the pedagogical ontology, the learner state ontology and the Cognitive Spacetime trajectories, this set reduction can be performed in separate processes. finally, the result of each reduction process can be combined by basic set operations. In the simplest form the final result set may be built as the intersection of all single set reductions.

At the beginning of this section I indicated that the knowledge base in the adaptivity cycle is not limited by the ontology and the Cognitive Spacetime approach. Defining the inference process as a combination of multiple set reductions makes it extremely scalable and expandable. We can think of any other method to be included as just another set reduction process which is illustrated by figure 2.28. For example, deep learning algorithms could be used to complement the adaptivity cycle. Especially pattern recognition in time series data — as they

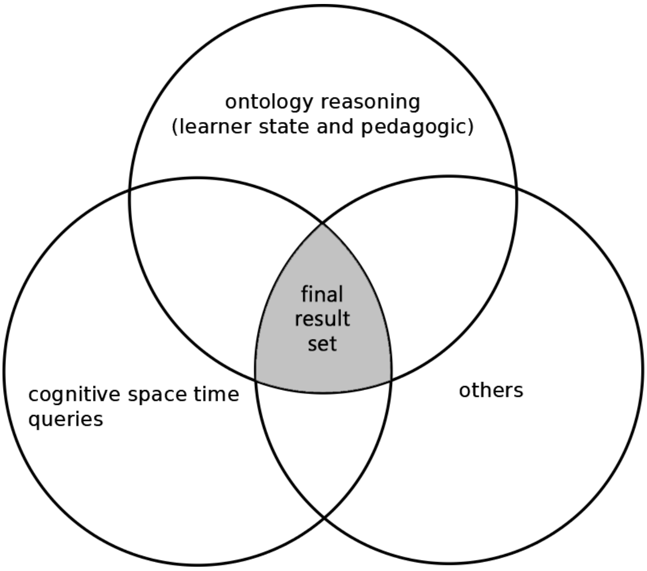


Figure 2.28: Inference process as a set reduction result

are the basis of spatio-temporal trajectories — might be an interesting extension.

2.4.4 Generating Learning Instructions

Before running into the next iteration, the adaptivity cycle may close with the generation of a learning recommendation. The input for this step is the result of the set reduction in the inference process. Basically the output of the set reduction process is only a set of learning objects. This set has to be transformed in a human-friendly message. An intelligent recommendation unit may use meta-data descriptions of learning objects to create more descriptive recommendations.

If semantic relationships in the form of an ontology are available the recommendation unit may even include justifications telling the learner why a learning object was recommended to her.

If the learning instructions are implemented as non-binding recommendations, the adaptivity cycle will be tolerant of users who ignore single recommendations. With each of the learner's activities, a new run through the adaptivity cycle is provoked. And within each iteration of the cycle the learner's current state within the learning environment will be reconsidered. I have described two possible representations of learners and their position within a learning environment: an ontology approach and the Cognitive Spacetime model. With any run through the adaptivity cycle the learner visits other learning objects which correspond to new semantic relationships to different parts of the ontology representation. In the Cognitive Spacetime model, the learner's deviations from given recommendations will mean that the her cognitive position moves away from certain trajectories but becomes closer to other trajectories. Both set reduction processes — the ontology based and the process based on Cognitive Spacetime — will therefore produce new recommended sets of learning objects. If, on the other hand, a learner follows recommendations she will converge to parts of the ontology and trajectories in the Cognitive Spacetime.

If an adaptivity cycle is implemented in the form described here, it will offer great advantages. Because the adaptivity cycle is based on a reciprocal influence between the learner and the machine, it works on the basis of incomplete and uncertain knowledge. We may start from an even vague image of a learner and the longer she and the machine interact the more both the learner and the machine will comply to each other. This is a fundamental difference compared to user models which presuppose more or less extensive prior knowledge about learners' behaviors.

3 Hypercube Database

In Cognitive Spacetime a learner's history is described by a n-dimensional trajectory. n denotes the number of meta-data items which constitute the dimensions of the Cognitive Spacetime. The strength of the Cognitive Spacetime is that it abstracts heterogeneous data to trajectories which are only geometric objects located in a n-dimensional space and a $(n+1)$-dimensional spacetime respectively. Geometric objects and Cognitive Spacetime as a whole are predestined to be modeled with a spatio-temporal database (STDB). I already outlined that the Cognitive Spacetime was implemented in the form of a prototype database system which I call the "Hypercube Database". The name alludes to the concept of the INTUITEL Hypercube because it provided the idea for the Cognitive Spacetime. This chapter provides a detailed description of the Hypercube Database and its implementation.

The Hypercube Database is implemented as a Java application. It consists of four sub-modules which are the *Database Access Module*, the *Vector Module*, the *Hypercube Module* and the *API*. Figure 3.1 shows the overall software design of the system.

3.1 Database Access Module

The Database Access Module encapsulates the functionality of a TimeDB back end. The TimeDB back end is an external

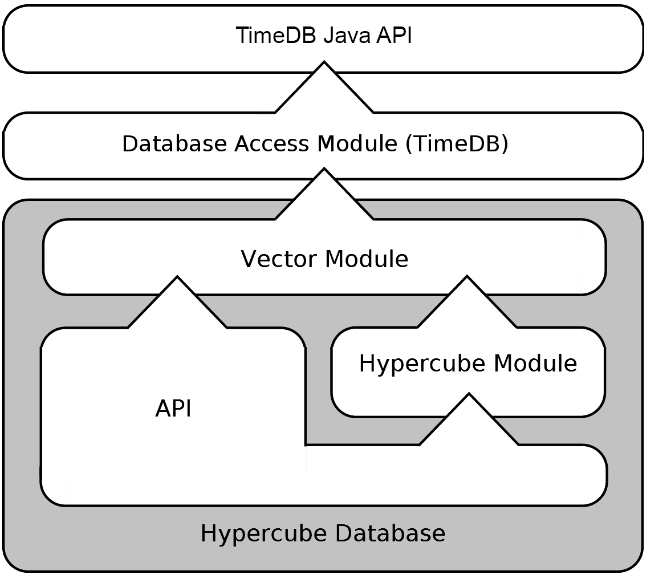


Figure 3.1: Architecture of the Hypercube Database

Java Component which is used for the temporal dimension of the Cognitive Spacetime. The Database Access Module provides an interface for performing basic database functions like opening or closing a database connection to TimeDB. Furthermore, temporal queries can be executed using the ATSQL query language [15, 16, 48, 62, 104]. ATSQL queries are delegated to the TimeDB module. In principle, the TimeDB back end can be replaced by another temporal database. As long as the respective temporal database implements the ATSQL query language other components which use the Database Access Module are not affected.

3.2 Vector Module

The Vector Module uses the interface of the Database Access Module. Its task is the transformation of single measuring points into time-dependent vectors. These vectors are synonymous with the cognitive position vectors in the Cognitive Spacetime. The Vector Module uses the Database Access Module to store representations of these vectors in the Time DB back end. Each attribute of such a vector is equal to one of the n dimensions in Cognitive Spacetime. More specifically, a vector attribute is the time-dependent value of a specific meta-data item. Each meta-data item may be updated at arbitrary discrete time points.

Consider an individual learner for whom the values of n variables are measured at arbitrary points in time. We regard the lastly measured value as valid until it is updated by a new value. The event of an attribute update is called "measuring point" and is formed by a quadruple mp=(trajectory, attribute, value, time) identifying the learner (trajectory), the attribute name, the update value of the attribute and the time point of the update. The listing below describes how measuring points are transformed into cognitive position vectors. It illustrates the insertion of attribute updates and the time-dependent evolution of the vectors for a three-dimensional Cognitive Spacetime containing three respective meta-data items. In the following I use the term "attribute" synonymously for meta-data item. the three attributes are denoted as a_1, a_2 and a_3.

Before the first update all attributes have an initial value — for example 0 — from start to eternity.

$$\begin{aligned} a_1 &= 0 \; for \; t \; \in [0, forever) \\ a_2 &= 0 \; for \; t \; \in [0, forever) \\ a_3 &= 0 \; for \; t \; \in [0, forever) \end{aligned}$$

Suppose that three measuring points for one individual learner are inserted at three different time points t_1, t_2 and t_3:

$$\begin{aligned} a_2 &= 3 \quad at \; t_1 \\ a_2 &= 5 \quad at \; t_2 \\ a_1 &= 7 \quad at \; t_3 \end{aligned}$$

After these updates the cognitive position vector has the following time-dependent values:

$$a_1 = \begin{cases} 0 & for \; t \in [0, t_3) \\ 7 & for \; t \in [t_3, forever) \end{cases}$$

$$a_2 = \begin{cases} 0 & for \; t \in [0, t_1) \\ 3 & for \; t \in [t_1, t_2) \\ 5 & for \; t \in [t_2, forever) \end{cases}$$

$$a_3 = 0 \; for \; t \in [0, forever)$$

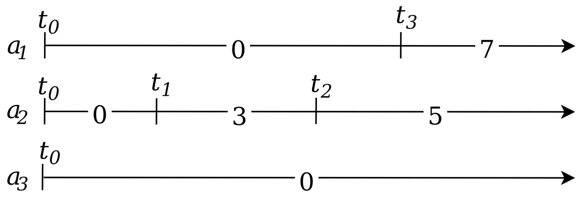


Figure 3.2: Example 3D-vector with three updates

Figure 3.2 illustrates the update procedure and the resulting time lines of the three attributes a_1, a_2 and a_3. The Hypercube Database includes functionality to visualize such time lines. The output is a color map for every trajectory. I have introduced this in section 2.2.1 (Worldlines of Learning). For clarification have again a look at an example as shown in figure 3.3. This visualization shows the vectors of three learners on a video platform and includes four variables for each learner. The time-dependent values of the attributes are encoded by colors. Essentially, figure 3.2 and 3.3 are equivalent representations of cognitive position vectors as trajectories.

The two figures also illustrate that time-related updates refer to single attributes. The TimeDB back end does not provide temporal support for single attributes. instead it only supports temporal functionality for complete database tuples. Therefore the Vector Module also implements a mapping to support attribute-wise temporal support. This is achieved by storing each attribute of each vector separately and associating it with the related vector by foreign key relations. It is the Vector Layer which joins single attributes to vectors at the time when queries are performed. For a more detailed description of the TimDB database schema see section 3.5.2 (ATSQL Representation).

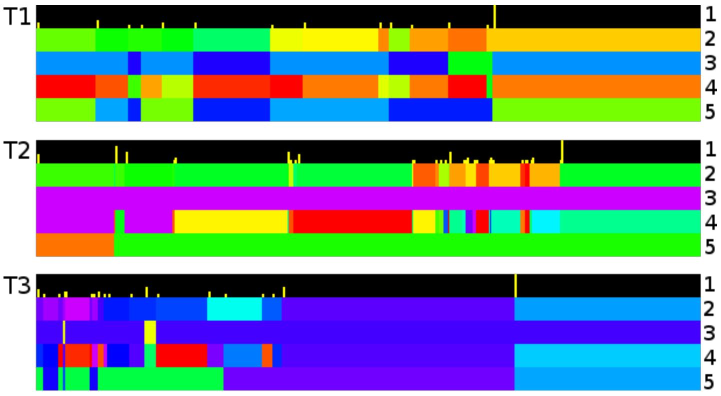


Figure 3.3: mobile users on a video platform (see section 2.2.1)

3.3 Hypercube Module

The Hypercube Module incorporates the actual model of the Cognitive Spacetime. The Vector Layer which was previously described, only manages vectors and their time-dependent attributes. Alone, these vectors do not yet represent trajectories. They can be regarded as trajectories when recorded over time but they are not yet real geometric objects. Transforming vectors into such geometric objects is therefore the special task of the Hypercube Module. The resulting trajectories are hyper-polylines which are built from linearly interpolated, interconnected segments. Beyond this, the Hypercube Module is in charge of spatial indexing and the implementation of spatio-temporal queries.

3.3.1 Trajectory Segments

Single attributes of a vector $v(t) = (a_1(t), a_2(t), \ldots, a_n(t))$ may contain non-simultaneous measuring points which is il-

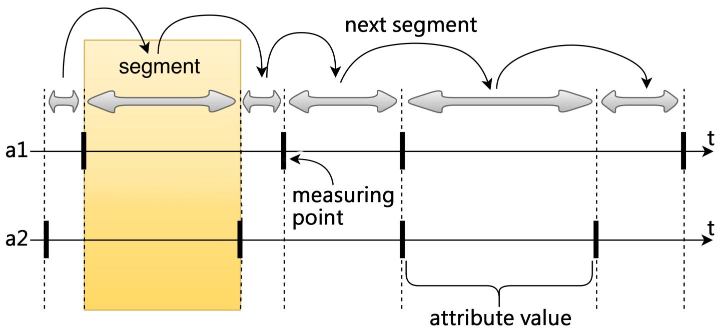


Figure 3.4: Transformation of vectors to segments

lustrated by figure 3.2. However, single trajectory segments are formed by the smallest time intervals which have no measuring points inside them. In order to group attribute time lines of a trajectory and to split it into segments the Hypercube Module performs a mapping as it is illustrated by figure 3.4.

The Vector Module represents its vectors by default as periodwise constant values. In order to built spatio-temporal trajectories from interconnected segments, it is necessary to interpolate the measured values. For this purpose, attribute values between two segment endings are interpolated by a linear function on each dimension. The interpolation does not take place persistently in the database. Instead, the database only stores non-interpolated data which is dynamically interpolated at the moment when spatio-temporal queries are performed. Note that one could also choose any other form of interpolation. For example we could also think of polynomial functions or even spline curves. However, linear interpolation is a good compromise between computational effort and approximation.

3.3.2 Spatial Indexing

The foundation for any form of spatio-temporal queries is an efficient indexing technique. In section 2.3.1 (Spatio-Temporal Indexing) I provided an elaborate discussion on multiple indexing techniques. I concluded that common index structures, which are based on the R-Tree family [7, 32, 53, 84, 125] are not appropriate for Cognitive Spacetime because their performance decreases significantly in high dimensions. The Hypercube Module therefore implements a grid index which is similar to the idea presented by the authors of the SETI project [17].

When trajectory segments are inserted into the grid, any segment is located completely inside a grid cell. If a segment intersects several grids, it is split into child segments such that every child segment lies within a cell. The splitting of segments costs computation time at the point of insertion. But on the other side a grid index has high advantages when it comes to the execution of search queries. The advantage of the grid index is explained best on the basis of a range query. In an R-Tree a range query is performed by traversing the tree and searching those bounding boxes which are contained or intersected by the query range. Bounding boxes may overlap which results in the traversion of multiple branches of the tree. Especially in high dimensions the overlapping volumes of the bounding boxes grow massively causing efficiency declines of the search process.

Conversely, in a grid index, the equivalent of a bounding box is a grid cell. But those grid cells are not organized in a search tree and they do not overlap. Instead, they are static and directly addressed. Each cell has the same size which is specified by the interval length on each dimension of the Cognitive Spacetime. Each cell is therefore defined by its lower and upper boundaries in space. Given a point in that space we can tell immediately from its coordinates in which cell it is located. This also applies to range queries. Given the lower and upper bound of a range query on each dimension we can directly compute the affected cells.

Figure 3.5 shows an example for a range query on a two-dimensional grid with an exemplary trajectory intersecting the range. The thick dashed rectangle represents the query window and the gray area comprises the affected grid cells. When a range query is performed the outer bounds of the query window define the coordinates of the cells that edge the area of interest. It is straight forward to compute all cells lying inside the area. Segments which are associated with the found cells can immediately be retrieved without further computation.

It must be noted that the number of cells can become very large if the query window is large and the grid has a high number of dimensions. This may even lead to a combinatorial explosion. However, this problem can be solved by applying a hierarchical grid index. This advanced version consists of multiple grids with different cell sizes. On the highest level the grid is very coarse and has large cells. At lower levels the grid becomes finer and cells become smaller. this is shown in figure 3.6. The size of the query window determines which grid index is used. If the query window is large a coarse grid is chosen. If the query window is of small extend a finer grid is applied. The prototype of the Hypercube Database does not implement a hierarchical grid index. It uses only one grid level but adding more levels may be a useful and easy extension. An example of such a hierarchical grid index is given by Hasan et al. [54] and Zheng et al. [123].

3.3.3 Query Implementation

The TimeDB back end provides temporal indexing as well as temporal queries. Based on the TimeDB back end and the

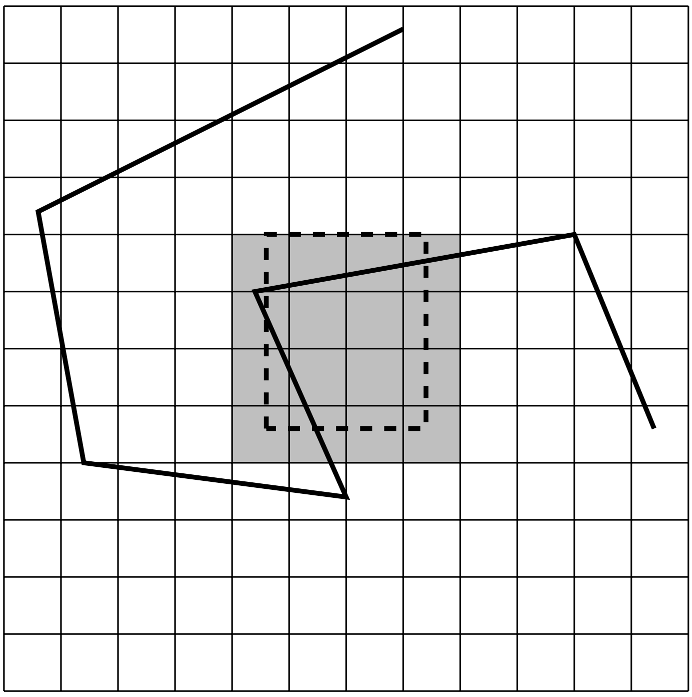


Figure 3.5: 2D Grid index and a range query (thick dashed rectangle)

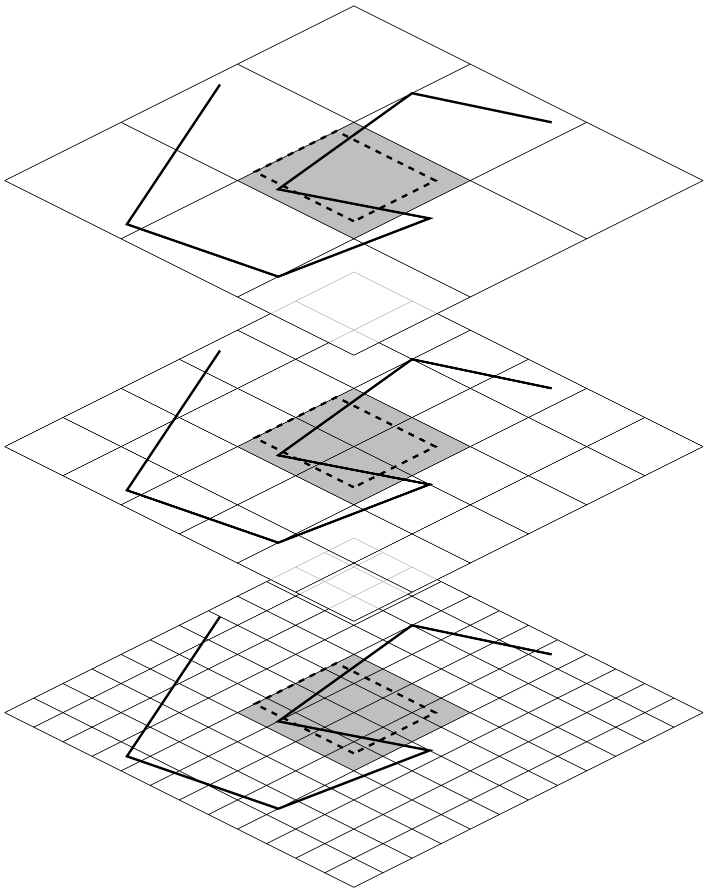


Figure 3.6: Hierarchical Grid Index

grid index structure described above, the Hypercube Module implements spatio-temporal queries for trajectories. For the spatial part, any query is divided into two steps: a cell-based query and a refinement step. In the first step the query computes all cells of the grid index which lie within the query range and which are intersected by it. Then all trajectory segments contained by these cells are returned. The result set of this step may contain segments which are located outside the actual query window. In the refinement step the exact positions of the segments are calculated and segments outside the query window are rejected. As any segment is associated with the trajectory it belongs to the query returns both affected segments and trajectories.

Implemented Queries

The concrete queries which are implemented in the Hypercube database were already explained in detail in section 2.3.2 (Spatio-Temporal Queries). A short overview is listed below.

Range Query This is the basic query from which the more sophisticated queries listed below are developed. It defines search intervals on each dimension of interest and results in a query window in the form of a hyper-rectangle.

Circular Query The circular query asks for objects which are at a maximum distance from a point of interest. In the Hypercube Database it is implemented with the Chebyshev distance metric by which we obtain a query window in the form of a hypercube. Using the Chebyshev distance the circular query can be directly applied as a range query with equal interval length on each dimension.

K Nearest Neighbor Query The K nearest neighbor query is implemented in the form of subsequent circular queries with iteratively growing radius. As the circular query uses the Chebyshev metric this accords to hypercube-shaped range queries with iteratively growing edge length. The iterations stop as soon as K trajectories are found.

Incremental K Nearest Neighbor Query Using a simple Nearest Neighbor Query we can only find trajectories that are closest to a specific point in space. In order to find nearest neighbors which are closest to trajectories the system performs an approximation by an incremental sequence of nearest neighbor queries. Sample points are selected from the trajectory of interest. Then nearest neighbor queries are applied for each sample point.

Chebyshev versus Euclidean Distance

In the Hypercube Database the range and circular query were implemented using the Chebyshev distance metric. This is also the case for the nearest neighbor queries as they are built from incremental circular queries. Actually any distance metric is possible. More information about this can be found with Prasath et al. who provide an overview and analysis considering the efficiency of different distance metrics for nearest neighbor classification [86].

Classical spatio-temporal databases and algorithms often use the Euclidean metric. This is because they usually deal with two or three-dimensional real world objects for which the Euclidean distance is an intuitive measure. However, the Hypercube Database deals with abstract objects in a high-dimensional space. Therefore, there is no universal argument for a specific distance metric. The Chebyshev distance was chosen because the circular query and the derived nearest neigh-

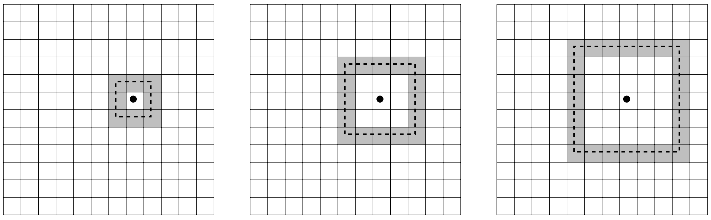


Figure 3.7: 2D Grid index and three iterations of a nearest neighbor query with Chebyshev metric

bor query can be implemented very simple on a grid index with rectangular structure. If the grid cells are square-shaped the implementation of nearest neighbor queries becomes extremely simple. In the Hypercube Database nearest neighbor queries are built from incremental circular queries with iteratively growing radius. With square-shaped cells, increasing the radius can be achieved by just adding a certain number of grid-cells on each dimension and in each direction. With each iteration only the extended cells around the previous search range are considered as the content of the previous cells is already known. Figure 3.7 shows three such iterations on a grid. The radius is increased by one cell per iteration.

Although the current version of the Hypercube Database works with the Chebyshev distance it can be extended by any other metric. The only thing that has to be changed is the construction of the circular query. For the Euclidean metric the neighborhood to be searched is constrained by a circle in 2D or a hypersphere in multiple dimensions. Cheema et al. propose nearest neighbor algorithms called "Circular Trip" and "Circular Arc Trip" [18, 19]. These algorithms also work with iteratively growing radius but they calculate those cells which intersect with a real circle. Figure 3.8 shows three iterations of a nearest neighbor query around a point object using the Euclidean distance and the affected cells on a grid index.

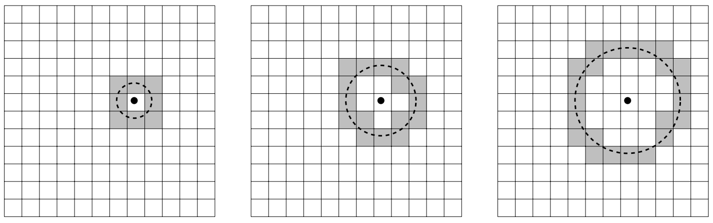


Figure 3.8: 2D Grid index and three iterations of a nearest neighbor query with Euclidean metric

Trajectory Clustering

An algorithm for trajectory clustering was not implemented in the Hypercube database prototype. However, the existing implementation can easily be extended for clustering by using the techniques I described in section 2.3.3 (Clustering). There, I introduced the shared nearest neighbor algorithm [61]. This algorithm can be easily built from the nearest neighbor algorithm, which is implemented in the Hypercube Database. The already existing nearest neighbor search can be used to determine the number of common neighbors that are shared by two trajectories. This metric then serves as a similarity measure for clustering.

Furthermore, I mentioned the CLIQUE (Clustering In QUEst) [3] and the STING (Statistical Information Grid) [119] algorithm. Both algorithms can use the information that is stored in the grid index to find regions in the Cognitive Spacetime which are densely covered by trajectories. In contrast to the shared nearest neighbor algorithm, these algorithms are the better choice if you intend to find trajectories that share common sub-trajectories but which may be different in other parts.

3.4 REST API

The functionality of the Hypercube database is accessed over a REST interface which is accessible using the HTTP protocol. The REST API provides the following methods:

- insertion of measuring points
- listing vectors
- visualization of vectors as color maps
- grid index creation
- spatio-temporal range and nearest neighbor queries

The following sections explain these REST methods in more detail.

3.4.1 Insertion of Measuring Points

The insertion of measuring points comprises two steps: preparing measuring points and submitting them. The REST call "/measuringpoint/prepare" puts a single measuring point into a queue, while the method "/measuringpoint/submit" finally inserts all queued measuring points into the database. This allows for efficient bulk insertion of large sets of measuring points and is useful if measuring points are inserted from already collected data from another data source. If measuring points are inserted from a live system the submit call may be sent immediately after every single measuring point preparation.

```
<REST END POINT>/measuringpoint/prepare \
?trajectory=<trajectory name> \
&name=<attribute name>&value=<value> \
&time=YYYY.MM.DD~HH:mm:ss \
&type=<attribute type>
```

```
<REST END POINT>/measuringpoint/submit
```

3.4.2 Listing Trajectory Attributes

Using the REST call "/vectors/periods" we obtain a time-ordered list of time periods and the values of the attributes for these periods.

```
<REST END POINT>/vectors/periods \
?trajectory=<trajectory name>
&begin=YYYY.MM.DD~HH:mm:ss
&end=YYYY.MM.DD~HH:mm:ss
```

The result of the REST call is an XML document. Each time period is defined as a vector containing the values of its attributes as they are valid for the according period. The vectors are chronologically ordered by their time periods. Basically, this vector output already represents a trajectory as interconnected segments. However, the attribute values are not interpolated in this output. The example below is an excerpt of the data from the MediaWiki example I presented in section 2.2.1 (Worldlines of Learning). The "view" attribute holds the numeric id of the latest Wiki article a student has viewed. The "edit" attribute indicates which article a student has edited. The student of the example below has swapped between viewing and editing the same article (identified by its id 47) which lead to the following data:

```
<hypercubedb>
  <vector begin="2017/1/11~11:58:37"
          end="2017/1/11~12:59:51">
          <attribute name="view" value="47"/>
  </vector>
  <vector begin="2017/1/11~12:59:51"
          end="2017/1/11~13:0:59">
    <attribute name="view" value="47"/>
    <attribute name="edit" value="47"/>
  </vector>
  <vector begin="2017/1/11~13:0:59"
```

```
                end="2017/1/11~20:38:11">
    <attribute name="view" value="47"/>
    <attribute name="edit" value="47"/>
  </vector>
  <vector begin="2017/1/11~20:38:11"
                end="2017/1/11~20:38:49">
    <attribute name="view" value="47"/>
    <attribute name="edit" value="47"/>
  </vector>
  <vector begin="2017/1/11~20:38:49"
                end="forever">
    <attribute name="view" value="47"/>
    <attribute name="edit" value="47"/>
  </vector>
</hypercubedb>
```

Using the REST method "/vectors/csv", a trajectory can also be obtained in the form of CSV data. This method takes the same parameters as above.

```
<REST END POINT>/vectors/csv \
?trajectory=<trajectory name> \
&begin=YYYY.MM.DD~HH:mm:ss \
&end=YYYY.MM.DD~HH:mm:ss
```

The output is provided as a comma-separated list, with each row representing an attribute and the last row listing the beginning of the according time period in the form a UNIX time stamp. This output type can be used for further processing. For example figure 2.6 on page 45 was generated with such CSV data using the programming language "R".

```
stud15_view,47,47,47,...
stud15_edit,47,47,47,...
stud15_time,1484135991,1484136059,1484163491,...
```

3.4.3 Visualizing Vectors

In section 2.2.1 (Worldlines of Learning) I provided multiple examples of trajectory visualizations using a color map technique. These images are generated using the "graphics" REST method. Each method call generates such a graphical visualization for one single trajectory.

```
<REST END POINT>/graphics \
?trajectory=<trajectory name> \
&begin=YYYY.MM.DD~HH:mm:ss \
&end=YYYY.MM.DD~HH:mm:ss \
&width=<image width in px> \
&ranges=<range list>
```

The value for the "ranges" parameter is a list of attributes and their value ranges:

```
&ranges=view=0-45;edit=0-45
```

This parameter is used to define the color range of the color map for the respective attribute. If a range for an attribute is not specified, the system will take the minimum and maximum value of the attribute as it is stored in the database and distribute the colors over that range. If attributes of multiple trajectories are to be compared, the range parameter should be specified to obtain the same color range for each trajectory.

3.4.4 Grid Index Creation

The REST method "/trajectories/index/create" is used to construct a grid index over existing trajectories. The "grids" parameter determines how many intervals on each dimension shall be used. The number of created cells equates to n^g with n denoting the number of dimensions and g denoting the value for the "grids" parameter. In this version of the implementation the extend of a grid cell depends on the distribution of the data on each dimension. If square-shaped cells are required, the data should be normalized before insertion takes place.

```
<REST END POINT>/trajectories/index/create \
?dimensions=<attribute names> \
&grids=<interval count>
```

3.4.5 Executing Spatio-Temporal Queries

The REST API provides an end point to formulate range and k nearest neighbor queries. The queries are pseudo-SQL statements which means that they are not based on a fully-developed database language. They are rather function calls translated into an SQL-like structure. The rationale behind is that in future work statements of such queries should be integrated in a complete spatio-temporal query language.

A spatio-temporal range query takes as its arguments a time period, a comma-separated list of trajectories and a "where" clause which contains a conjunction of ranges for each dimension. If the trajectory list is left empty the query refers to all trajectories stored in the database. Otherwise the search is restricted to this set of trajectories.

```
VALIDTIME
PERIOD[YYYY.MM.DD~HH:mm:ss-YYYY.MM.DD~HH:mm:ss)
SELECT TRAJECTORIES[<trajectory list>]
WHERE <attribute name> BETWEEN(<lower>,<upper>)
AND <attribute name> BETWEEN...
```

The arguments of a nearest neighbor query include a trajectory list which may be empty and the parameter "K" defining how many neighbors are to be found. The "where" clause contains a conjunction of query points.

```
SELECT NEAREST NEIGHBOR[<trajectory list>](<K>)
WHERE NEARTO{<query point>}
AND NEARTO{...
```

A query point must be defined by its coordinates in the Cognitive Spacetime and may include an optional time period for which this query point is valid. The listing below gives an example of a query point including a time period.

```
SELECT NEAREST NEIGHBOR[](3)
WHERE NEARTO {
  PERIOD[2017.01.23~12:00:00-2017.01.28~11:30:00);
  edit=47;
  view=35
}
AND NEARTO{...
```

3.5 Database Implementation

The Hypercube Database uses TimeDB for temporal indexing. TimeDB is a fully functional temporal database using the ATSQL query language [15, 16, 48, 62, 104]. ATSQL is designed as a super set of SQL, which makes it possible to combine temporal and non-temporal relations and queries. In contrast to standard SQL, ATSQL provides temporal predicates and operators to work on time periods. The TimeDB structure of the Hypercube Database fulfills two functions: storing arbitrary measuring points to temporal attributes and mapping them to vectors and trajectory segments.

3.5.1 Entity Relationship Model

The TimeDB database itself does actually not store trajectory segments as real geometric objects. Instead it links attributes to the trajectories they belong to. It uses time periods to define which parts of the trajectory form a segment.

The entity relationship model of the TimeDB structure is shown by figure 3.9. A trajectory has multiple attributes. An attribute is identified by its name and the trajectory it belongs to. If the same data is measured for several individuals, they will have attributes with the same name but each of them is associated with a unique trajectory. An attribute has also a specified type which may be either string, boolean, double or integer. An attribute has multiple values which are distinguished by consecutive time periods for which they are valid.

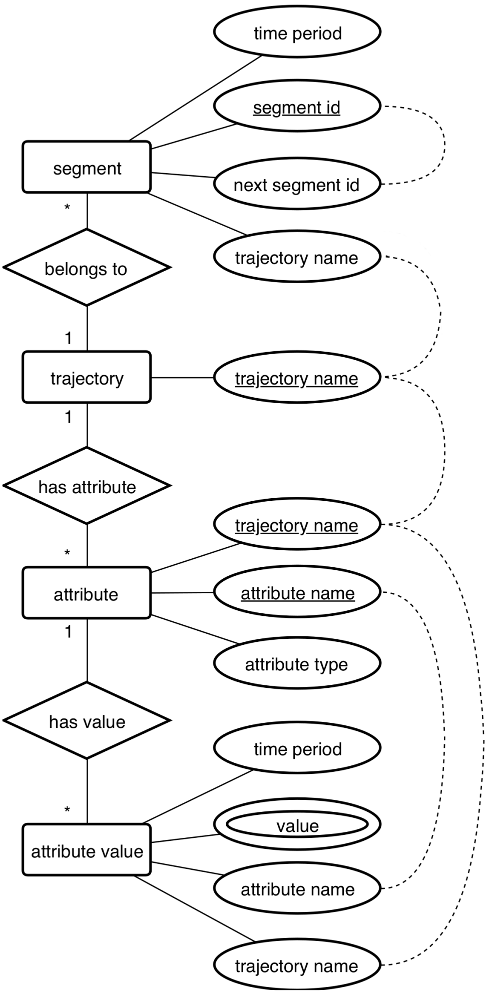


Figure 3.9: E-R model of the TimeDB structure

The value of an attribute consists of different representations which are string, boolean, double or integer. The type of the attribute determines which representation is the primary one.

At the highest level there are segments which represent subsequent time periods of a trajectory. The segment relation only contains the information which parts of a trajectory are valid for certain time periods and which segment succeeds another one. This means that the geometric information about trajectories is not contained in the database explicitly. The information rather concludes from a specific linking of data with time periods.

The dotted lines in figure 3.9 indicate foreign key relations. An attribute value is linked with its attribute by the keys "attribute name" and "trajectory name". An attribute and a segment are linked with trajectories by the key "trajectory name". Moreover, a segment is linked to its temporal successor by the "next segment id" key. Note that subsequent segments could also be identified by their time periods. However, linking consecutive segments is easy to implement and more efficient.

3.5.2 ATSQL Representation

In the following I show how specific database queries and structures are implemented using the ATSQL language.

Table Construction

The listings below show how the tables "trajectory", "attribute", "attributevalue" and "segments" are constructed. Only "attributevalue" and "segment" are temporal tables as they include time periods. In ATSQL time periods are not created as explicit columns. Instead a temporal table is created using the key phrase "AS VALIDTIME" concluding the

SQL definition. The ATSQL statements use the following abbriviations: "aname" (attribute name), "tname" (trajectory name) and "atype" (attribute type).

```
CREATE TABLE trajectory(tname VARCHAR(200));

CREATE TABLE attribute(
        aname VARCHAR(200),
        tname VARCHAR(200),
        atype VARCHAR(30)
);

CREATE TABLE attributevalue(
        aname VARCHAR(200),
        tname VARCHAR(200),
        doublevalue NUMBER,
        intvalue INTEGER,
        stringvalue VARCHAR(300),
        boolvalue NUMBER(1)
) AS VALIDTIME;

CREATE TABLE segment(
        id VARCHAR(200),
        next_id VARCHAR(200),
        tname VARCHAR(200)
)AS VALIDTIME;
```

Insertion of Measuring Points

Trajectories and their attributes do not have to be configured. Instead, trajectories and their attributes are created in the database when names appear in a REST call for the first time. The first insertion of a measuring point belonging to a specific attribute of a trajectory at time t_0 will create an attribute with the time period $[t_0, forever)$. Any following measuring point causes a splitting of the vector that embeds the measuring point. The embedding vector is deleted and replaced by two consecutive vectors. The measuring points do not have to be inserted in chronological order, it is the definition of the time point in the REST call, which matters. Consider the following example:

1. The first measuring point is inserted at t_0 which results in a single attribute value which is valid for $[t_0, forever)$.
2. The second measuring point is inserted at $t_2 > t_0$ leading to two attribute values for $[t0, t2)$ and $[t2, forever)$.
3. The third measuring point is inserted at t_1 with $t_0 < t_1 < t_2$ and the result is three vectors for the time periods $[t_0, t_1)$, $[t_1, t_2)$, $[t_2, forever)$.

The ATSQL statement for a single measuring point insertion is listed below:

```
VALIDTIME
PERIOD [YYYY.MM.DD~HH:mm:ss-YYYY.MM.DD~HH:mm:ss)
INSERT INTO attributevalue VALUES(
  '<attribute name>','<trajectory name>',
  <double val>,<int val>,'<string val>',<bool val>,
  <internal id>);
```

Trajectory Segmentation

The "attributevalue" table stores single attributes according to their valid time periods. A trajectory comprises several time lines of such attributes. A segment of a trajectory is defined by the smallest time interval which does not intersect or contain a time period of other attributes belonging to the same trajectory, which is again illustrated in figure 3.10.. These time intervals are stored in the "time period" field in the "segment table".

Mathematically this segmentation can be expressed as follows. Let T denote a trajectory. Let a_i be the i^{th} attribute of T. Let $m_{a_i j}$ be the j^{th} measuring point of a_i and let $t(m_{a_i j})$ be the respective time point. Finally, let $S(T)$ denote the "segmentation" of T over its attributes a_i. A segmentation

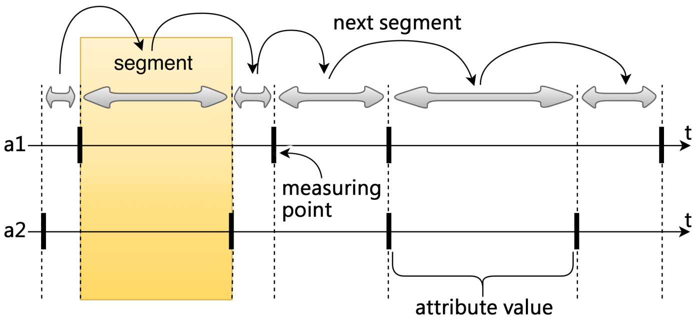


Figure 3.10: Transformation of vectors to segments

over the attributes of a trajectory is defined as the sequence of consecutive time intervals $\Phi_k = [\phi_{k_{start}},\ \phi_{k_{end}})$ that result from dividing the time line by all measuring points m_{a_ij} of all attributes a_i:

$$\begin{aligned} S(T) = \quad & \{[t(m_{a_cd}), t(m_{a_ef})) \mid t(m_{a_cd}) < t(m_{a_ef}) \ \bigwedge \\ & \nexists\ m_{a_gh} : t(m_{a_cd}) < t(m_{a_gh}) < t(m_{a_ef})\} \end{aligned}$$

Basically a database query language like SQL or ATSQL works on algebra of sets and first order logic. The expression above can therefore be transformed into an ATSQL statement yielding the lower and upper bounds of the respective time intervals. The result of this query is inserted into the "segment" table in the form of time periods. The ATSQL query below performs the desired segmentation of a trajectory. This query also illustrates how temporal predicates and operators are used with the ATSQL language.

```
NONSEQUENCED VALIDTIME SELECT
BEGIN(VALIDTIME(a)), BEGIN(VALIDTIME(b))
FROM attributevalue AS a, attributevalue AS b
WHERE
```

```
  a.tname='<trajectory name>' AND
  b.tname='<trajectory name>' AND
  BEGIN(VALIDTIME(a)) PRECEDES
  BEGIN(VALIDTIME(b)) AND
  NOT EXISTS (
    VALIDTIME SELECT
    BEGIN(VALIDTIME(z))
    FROM attributevalue AS z
    WHERE
      z.tname='<trajectory name>' AND
      BEGIN(VALIDTIME(a)) PRECEDES
      BEGIN(VALIDTIME(z)) AND
      BEGIN(VALIDTIME(z)) PRECEDES
      BEGIN(VALIDTIME(b))
);
```

3.5.3 Segment Interpolation

The database itself does not store interpolated trajectory segments. Interpolation of segments is done at run time in specific scenarios. When a grid index is created, segments may intersect the boundaries of cells. These segments have to be split at the points of intersection. For these intersection points, interpolated data is needed. Another scenario is the refinement step of a spatio-temporal query. In the preliminary step the spatio-temporal query uses information from the grid index telling only which segments are located in the affected cells of the query.

In the refinement step, exact distances between query objects and segments are calculated. For this calculation the concrete geometric position of the segments is taken which requires interpolated data. For the interpolation of a segment, the coordinates of its start and end point are obtained from the "attributevalue" table. Then a linear interpolation of the attribute values is performed.

Let $a_i(t)$ denote the non-interpolated value of an attribute belonging to a single trajectory at a specific time point t. Let further be $Int(a_i(t))$ the linearly interpolated value. In order to calculate $Int(a_i(t))$ we first determine the starting and ending time point of the segment which contains the requested time point t. Let them be denoted by t_{start} and t_{end}. They are obtained from the "attributevalue" table. Note that in TimeDB the value of an attribute at its ending time point equates to the value of the successive time period because TimeDB works with time intervals which are open on the right.

First we calculate the slope of the interpolation function:

$$slope(a_i(t)) = \frac{a_i(t_{end}) - a_i(t_{start})}{t_{end} - t_{start}}$$

Finally, we get the interpolated value:

$$Int(a_i(t)) = a_i(t_{start}) + slope(a_i(t)) \cdot (t - t_{start})$$

3.5.4 Grid Index

The grid index is created independently from the underlying data. In the first place only measuring points are inserted into the database. The grid index can be created afterwards at any point in time. Consider the database structure as it was explained in section 3.5.1 (Entity Relationship Model). The information about trajectory segments is stored in the "segment" table containing time periods and segment ids. This information is only linked with the data stored in the "attributevalue" table. This way the entire segmentation and hence all information about trajectories is stored independently from the originally inserted data. This allows us to first collect data and then decide about the granularity of the grid index based on the distribution of the data. In addition, it would also be possible to first create the index and to put the respective segments into the grid at the moment when measuring points are inserted. However, this was not implemented in the Hypercube Database prototype because it was designed as an experimental system.

The grid is created by subdividing the entire value range on each dimension into an equal number of intervals. If the data is not distributed equally on each dimension — which is very likely — we obtain a grid in the form of hyper-rectangulars. To obtain a square-shaped grid, the data should be normalized on each dimension before the grid is created. An advanced version of the system might also allow for defining interval numbers and lengths on each dimension separately.

Let n be the number of dimensions, let m denote the number of intervals on a dimension and let l_i denote the length of an interval on the i^{th} dimension. The intervals along a single dimension are enumerated from 1 to m. Suppose there is a n-dimensional vector $v(t)$ which is defined by its attribute values:

$$v(t) = \begin{pmatrix} a_1(t) \\ a_2(t) \\ \vdots \\ a_n(t) \end{pmatrix}$$

From the vector v we can directly compute the coordinates of the grid cell $c(t)$, in which this point is located.

$$c(t) = \begin{pmatrix} \lceil a_1(t)/l_1 \rceil \\ \lceil a_2(t)/l_2 \rceil \\ \vdots \\ \lceil a_n(t)/l_n \rceil \end{pmatrix}$$

Note that although the time parameter t is used here, the grid index implements only spatial indexing whereas time is managed by the TimeDB back end. When a segment is inserted into the grid index, we determine the spatial coordinates of its start and end point. The TimeDB back end uses time periods in the form $[t_{start}, t_{end})$ which are open on the right. Therefore, the ending coordinates of a segment equate to the starting coordinates of its successive segment. Using the above calculation for $c(t)$ the cell coordinates for both the start and end point of the segment are computed. If both the starting and the ending coordinates are located in the same cell, the segment id from the "segment" table is associated with this cell. If the segment intersects two or more cells it is split into child segments. For this purpose the attributes of the segment are linearly interpolated at the cell boundaries and their respective time points are deduced. These time points are used to update the "segment" table inserting the time periods of the child segments. These child segments are assigned new ids which are associated with the affected grid cells.

Searching the grid index for segments is straight forward. Remember that the Hypercube Database uses range queries as the basis for more sophisticated queries. If the search window of a range query has equal lengths on each dimension and if we use the Chebyshev distance, we obtain a circular query. Nearest neighbor queries are then composed of incremental circular queries. Therefore it is sufficient to define a simple range query as the other queries are built from it and behave in an analogue way.

A range query is constrained by its lower and upper boundary on each dimension. Again, let l_i denote the length of an interval on the i^{th} dimension. Let $range_{i,lower}$ denote the lower and let $range_{i,upper}$ denote the upper boundary of the query range with respect to the i^{th} dimension in the Cognitive Spacetime. From this information we can immediately calculate the number c_i of the affected lower and upper interval.

$$\begin{aligned} c_{i,lower} &= \lceil range_{i,lower}/l_i \rceil \\ c_{i,upper} &= \lceil range_{i,upper}/l_i \rceil \end{aligned}$$

From this calculation we obtain the set $\mathfrak{C}_i$ containing all affected interval numbers for the i^{th} dimension:

$$\mathfrak{C}_i = \{c_i \in \mathbb{Z} \mid c_{i,lower} \leq c_i \leq c_{i,upper}\}$$

Now, let $\mathfrak{R}$ denote the set containing all n-dimensional grid cells which are covered by the range query. $\mathfrak{R}$ is computed as the cartesian product of all interval number sets $\mathfrak{C}_i$:

$$\mathfrak{R} = \bigtimes_{i=1}^{n} \mathfrak{C}_i$$

An element of $\mathfrak{R}$ is an ordered n-dimensional tuple of interval numbers c_i. Each of these tuples addresses a specific grid cell that is affected by the query range. If the query window affects many intervals on multiple dimensions, the cardinality of $\mathfrak{R}$ can become extremely large which can even lead to combinatorial explosion. However, this problem is easy to solve. As I have explained in section 3.3.2 (Spatial Indexing) an improvement is achieved with a hierarchical grid index with multiple levels of granularity. Depending on the extend of query ranges either a grid with lower or higher granularity can be chosen.

4 Summary and Conclusion

In this work I have introduced the Cognitive Spacetime model which is a novel method to make learning behavior measurable and analyzable. A dominant problem in learning analytics has always been the fact that we are faced with complex data from heterogeneous sources. With Cognitive Spacetime, such data can be modeled as spatio-temporal trajectories in an abstract high-dimensional space. The problem of analyzing learning histories is reduced to a purely geometric problem which can be solved on the basis of very simple distance measures whereby similarities among learners are expressed by spatial and temporal nearness.

According algorithms for nearest neighbor searches and clustering are derived from the field of Geo Informational Systems. In section 2.3.2 (Spatio-Temporal Queries) I have developed a hierarchical framework of spatio-temporal queries which is easy to implement and can be adopted to multiple distance metrics and indexing techniques. Remember that the basic query of this framework is a simple range query on the top of which circular queries, nearest neighbor queries and clustering can be implemented. In case the distance metric is modified or the underlying spatial index structure is changed it is only the range query which has to be adapted. Therefore this framework offers a workable and extendable tool set for the implementation of a spatio-temporal database system. In particular, the proposed grid index structure is efficient for even high-dimensional spaces.

However, such a database system is not enough. We also need respective meta-data to build the Cognitive Spacetime in a way that can utilize spatio-temporal nearness as a useful metric. In section 2.2.3 (Learning Histories as Spatio-Temporal Trajectories) I introduced a guideline to annotate learning material in an appropriate way and I explained how the Cognitive Spacetime is created from this. Note that there is not one Cognitive Spacetime. Instead there are many possible Spacetimes depending on the set of meta data that is used. Which set of meta data is useful will strongly depend on context and purpose. Fortunately, collecting data and annotating them are two independent processes. We can first gather information about the content on which learners progress and annotate the information afterwards. This way we can construct multiple Cognitive Spacetimes for the same content and perform different analysis on them.

The Cognitive Spacetime has a particular characteristic: it does not describe learners in the first place. Instead it describes objects and abstract places with which learners interact. Therefore, we actually do not analyze learners themselves but the artefacts and traces they leave behind. This approach is not unusual. You can find it in cultural and social sciences and especially in archeology. We can learn a lot about people's history and minds by examining their legacies and records. Actually this is the preferred method if we can not talk directly to the respective persons. Cognitive Spacetime is therefore best understood as an artificially constructed place which makes learners leave some artefacts and traces which we can analyze.

Once a Spacetime is defined we are not limited to analyzing data. We can furthermore create adaptive learning environments which support the learner in an interactive style. In the introductory chapter under section 1.2 (Computability) I have discussed the essence of algorithms and the fact that basically they are no more than mechanical step-by-step procedures. However, I have also introduced the concept of the adaptive Turing machine which is absolutely sufficient to implement a mutually adaptive human-machine interaction. The Cognitive Spacetime model is perfectly suited to build such an adaptive system without the need for too exhaustive previous knowledge about a learner. Indeed, it enables us to implement adaptive systems which works on very fuzzy and incomplete knowledge.

In order to create a model of a learner there are initially two obvious solutions. First we can try to model a hypothetical learner by investigating learning behaviors and influencing factors as broadly as possible and create some kind of user model from which the system derives its instructional procedures. Second we might use deep learning algorithms to find patterns and perform probability-based forecasts for learning behavior. Both approaches require extensive gathering of data as a preliminary step. A third solution is in fact the Cognitive Spacetime model which does not require a too extensive previous knowledge.

It may occur that we do not know anything about a learner at the beginning. In this case an adaptive learning environment which is based on Cognitive Spacetime might recommend an initial learning pathway to the user. This initial pathway may be constructed by a teacher as a default version. Let us suppose that the learner is not forced but only recommended to follow the system's instructions and that the learning environment is open enough to support a constructivist learning style. If these two criteria are satisfied, the learner is free enough to deviate from the system's recommendations. The deviations will then feed the mutual adaptivity cycle and provoke according updates of the knowledge base incorporated by the system. Concretely, this means that the learner's trajectory moves away from the initial pathway but it gets closer to other trajectories in the Cognitive Spacetime. The adaptive learning environment will then generate new recommendations. This is comparable to a GPS navigation system which does not force a driver to return to the original route at any cost. Instead, it calculates new alternative routes with every deviation from the initial route. In this concept, deviations of a learner are not a casual evil. Instead they are absolutely appreciated and needed to feed the adaptivity cycle. At the same time the deviating trajectory of this particular learner contributes to the growth of the knowledge base.

The mutual adaptivity cycle is also similar to how a teacher interacts with her students. The teacher may start from a stereotypical image of a student, providing a provisional orientation. Continuous feedback and interaction will then refine the teacher's and the student's image and behavior. The quality of teaching and learning therefore relies on the quality of this communication.

Remember the Turing test which I explained in section 1.5 (The Empathetic Algorithm). The Turing test is supposed to decide whether a machine can be considered "intelligent". The special feature of this test is its focus on the mere interaction between a human and the machine. The model of the adaptive Turing machine and its mutual interaction with the user reflects the concept of the Turing test. With the mutual adaptivity cycle "intelligence" is neither located with the user nor inside the algorithm. "Intelligence" in this sense is something that is constructed within the human-machine interaction. It does not exist in a physical place like the user or the machine.

Let me now sketch a scenario how the Cognitive Spacetime model could be used within an organization. Throughout this work I have used the term "Learning Object" (LO). A LO may be anything that can be regarded as an atomic entity of learning content. This may be a book, a pdf file, a video clip, an audio file or it may be a course lesson taking place at a physical location. It may even be a person having some specific knowledge or expertise. Generally speaking it may be any form of resource which provides people with information. In the world of the semantic web any resource has a unique identifier — usually encoded by a URI (Uniform Resource Identifier). In an organization, which may be a university, a company or anything else, resources of information may be provided by multiple sources. These may be web servers, document management systems, learning management systems, streaming platforms or catalogs and databases containing information about non-digital resources. It is insignificant by which source a resource is provided. Whenever such a resource can be identified by a URI it can also be described with any set of meta data.

In order to establish adaptive environments based on the Cognitive Spacetime model we need the following: first we need an instance of the Hypercube Database (HCDB). Second we need a semantic layer which enriches the resource identifiers (URIs) with appropriate meta data. Third we need a notification mechanism to inform the HCDB instance when someone accesses a resource together with the meta data of that resource.

The semantic layer may be a central or a distributed system which only links the URIs with meta data. One possible solution may be one central or multiple instances of a Semantic MediaWiki. This layer holds the URIs of the LOs wherever they may be located. Within the MediaWiki these LOs have to be annotated with meta data which form the dimensions of the Cognitive Spacetime. The advantage of using Semantic MediaWiki is that its content and semantic annotations are both human-readable and machine-processable.

Now suppose that data sources like a web server, a document management systems, a learning management system or any other type of platform implement an intermediate process to send notifications to the HCDB instance. The intermediate process first contacts the responsible MediaWiki instance to retrieve the meta data associated with the respected URI. The MediaWiki will hold only LO-specific meta-data. If learner-specific meta-data is to be included, the intermediate process has to add this information. Then the intermediate process notifies the HCDB instance using the REST interface which I described in section 3.4 (REST API). Besides the meta-data this notification must contain a unique identifier of the person who accesses the resource. The identifier may be a pseudonym to protect privacy.

Technically, this implementation is pretty clear. However, a fundamental requirement is the definition of a meta data set which satisfies the conditions I defined in section 2.2.3 (Learning Histories as Spatio-Temporal Trajectories). Although I provided some experimental data in section 2.2.1 (Worldlines of Learning) to illustrate the concept of spatio-temporal nearness, there is no universal advice for the creation of an appropriate meta data set. The work I present here is to be understood as a theoretical foundation. The development of good meta data sets as well as their empirical proof must therefore be the issue of future work.

Furthermore, the Hypercube Database prototype is an experimental system which served as a proof-of-concept and a test setup to validate the discussed data structures and algorithms. It is — in my opinion — not yet a fully-operable database system. To enhance the HCDB prototype it should be programmed more closely to the underlying hardware. This means in particular that the indexing technique should consider the way data is physically stored on the hardware in order to optimize access speed. Even more improvements could be achieved by more sophisticated parallelization techniques. Furthermore, the HCDB prototype does not provide a real query language. It only implements a pseudo language for range and nearest neighbor queries as I explained in section 3.4.5 (Executing Spatio-Temporal Queries). A possible enhancement might consist of integrating a range and nearest neighbor query syntax into the ATSQL query interface.

Bibliography

[1] T. Abraham and J. F. Roddick. Survey of spatio-temporal databases. *Geoinformatica*, 1999.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms*. Springer Berlin Heidelberg, 1993.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, 1998.

[4] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, 1999.

[5] G. Antoniou, E. Franconi, and F. van Harmelen. Introduction to semantic web ontology languages. In *Reasoning Web*, 2005.

[6] J. B. Arbaugh and Raquel Benbunan-Fich. The importance of participant interaction in online environments. *Decision Support Systems*, 2007.

[7] L. Balasubramanian and M. Sugumaran. A state-of-art in r-tree variants for spatial indexing, 2012.

[8] O. Balovnev, M. Breunig, A. B. Cremers, and S. Shumilov. Extending geotoolkit to access distributed spatial data and operations. In *Proceedings of the 12th International Conference on Scientific and Statistica Database Management*, 2000.

[9] S. Bechhofer et al. Owl web ontology language reference.

[10] L. T. Benjamin. A History of Teaching Machines. *American Psychologist*, 1988.

[11] S. Berchtold, D. A. Keim, and H. Kriegel. The x-tree: An index structure for high-dimensional data. In *Proceedings of the 22th International Conference on Very Large Data Bases*, 1996.

[12] W. Bhuasiri, O. Xaymoungkhoun, H. Zo, J. J. Rho, and A. P. Ciganek. Critical success factors for e-learning in developing countries: A comparative analysis between ict experts and faculty. *Computers & Education*, 2012.

[13] M. Böhlen. Managing temporal knowledge in deductive databases, 1994. Dissertation submitted to the Swiss Federal Institute of Technology Zurich.

[14] M. H. Böhlen. Temporal database system implementations. *SIGMOD Rec.*, 1995.

[15] A. Carvalho, C. Ribeiro, and A. Sousa. Spatial timedb - valid time support in spatial dbms. In *Proceedings of the 2nd International Advanced Database Conference*, 2006.

[16] A. Carvalho, C. Ribeiro, and A. Sousa. A spatio-temporal database system based on timedb and oracle spatial. In *Research and Practical Issues of Enterprise Information Systems*. Springer US, 2006.

[17] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with seti. In *Proceedings*

of the Conference on Innovative Data Systems Research (CIDR), 2003.

[18] M. A. Cheema, , Y. Yuan, and X. Lin. Circulartrip: An effective algorithm for continuous knn queries. In *Advances in Databases: Concepts, Systems and Applications*. Springer Berlin Heidelberg, 2007.

[19] M. A. Cheema. *CircularTrip and ArcTrip: Effective Grid Access Methods for Continuous Spatial Queries*. PhD thesis, 2007.

[20] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, Yu Zheng, and Xing Xie. Searching trajectories by locations: An efficiency study. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010.

[21] J. Chomicki, D. Toman, and M. H. Böhlen. Querying atsql databases with temporal logic. *ACM Transactions on Database Systems (TODS)*, 2001.

[22] A. Church. A note on the entscheidungsproblem. *The Journal of Symbolic Logic*, 1936.

[23] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 1936.

[24] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 1940.

[25] C. Combi et al. Querying temporal clinical databases with different time granularities: the gch-osql language. In *Proceedings of the Annual Symposium on Computer Applications in Medical Care*, 1995.

[26] V. T. de Almeida, R. H. Güting, and T. Behr. Querying moving objects in secondo. In *7th International Conference on Mobile Data Management (MDM'06)*, 2006.

[27] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 2008.

[28] E Delen, J. Liew, and V. Willson. Effects of interactivity and instructional scaffolding on learning: Self-regulation in online video-based environments. *Computers & Education*, 2014.

[29] V. Dimitrova and M. Lüdmann. *Zur Entwicklung der Fähigkeit zur Perspektivenübernahme*. Springer, 2014.

[30] H.C. Doets and van Eijck D.J.N. *The Haskell Road to Logic, Maths and Programming.* College Publication, 2012.

[31] W. Dreyer, A. Kotz Dittrich, and D. Schmidt. Using the calanda time series management system. *SIGMOD '95 Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, 1995.

[32] A. G. Emslie. An efficient indexing structure for trajectories in gis. 2007.

[33] L. Ertoz, M. Steinbach, and V. Kumar. A new shared nearest neighbor clustering algorithm and its applications. In *Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining*, 2002.

[34] M. Erwig. Design of spatio-temporal query languages. In *NSF/BDEI Workshop on Spatio-Temporal Data Models of Biogeophysical Fields for Ecological Forecasting*, 2002.

[35] M. Erwig and M. Schneider. STQL — a spatio-temporal query language. In *Mining Spatio-Temporal Information Systems*, 2002.

[36] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial

databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[37] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Nearest neighbor search on moving object trajectories. In *Advances in Spatial and Temporal Databases.* Springer Berlin Heidelberg, 2005.

[38] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Algorithms for nearest neighbor search on moving object trajectories. *GeoInformatica*, 2007.

[39] K. Fuchs and P. A. Henning. *Computer-Driven Instructional Design with Intuitel: An Intelligent Tutoring Interface for Technology-Enhanced Learning.* River Publishers Series in Innovation and Change in Education, 2017.

[40] K. Fuchs and P. A. Henning. Cognitive space time: A model for human-centered adaptivity in e-learning. In *Proceedings of the 24th IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), 2018*, 2018.

[41] K. Fuchs, P. A. Henning, and M. Hartmann. Intuitel and the hypercube model – developing adaptive learning environments. *Journal on Systemics, Cybernetics and Informatics: JSCI, 14(3)*, 2016.

[42] Y. Gao, C. Li, G. Chen, L. Chen, X. Jiang, and C. Chen. Efficient k-nearest-neighbor search algorithms for historical moving object trajectories. *Journal of Computer Science and Technology*, 2007.

[43] K. Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für Mathematik 38(1)*, 1931.

[44] L. R. Goldberg. The structure of phenotypic personality traits. *American Psychologist*, 1993.

[45] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. dedale, a spatial constraint database. In *Database Programming Languages*. Springer Berlin Heidelberg, 1998.

[46] S. Grumbach, P. Rigaux, and L. Segoufin. Efficient multi-dimensional data handling in constraint databases, 1998.

[47] S. Grumbach, P. Rigaux, and L Segoufin. On the orthographic dimension of constraint databases. In *Database Theory — ICDT'99*. Springer Berlin Heidelberg, 1999.

[48] H. Guo, Y. Tang, X. Yang, and X. Ye. *Improvement and Extension to ATSQL2*. Springer Berlin Heidelberg, 2010.

[49] R. H. Güting, T. Behr, V. Almeida, Z. Ding, F. Hoffmann, and M. Spiekermann. Secondo: An extensible dbms architecture and prototype. Technical report, FernUniversität in Hagen, 2004.

[50] R. H. Güting, T. Behr, and J. Xu. Efficient k-nearest neighbor search on moving object trajectories. *The VLDB Journal*, 2010.

[51] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, E. Nardelli, M. Schneider, and J. R. R. Viqueira. Spatio-temporal models and languages: An approach based on data types. In *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, 2003.

[52] R.H. Güting and M. Schneider. *Moving Objects Databases.* Morgan Kaufmann Publishers, 2005.

[53] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM*

SIGMOD International Conference on Management of Data, 1984.

[54] M. Hasan, Muhammad A. C., W. Qu, and X. Lin. Efficient algorithms to monitor continuous constrained k nearest neighbor queries. In *Database Systems for Advanced Applications*. Springer Berlin Heidelberg, 2010.

[55] P. A. Henning, F. Heberle, A. Streicher, A. Zielinski, C. Swertz, J. Bock, and S. Zander. Personalized web learning: Merging open educational resources into adaptive courses for higher education. In *UMAP Workshops*, 2014.

[56] P. A. Henning et al. Intuitel – intelligent tutorial interface for technology enhanced learning. In *Proceedings of the 22nd UMAP Conference*, 2014.

[57] P. A. Henning et al. Learning pathway recommendation based on a pedagogical ontology and its implementation in moodle. In *Proceedings of the DeLFI 2014 conference, GI Lecture Notes in Informatics*, 2014.

[58] D. Hilbert and W. Ackermann. *Grundzüge der theoretischen Logik*. Springer, 1928.

[59] D. Hilbert and W. Ackermann. *Principles of mathematical logic*. Chelsea Pub. Co., 1950.

[60] I. Horrocks, B. Motik, and Z. Wang. The hermit owl reasoner. In *ORE*, 2012.

[61] R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, 1973.

[62] C. S. Jensen. Temporal database management, 2000. Dissertation.

[63] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *Advances in*

Spatial and Temporal Databases. Springer Berlin Heidelberg, 2005.

[64] V. Kalyan-Masih. Cognitive egocentricity of the child within piagetian developmental theory. *Transactions of the Nebraska Academy of Sciences*, 1973.

[65] S. Kisilevich, F. Mansmann, M. Nanni, and S. Rinzivillo. *Spatio-temporal clustering.* Springer US, 2010.

[66] A. Y. Kolb and D. A. Kolb. The kolb learning style inventory—version 3.1 2005 technical specifications, 2005.

[67] D. A. Kolb. *Experiential learning: Experience as the source of learning and development.* PrenticeHall, 1984.

[68] B. Kosko. Fuzziness vs. probability. In *International Journal of General Systems*, 1990.

[69] J. Lee, J. Han, and K. Whang. Trajectory clustering: A partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 2007.

[70] Torsten Leidig. L3 – towards an open learning environment. *Journal on Educational Resources in Computing (JERIC)*, 2001.

[71] S. Liaw. Investigating students' perceived satisfaction, behavioral intention, and effectiveness of e-learning: A case study of the blackboard system. *Computers & Education*, 2008.

[72] J. Limberg and R. Seising. Representing fuzzy sets in the hypercube part ii: Developments in the life sciences. *2007 2nd International Workshop on Soft Computing Applications*, 2007.

[73] C. Manolis, D. J. Burns, R. Assudani, and R. Chinta. Assessing experiential learning styles: A methodological

reconstruction and validation of the kolb learning style inventory. *Learning and Individual Differences*, 2013.

[74] M. R. Martinez-Torres, S. L. Toral Marin, F. Barrero Garcia, S. Gallardo Vazquez, M. Arias Oliva, and T. Torres. A technological acceptance of e-learning tools used in practical and laboratory teaching, according to the european higher education area. *Behaviour & Information Technology*, 2008.

[75] N. Meder. *Didactic requirements of learning environments: the webdidactics approach of L3.*

[76] H. Minkowski. *Raum und Zeit.* 1909.

[77] H. Minkowski and V. (ed.) Petkov. *Minkowski's papers on relativity.* Minkowski Institute Press, 2012.

[78] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 1983.

[79] F. Murtagh and P. Contreras. Methods of hierarchical clustering. *CoRR*, 2011.

[80] M. Nanni and D. Pedreschi. Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems*, 2006.

[81] M. Neteler, M. Hamish Bowman, M. Landa, and M. Metz. Grass gis: A multi-purpose open source gis. *Environmental Modelling & Software*, 2012.

[82] N. Pelekis, I. Kopanakis, G. Marketos, I. Ntoutsi, G. Andrienko, and Y. Theodoridis. Similarity search in trajectory databases. In *Temporal Representation and Reasoning, 14th International Symposium on*, 2007.

[83] Stephen Petrina. Sidney Pressey and the Automation of Education, 1924-1934. *Technology and Culture*, 2004.

[84] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000.

[85] J. Piaget and B. Inhelder. *The Psychology Of The Child.* 1972.

[86] V. B. Surya Prasath, Haneen Arafat Abu Alfeilat, Omar Lasassmeh, and Ahmad B. A. Hassanat. Distance and similarity measures effect on the performance of k-nearest neighbor classifier - a review. *CoRR*, 2017.

[87] S. L Pressey. A Machine for Automatic Teaching of Drill Material. *School & Society*, 1927.

[88] L. Relly, H.-J. Schek, O. Henricsson, and S. Nebiker. Physical database design for raster images in concert. In *Advances in Spatial Databases.* Springer, 1997.

[89] L. Relly, H. Schuldt, and H. Schek. Exporting database functionality - the concert way. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1998.

[90] P. Rigaux, M. Scholl, L. Segoufin, and S. Grumbach. Building a constraint-based spatial database system: model, languages, and implementation. *Information Systems*, 2003.

[91] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, 1995.

[92] N. L. Sarda. Design of an information system using a historical database management system. In *Proceedings of the 8th. Annual International Conference on Information Systems*, 1987.

[93] N. L. Sarda. Extensions to sql for historical databases. *IEEE Transactions on Knowledge and Data Engineering*, 1990.

[94] D. Schmidt et al. Calanda - a complete solution for time series management in banking, 1995. UBS, Zurich.

[95] R. Schulmeister. Students, internet, elearning and web 2.0.

[96] R. Schulmeister. *Kognitive Heterogenität von Studierenden.* Beltz, 1985.

[97] R. Schulmeister. *Diversität von Studierenden und die Konsequenzen für eLearning.* Waxmann, 2004.

[98] R. Seising. Representing fuzzy sets in the hypercube part i: A historical note. *2007 2nd International Workshop on Soft Computing Applications*, 2007.

[99] H. M. Selim. Critical success factors for e-learning acceptance: Confirmatory factor models. *Comput. Educ.*, 2007.

[100] R. L. Selman. *The Growth of Interpersonal Understanding: Developmental and Clinical Analyses.* Acadmeic Press, Inc., 1980.

[101] R. Shearer, B. Motik, and I. Horrocks. Hermit: A highly-efficient owl reasoner. In *OWLED*, 2008.

[102] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2007.

[103] B. F. Skinner. Teaching Machines. *Science*, 1958.

[104] A. Steiner. A generalisation approach to temporal data models and their implementations, 1998. Dissertation

submitted to the Swiss Federal Institute of Technology Zurich.

[105] Kurt Stocker. The theory of cognitive spacetime. *Metaphor and Symbol*, 2014.

[106] A. Streicher, F. Heberle, and B. Bargel. Intuitel - deliverable 3.2: Specification of the learning progress model. intuitel resources, 2013. visited on 2018-06-28.

[107] Suman and P. Rani. A survey on sting and clique grid based clustering methods. *International Journal of Advanced Research in Computer Science*, 2017.

[108] C. Swertz, A. Barberi, A. Forstner, A. Schmölz, P. A. Henning, and F. Heberle. A sms with a kiss. towards a pedagogical design of a metadata system for adaptive learning pathways. In *Proceedings of EdMedia: World Conference on Educational Media and Technology 2014*, 2014.

[109] C. Swertz et al. A pedagogical ontology as a playground in adaptive e-learning environments. In *Proceedings of INFORMATIK 2013, conference, GI Lecture Notes in Informatics*, 2013.

[110] A. u. Tansel. Temporal relational data model. *IEEE Trans. on Knowl. and Data Eng.*, 1997.

[111] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 1937.

[112] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society*, 1938.

[113] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX, 1950.

[114] E. Verdú, L. M. Regueras, M. J. Verdú, J. P. de Castro, D. Kohen-Vacs, E. Gal, and M. Ronen. Intelligent tutoring interface for technology enhanced learning in a course of computer network design. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, 2014.

[115] E. Verdú Pérez, J.P. De Castro Fernández, M.J. Verdú Pérez, L. Regueras Santos, and P.A. Henning. Intelligent tutoring interface for technology enhanced learning with moodle. In *EDULEARN13 Proceedings*, 5th International Conference on Education and New Learning Technologies, 2013.

[116] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings 18th International Conference on Data Engineering*, 2002.

[117] D. Šidlauskas, S. Šaltenis, C. W. Christiansen, J. M. Johansen, and D. Šaulys. Trees or grids?: Indexing moving objects in main memory. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2009.

[118] B. J. Wadsworth. *Piaget's theory of cognitive and affective development: Foundations of constructivism.* Longman Publishing, 1996.

[119] W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

[120] A. N. Whitehead and B. Russell. *Principia Mathematica.* Cambridge University Press, 1910.

[121] Y Xia, R Wang, X Zhang, and H. Y. Bae. Grid-based k-nearest neighbor queries over moving object trajecto-

ries with mapreduce. *International Journal of Database Theory and Application*, 2017.

[122] B. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings 14th International Conference on Data Engineering*, 1998.

[123] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 2013.

[124] A. Zielinski and J. Bock. A case study on the use of semantic web technologies for learner guidance. In *Proceedings of the 24th International Conference on World Wide Web*, 2015.

[125] G. Zimbrao, Jano. M. de Souza, and V. T. de Almeida. The temporal r-tree.