On the building of optimal binary trees for Spelling Interfaces

Dissertation

der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. Nat.)

vorgelegt von
Dipl.-Ing. Mikhail Tregoubov, M. Sc.
aus Moskau

Tübingen
2005

Tag der mündlichen Qualifikation:	19.10.2005
Dekan:	Prof. Dr. Michael Diehl
1. Berichterstatter:	Prof. Dr. Wolfgang Rosenstiel
2. Berichterstatter:	Prof. Dr. Niels Birbaumer

Widmung

Im Andenken an meinen Vater, Dr. Jouri Perelmouter, den besten Vater, den man sich wünschen kann.

Danksagung

Diese Arbeit entstand neben meiner Tätigkeit als Unternehmensberater der Accenture GmbH.

Ich möchte an dieser Stelle allen danken, die mich während dieser Zeit unterstützt und somit die Entstehung dieser Arbeit ermöglicht haben.

Besonders danke ich von meinem ganzen Herzen meinen Eltern, vor allem meinem Vater Dr. Jouri Perelmouter, dem ich diese Arbeit widmen möchte. Es waren meine Eltern, die mich bei der Erstellung dieser Arbeit sowie bei der Prüfungsvorbereitung stets unterstützt, ermutigt und gestärkt haben. Sie haben wahrhaftig diese Arbeit ermöglicht.

Mein Dank gilt in ganz besonderem Maße meinem Doktorvater Herrn Professor Dr. Wolfgang Rosenstiel. Ich möchte ihm für seine ganz hervorragende Betreuung und Unterstützung meiner Arbeit, Begutachtung meiner Dissertation sowie für sehr nützliche fachliche Diskussionen, wertvolle Kritik, Hilfsbereitschaft und organisatorische Hinweise danken.

Besonderer Dank gebührt auch meinem zweiten Gutachter Herrn Professor Dr. Niels Birbaumer, der mich insbesondere in der Anfangsphase der Arbeit sehr stark unterstützt hat. Er hat diese Arbeit mit seinen Ideen, seiner Erfahrung und vielen fruchtbaren Diskussionen immer wieder vorangetrieben und hat stets ein offenes Ohr für alle anfallenden Probleme gehabt.

Ferner danke ich Herrn Professor Dr. Klaus-Jörn Lange für seine Hilfsbereitschaft im Hinblick auf meine Prüfungsvorbereitung.

Stuttgart, Oktober 2005

Mikhail Tregoubov

TABLE OF CONTENTS:

TABLE OF FIGURES:

Zusammenfassung

Die nachfolgende Arbeit befasst sich mit den Problemen der nicht-fehlerfreien Kommunikation mit Hilfe der binären Kommunikationsgeräte, die insbesondere von den an dem sogenannten „locked-in“ Syndrom leidenden Patienten verwendet werden.

Die praktische Motivation für diese Arbeit besteht vor allem darin, die Lebensqualität für solche Patienten durch Verbesserung ihrer Kommunikationsmöglichkeiten zu erhöhen. Das „locked-in“ Syndrom wird in dem „*Pschyrembel Klinisches Wörterbuch“*, Seite 974, als „Unfähigkeit, sich bei erhaltenem Bewusstsein sprachlich oder durch Bewegungen spontan verständlich zu machen“ definiert. In den meisten Fällen wird dieses Syndrom durch eine vollständige oder fast vollständige Paralyse aufgrund eines Gehirn- oder Rückenmarkschadens, oder aber durch eine Erkrankung namens amyotrophe Lateralsklerose verursacht. Die meisten Patienten, die an dem „locked-in“ Syndrom im fortgeschrittenen Stadium leiden, verfügen über keine motorischen Fähigkeiten mehr und sind bei der Kommunikation mit ihrer Umwelt auf sogenannte Brain-Computer Interfaces (BCI’s) angewiesen, wobei man unter Brain-Computer Interfaces Geräte und Verfahren versteht, die für die Kommunikation zwischen dem menschliche Gehirn und einem Computer dienen. Solange die motorischen Fähigkeiten noch vorhanden sind, können sie zur Bedienung motorischer Kommunikationsgeräte eingesetzt werden.

Die Mehrheit der motorischen Kommunikationsgeräte und BCI’s funktionieren so, dass dem Patienten eine Sequenz von Zeichen und / oder Buchstaben angezeigt wird, aus der der Patient bestimmte Buchstaben auszuwählen versucht, um so zu gewünschten Worten und Phrasen zu kommen. In den meisten Fällen kann der Patient aufgrund seiner Behinderung lediglich angeben, ob er eine bestimmte Sequenz der Buchstaben in betracht ziehen möchte oder nicht, so dass man mit einem binären Signal „Ja/Nein“ zu tun hat und in diesem Zusammenhang auch von binären Spelling Interfaces spricht. Eine weitere Herausforderung stellt die Tatsache dar, dass die Patienten, die weder ihr Gehirn noch ihre verbleibende Motorik vollständig kontrollieren können, auch diese eingeschränkte Wahl nicht immer genau treffen und fehlerhaft eine falsche Sequenz auswählen können, was zu einer hohen Fehleranfälligkeit insbesondere im Falle der BCI’s führt. Die Verbesserung der Kommunikationsmöglichkeiten kann daher zweierlei erreicht werden: zum einen durch Beschleunigung der Kommunikationsgeschwindigkeit und zum anderen durch Erhöhung

der Fehlerfreiheit. Das Kapitel 3 dieser Arbeit beschäftigt sich mit einer ausführlichen Analyse der existierenden binären Spelling Interfaces, deren Funktionsweise, Eigenschaften und Leistungsfähigkeit. Es wird gezeigt, dass ausschließlich der an der Universität Tübingen entwickelte Thought-Translation Device (TTD) praktische Anwendung gefunden hat.

Der Schwerpunkt dieser Arbeit liegt also darin, die Kommunikation sowohl zu beschleunigen als auch fehlerfreier zu machen, was durch Optimierung von BCI's einerseits und Berücksichtigung der patientenspezifischen Besonderheiten andererseits erreicht werden kann. Um dies zu erreichen, wurde im Kapitel 4 zuerst ein Kriterium entwickelt, welches diesem Optimierungsproblem entspricht. Das Kriterium beschreibt die mathematische Erwartung der Anzahl der Schritte, die erforderlich sind, um eine gewünschte Buchstabe zu schreiben, wobei die eventuell gemachten Fehler sowie deren Berichtigung mit Hilfe der sogenannten „delete"-Funktion berücksichtigt wird. Die „delete"-Funktion wirkt dabei wie die Backspace-Taste und, wenn sie ausgewählt wird, löscht die letzte geschriebene Buchstabe. Die Sequenz der Buchstaben entspricht einem binären Baum; die „Ja/Nein" Wahl entspricht der Wahl des linken oder des rechten Teibaums.

Um das oben beschriebene Problem hinsichtlich der schnelleren und fehlerfreieren Kommunikation zu lösen, muss man eine Sequenz der Buchstaben aufstellen, bei der das entwickelte Kriterium sein Minimum erreicht. Es handelt sich dabei also um ein Optimierungsproblem, welches dadurch gelöst wird, dass man Algorithmen entwickelt, mit dessen Hilfe eine oder mehrere optimalen Sequenzen der Buchstaben durch Synthese aufgebaut werden. Dabei müssen zwei Aspekte besonders stark berücksichtigt werden und stellen nicht triviale Herausforderungen dar. Zum einen ist es die Tatsache, dass die Kommunikation nicht fehlerfrei abläuft. Dies bedeutet, dass der gewünschte Teilbaum mit einer Auswahlwahrscheinlichkeit keiner 1 ausgewählt wird, wobei sich die Auswahlwahrscheinlichkeiten für den linken und den rechten Teilbaum im Allgemeinen voneinander unterscheiden. Zum anderen muss berücksichtigt werden, dass diese Auswahlwahrscheinlichkeiten auch für unterschiedliche Patienten verschieden sind.

Die entwickelten Algorithmen tragen diesen beiden Anforderungen Rechnung, wobei die Gründe der praktischen Anwendbarkeit erfordern, dass mehr als ein Algorithmus entwickelt wurde. Als erstes wurde ein Algorithmus entwickelt, welches die Darstellung der binären

Bäume in Form der sogenannten P-Sequenzen ermöglicht, und zwar dadurch, dass es die Dekodierung der P-Sequenzen als binäre Bäume gestattet. Es sei gesagt, dass die P-Sequenzen eine der handlichsten und bequemsten Darstellungsformen für die binären Bäume sind.

Als nächstes wurde ein sogenanntes „Full-Search“-Algorithmus entwickelt, welches die Erstellung aller möglichen binären Bäume für ein bestimmtes Alphabet, verknüpft mit bestimmten Auftritthäufigkeiten der einzelnen Buchstaben, sowie für bestimmte, patientenspezifische Werte der Auswahlwahrscheinlichkeiten für das „Ja/Nein“-Signal ermöglicht.

Da dieses Algorithmus das Optimierungsproblem zwar prinzipiell löst, für die praktische Anwendung jedoch aufgrund seiner sehr langen Rechenzeiten jedoch weniger geeinigt ist, wurden zwei weitere Algorithmen entwickelt, die sich mit dem stark inhomogenen Fall (d.h. Auswahlwahrscheinlichkeiten für das „Ja“- und für das „Nein“-Signal sind sehr unterschiedlich) und mit dem homogenen Fall (d.h. beide Auswahlwahrscheinlichkeiten sind gleich, jedoch ungleich 1) befassen. Diese Algorithmen ermöglichen erhebliche Beschleunigung hinsichtlich der Rechenzeit im Vergleich zu dem „Full-Search“-Algorithmus. Dies wird anhand diverser Vergleiche und Beispiele demonstriert. Es sei gesagt, dass alle drei Algorithmen die Synthese der optimalen binären Bäume ermöglichen, also Bäume, die hinsichtlich der Kommunikationsgeschwindigkeit und Fehlerfreiheit nach dem oben aufgeführten Kriterium am besten geeignet sind. Zusammen decken sie die komplette Bandbreite an möglichen Fällen ab und stellen in Kombination die einzige bekannte universelle Methode dar, bestehend aus Kriterien und Algorithmen, die eine Synthese der individuell passenden Spelling Interfaces ermöglicht, denn sie berücksichtigt individuelle Auswahlwahrscheinlichkeiten eines bestimmten Patienten. Dadurch kann sichergestellt werden, dass ein Spelling Interface, welches mit Hilfe dieser Methode erstellt wurde, das aus der Sicht der Patienten optimale Interface darstellt und ermöglicht, die schnellstmögliche und möglichst fehlerfreie Kommunikation zu führen.

Die letzte Sektion im Kapitel 4 befasst sich mit dem integrierten Computertool, welches zwecks praktischer Anwendbarkeit der entwickelten Algorithmen und Erzeugung der binären Bäume für die Verwendung durch Patienten entwickelt wurde, wobei die individuellen Fertigkeiten und Fähigkeiten eines Patienten einerseits und sowohl die neu

entwickelten als auch die bestehenden Algorithmen (wie z.B. das Perelmouter-Birbaumer Algorithmus) andererseits berücksichtigt wurden.

Das Kapitel 5 beinhaltet die Zusammenfassung und den Ausblick, wobei es klar gemacht wird, dass sich die Problemstellung der nicht-fehlerfreien Kommunikation mit Hilfe der binären Kommunikationsgeräte nicht nur auf die Kommunikation mit gelähmten Patienten beschränkt.

1. Introduction and Motivation

1.1. Motivation

Accomplishments of surgery, increasing level of efficiency of life-support systems as well of intensive care result in the increasing number of patients who survive severe accidents or diseases, such as stroke in the brain stem, tumors, encephalitis, some forms of polyneuritis or brain injuries. Although patients survive, they can suffer from serious injures of the brain and spinal cord, resulting, in the worst case, in paralysis or in the so-called locked-in syndrome – “active mind in a paralyzed body” [33, 34, 49]. A very common reason for paralysis and locked-in symptom is, for instance, amyotrophic lateral sclerosis or ALS. ALS involves a continuously progressive degeneration of central and peripheral motor neurons, usually (but not exclusively) after the age of 30. The question of causes of ALS has not been answered yet, and effective therapeutic strategies still have to be developed. The incidence of ALS is about 1.5-2:100,000 [8], with an increasing trend [11], thus, leading to a comparably high ratio of locked-in patients. Other causes for the locked-in symptom that are described above also take their toll. Therefore, the ever increasing number of locked-in patients remains an important concern for today’s society.

Locked-in patients, suffering either from ALS or other diseases, remain conscious and can both hear and see their surroundings, although they are unable to use their muscles and, therefore, to communicate with their environment either vocally or manually (using body language, per writing or keyboard etc.). Hence, in order to maintain at least a minimal degree of quality of life they need artificial means for communication. Usually, such communication devices provide a patient with a sequence of letters. Using his / her remaining motor ability (e.g., finger movements or blinks), the patient tries to choose the correct letters, thereby creating words and phrases [2]. The remaining motor ability can be so scanty that only the binary (“Yes/No”) communicational signal could be delivered to a device (for example, to a so-called "head-switch" communication device).

A definite drawback of such communication devices for artificial spelling is their error-proneness. Using such systems for alternative communication with motor response control, a patient very often is not able to produce an absolutely reliable single binary response, especially if suffering from very severe motor paralysis. This means that in many cases the

binary signal will be delivered to the device with an error. Moreover, the progressive disease can lead to further muscle atrophy, when even minimal movements, which are required to operate such communication devices, can become impossible for a patient.

As a result, motor communication becomes extremely restricted or even impossible. In order to overcome this problem, so-called direct brain-computer communication devices are being developed for fully locked-in patients. These devices allow muscle-independent communication and create a direct communication channel between the patient's brain and the computer, making possible for a patient to select different characters or menu items using their electrical brain activity [26, 50]. The first such device, which has been already *successfully* used for practical communication with ALS-patients [7, 25], is the TTD (Thought-Translation Device). It was developed by Prof. N. Birbaumer and his group at the University of Tübingen and is based on the principle of the biofeedback of slow cortical potentials (SCP).

Brain-computer interfaces are even more error-prone than artificial spelling devices based upon movements. The reason for that is the obvious fact that the muscle control prevails over the brain control. Nevertheless, in case of fully locked-in patients brain-computer interfaces represent the only existing possibility of communication with the environment. As described above, in many cases the disease is progressing, thus, leading sooner or later to a fully locked-in syndrome. Therefore, as the second important concern one can mention the fact that brain-computer interfaces become a single mean of communication for locked-in patients in a long run.

The third concern deals with the error-proneness itself and its psychological consequences. Especially for locked-in patients it is extremely important to have a reliable communication channel, since this is the only one communication option that they possess. On the other hand, due to non-trivial skills to voluntarily control the brain activity, the communication via brain-computer interfaces is conducted essentially slower and with more errors than verbal or motor communication. Hence, both the communicating patient and his / her environment / relatives need to devote a very high amount of patience and time in order to achieve a desired outcome. In many cases, however, either patients their selves or their relatives or both are so depressed and dispirited by a low communication rate that their initial wish for communication is distinguished, leading, in the worst case, even to a complete

abandonment of communication attempts. Every acceleration of either communication rate or accuracy of communication will, therefore, lead to an increasing level of quality of life for locked-in patients. This can be achieved through optimization of brain-computer interfaces used for communication in terms of increasing of error-propones and adoption to individual patient's characteristics. These are, first of all, the individual needs of the patient as well as his / her skills with respect to voluntary control of brain activity.

1.2. Problem description

The initial problem that is covered in this thesis consists of all three main concerns presented above and deals primarily with increasing of quality of life for locked-in patients, who are or will be using brain-computer interfaces for communication. We strive to solve this problem through increasing of communication speed and accuracy due to both optimization of interfaces and adoption to individual patient's characteristics. As we will see further on, this initial problem will lead to a general problem of a non-zero error communication and can be applied to all types of binary communication devices.

In order to solve the problem, we need to set a criterion that matches this problem and to develop a method of creation of optimal brain-computer interfaces, which corresponds to this optimization criterion. Hereby the individual characteristics of the patient need to be taken into account. The aim of this research is to find such criterion, to develop optimization methods for it and to implement these methods in form of mathematical algorithms and of a corresponding computer tool. The results of this work are intended, first and foremost, for the Thought-Translation Device and its applications, but could be also applied to all type of binary augmentative communication devices.

1.3. Thesis outline

In the next chapter we will introduce the most important definitions and descriptions of terms that are used in this thesis and are related to mathematics, information technologies and brain-computer interfaces. Additionally to this, explanation of optimization problems and algorithms will be provided. Also, the complete description of the Huffman's algorithm will be given, since it plays a crucial role in creation of optimal brain-computer interfaces. These definitions and explanations will build the required foundation for understanding both mathematical and psychological relations and, thus, will make it possible to follow presented deductions even for a reader without large expertise in these areas.

The chapter 3 deals with the state of the art of brain-computer interfaces. In the first section of this chapter all major existing binary Spelling Interfaces will be presented, together with their principles and the most important characteristics. Also some additional insights with respect to EEG activity and psychological relations will be given. Since the Thought-Translation Device plays the most important role of all existing BCI's from the practical point of view, its thorough description will be provided. In addition, at the end of the section an overall summary for described interfaces will be given, whereat the major focus will be put on practical applicability. This section is based upon a comprehensive literature research.

In the second section of the chapter 3 the general problem definition of the non-zero error communication will be given. Algorithms that can provide solution of this general problem represent the major focus of this thesis. The relevance of the general problem to particular aspects of building of optimal Spelling Interfaces for individual handicapped patients will be shown. Besides, we will discuss known approaches that seek to solve this problem and show their advantages and disadvantages.

The fourth chapter deals with actual algorithms that solve the problem of non-zero error communication. Since this problem represents an optimization problem, the first step is to develop a practically relevant optimization criterion. In the next step we will present a newly developed algorithm for decoding of P-sequences and the so-called Full-Search algorithm. In order to further accelerate the search, two additional algorithms – the inhomogeneous algorithm and the merging algorithm – have been developed. They will be compared with the Full-Search algorithms in order to show the search acceleration. The last section of the chapter 4 will be focused on the developed integrated computer tool that allows creating optimal Spelling Interfaces for handicapped patients, taking into consideration individual patient's abilities and skills.

The final chapter of the thesis provides an overall summary and gives an outlook with respect to further development potentials for Spelling Interfaces.

2. Fundamentals

In the following sub-chapters one can find definitions and descriptions of terms, structures and methods used in this thesis. Especially areas dealing with binary trees, optimization problems and devices for brain communication and binary communication are wide-ranging covered.

The most fundamental definitions, especially with respect to the graph theory, binary trees and optimization problems, are given according to [9].

2.1. Graphs, trees and data structures

Directed and undirected graphs

A ***directed graph*** *G* is a pair (*V*, *E*), where *V* is a finite set and *E* is a binary relation on *V*. The set *V* is called the ***vertex set*** of *G*, and its elements are called ***vertices***. The set *E* is called the ***edge set*** of *G*, and its elements are called ***edges***. In a so-called ***undirected graph*** *G* = (*V*, *E*), the edge set *E* consists of unordered pairs of vertices, rather than ordered pairs.

Connected graphs

An ***undirected graph*** is ***connected*** if every pair of vertices is connected by a path.

Free trees

A ***free tree*** is a connected, acyclic, undirected graph. A ***rooted tree*** is a free tree, in which one of the vertices is distinguished from the others. The distinguished vertex is called the ***root*** of the tree. Vertices of a rooted tree are often referred to as ***nodes*** of the tree.

Parents, children, leaves

If the last edge on the path from the root *r* of the tree *T* to a node *x* is *(y, x)*, then *y* is the ***parent*** of *x*, and *x* is the ***child*** of *y*. The root is the only node in *T* with no parents. If two nodes have the same parent, they are ***siblings***. A node with no children is an ***external node*** or ***leaf***. A nonleaf node is an ***internal node***.

Degree, depth, height

The number of children of a node *x* in a rooted tree *T* is called the ***degree*** of *x*. The length of the path from the root *r* to a node *x* is the ***depth*** of *x* in *T*. The largest depth of any node in *T* is the ***height*** of *T*.

Binary trees

A ***binary tree*** is a structure defined on a finite set of nodes that either:
contains no nodes, or is comprised of three disjoint sets of nodes: a ***root*** node, a binary tree called its l***eft sub-tree***, and a binary tree called its ***right sub-tree***.

Full binary trees

A ***full binary tree*** is a binary tree, in which each node is either a leaf or has degree exactly 2.

Means of description of binary trees

There is a variety of ways and methods how to describe given binary trees. An ***inorder tree walk***, for instance, prints the root of the sub-tree between the values in its left sub-tree and its right sub-tree. A ***preorder tree walk*** prints the root of the tree before the values in either sub-tree. A ***postorder tree walk*** prints the root of the tree after the values in either sub-tree.

In a so-called ***Polish notation*** the set *B* of binary trees is recursively defined by the equation $B = \square + \bigcirc BB$, where ○ is an internal node and □ is an external node or leaf. Here a preorder tree walk is used – firstly visiting the root and then – all the leaves walking anti-clockwise. For example, the tree in the Figure 2-1 can be notated in this way as ○○□○○□□□○□□.

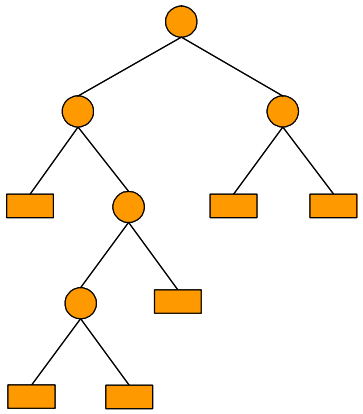

Figure 2-1: A full binary tree with 6 leaves

We denote by $|T|$ the number of leaves of the tree T and by $r(T)$ the degree of the root of the tree T, i.e. $r(T) = 0$, if $T = \square$, and $r(T) = 2$ otherwise. Let TL and TR be the left and the right sub-trees of T, and $\circ T1T2$ – the product of $T1$ and $T2$, which admits $T1$ as the left sub-tree and $T2$ as the right sub-tree.

Two trees T and T' are in ***B-order***, $T < T'$ if

1) $r(T) < r(T')$, or
2) $r(T) = r(T')$ and $T_L < T'_L$ or
3) $r(T) = r(T')$ and $T_L = T'_L$ and $T_R = T'_R$.

The ***P-sequence*** of a binary tree T is the integer sequence $(p_T(1), \ldots, p_T(|T|-1))$, where $p_T(i)$ is the number of internal nodes $\circ$ written before the leaf i in the Polish notation of T. For the tree in the Figure 2-1, for instance, the P-sequence is (2, 4, 4, 4, 5).

Stacks and queues

Stacks and queues are dynamic sets, in which the element removed from the set is pre-specified. In a ***stack***, the element deleted or removed from the set is the one most recently inserted: the stack implements a ***last-in, first-out***, or ***LIFO***, policy. In a ***queue***, the element deleted is always the one that has been in the set for the longest time: the queue implements a ***first-in, first-out***, or ***FIFO***, policy.

Alphabets

An ***alphabet*** is a finite set of symbols.

Catalan numbers

The number of different binary trees on *n* internal nodes b_n is given by the *n*-th Catalan number [e.g. 22]:

$$b_n = Cat(n) = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n!)}{(n!*(n+1)!)}$$

An asymptotic estimation for b_n is:

$$b_n = \frac{4^n}{\sqrt{\pi} n^{3/2}}(1 + O(1/n))$$

As one can see from this formula, the *n*-th Catalan number, and therefore the number of different binary trees grows exponentially with *n*.

For many reasons, including the synthesis of binary SI, it is often useful to have the whole list of all possible shapes of trees of a certain type – a so-called ***enumeration*** all these trees. In such cases it is advantageous to have some concise representation for the structure of each tree. An ideal situation [28] is the one where an interval of Cat(*n*) integers has "one-to-one" correspondence with the trees on *n* internal nodes. Fortunately, many papers have appeared, which contain algorithms for generating all binary trees of a certain type [23, 28, 35, 42, 47, 53]. Typically, the trees are encoded as integer sequences and those sequences are, in turn, generated lexicographically. There is a lot of such coding methods – coding based on permutations (tree permutations [43], ballot sequences [42], permutation pairs, level sequences, Zaks' sequences [53] etc.), coding based on rotations [28] and so on. As noted in [28], "different coding methods highlight different aspects of the tree structure and a good choice emphasizes the aspect that is essential for the application". We will use further on so-called P-sequences, which are *n*-tuples of numbers of preceding internal nodes for each leaf in preorder.

2.2. Optimization problems, greedy algorithms and greedy-choice property

Optimization problems deal with either maximization or minimization of certain criteria. In such problems there can be many possible solutions. Each solution corresponds to a particular value of the criterion or criteria, and one attempts to find a solution with the optimal (minimum or maximum) value. Such a solution is called *an* optimal solution to the problem, as opposed to *the* optimal solution, since there may be several solutions that achieve the optimal value.

Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step. For many optimization problems, however, going through the whole variety of choices in order to determine the best choice is overkill. Thus, more efficient algorithms are required. A ***greedy algorithm*** always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in order to achieve a globally optimal solution, requiring in general essentially fewer steps than algorithms of dynamic programming.

Greedy algorithms do not always yield optimal solutions. In order to be able to say whether a greedy algorithm actually solves a particular optimization problem, one should use a so-called ***greedy-choice property***: a globally optimal solution can be arrived at by making a locally optimal (greedy) choice. Therefore, the usage of greedy algorithms requires that it is proven that a greedy choice at each step yields a globally optimal solution, and thus, a greedy-choice property is fulfilled.

2.3. Huffman codes

Huffman codes represent a widely used and very effective technique for a number of greedy problems, especially for compressing data. In order to compress a set of characters each character is represented by an optimal binary characteristic code or an optimal unique binary sting that is built with help of the Huffman's greedy algorithm based upon a table of the frequencies of characters.

Let us assume one has to compress (which is to write in the most compact way) some amount of data based on an alphabet consisting of six different characters (chart 2-2). If

one uses a co-called ***fixed-length code***, one needs for instance 3 bits in order to represent these six characters.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Chart 2-2: A character-coding problem. Developed fix-length-codes and variable-length codes

If each character is assigned a 3-bit codeword, a data set of 100,000 characters can be encoded in 300,000 bits. Using the ***variable-length code*** built upon the Huffman's algorithm, the same data set can be encoded in 224,000 bits.

2.3.1. Prefix codes and the cost function

Codes, in which no codeword is also a prefix of some other codeword, are called ***prefix codes***. It is possible to show that the optimal data compression achievable by a character code (which is writing down data in the most compact way) can always be achieved with a prefix code.

Given a tree T corresponding to a prefix code, it is a simple matter to compute the number of bits required to encode some particular amount of data. For each character c in the alphabet C, let $f(c)$ denote the frequency of c and let $d_T(c)$ denote the depth of the c's leaf in the tree. $d_T(c)$ is at the same time the length of the codeword for the character c. The number of bits required to encode the data is thus

$$B(T) = \sum_{c \in C} f(c) d_T(c),$$

which is denoted as the ***cost function*** or the ***cost*** of the tree T. In general, optimization problems with respect to trees often strive to either maximize or minimize the cost function.

2.3.2. Constructing a Huffman code

The Huffman algorithm builds the tree *T* corresponding to the optimal code in a bottom-up manner. It begins with a set of |*C*| leaves and performs a sequence of |*C*|-1 co-called "merging" operations to create the final tree.

In the following example *C* is a set of *n* characters and each character $c \in C$ is an object with a defined frequency *f [c]*. A priority queue *Q*, keyed on *f*, is used to identify the two least-frequent objects to merge together. The result of the merger of two objects is a new object, whose frequency is the sum of the frequencies of the two objects that were merged.

Huffman's Algorithm

$n \leftarrow |C|$

$Q \leftarrow C$

for *i*=1 to *n*-1 do

extract characters *x* and *y* with the lowest values of the cost from *Q*

make a new sub-tree *z*, where *x* and *y* are the left and the right leaf respectively and the cost of *z* is *f [z] = f [x] + f [y]*

insert *z* into *Q*

return *Q*

In the example corresponding to the chart 2-2, Huffman's algorithm proceeds as shown in the figure 2-3. Since there are 6 letters in the alphabet, the initial queue size is *n = 6*, and 5 merge steps are required to build the tree. The final tree represents the optimal prefix code. The codeword for a letter is the sequence of edge labels on the path from the root to the letter (leaf).

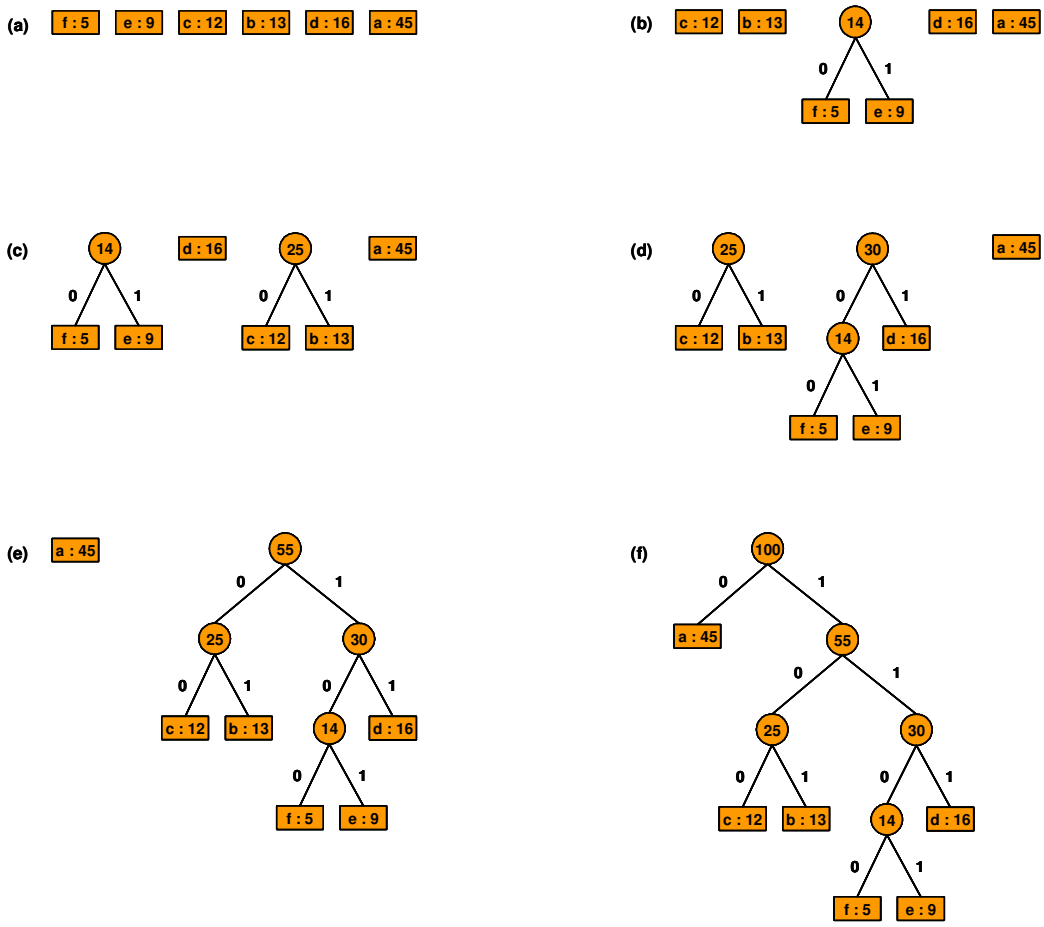

Figure 2-3: The steps of Huffman's algorithm for the alphabet and the frequencies given in the chart 2-2

To prove that the greedy-algorithm of Huffman is correct, two following lemmas are required and sufficient. The first lemma shows that the greedy-choice property holds.

Lemma 2-1

Let C be an alphabet, in which each character $c \in C$ has frequency $f[c]$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C, in which the codewords for x and y have the same length and differ only in the last bit.

Proof:

The idea of the proof is to take the tree T representing an arbitrary optimal prefix code and modify it to make a tree representing another optimal prefix code such that the characters x

and *y* appear as sibling leaves of maximum depth in the new tree. If one can do this, then their codewords will have the same length and differ only in the last bit.

Let *b* and *c* be two characters that are sibling leaves of maximum depth in *T*. Without loss of generality, one can assume that $f[b] \le f[c]$ and $f[x] \le f[y]$. Since $f[x]$ and $f[y]$ are the two lowest leaf frequencies, in order, and $f[b]$ and $f[c]$ are two arbitrary frequencies, in order, we have $f[x] \le f[b]$ and $f[y] \le f[c]$. As shown in the figure 2-4, one can exchange the positions in *T* of *b* and *x* to produce a tree *T'*, and then one exchanges the positions in *T'* of *c* and *y* to produce a tree *T''*. According to the above definition of the cost, the difference in cost between *T* and *T'* is:

$$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) = f[x] d_T(x) + f[b] d_T(b) - f[x] d_{T'}(x) - f[b] d_{T'}(b) =$$
$$= f[x] d_T(x) + f[b] d_T(b) - f[x] d_T(x) - f[b] d_T(b) = (f[b] - f[x])(d_T(b) - d_T(x)) \ge 0,$$

because both $f[b] - f[x]$ and $d_T[b] - d_T[x]$ are nonnegative.

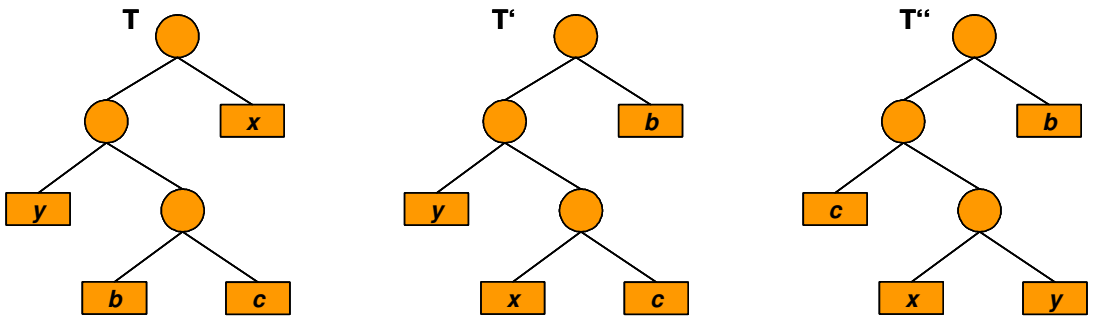

Figure 2-4: The key step in the proof of the lemma 2-1. In the optimal tree T, the leaves b and c are two of the deepest leaves and are siblings

More specifically, $f[b] - f[x]$ is nonnegative because *x* is a minimum-frequency leaf, and $d_T[b] - d_T[x]$ is nonnegative because *b* is a leaf of maximum depth in *T*. Similarly, because exchanging *y* and *c* does not increase the cost, $B(T') - B(T'')$ is nonnegative. Therefore, $B(T'') \le B(T)$, and since *T* is optimal, $B(T) \le B(T'')$, which implies $B(T'') = B(T)$. Thus, *T''* is

an optimal tree, in which *x* and *y* appear as sibling leaves of maximum depth, from which the lemma follows.

This lemma implies that the process of building up an optimal tree by mergers can, without loss of generality, begin with the greedy (local optimal) choice of merging together those two characters of lowest frequency. Since the Huffman's algorithm is a greedy-algorithm, it chooses out of all possible mergers at each step the one that incurs the least cost.

Lemma 2-2

Let *T* be a full binary tree representing an optimal prefix code over an alphabet *C*, where frequency *f[c]* is defined for each character $c \in C$. In the next step, let us consider any two characters *x* and *y* that appear as sibling leaves in *T*, and let *z* be their parent. Then, considering *z* as a character with frequency *f [z] = f [x] + f [y]*, the tree $T' = T - \{x, y\}$ represents an optimal prefix code for the alphabet $C' = C - \{x, y\} \cup \{z\}$.

Proof:

We first show that the cost *B(T)* of tree *T* can be expressed in terms of the cost *B(T')* of tree *T'* by considering the component costs according to the cost definition above. For each $c \in C - \{x, y\}$, we have $d_T[c] = d_{T'}[c]$, and hence, $f[c]d_T(c)=f[c]d_{T'}(c)$. Since $d_T(x)=d_T(y)=d_{T'}(z)+1$, we have

$$f[x]d_T(x) + f[y]d_T(y) = (f[x] + f[y])(d_{T'}(z) + 1) = f[z]d_{T'}(z) + (f[x] + f[y]),$$

from which we conclude that

$$B(T) = B(T') + f[x] + f[y].$$

If *T'* represents a nonoptimal prefix code for the alphabet *C'*, then there exists a tree *T''*, whose leaves are characters in *C'* such that *B(T'')<B(T')*. Since *z* is treated as a character in *C'*, is appears as a leaf in *T''*. If we add *x* and *y* as children of *z* in *T''*, then we obtain a prefix code for *C* with cost *B(T'') + f [x] + f [y] < B(T)*, contradicting the optimality of *T*. Thus, *T'* must be optimal for the alphabet *C'*.

2.4. Brain-computer interfaces and binary augmentative communication

Since remaining motor abilities of locked-in patients can be extremely sparse, communication devices based upon muscle movement are not suitable for many such patients (see also chapter 1). Alternative approaches were suggested, based upon either electric brain activity or so-called autonomic functions (i.e. skin temperature, heart rate etc.). However, according to [26], the very slow rate of responsibility and the high metabolic noise of many autonomic responses, as well as the high incidence of pathological changes in locked-in patients, make autonomic functions useless for precise and reliable communication. So far only the usage of electrical brain activity, which can be either event-dependent or not, has proved itself being useful. Devices using electrical brain activity for communication are commonly referred to as ***brain-computer*** interfaces. This sub-chapter is dedicated to basics of brain-computer interfaces.

2.4.1. Electric brain activity

The ***electroencephalogram (EEG)*** refers to the electrical signals of the brain. In particular, EEG refers to electrical activity arising from neurons in the cerebral cortex and is usually recorded non-invasively (that is, without surgical operation) from the scalp of a patient according to the international 10-20 system* (figure 2-5).

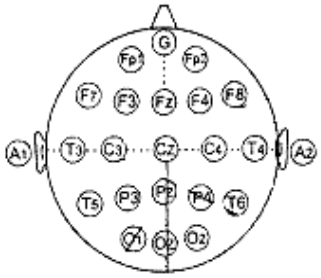

Figure 2-5: Electrode placement on the scull according to the international 10-20 system

* Since shape of heads (or skulls) varies from individual to individual, it is impossible to set positions for electrode placement absolutely. Therefore, electrodes are positioned in relative (instead of absolute) distances. In the international 10-20 system these distances make up 10% and 20% of the total scope of the individual skull. The capital letters refer to the lobes of the brain (F = frontal: T= temporal; P= parietal; O= occipital). Electrodes placed on the left hemisphere are marked with odd numbers, those on the right with even numbers, and those sagittal with z.

The electrical activity of the brain measured with help of EEG includes spontaneous electrical activity of the cerebral network as well as the cortical responses to external or internal events.

Electrical brain responses that are linked to physical stimuli or behavioral responses are called ***event-related potentials***. They are characterized and named based upon their voltage amplitude and their latency in relation to stimulus onset (figure 2-6) [26].

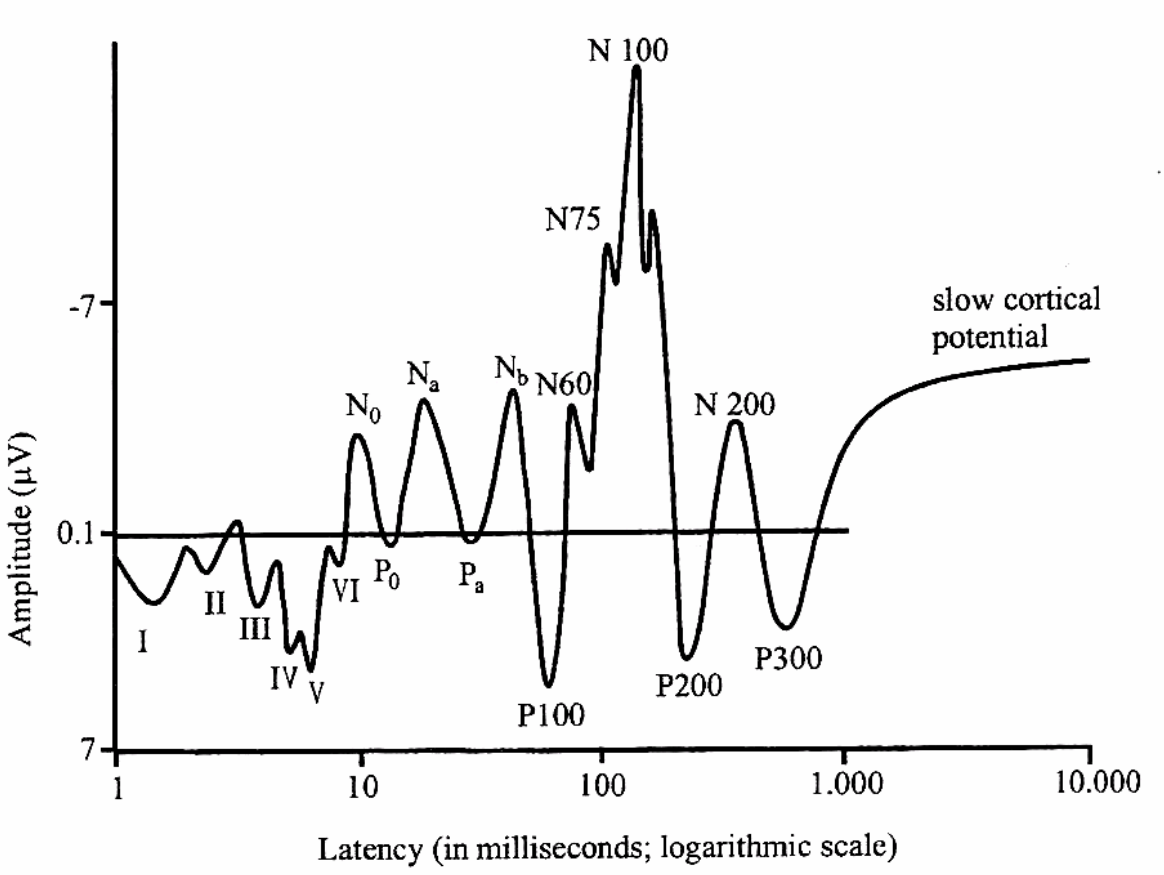

Figure 2-6: Averaged event-related potentials as responses to acoustic stimuli

For example, the negative peak that arises 100 ms after the occurrence of the stimulus is called N100, the positive peak 300 ms after the occurrence of the stimulus is called P300 etc.

We differentiate between ***exogenous*** and ***endogenous*** event-related potentials. Exogenous potentials occur up to 100 ms after the occurrence of the stimulus itself and are rather obligatory responses to the presentation of physical stimuli, therefore depending more on the physical parameters of the stimulus, rather than on psychological aspects of the patient or the situation. Endogenous potentials occur at least 100 ms up to several seconds later than the stimulus occurred and rely first of all on behavioral and psychological processes related to the event. Event-related potentials with the latency ranging from 500 ms up to 10 s or more are called ***slow cortical potentials***. Since cortical potentials play an absolutely crucial role for various brain-computer interfaces and are, therefore, extremely important for this research, they are discussed more thoroughly in the chapter 3.
The EEG normally contains characteristic bands of different frequencies [26]. Electrical signals between 8 and 13 Hz are referred to as ***alpha band***, which is most prominent over occipital, parietal and posterior regions in the state of relaxed wakefulness with closed eyes. The μ***-rhythm*** is defined as arch-shaped electric activity over the sensorimotor cortex in the alpha band. Contrary to the visual alpha rhythm, it is not dependent on vision, but is linked to motor activity, motor imagery and movement conduction and preparation. The ***beta band*** deals with frequencies above 13 Hz and is measured mainly over frontal and central skull regions. The ***theta band*** covers frequencies from 4 Hz to 8 Hz and is only weakly expressed in an EEG of a normal waking adult. Frequencies from 0.5 Hz to 4 Hz form the ***delta band***. They are present only in deep sleep or anesthesia. Frequencies above 30 Hz are referred to as the ***gamma band*** and have been related to the synchronization of neuronal assembles involved in the generation of mental representation [26].

2.4.2. Brain-computer interfaces

Brain-computer interfaces can be divided into two large groups based upon two founding approaches: the ***direct recognition approach*** and the ***operant conditioning approach***. The direct recognition approach deals with direct and immediate recognition of brain potentials without prior training or conditioning, whereat it is assumed that specific and detectable event-related brain potentials or EEG power spectra reflect the confrontation of the individual with certain kinds of stimuli [12] or engagement in certain cognitive tasks [18, 19].

The operant conditioning approach, on the other hand, is based upon principles of learning psychology and assumes that the individual can bring EEG signals under his or her voluntary control and, therefore, establish control of the target EEG response by means of sensory feedback, positive reinforcement of correct behavior, or both. Sensory feedback, for instance, can be provided by the movement of a cursor on a computer screen. With help of brain-computer interfaces based upon operant-conditioning patients receive continuous feedback of their target brain response and learn to produce or withhold certain levels of the EEG signal. This signal can be reflected in certain voluntary movement of symbols on the computer screen or in voluntary changing of frequencies of sounds according to the amplitude of the brain response. Successful learning using reinforcement and shaping of the response lead thus to obtaining of a new, non-motor skill – the voluntary control of the EEG signal.

From the clinical point of view and from the point of view of increasing of patient's quality of live, there are two major criteria of brain-computer interfaces: ***speed*** and ***accuracy***. Speed deals first of all with the time required to select a character or, for menu-based interfaces, an intended menu item. For locked-in patients, there is presumably a limit, below which the speed and, therefore, the communication rate should not fail. This limit certainly depends on the motivation and the patience of individual patients and their social environment. However, cases have been reported [26], when patients dropped off communication attempt due to slow communication.

Accuracy is often defined as percentage of correct responses, that is, correct selections per time interval or per attempt. Accuracy, therefore, does not apply to the correctness of the final message. Since individuals can not absolutely control their responses, it is extremely important for successful communication and for acceptance of brain-computer interfaces through patients to increase the level of accuracy. As we will see in the chapter 3, the voluntary control of responses can vary a lot from individual to individual. In case of the Thought-Translation Device, for instance, it can differ from 20% to 90%. Although the level of accuracy for devices based upon the operation conditioning approach can be increased through patient's training, this is often not sufficient for accurate communication. In order to evaluate brain-computer interfaces from the standpoint of accuracy, it is important to take into account, whether means of error correction is available. Basically, there are two possible means of correction: the delete option and the step-back option.

The ***delete option*** functions similar to the backspace key on the computer keyboard, that is, it deletes of the last written character and creates of the original state before *the last writing attempt* has been started.

The ***step-back option*** resumes the state before *the last action* has been taken; therefore, returning the patient one level higher than the last level he or she was at. Both the delete option and the step-back option have their particular advantages and disadvantages and can be the more or less preferred means of the error correction, depending of the general design of the particular interface. In brain-computer interfaces, the delete option has been proven itself being more convenient for patients.

From the psychological point of view, especially from the point of view of training of patients, the easiest and most reliable way to achieve the highest values of both speed and accuracy with help of the voluntary control of the EEG signal is to make use of the easiest reveal of this skill – the binary signal (i.e. “yes” or “no”). With respect to the EEG such signals can be given, for instance, through voluntary positivity or negativity of certain potentials. Since a binary signal itself is sufficient for successful communication (i.e. binary computer code), it can serve as a basis for successful communication. Devices that use binary signals for communication with help of electric brain activity are called ***binary brain-computer interfaces***.

In the chapter 3 – “State of the Art of Binary Spelling Interfaces” – which gives an overview of and compares existing brain-computer interfaces, we will see that only binary brain-computer interfaces based upon operation conditioning have found practical application for locked-in patients so far.

3. State of the Art of Binary Spelling Interfaces

As mentioned in the chapter 2, the absolutely sufficient and the most effective way to use brain-computer interfaces for communication is to produce a *binary* signal, either as a result of application of certain stimuli (the direct recognition approach) or as a result of the voluntary controlling of EEG signals (the operant conditioning approach). Therefore, the most brain-computer interfaces described in the available literature are *binary spelling interfaces*, which belong to one of the both above groups, depending on whether training to self-regulate an EEG response is necessary for usage of the brain-computer interface or not.

In the following sub-chapters one can find descriptions and functional principles of major brain-computer interfaces developed till now. The focus will be put on the so-called Thought-Translation Device, developed by Professor N. Birbaumer and co-workers [e.g. 36], as the device most successfully used in the practice.

3.1. Overview of the existing binary spelling interfaces, their principles and characteristics

3.1.1. The P300 brain-computer interface

As mentioned in the chapter 2, the P300 is a positive peak of the EEG that occurs approximately 300 to 400 ms after stimulus onset. The P300 is endogenous and event-related, which means that it relies on behavioral and psychological processes linked to some particular rare event. The P300 potential consists of two components – the P300a potential that occurs earlier and is often linked to the newness of the event, and the P300b potential that occurs later and is important for the practical usage in brain-computer interfaces. The P300b potential is recorded at the electrode location Pz (compare the figure 2-5). The best way to measure the P300 potential is to do it in the so-called oddball paradigm: the test person is confronted with a row of rare and frequent events. Additionally, the test person has to conduct a task of categorizing these events in some way, for instance, counting them. Under these circumstances, desired rare events lead to occurrence of P300 potentials. The rarer the desired event is, the larger is the amplitude of the P300 potential.

Since the P300 potential is endogenous and event-related, patients do not need to be trained in order to be able to generate this potential, as it happens automatically and voluntary as a response to rare task-relevant events. These properties of the P300 potential were employed for the practical application in brain-computer interfaces, e.g. as described in [10]. For these purposes the square matrix of the size 6x6 was presented to test persons. This matrix contained 36 symbols – letters of the English alphabet, numbers and the space symbol, used for word separation (see the figure 3-1). In order to generate task-related events, rows and columns of the matrix flashed in some random order, and the test person had to count the number of times the cell with the target letter would flash. In each trial each of six rows and each of six columns of the matrix would flash once for the period of time of 100 ms, thus resulting in 12 events per trial. Since the target cell would always flash twice – once contained in the target row and once contained in the target column, these two task-relevant events can be considered as rare as compared to the other events. Thus, only flashing of rows or columns containing the target cell resulted in occurrence of the prominent P300b potential. Flashing of the other rows or columns did not produce prominent P300 potentials (compare the figure 2-6). The measurements were conducted online in the time frame of 100 ms before to 500 ms after the occurrence of the event, whereat the EEG background noise was taken into account. According to [10], with accuracy of 80%, up to 7.8 characters per minute could be written.

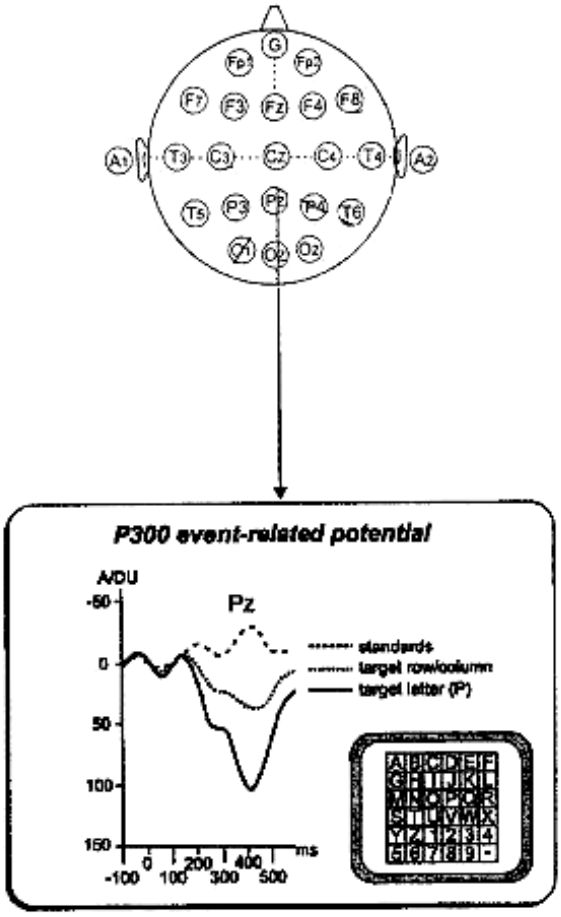

Figure 3-1: The P300 brain-computer interface – electrode placement points, EEG-activity and visualization

Altogether, the P300 brain-computer interface was tested with four patients suffering from complete or incomplete paraplegia. Due to various reasons, the P300 brain-computer interface is not in practical use.

3.1.2. The visual event-related potential brain-computer interface

Different visual stimuli, such as flickering illumination or flashes, can elicit so-called visual even-related potentials. These potentials can differ in their amplitude and frequency and can, in turn, be used for control of brain-computer interfaces [30]. An important pre-requirement with respect to the usage of such interfaces is, however, the ability of the patient to control the gaze direction. All described visual event-related potential BCI's function as follows: a patient is presented with a screen containing various items for

selection. First of all, the patient has to focus on the desired item and, therefore, to fixate his / her eyes on it. Then items on the screen are sequentially visually activated (e.g. through flickering illumination), and the brain-computer interface detects the visual event-related potential elicited by the desired item, which, in turn, results in its selection.

One of well-known visual event-related BCI's is the one developed by Sutter and Tran [30, 45, 46, 48], where a patient is presented an 8x8 matrix containing either characters or other items for selection. Cells of the matrix were activated by flickering illumination in a pseudo-random order. The matrix cell that elicited the largest event-related potential amplitude over the occipital cortex was considered being the target cell a patient wanted to select.

This BCI was tested in 60 participants without disabilities, whereby successful communication was reported. However, different issues and problems occurred when using the BCI in patients. In patients suffering from neurological impairment communication problems occurred due to large muscular artifacts from neck muscles, which interfere with the visual event-related potential recording. Limitations in patients suffering from cerebral palsy were reported due to artifacts caused by involuntary athetoid movements. Limitations in patients suffering from amyotrophic lateral sclerosis were linked to artifacts caused by fasciculation. Attempting to overcome these limitations, the authors decided to implant an electrode between the dura mater and the scull of one patient suffering from amyotrophic lateral sclerosis. In this case the authors reported a very high communication rate of 10 – 12 words per minute. According to [26], however, the usage of the system is limited to this only one patient. No data about training duration or accuracy of this system are available.

3.1.3. The EEG brain-computer interface based upon motor imagery or cognitive tasks

Different cognitive tasks, such as imagining a rotation of a three-dimensional geometric figure or imagining a movement of a finger, lead often to distinct, task-specific distribution of EEG frequency patterns over scalp. These patterns can, in turn, be recognized and can serve as a basis for brain-computer communication. Typically, such interfaces deal rather with encoding of special commands than with written communication. For instance, an EEG recorded during mental rotation of a geometric figure after imagining a movement of one specific finger could turn on a TV. An EEG recorded while imagining a movement of one

specific finger after a movement of another specific finger could turn off a TV. This happens, again, due to the fact that such EEG patterns are distinct and task-specific.

In case of motor imagery this pattern has been proven in series of extensive studies [e.g. 38]. These studies have also shown that the accuracy of communication improves significantly when different parameters of experimental setup, such as electrode position or frequency range of an EEG, are optimized for particular patients. For example, at electrode positions over the central motor cortex (C3 and C4, compare the figure 2-5) researchers could record an event-related desynchronization of an EEG over the hemisphere [26]. This desynchronization occurred due to imagined movement, whereat synchronization of the EEG was kept over the ipsilateral hemisphere (see figure 3-2). This difference in the EEG frequency patterns over both hemispheres made possible to control a cursor on a computer screen [39]. At the moment the same principle is used in a brain-computer interface, which allows a patient suffering from tetraplegia to control a device to open and close his hand (referred to as an orthosis). This patient needed overall 62 training sessions lasting for over 5 months in order to control the orthosis over the corresponding brain-computer interface with accuracy of 90% to 100%. He remains the only one patient using this BCI based upon motor imagery.

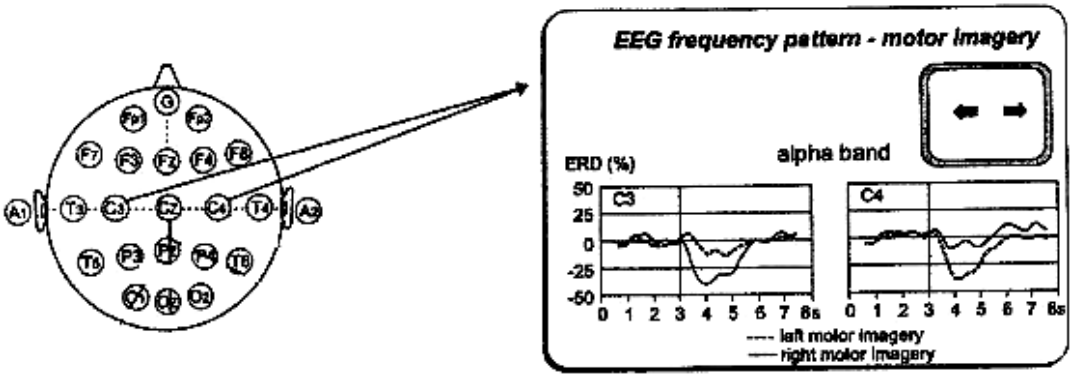

Figure 3-2: The brain-computer interface using EEG frequency patterns related to motor imagery

In case of usage of EEG frequency patterns related to other mental tasks for communication an extensive research was conducted as well [1, 18, 19]. The following mental tasks were investigated [26]:

- Thinking of nothing in particular (baseline task);

- Solving a multiplication problem;
- Mentally rotating a complex three-dimensional block figure*;
- Mentally writing a letter to a friend without vocalization;
- Visualizing numbers being written on a black-board sequentially, with each number being erased before the next one is written.

These studies claim to allow an offline classification of the distribution of spectral power with an accuracy of 80% - 90%, thus making possible to conclude from the particular EEG power spectra characteristic to a particular task. The studies dealt, however, purely with healthy participants. No attempt of applying these principles for communication with handicapped patients has been described so far.

3.1.4. The action potential frequency ("firing rate") brain-computer interface

The principle of such brain-computer interfaces is based upon the fact that the action potential of a neuron is a so-called "all-or-none" signal of constant amplitude. Under the assumption that patients suffering from the locked-in syndrome can obtain control over the number of these signals (that is, the action potential frequency or firing rate of neurons in the central nervous system), this principle can be used to encode information and, therefore, be employed for communication. Since it is not possible to record action potentials directly from the scalp, electrodes have to be placed into cortical tissue of a patient, thus requiring an invasive operation.

A number of studies has been conducted with monkey in order to investigate the action potential firing of cortical neurons [e.g. 20]. Glass electrodes containing so-called "proprietary neurotrophic factors" were implanted into cortical tissue. Neurotrophic factors were intended to allow adjacent neurons to grow into tips of electrodes. Next, the neuronal activity was measured directly from cortical tissue. These studies demonstrated that in monkeys implanted electrodes can remain active for 15 to 16 months, thus making possible the conduction of training sessions as well as actual communication. In the next step, a female patient suffering from amyotrophic lateral sclerosis and being close to a total locked-

* The task of mentally rotating a three-dimensional figure was conducted as follows: first, participants had to study a drawing of a figure for 30 seconds. Then the drawing was removed, and participants were instructed to imagine the figure rotation around its axis.

in syndrome underwent a procedure of an electrode emplacement. Two electrodes were implanted in the hand area of the right motor cortex, and approximately two weeks after action potentials could be recorded. Subsequently, training sessions were conducted, which allowed the patient to increase or decrease her firing rate voluntary in order to produce a binary signal. During training sessions, both visual and auditory feedback was given. Unfortunately, the patient died before she could actually use the brain-computer interface for communication. According to [21], another patient underwent the emplacement of electrodes and was able to successfully communicate at a maximal rate of 3 letters per minute on the 423rd day after the operation.

Altogether, the necessity of invasive electrode emplacement, which is required for action potential frequency brain-computer interfaces, results in their very limited applicability.

3.1.5. The μ-rhythm based brain-computer interface

M-rhythm based brain-computer interfaces rely upon ability to control the μ-rhythm of an EEG. As described in the chapter 2, the μ-rhythm is not dependent on vision, but is linked to motor activity, motor imagery and movement conduction and preparation. The fact that the μ-rhythm can be controlled, has been demonstrated in cats [52] and in humans [27]. In later investigations healthy participants were able to demonstrate a skill of moving a cursor on a computer screen towards a target either at the top or at the bottom with help of the μ-rhythm control [51]. Hereby the direction of a cursor movement was determined through the voltage of the μ-rhythm frequency (8 – 12 Hz). Increased amplitude led to the movement towards the top target, decreased amplitude towards the bottom target (see the figure 3-3). Several weeks of training were required for healthy participants in order to reach an accuracy of approximately 90%.

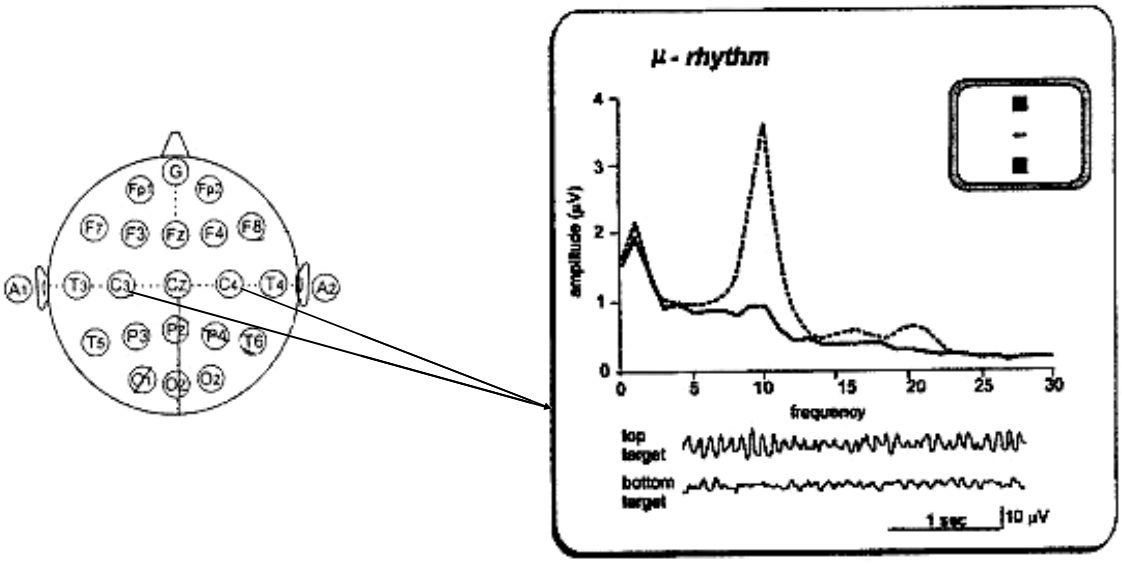

Figure 3-3: μ -rhythm based brain-computer interface – electrode placement points, EEG-activity and visualization

In further studies the same group managed to identify and, subsequently, to improve parameters that affect the accuracy and the speed of communication. Such, it was determined individually, what electrode sites are more advantageous from the point of view of recording the μ -rhythm [e.g. 29]. Later on, studies were spread from healthy participants only to patients suffering from disabilities such as amyotrophic lateral sclerosis and spinal cord injuries. Both healthy participants and handicapped patients had to attend trainings prior to actual usage of brain-computer interfaces for communication. The reported accuracy after this training amounts to up to 90%. According to [26], several patients with different diagnoses currently employ the μ -rhythm based brain-computer interface in the lab.

The main disadvantage of the μ -rhythm based BCI is the fact that the μ -rhythm itself is linked to motor activities. Since paralyzed patients tend, in general, to loose their ability to control and, later on, to imagine the control over motor activities, their ability to voluntarily influence their μ -rhythm also tend to fade away. Thus, at some point of time it can become impossible for them to communicate with help of μ -rhythm based interfaces. Here it needs to be emphasized again that general skills and behavior of healthy participants and

handicapped patients can be very different, therefore, making the aspect of the practical applicability of brain-computer interfaces even more important.

3.1.6. The brain-computer interface based upon slow cortical potentials (the Thought-Translation Device or TTD)

As described in the chapter 2, slow cortical potentials play an extremely important role in the applied brain-computer communication. As we will see, the Thought-Translation Device based upon cortical potentials is currently the only one BCI device that is in the practical use for severe or totally paralyzed patients both at home or in nursing homes. Therefore, it is required to create a better understanding for the nature of slow cortical potentials and to discuss them and the TTD more thoroughly.

As shown above, slow cortical potentials (SCP) are shifts in the depolarization level of the upper cortical dendrites with frequency less than 1-2 Hz and amplitude from 10 to 100 μV. The vertical arrangement of pyramidal cells in the cortex is essential for the generation of far field potentials measured in the electroencephalogram (EEG). The apical dendrites of pyramidal cells are located in cortical layers I and II. Depolarization of the apical dendrites leading to SCP generation is dependent on sustained afferent intracortical and thalamocortical input to layers I and II and on simultaneous depolarization of large pools of pyramidal neurons. The SCP amplitude recorded from the scalp depends on the synchronicity and intensity of the afferent input to layers I and II [44]. Negative SCP are the sum of synchronized ultraslow excitatory postsynaptic potentials (EPSP) at the apical dendrites with the source (electrical positivity) in deeper layers (III and IV) near or at the soma. If an EPSP is initiated, a net inflow of cations occurs at the site of the membrane located below the excitatory synapse. This inflow of positive electrical charge is referred to as a sink (electrical negativity) of the potential gradient, which develops along both sides of the neuronal membrane. In the case of cortical negativity the sink occurs at the apical dendrites [31, 44]. To long lasting positive SCP may contribute several physiological mechanisms. For instance, a cortical positivity can be elicited by sinks (electrical negativity) in layer III and IV due to synchronous excitatory activity of pyramidal synapses at the axon hillock.

The depolarization of cortical cell assemblies reduces their excitation threshold. Firing of neurons in regions responsible for specified motor or cognitive tasks is facilitated and therefore cortical resources are provided. Whenever a task-relevant event is expected cortical excitation thresholds are lowered in the corresponding cortical cell assemblies to facilitate neuronal excitation. The negative SCP amplitude shifts can be recorded in the EEG using typical experimental paradigms that elicit negative SCP shifts [6, 41]. Negative amplitude shifts grow with increasing attentional or cognitive resource allocation.

A strong relationship between self-induced cortical negativity and reaction time, signal detection and short-term memory performance has been reported in many studies in humans [3, 26]. For example, cognitive task (e.g., memory retrieval) or motor tasks (e.g., pressing a button) are performed significantly better when presented after spontaneous or self-induced cortical negativity. Studies of Prof. Birbaumer and co-workers [4, 6], as well as other researchers have demonstrated that healthy subjects and neurological patients can attain reliable control over their SCP amplitude at vertex, frontal and parietal locations with operant conditioning. Moreover, subjects can learn to control SCP differences between the left and right hemisphere [5]. In the operant control condition participants have to generate SCP shifts of different amplitude and polarity. Participants were reinforced for increasing or reducing negative SCP shifts, depending on the discriminative stimuli (i.e. the letters A or B presented on the screen), where continuous feedback of the SCP amplitude shifts was presented to a participant.

This skill to voluntarily control SCP represents a basis for corresponding brain-computer interfaces. These BCI’s are binary in their nature and are usually referred to as Thought-Translation Devices or TTD’s.

In the original version of the Thought Translation Device (TTD) the self-regulation training and feedback is realized as follows [25]. Participants observe two rectangles (goals); one at the top and one at the bottom of the computer screen. A graphic signal referred to as a "ball" informs the subjects about their SCP shifts. Participants are instructed to move the ball towards either the top or the bottom rectangle as indicated on the screen. Participants are instructed how to move the ball. They are only advised to be attentive to the feedback and sort out the most successful mental strategy. Rhythmic acoustical timing signals are introduced to shorten the time for generation of a particular brain response. A high-pitched

tone marks the beginning of a trial. The first two seconds after the high-pitched tone represent a passive phase, during which the ball does not move. A low-pitched tone introduces an active feedback phase and signals that the ball could be moved. During the active phase subjects attempt to move the ball and the feedback is provided to them.

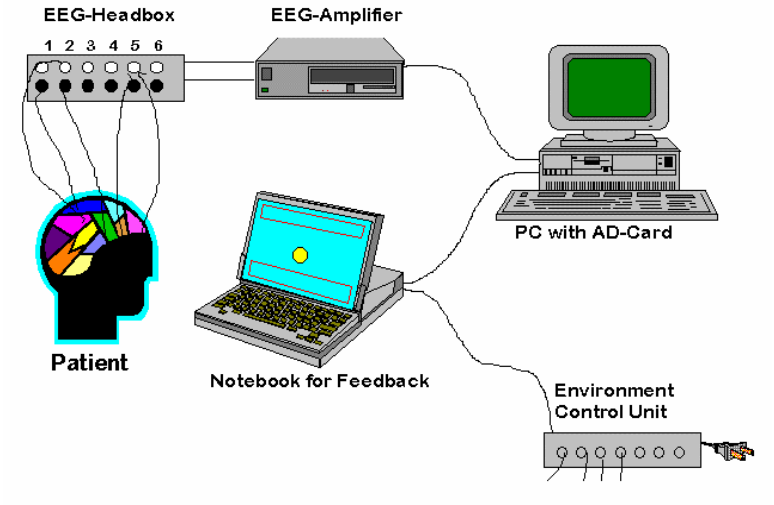

Figure 3-4: The experimental setup of the TTD

The average EEG within a 500 ms interval prior to the low-pitched tone serves as baseline. The ball providing feedback moves in proportion to the integrated EEG of the active phase, which refers to the average of the baseline. The active phase lasts from 2 to 5 s depending on a particular patient. Therefore, the duration of a trial varies from 4 to 7 s, but has to be constant within one session. One training session usually consists of 100 trials. The experimental setup of the TTD [25] can be seen in the figure 3-4.

The EEG is recorded from Cz – a spot on a human scalp according to the international 10-20 system (compare paragraph 2.4.1 and figure 3-5). SCP changes at the vertex move the ball in the vertical direction. Cz is recorded against both A1 (left mastoid) and A2 (right mastoid) at a sampling rate of 256 Hz. A simulated Cz-linked mastoid channel is calculated on-line as ½ [(Cz-A1) + (Cz-A2)]. At the beginning of the trial one of the rectangles is illuminated indicating that it is the target, towards which the ball has to be moved (discriminative stimulus). The rectangles can be illuminated either in random order or in a sequence given by a trainer. Whenever a rectangle is hit as required it flashes as a positive reinforcement. The criterion for a SCP amplitude shift required to move the ball into the rectangle (the hit amplitude) is set individually [24, 25]. Whenever the SCP amplitude shift

exceeds the corresponding threshold, the ball touches the rectangle, and the computer counts a hit. In a more sophisticated version of the thought translation device a smiling face saying "very well", "fantastic" etc. is shown to enhance the reinforcement value. As a rule, patients achieve a stable response accuracy of more than 70% after 3-5 months with 1-2 training sessions per week; each training session consists of 7-12 sessions comprised of 70-100 trials.

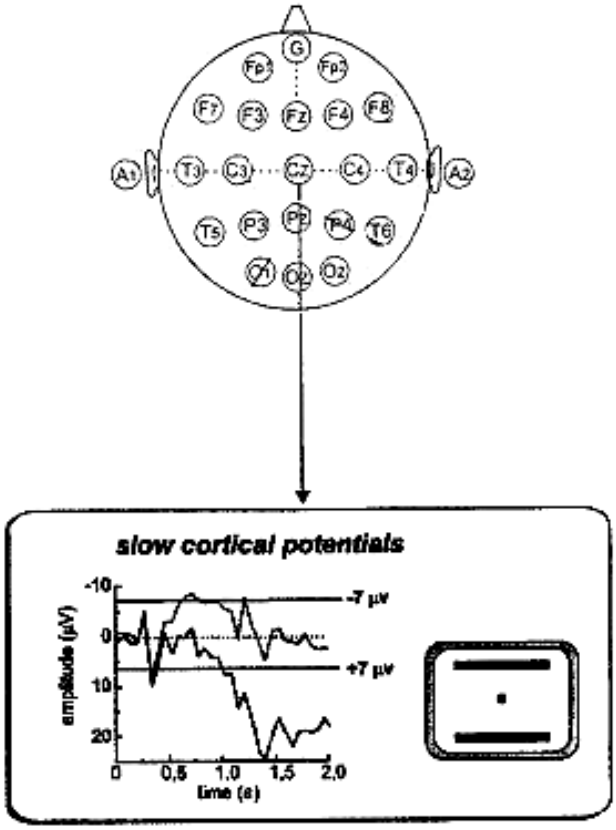

Figure 3-5: The TTD – electrode placement points, EEG-activity and visualization

When the stable patient response accuracy about 70% has been reached, the patient is provided with a Language Support Program (LSP) [36]. The LSP presents letters for selection in form of alternating sets in the bottom goal, while selected letters (a written text) appear in the top goal. The letters (or groups of letters) are presented in some particular, individual for each patient order, which we will discuss below in the section devoted to SI. If the patient wants to select one letter in the currently presented on the screen set, he or she

has to generate a positive SCP shift; otherwise this SCP shift must be nonnegative. Once the set on the screen has been selected, the patient has to select among its subsets until he or she reaches the desired letter, which appears in the written text.

At the beginning patients have to select only among few letters and they have to copy words given by the trainer. This step of the training is called a copy-spelling. Increasing gradually the number of letters and the length of words to copy, the trainer assures a confident patient handling of the TTD. During this phase the response accuracy grows up to about 75%. In the figures 3-6 and 3-7 typical selection and rejection probabilities depending on the number of training sessions are presented (the data are obtained from [36]). It could be seen that the TTD can also be adapted to a particular patient through the varying of a selection threshold value.

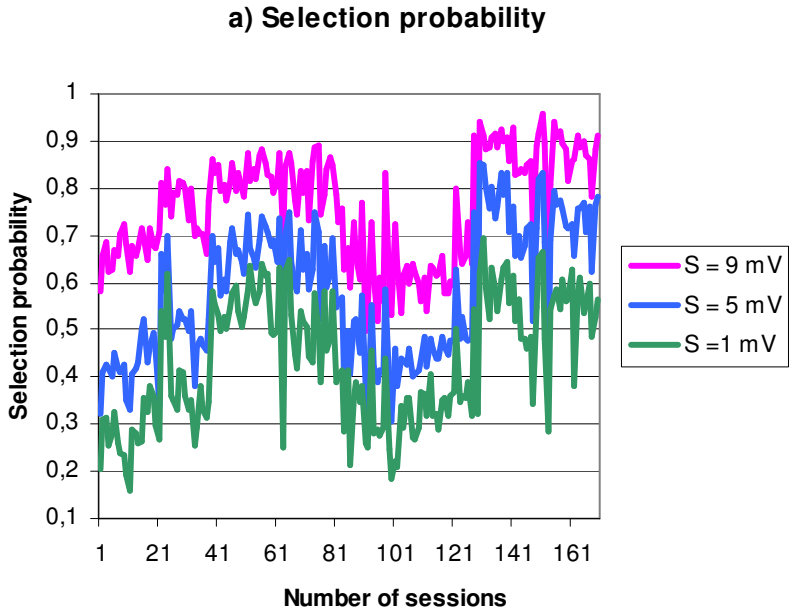

Figure 3-6: Probabilities of selection for a locked-in patient[*]

[*] Probabilities of selection (a) and rejection (b) for a locked-in patient, who learned to operate a Thought-Translation-Devices by producing positive SCP during 170 training sessions for different threshold values of SCP: 1) S = 1 µV, 2) S = 5 µV and 3) S = 9 µV.

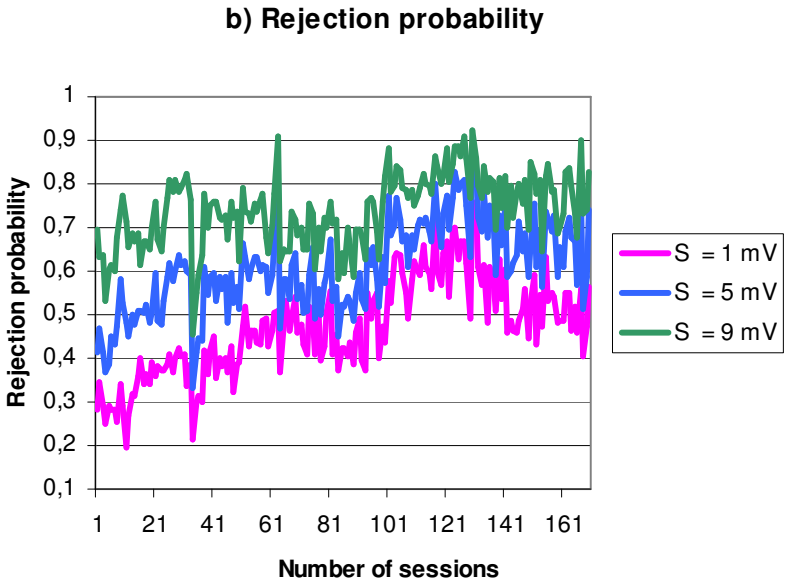

*Figure 3-7: Probabilities of rejection for a locked-in patient**

After this copy-spelling phase patients are allowed to use a free-spelling mode of the Language Support Program. They can write whatever they want to write and so communicate with their environment. In this manner the first completed text ever written with self-regulation of brain signals was communicated and published [7]. This spelling communication mode is not the only one application of the TTD. Some patients are provided with a menu containing preconceived expressions, which were grouped according to their meaning and correspond to patients' daily needs. Patients can not only communicate wordy, but can also independently switch on/off the TV, radio, alarm, light [25], navigate through the internet a special brain controlled internet browser [14]. They can even switch on/off the TTD itself without any assistance [17].

With respect to the practical usage of the Thought-Translation Device it has to be mentioned that altogether 12 patients have been using this device successfully for several months or even years. All of these patients are strongly handicapped; some are completely paralyzed (including artificial feeding and artificial ventilation). A medium of 20 to 40 training sessions per patient was required to achieve stable responses and the accuracy rate of 65% to over 90%. It was also determined that accuracy rates lower than 65% lead to very high error rates that basically make communication impossible. Hence, from the practical

point of view, the first step always has to be conducting copy-spelling trainings and, thus, achieving sufficient accuracy of *at least* 65% - 70%.

3.1.7. Limitations of BCI's, comparison of the existing BCI's and summary

Brain-computer interfaces described above undergo different limitations. According to [26], the main limitations are the following:

- Habituation;
- Interference and distraction;
- Impaired visual system;
- Instability of EEG frequency bands;
- Invasive recording.

Let us take a closer look at these limitations and their impact on the analyzed brain-computer interfaces.

Habituation

Especially the P300 BCI is well-known for its habituation issues. It depends on the nature of the P300 event-related potential and has been shown in several publications [40]. In the μ-rhythm-based BCI and in the Thought-Translation Device the long training periods exposed that the correspondent EEG responses do not habituate and remain the same for weeks, months, and years.

Interference and distraction

This kind of limitations occurs especially in an "every day's life" type of communication, when patients use the BCI device for communicating their actual needs and not only for following trainer's instructions as a part of a training session. Here, again, particularly the P300 BCI is the subject of interference and distraction due to its nature. As described above, the usage of the P300 BCI requires on the one hand counting the number of flashes of the matrix cell containing the target letter. On the other hand, a patient has to concentrate on the message he or she is going to communicate, thus, thinking of what letters, words and sentences are going to be used. This situation requires split attention and, therefore, leads to distracted concentration and lower selection accuracy.

Other BCI's that are based upon production of specific EEG patterns (e.g. the TTD) can also be a subject of interference and distraction, whereat the mechanism is different. In this case it is a correlation or interference of thoughts a patient tries to communicate and cognitive responses a patient executes in order to generate a specific EEG pattern. Since generation of desired EEG patterns occurs individually for each patient, and the used techniques are individual too, it is impossible to foresee what thoughts would lead to interferences. Nevertheless, since the control of EEG patterns can be learned, upcoming interferences can be overcome through additional training. For instance, in case of the TTD it has been reported for to patients that their accuracy and performance decreased significantly during the transfer from the cursor movement phase to the copy-spelling phase and from the copy-spelling phase to free communication [7]. However, patients managed to return to the high level of accuracy and performance again after several additional weeks of training.

Impaired visual system

While using BCI's based upon visual event-related potentials, patients have to fixate their eyes at items they want to select. In case patients occasionally move their eyes off the target, for instance, as a result of poor control of eye muscles, a communication error takes place, since false target is fixated. Therefore, visual attention is absolutely critical for BCI that are based purely on visual attention, such as the P300 BCI. However, both visual attention and the visiomotor system may be impacted in paralyzed patients. Moreover, in case of the locked-in syndrome even eye muscle paralysis can occur, thus, making it impossible for a patient both to control horizontal and vertical movement of his eyes and to move his eyelids voluntarily. Also the eyesight can decrease. Hence, BCI's related only on visual event-related potentials are not suitable for patients, who already suffer from visual impairment, and can become unsuitable for other paralyzed patients as their disease is progressing. Consecutively, BCI's related upon operant learning require additional feedback options apart from visual stimuli. For example, the Thought-Translation Device possesses auditory and tactile feedback modes, thus, reducing possible impacts of decreasing visual abilities.

Instability of EEG frequency bands

EEG responses are characterized through the fact that they do not necessarily remain the same, but can vary during conduction of cognitive tasks [e.g. 41]. This instability leads to

difficulties with respect to correct recognition and classification of frequency bands. It has been shown, for instance, that even within a very short period of time of less than one hour an EEG pattern changed significantly while a patient underwent a very simple aversive conditioning paradigm [13]. In case of the TTD, however, no major issues with respect to unstable EEG responses have been reported.

Invasive recording

Some brain-computer interfaces – e.g. the action potential frequency brain-computer interface – are based upon invasive recording techniques that require surgical intervention. These techniques lead, on the one hand, to increasing costs and risks that are typical for any kind of brain neurosurgery. On the other hand, the positive outcome of such comprehensive techniques is rather very limited, if humanly existing. It could not be shown that the signal-to-noise ratio for electrical responses is higher in case of invasive recording than for non-invasive recording from the scalp. Besides, invasive recording always results in very long training periods and implanted electrodes remain intact for a limited length of time, thus, resulting in inadequate overall communication from the patient's point of view. Hence, the preference is mostly given to non-invasive BCI's, such as the TTD.

Altogether, the TTD is less impacted by existing limitations than other BCI's, which results in its higher practical applicability. Below one can find an overall table, which allows its comparison with other existing BCI's from the point of view of speed, accuracy and practical usage by patients.

BCI	Speed	Accuracy	Practical usage
P300 event-related potential	7.8 characters per minute	~ 80%	4 patients, no locked-in patients
Visual event-related potential	10-12 words per minute	Not reported	1 locked-in patient
EEG frequency patterns – motor imagery	No letter selection, but orthosis	90% - 100%	1 locked-in patient
EEG frequency patterns – cognitive tasks	Not reported	70% - 90%	None

Action potential firing rate	3 characters per minute	Not reported	1 locked-in patient
M-rhythm	No letter selection	~ 90%	Several patients
TTD - slow cortical potentials	0.15-3 characters per minute	70% - 90%	12 locked-in patients (partially or fully locked-in)

Table 3-8: Comparison of the existing BCI's

Thus, we have learned what kinds of principles are used to allow direct communication between the brain and the computer, what brain-computer interfaces are built upon these principles and how the actual communication is carried out. We have also seen that the majority of BCI's is binary. We have figured out that the Thought-Translation Device represents the only existing BCI that is in practical use for successful communication with fully locked-in patients for a longer period of time and that it is characterized through relatively high speed and accuracy, relatively low average training duration and the step-back option as a mean of error correction. Besides, the TTD is less affected by common BCI limitations than other BCI's.

3.2. Problem definition for optimal BCI's in terms of coding and communication theory

3.2.1. Problem definition for a non-zero error communication

All brain-computer interfaces described above can deal with both letters and menu items. Generally speaking, both letters and items are objects of some particular alphabet, upon which a BCI is based. For simplicity and due to issues of practical usage, in the following we will discuss alphabets consisting of letters. However, all formulas, results and deductions can be applied to alphabets consisting of menu items or other objects.

With the only one exception of the P300-based BCI, which operates randomly due to its nature, all described brain-computer interfaces deal with a *sequence* of letters presented to a patient, from which he or she has to choose. These letters can be presented either separately or in sets and subsets. The choice itself happens always in a binary manner, i.e. the patient either *selects* or *rejects* a particular letter or a set, in which the letter is contained.

The sequence, in which letters are presented to a patient, influences greatly both the speed of communication and its accuracy and, therefore, is absolutely crucial for the acceptance of the BCI as a means of communication by a patient. Cases have been reported [e.g. 26], when patients refused to use particular BCI's due to slow and inaccurate communication. As one can easily recognize, an optimal sequence can vary from case to case and relies both on the alphabet itself and on the skills and capabilities of a particular patient. Thus, for instance, optimal sequences can differ for German and English languages, even if all other parameters are the same. On the other hand, somebody whose level of voluntarily control of his or her responses is low tends to make more mistakes and, therefore, needs a different sequence than a person who hardly makes any wrong choices, while using the same language.

From the point of view of communication theory, the sequence of subsets, in which a particular letter is presented, represents essentially a prefix code of this letter. Each letter in the alphabet possesses its own unique code, through which it can be specifically identified. The length of the code corresponds with the number of steps that are required to reach the desired letter in one attempt, thus, not yet taking into account possible mistakes. Since prefix codes and trees are closely related (compare chapter 2), the length of the code represents consequently the depth, in which the desired letter is located in a corresponding tree, whereat the depth can vary from letter to letter.

The psychological problem of creation of an optimal sequence for handicapped patients leads, therefore, to a general communication problem of creation of an optimal variable-length code for some particular alphabet. The term *optimal* implies from the psychological point of view achieving the best compromise between high speed of communication and high accuracy. From the point of view of communication theory it means that it is required to define a correspondent cost function, which, in turn, needs to be either maximized or minimized in order to create an optimal binary characteristic code (compare chapter 2).

The problem of creation of an optimal variable-length code is a well-known problem, which arose first in communication theory and computer science in the context of data compression. It can be solved with help of the Huffman's algorithm through bottom-up creation of optimal binary trees. However, it can only be solved this way under the absolutely crucial assumption that the selection of right or left edges in binary trees

happens exactly as desired, e.g. the proper edge is always selected if desired. Hence, it implies that there is no noise or other disturbing factors. Besides, it implies there is no need for any means of error correction. In case of brain-computer interfaces or other devices apart from computers such kind of selection is not present at all times. For BCI's, for instance, the success of desired selection of a proper edge (which, in turn, corresponds to successful selection or rejection of some subset of letters) greatly depends on the BCI itself, the level of patient's training, environmental factors and is, therefore, strongly dependent on particular patient. As shown in the section 3.2, for some BCI's results can be effectively enhanced through training. Usually, one speaks about selection or rejection probabilities, which are individual for each patient (compare section 3.5.6). Thus, in many cases and, in particular, in case of BCI's one of crucial assumptions, upon which the Huffman's algorithm is based – the equity of selection and rejection probabilities to 1 – is not fulfilled. Patients can and do make mistakes while selecting letters or sets of letters. These mistakes must be taken into account from both points of view: creation of an optimal sequence and providing a means of error correction, for instance, with help of a delete option.

Thus, the Huffman's algorithm itself does not cover this more general problem of communication. In the accessible literature only two approaches are descried, which strive to solve this problem. We will discuss both approaches below in the section 3.2.2.

Therefore, the overall problem this thesis deals with can be defined as follows:

To define an optimization criterion and to find corresponding algorithms for creation of an optimal variable-length binary characteristic code (e.g. an optimal binary tree) for some particular alphabet with a means of error correction, whereat code signals are delivered with possible errors (e.g. in an optimal binary tree occasionally wrong edges are taken and selection and rejection probabilities are not equal to 1).

As one can easily see from the above definition, the overall problem represents a general problem of communication and covers a very broad specter of applications. It can be used in *any* system dealing with binary data compression and transmission disturbances, when binary signals are delivered with undesirable noise. The problem is, therefore, not limited to binary brain-computer interfaces, but rather completely fulfills the requirements for optimal BCI's. However, since the problem definition itself arose from the usage of BCI's in general

and the Thought-Translation Device in particular, all examples, calculations and practical usage will be, first and foremost, discussed in the BCI context.

3.2.2. Known approaches for a non-zero error communication problem

As mentioned above, Huffman invented a greedy algorithm that constructs an optimal prefix code called a Huffman code. This algorithm builds the tree *T* corresponding to the optimal code in a bottom-up manner. It begins with a set of $|\Omega|$ leaves and performs a sequence of $|\Omega|$-1 "merging" operations to create a final tree. The result of the merger of two objects is a new object, which frequency is the sum of the frequencies of the two objects that were merged.

In terms of BCI's, this optimization corresponds to minimization of the expectation of the number of selection steps, which are needed to write one letter from the alphabet in the case when each single selection can be made without any errors (selection probability *p* equals rejection probability *q* equals 1) and therefore, no delete-option is needed. In the common case of non-zero error communication, a synthesis of the optimal tree has to take into account that the path from the root to the leaf, where a letter $c \in \Omega$ is placed, consists of $x_T(c)$ left- and $y_T(c)$ right-edges, so that the depth of the letter *c* accounts to $d_T(c) = x_T(c) + y_T(c)$, and the probability to achieve this leaf (i.e. to write the letter into a final text) without any error is $p^{x_T(c)} q^{y_T(c)}$. This means also that with the probability of $1 - \sum_{c \in \Omega} f(c) p^{x_T(c)} q^{y_T(c)}$ we will need a correction with the delete-option after every writing attempt. This delete-option must be built into the tree. Thus, again, the Huffman's algorithm itself can not be used in such a situation and a new other method need to be developed in order to optimize a Spelling Interface. Below we will discuss two approaches to the problem, which were published in the last time.

In spite of the world wide application of spelling communication systems for the physically disabled, who control them with a binary signal using their limited motor abilities, we can not, with exception of [36] and [37], find sources in the accessible literature, which contain an analysis or estimation of the actual effectiveness for their spelling interfaces. As described in the chapter 3, most of communication systems operate on the following two ways. Either they use an alphabetical or frequency ordered sequential presentation of letters. Or a set of

letters is arranged as a two-dimensional array with the possibility to select a letter in a „row-column“ manner. For the overwhelming majority of such systems the binary signal is assumed to be delivered into the spelling interface without errors. Thus, these systems are not equipped with means for correction, which is usually sufficient in case when the error probability is substantially higher than zero. On contrary, the publications [36] and [37] mentioned above deal exactly with the TTD Spelling Interface, and, therefore, will be thoroughly analyzed here.

The paper [36] discusses the design of a LSP interface (Spelling Interface for the TTD), including the selection of a set of symbols, the organization of their presentation and the LSP equipment with an error correction means (a *go-back* function). Authors used a set consisting of 32 (i.e., 2^5) symbols: 29 letters of the German alphabet, 2 punctuation marks {,} and {.}, and the symbol {-} indicating space between words. This symbol set makes up the basis for different spelling structures, which may be described in terms of „parent-children” relations. The whole set Ω as a parent-set consists of children $\Omega^1{}_1$, $\Omega^1{}_2$,..., which are disjoint subsets of the parent, and represents the highest level of the SI structure (level 1). Each subset $\Omega^i{}_k$ at the level *i* is the parent of disjoint subsets $\Omega^{i+1}{}_n$, $\Omega^{i+1}{}_{n+1}$ ... at the level *i+1*. This partition process continues until each child contains only one symbol. In terms of the graph theory, such structure may be described as a tree. The nodes of the tree are subsets of Ω, its edges connect parents with their children, and its end nodes (leaves) are subsets containing only one symbol. The particular subset of *alphabetical symmetrical* spelling structures was analyzed. These structures satisfy the following two properties:

1. The symbols in each subset are arranged in alphabetical order, and the division of each subset into its children subsets is also alphabetical.
2. All parents belonging to the same level generate an equal number of children.

There are 16 different symbol structures, which posses these properties. Each structure was analyzed and compared with others with the help of the following tree evaluation criteria: the average time (number of selection steps), which is necessary for the correct selection of one letter, the mean probability of the correct selection of one letter and the expectation of the number of trials which are necessary to write one letter if an upper limit of the writing time is given. The results concerning the third criterion are the most important

from the clinical viewpoint as they assess the complete SI design and allow choosing the best among others SI for any patient's values of selection probability and rejection probability. For an efficient calculation of this criterion a statistical modeling (simulation) was used. A computer program simulated the writing process for each symbol with any given spelling structure and any combination of the selection/rejection probability values and then estimated the statistic characteristics for a certain number of realizations. These characteristics were averaged and weighted by letter frequencies in German. As the result of this investigation the optimal symbol structure for every pair (selection probability, rejection probability) was found. These structures are used in the TTD (as the LSP) up to date.

There is no doubt that the results of this investigation are of significant theoretical and practical importance – the presented criteria, especially the expectation of the number of trials which are necessary to write one letter, are relevant to practical problems of augmentative communication. The first successful matter-of-fact application of the BCI-spelling was realized with the SI found in this way [7, 25]. At the same time it should be noted that this paper devotes to only the evaluation of the *given binary tree* and for the *given arrangement of letters* over it, but not to the problem of an optimal binary tree synthesis. Although the binary trees examined in this paper are relevant to practical communication (they use the over learned alphabetical order and a simple symmetry, so that each patient can easily predict, which set of symbols will be presented for selection at the next step within one writing session), it is clear that the search for the optimal SI can not be restricted only to such structures. Not every alphabet in a general sense (consisting of menu-items) can be a basis for alphabetical ordered symmetrical spelling structures. Besides, even when such structures can be built for a given alphabet, it is impossible to judge apriori whether these structures have advantage compared to some other letter arrangements.

In the paper [37] an algorithm for design of a spelling interface based on a modified famous Huffman's algorithm is presented. This algorithm pretends to build a full binary tree corresponding to the following optimization criterion - the average probability to write one character without any errors

$$\Phi = \sum_{c \in \Omega} f(c) \cdot A(c) \to \min,$$

where *c* is the character from the alphabet Ω, *f(c)* is the frequency of occurrence for *c* and *A(c)* is the probability of the unerring attainment of character *c* (i.e. the probability of reaching the leaf, where character *c* is located, without any errors). The algorithm is a certain generalization of Huffman's algorithm for the case when each parent node of the tree has left and right children with different weights $p<1$ and $q<1$, where *p* and *q* are the probabilities to select the left child and the right child respectively. The presented algorithm also builds the full binary tree corresponding to the optimal Φ in a bottom-up manner. It also begins with a set of n leaves and performs a sequence of *n*-1 merging operations to create the final tree. For each merging operation firstly the two least-frequent objects (characters or sets of characters) must be identified to merge together. The result of the merger of two objects is a new object, which frequency is the sum of the frequencies of the two weighted by *p* and *q* objects merged, respectively. When $p \le q$, and *a* and *b* denote these merged two objects then the frequency of the new object in case of $f(a) \le f(b)$ is $f_{new} = p \cdot f(a) + q \cdot f(b)$, otherwise it is $f_{new} = p \cdot f(b) + q \cdot f(a)$. In the second part of the paper a delete-option is discussed as a means to correct possible errors in writing and a method to build this delete-option into the optimal binary tree.

It should be noted here that the proof of the correctness of the algorithm the authors presented did not take into account a possible lost of the greedy-choice property upon some special circumstances. When selection probabilities *p* and *q* essentially distinct one from the other (a so-called “strong” inhomogeneous SI) and the number of letters is large, the globally optimal solution could not be arrived at by making a locally optimal (greedy) choice. It means that algorithms, which build the full binary tree in a bottom-up manner, can not be applied to this case. The globally optimal solution could not be arrived at by making a greedy choice just because the two least-frequent objects can belong to different parent nodes at different levels of the optimal binary tree. In this case the deepest leaves of the tree do not correspond to minimal values of probabilities to reach these leaves without any error.

To estimate these limitations of an algorithm practical applicability let us consider $p < q$. The analyzed condition is the following “a probability to reach any leaf at the (*i*-1)-th level of the tree must be not less than a probability to reach any leaf at the *i*-th level of the tree”. This condition is enough for the algorithm applicability and can be expressed as $p^{i-1} \ge q^i$,

because p^{i-1} is the minimal probability of the unerring attainment the (*i*-1)-th level and q^i is the maximal probability of the unerring attainment the *i*-th level of the tree. From this follows that $q^{i/(i-1)} \le p < q$. Thus, the value *p* must be limited not only with respect to its maximum but also with respect to it's minimum. Figure 3-9 demonstrates the left part of this inequity (a minimal value of the probability *p*) for different value *q* depending of the depth of the tree.

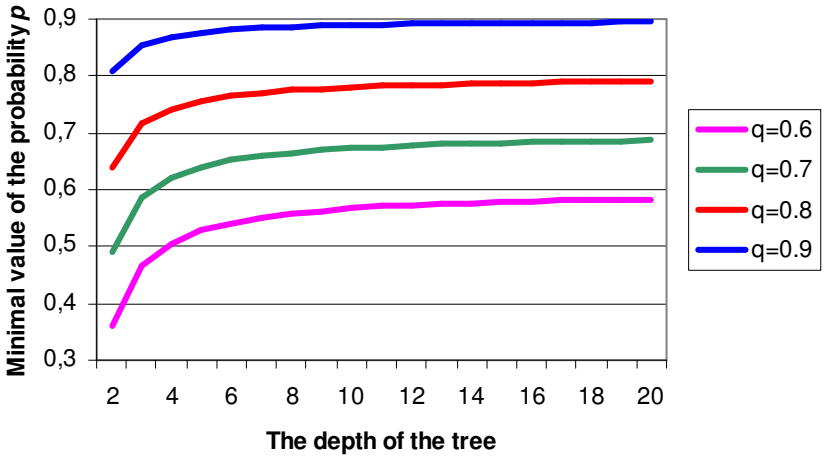

Figure 3-9: The minimal value of p, which limits the applicability of the algorithm, depending on the tree depth for different values of q

Altogether, the algorithm works very well in case of equal probabilities and represents the generalization of the Huffman's algorithms, which works only for the selection probabilities equal to 1. As could be seen in this figure, the algorithm presented in [37] can be applied to a relatively limited class of binary SI with non-equal probabilities: for example, if *q*=0.7 and under an assumption that the depth of the tree is not greater than 8 levels, the value of *p* must satisfy the inequity 0.665< *p* <0.7. If *q*=0.7 and under an assumption that the depth is not greater than 4 levels, the value of *p* must satisfy the inequity 0.621<*p*<0.7 etc. So, greedy algorithms can be applied only to "almost" homogeneous SI with equal (or almost equal) selection probabilities.

Thus, papers devoted to the optimization problem for binary SI deal either with methods of evaluations for given SI or with the optimization method, which solves the problem for the criterion "the average probability to write one character without any errors" for *almost equal selection probability values*. It means that the overall problem described above can not be denoted as completely solved and any successful effort of further investigation in this field

and further development of new methods for an optimal binary SI synthesis will be very valuable and beneficial. This is especially factual taking into account the value the faster and more accurate communication will bring to handicapped patients.

In the chapter 4 we will present a number of approaches and algorithms dealing with exactly with the overall problem of a non-zero error communication.

4. Algorithms for the optimal building (synthesis) of binary SI's

4.1. Optimization criterion

As we have seen above, a binary SI consists of a binary tree with n internal nodes (which means n+1 leaves), a pair of selection probabilities (p, q), where p and q are probabilities of selecting respectively the left child and the right child when desired, and an alphabet Ω, which consists of exactly n letters or menu items $\{c_i\}$ with corresponding frequencies of occurrence $\{f(c_i)\}$. In each leaf of the tree one letter from the alphabet is placed, and one remaining leaf is used as a delete-option leaf (or δ-leaf), which selection means a deletion of the last written letter from a written text. A distribution of letters over the tree, which defines an order of selection, is also an important component of SI.

The most natural criterion for the evaluation of different SI's from the point of view of practical usage by handicapped patients is the average expectation M of the number of trials (or writing steps) that are necessary to write one letter (to select one menu item) taking into account a needed correction of possible writing mistakes. For the SI with the "delete" option δ this criterion could be expressed as following. Let p_i be a probability to write the i^{th} letter (to achieve the i^{th} leave in the tree) without errors, in the binary case

$$p_i = p^{x_i} q^{y_i},$$

where $x(i)$, $y(i)$ are numbers of "left" and "right" steps from the root to the i^{th}-leave in the tree respectively. A corresponding number of steps (left and right) can be, therefore, expressed as

$$S_i = x(i) + y(i).$$

Let p_δ and S_δ be the probability to achieve the δ-leave in the tree without errors and the corresponding number of steps respectively; R – the expected number of steps by a so-called "false attempt". The last means the number of steps towards a needed letter and then, after an occasional error has been made, the number of steps needed to escape the SI-tree and to start a new attempt. Here we assume that R is a constant for the given tree cardinality n+1 and will discuss this assumption later on.

The expectation of number of steps needed to achieve, if desired, the δ-leave in the tree can therefore be expressed as:

$$M_\delta = p_\delta S_\delta + (1 - p_\delta)(R + 2M_\delta).$$

The term $p_\delta S_\delta$ represents the average number of steps required to achieve the δ-leave in the first attempt, which will happen with the probability of $p_\delta > 0.5$. If the probability p_δ is not larger than 0.5 it means that the patient is effectively not capable of deleting the wrong written letter and, therefore, requires further training sessions (as we have seen in the section 3, average selection and rejection probabilities for TTD after the standard training course account to over 75% or 0.75). If the δ-leave could not be reached in the first attempt; we need to leave the tree spending *R* steps, to cancel the false letter and to repeat the attempt. This, in turn, will happen with the probability of (1- p_δ) etc. It gives us the formula

$$M_\delta = (p_\delta S_\delta + (1 - p_\delta)R)/(2p_\delta - 1).$$

The expectation of number of steps needed to write the i^{th} letter (if desired) taking into account the required correction of all mistakes by deleting wrong written letters with the delete option can be written in the same way as:

$$M_i = \sum_{j=0}^{\infty} (1 - p_i)^j \left[p_i S_i + (1 - p_i)(R + M_\delta) \right] = S_i + \frac{(1 - p_i) p_\delta (S_\delta + R)}{p_i (2p_\delta - 1)}.$$

Thus, the optimization criterion - the average expectation of the number of trials, which are necessary to write one letter- we can write as

$$M = \sum_{i=1}^{n} f_i \left[S_i + \frac{(1 - p_i) p_\delta (S_\delta + R)}{p_i (2p_\delta - 1)} \right] = \sum_{i=1}^{n} f_i M_i \to \min, \qquad (4.1)$$

where f_i - the frequency of i^{th} letter in the alphabet and *n* – the number of letters contained in this alphabet.

In order to express *R = R(n*+1*)* for the binary case we can use an estimation of Munro [32] (see also [43]), who described random walks on binary trees. Such a walk starts at the root and takes a right or a left sub-tree with probability *p* or 1-*p* respectively. The walk stops, when a chosen sub-tree is empty. Munro proves that for *p*=0.5 (meaning that probabilities to take the left or the right sub-tree are equal, which, in our case, is equal to *p*=*q*) the average number of random steps is

$$R(n+1) = 2 - \frac{6}{n+3},$$

and, therefore, R depends solely on the cardinality (that is, number of leaves) of a given tree.

It should be noted here that the above formulas work not only for binary SI's, but also for k-ary SI's. In case of k-ary SI's, however, *R* needs to be taken into account accordingly.

In case the binary tree is given, an optimal distribution of the given set of letters over the tree could be easily found following the next procedure: in case there are no leaves in the tree with p_i>0.5 the tree does not need to be analysed further. Otherwise the delete option δ is placed consequently in those leaves of the tree, which p_i is higher than 0.5, and for the values p_δ and S_δ, which are obtained in this way, the optimal letters' distribution over the leaves is the distribution, when the most frequent letter is placed in the leaf with the minimal value of M_i. The second most frequent letter is placed in the leaf with the second lowest value of M_i etc.

This property obviously follows directly from the features of the bi-linear form (4.1), in particular, from the fact that in case $f_1 \geq f_2 \geq \ldots \geq f_n > 0$ the bi-linear form

$$\Delta = \sum_{i=1}^{n} f_i \lambda_i$$

reaches its minimal value, if and only if $0 < \lambda_1 \leq \lambda_2 \ldots \leq \lambda_n$. To show this let us assume that

$$\Delta_1 = \sum_{i \neq s,t} f_i \lambda_i + f_s \lambda_t + f_t \lambda_s < \Delta = \sum_{i=1}^{n} f_i \lambda$$

and calculate

$$\Delta_1 - \Delta = f_s \lambda_t + f_t \lambda_s - f_s \lambda_s - f_t \lambda_t = (f_s - f_t)(\lambda_t - \lambda_s) > 0,$$

which contradicts our assumption. It means that for the given binary tree we need only to place the delete option δ, the other letters can be placed automatically. Thus, for the given binary tree with *n*+1 leaves there are exactly *n*+1variants to calculate criterion values, which need to be compared with each other in order to obtain the optimal tree.

One can also note that for binary trees and for $\forall p_i \equiv 1$ ($p \equiv q \equiv 1$) there is no need of the δ-leave in the tree and the solution of the problem (4.1) can be obtained with help of the

famous Huffman's algorithm [9, 15]. Besides that, in order to express the criterion (1) we used no assumption connected with the "binary nature" of SI. It means that this criterion could be used also for k-ary SI, where $k>2$.

4.2. Decoding of P-sequences for criterion computation and the Full-Search algorithm

4.2.1. Algorithm for decoding of the given P-sequence

As we have seen in the previous section, for a given binary tree an optimal distribution of letters can be easily found through computation of the criterion (4.1). Thus, for our purposes we need to compute values of the optimization criteria (4.1) and to compare these values in order to choose the minimum. To calculate these values in the effective way, we need a decoding algorithm, which can compute pairs of left and right steps (x_i, y_i) for all leaves $i \in [0, n+1]$ from the given P-sequence $(p_1, p_2, \ldots, p_n)$, since with help of P-sequences one can describe a binary tree in the easiest way. Such an algorithm can be written as shown below. Note that a stack S, which implements a LIFO (last-in, first-out) policy for a set of leaves' "power" coordinates $\{(x, y)\}$ is used. Here x represents the number of left steps and y – the number of right steps required to achieve a particular letter.

Algorithm 4-1 – Decoding of the given P-sequence

$p_0 \leftarrow 0$

$p_{n+1} \leftarrow p_n$

$x \leftarrow 0$

$y \leftarrow 0$

Push(S, (x, y))

for $i \leftarrow 1$ **to** $n+1$ **do**

    (x, y) $\leftarrow$ Pop(S)

    for $j \leftarrow 0$ **to** $p_i - p_{i-1} - 1$ **do**

        $x_R \leftarrow x$

        $y_R \leftarrow y + 1$

        Push(S, (x_R, y_R))

        $x \leftarrow x + 1$

    enddo

$x_i \leftarrow x$

$y_i \leftarrow y$

enddo

Example 4-1

This algorithm can be illustrated with the following example. The P-sequence $P = (2, 4, 4, 4, 5)$ is given. The corresponding binary tree looks as shown in the figure 4-1, it is the same tree as in the figure 2-1, but each leaf of it has been assigned a number. We show all calculations for each leaf $i = 1, 2, \ldots, n+1$.

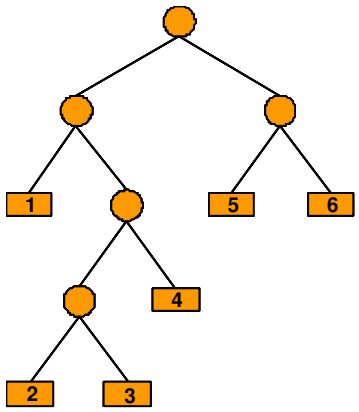

Figure 4-1: A full binary tree with 6 numerated leaves

We follow the algorithm step by step and describe the values of all variables and the stand of the stack *S,* which will be filled (with the standard Push-procedure) and popped (with the standard Pop-procedure) from the left to the right. We begin with $p_1=2$, $p_2=p_3=p_4= 4$, $p_5=5$.

Step 0. $p_0 = 0$; $p_6 = 5$; $(x, y) = 0$; $S = \{(0, 0)\}$

Step 1. $i = 1$; $j = 0$; $(x, y) = (0, 0)$; $(x_R, y_R) = (0, 1)$; $S = \{(0, 1)\}$; $(x, y) = (1, 0)$

$i = 1$; $j = 1$; $(x, y) = (1, 0)$; $(x_R, y_R) = (1, 1)$; $S = \{(1, 1), (0, 1)\}$; $(x, y) = (2, 0)$

$(x_1, y_1) = (2, 0)$

Step 2. $i = 2$; $j = 0$; $(x, y) = (1, 1)$; $(x_R, y_R) = (1, 2)$; $S = \{(1, 2), (0, 1)\}$; $(x, y) = (2, 1)$

$i = 2$; $j = 1$; $(x, y) = (2, 1)$; $(x_R, y_R) = (2, 2)$; $S = \{(2, 2), (1, 2), (0, 1)\}$; $(x, y) = (3, 1)$

$(x_2, y_2) = (3, 1)$

Step 3. $i = 3$; $(x, y) = (2, 2)$; $S = \{(1, 2), (0, 1)\}$;

(x_3, y_3) = (2, 2)

Step 4. $i = 4$; $(x, y) = (1, 2)$; $S = \{(0, 1)\}$;

(x_4, y_4) = (1, 2)

Step 5. $i = 5$; $j = 0$; $(x, y) = (0, 1)$; $(x_R, y_R) = (0, 2)$; $S = \{(0, 2)\}$; $(x, y) = (1, 1)$

(x_5, y_5) = (1, 1)

Step 6. $i = 6$; $(x, y) = (0, 2)$; $S = \varnothing$;

(x_6, y_6) = (0, 2)

4.2.2. The Full-Search algorithm

The algorithm 4-2 for generation of all binary trees with n internal nodes (the Full-Search algorithm) bases on the following lemma and the following two theorems.

Lemma 4-1

Given two trees T_1 and T_2, the p-sequence of the product of T_1 and T_2 is

$$p_{OT_1T_2} = (p_{T_1}(1)+1,..., p_{T1}(|T_1|-1)+1, |T_1|, p_{T_2}(1)+|T_1|,..., p_{T_2}(|T_2|-1)+|T_1|).$$

Theorem 4-1

An integer sequence (p_1, p_2, …, p_n) is the P-sequence of a binary tree with n internal nodes, if and only if:

1) $p_n = n$
2) $\forall i \in [1, n\text{-}1]$: $p_i \le p_{i+1}$
3) $\forall i \in [1, n\text{-}1]$: $p_i \ge i$

Theorem 4-2

Given two trees T_1 and T_2, such that $|T_1| = |T_2|$, T_1 and T_2 are in B-order, so that $T_1 < T_2$, if and only if the P-sequences for T_1 is lexicographically less then the P-sequences for T_2.

The following algorithm generates, corresponding to the both theorems, all P-sequences in the lexicographical order beginning with (1, 2, …, n) and ending with (n, n, …, n).

Algorithm 4-2 – The lexicographical generation of all P-sequences for binary trees with n internal nodes (e.g., n+1 leaves) (the Full-Search algorithm)

for i =1 **to** n **do**
$p_i = i$
enddo
while $i = \max\{k|\ p_k < n\}$ *exists* **do**
$p_i \leftarrow p_i + 1$
for $j = i + 1$ **to** n **do**
$p_j \leftarrow \max\{p_i, j\}$
enddo
enddo

Generating all P-sequences of the length n we can compute values of the optimization criterion (4.1) and compare these values in order to choose the minimum. Thus, for relatively small values n the problem of optimization of binary SI is solved: we generate all P-sequences with the Full-Search algorithm 4-2 and compute the corresponding values of our criterion (4.1), using the decoding algorithm 4-1 and looking for an optimal letters' distribution over the binary tree. Our calculation experiments have shown that for optimisation of binary SI over the alphabet with 14 letters and one delete option (14 internal nodes tree) we need with this Full-Search algorithm about 5 min, when calculating on an Intel-Pentium (R) IV CPU with 2GHz. When the number of letters increases the computation time grows dramatically – the Catalan numbers raise exponentially with n. Therefore, to solve the problem for $n > 14$ we need some other generating algorithm. Further on we describe a heuristic idea and a correspondent algorithm, which allows accelerating the search for the optimal binary SI for the case of the inhomogeneous SI.

4.2.3. The usage of asymmetrical features of the SI's and the inhomogeneous algorithm

The difference between p and q can severely influence the shape of the tree: if $p < q$, where p is the probability to select a left sub-tree T_L and q is the probability to select a right tree T_R, the probability to find the optimal tree with $|T_L| < |T_R|$ is essentially higher than to find an optimal tree with $|T_L| > |T_R|$. On the other hand, P-sequences have the following property: if in the P-sequence $p_{i+1} - p_i = s_i > 0$, it means that the i+1-leaf is the left leaf of the tree and

a left branch, where this leaf is placed, has at least s_i leaves on it. Therefore, one can try to estimate values of s_i based upon the ratio of possible leaves in left and right (sub-) branches. This ratio, in turn, depends on values of p and q. Hence, the idea is to set a threshold for values s_i ($i \in [1, n-1]$) so, that the "left careen" of the tree shape is restricted. It should be noted here that these threshold values s_i strongly depend not only on the selection probability values, but also on i – the place in the P-sequence, since it must be taken into account, how many leaves remain in the tree by the preorder walking. Such information can be obtained, for example, from asymmetrical features of the completed binary tree with the selection probabilities $p \neq q$ and the depth n. Further on in this section we will assume for simplicity that $p < q$. This assumption could be always fulfilled through a simple change of direction and gives us the possibility to discuss here in terms of "left-right".

Let C be the completed binary tree with the depth n, which means that all leaves of C have the same depth n and all internal nodes have the degree of 2. Based upon this completed binary tree we try to estimate a ratio of possible leaves in left and right (sub-) branches. Possible leaves are obviously leaves that add to the values of the criterion (4.1) as little as possible. Hence, we only need to take into account nodes that satisfy the inequality

$$S_i + \frac{(1-p_i)q(1+R)}{(2q-1)p_i} \leq n + \frac{(1-q^n)q(1+R)}{(2q-1)q^n}, \qquad (4.2)$$

where $S_i = x(i)+y(i)$ is the number of steps needed to rich the i-node, $p_i = p^{x(i)}q^{y(i)}$ is the probability to rich the i-node without any error, R is the Munro's estimation of the number of random steps along the tree. Nodes (internal and leaves) that satisfy this inequality, add to the values of the criterion (4.1) not more than the leaf adds, which is placed on the maximal remoteness from the root in the right branch of the tree. This leaf belongs to the tree with the P-sequence (n, n, …, n) and can serve as a measure of the minimal possible ("best possible") contribution v (which equals to the right term in the inequality) from the maximum depth n to the criterion (4.1). All the nodes that do not satisfy this inequality are wittingly "unfavourable" to the criterion (4.1) and, therefore, do not need to be taken into account. The inequality allows us to sort out all nodes, which are wittingly "unfavourable" for the criterion (4.1), since their contributions are greater than v and their distance from the root is less than n. Therefore, through analysis of C we estimate a ratio of possible leaves in left and right branches and determine the threshold for s_1. We can repeat this procedure for

other s_i using corresponding completed binary trees and thereby reduce the number of possible P-sequences.

The simplest estimation for the threshold value can be based on the ratio of number of nodes in the left and in the right branches of the tree. It means that the threshold value s_1 for two first members of the P-sequence, s_1: $p_2 - p_1 \le s_1$, can be calculated as:

$$s_1 = \text{entire}\left(\frac{n \cdot N_L(n)}{N_L(n) + N_R(n)}\right),$$

where $N_L(n)$ is the number of nodes in the left branch of the tree with the depth n, which satisfy the inequality (4.2), and $N_R(n)$ is the same number for the right branch. All other values of s_j ($\forall j < n$) can also be calculated in the same fashion – through substitution of (n-j) instead of n in the formula above.

Taking into account all these considerations, we can now formulate features of P-sequences for the case of inhomogeneous SI as:

1) $p_n = n$;
2) $\forall i \in [1, n\text{-}1]$: $p_i \le p_{i+1}$;
3) $\forall i \in [1, n\text{-}1]$: $p_i \ge i$;
4) $\forall i \in [1, n\text{-}1]$: $p_{i+1} - p_i \le s_i$

The next algorithm generates lexicographically all P-sequences with such features and will be used as the main algorithm for a search for the optimal inhomogeneous SI.

Algorithm 4-3 – Generation of P-sequences for inhomogeneous binary SI's with n+1 leaves

for $i = 1$ **to** n **do**
$p_i = i$
enddo
while $i = \max\{k |\ p_k < \min\{p_{k-1} + s_k, n\}\}$ *exists* **do**
$p_i \leftarrow p_i + 1$
for $j = i + 1$ **to** n **do**

$p_j \leftarrow \max\{ p_{j-1}, j\}$

enddo

enddo

By generating all such P-sequences, computing and comparing values of the optimization criterion we can solve the optimization problem, sparing a generation of wittingly "unfavorable" binary trees.

4.2.4. Comparison of the Full-Search algorithm and the inhomogeneous algorithm (example)

The following example helps us to compare the both algorithms described above from the point of view of the value of the optimization criterion and the number of calculation steps.

Example 4-2

Let us consider an alphabet *A,* which consists of 14 letters *A*={a, b, c, d, e, f, g, h, i, j, k, l, m, n} with frequencies of occurrence 0.2, 0.15, 0.12, 0.11, 0.10, 0.08, 0.07, 0.05, 0.04, 0.03, 0.02, 0.015, 0.01 and 0.005 respectively. Table 4-2 illustrates 6 different examples (different values of selection probabilities) of SI-optimization over the alphabet *A* with both algorithms: the Full-Search algorithm (Algorithm 4-2) and the search based on asymmetrical SI features – our main search algorithm (Algorithm 4-3). In this table one can see the number of generated trees, the founded optimal P-sequence, the order of letters (in preorder) and the minimal value of the optimization criterion.

Selection probabilities	Algorithm 4-2 (a full-search algorithm)	Algorithm 4-3
p=0.5 *q*=0.7	Number of trees = 2674440 (3, 4, 4, 6, 7, 7, 10, 10, 10, 11, 12, 13, 14, 14) a, c, b, d, h, e, m, j, f, g, i, k, n, l, δ Criterion =54.835839	Number of trees = 516394 (3, 4, 4, 6, 8, 8, 8, 10, 10, 11, 12, 13, 14, 14) a, c, b, d, m, j, e, h, f, g, i, k, n, l, δ Criterion =54.835839
p=0.6 *q*=0.7	Number of trees = 2674440 (4, 4, 5, 5, 8, 8, 8, 11, 11, 11, 14, 14, 14, 14) f, c, d, a, i, h, b, l, k, g, n, m, j, e, δ Criterion =34.339081	Number of trees = 1276462 (4, 4, 5, 5, 8, 8, 8, 11, 11, 13, 13, 13, 14, 14) f, c, d, a, i, h, b, l, k, n, m, j, g, e, δ Criterion =34.339081
p=0.6 *q*=0.8	Number of trees = 2674440 (3, 4, 4, 6, 7, 7, 9, 9,10, 11, 12, 13, 14, 14) b, f, a, g, i, c, j, d, e, h, k, m, n, l, δ Criterion =20.633935	Number of trees = 980863 (3, 4, 4, 6, 7, 7, 9, 9,10, 11, 12, 13, 14, 14) b, f, a, g, i, c, j, d, e, h, k, m, n, l, δ Criterion =20.633935
p=0.7 *q*=0.8	Number of trees = 2674440 (3, 4, 4, 6, 7, 7, 10, 10, 10, 11, 12, 14, 14, 14) a, c, b, d, h, e, l, j, f, g, i, n, m, k, δ Criterion =14.353286	Number of trees = 1541227 (3, 4, 4, 6, 7, 7, 10, 10, 10, 11, 12, 14, 14, 14) a, c, b, d, h, e, l, j, f, g, i, n, m, k, δ Criterion =14.353286
p=0.7 *q*=0.9	Number of trees = 2674440 (3, 4, 4, 6, 6, 8, 8, 9, 10, 11, 12, 13, 14, 14) c, f, a, g, b, i, d, e, h, j, k, m, n, l, δ Criterion =10.249402	Number of trees = 980863 (3, 4, 4, 6, 6, 8, 8, 9, 10, 11, 12, 13, 14, 14) c, f, a, g, b, i, d, e, h, j, k, m, n, l, δ Criterion =10.249402
p=0.8 *q*=0.9	Number of trees = 2674440 (3, 4, 4, 6, 6, 8, 8, 10, 10, 11, 12, 13, 14, 14) a, d, b, e, c, g, f, j, h, i, k, l, n, m, δ Criterion = 7.793403	Number of trees = 1337206 (3, 4, 4, 6, 6, 8, 8, 10, 10, 11, 12, 13, 14, 14) a, d, b, e, c, g, f, j, h, i, k, l, n, m, δ Criterion = 7.793403

Table 4-2: A comparison of both algorithms for 15-leaves binary SI over the alphabet A for the example 4-2

As could be seen from the table, both algorithms deliver the same results: all criterion values are the same. The difference in the number of generated trees demonstrates benefits of the consideration of asymmetrical SI features – these benefits are significant for cases with essential different values of *p* and *q* – the first example (*p*=0.5, *q*=0.7) shows more than 5-time acceleration in the search. The differences in the optimal trees in the

examples (for instance, for p=0.5, q=0.7 and p=0.6, q=0.7) show that the optimal tree can be not unique – here there are at least 2 optimal trees with the same value of the criterion. Figure 4-3 shows the computation acceleration for SI over the alphabet *A*. The computation acceleration is calculated as a ratio of the number of trees generated by the Full-Search algorithm and the number of trees generated by the main algorithm.

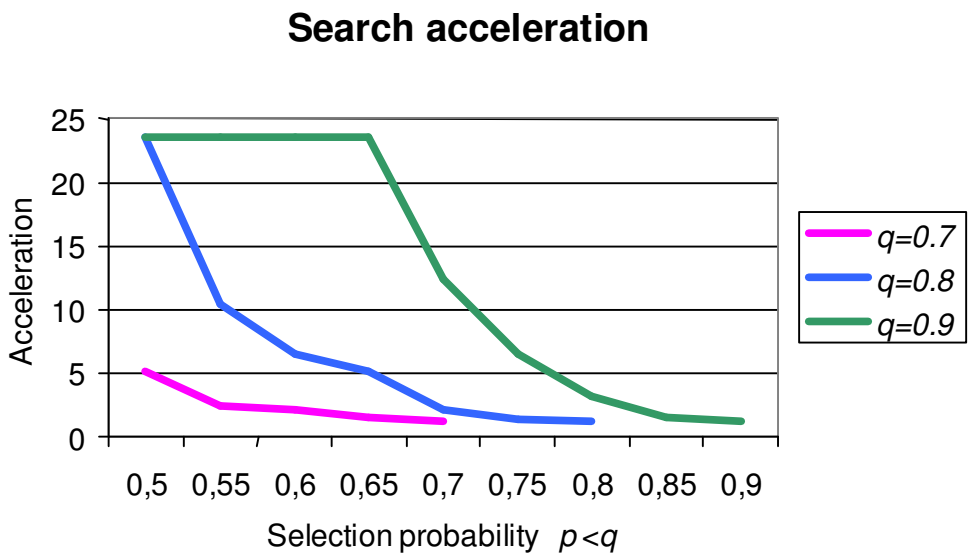

Figure 4-3: Search acceleration depending on selection probability value (Example 4-2)

4.3. Synthesis of optimal homogenous SI's, the auxiliary optimization problem and the merging algorithm

4.3.1. Necessity for an additional algorithm in case of homogeneous SI's

The main algorithm presented above works very well for selection probabilities p and q, which have strongly different values. However, it has almost no benefits in comparison with the Full-Search algorithm when both selection probabilities p and q are close to each other and, therefore, the SI is a homogeneous one. It is clear that in this case statistical features of binary trees can not help find and use some kind of asymmetry just because there is no asymmetry at all. Thus, a new approach has to be found in order to reduce a number of calculations in this homogeneous case.

One such approach is definitely building optimal sub-trees, which are parts of the optimal SI-tree and can be built before the Full-Search algorithm is applied. Below we will present the developed algorithm, which can, under some conditions, build optimal sub-trees from sub sets of letters and in this way to reduce the number of letters for the Full-Search algorithm. It can be helpful especially in the case when section probability values are about 0.6-0.9, which is the typical situation for SI-applications. We assume here that both selection probabilities have the same value *p*. It also should be noted here that in the particular case of $p=q=1$ the solution can be obtained with help of Huffman's algorithm.

When both selection probabilities have the same value *p*, the criterion of the optimization can be transformed as:

$$M(T) = \sum_{i=1}^{n} f_i d_T(a_i) + \frac{p^k (k+R)}{2p^k - 1}\left(\sum_{i=1}^{n} f_i p^{-d_T(a_i)} - 1\right), \qquad (4.3)$$

where *k* is the depth of the delete leaf δ and $d_T(a_i)$ is the depth of the i^{th} letter in the binary tree *T*. It could be seen that the second term of the above expression grows essentially with the increasing *k*. Such a behavior is even stronger, when values of *p* are relatively small (for $p \le (0.5)^{1/2} \approx 0.71$ this is an evident fact) and the cardinality of the alphabet *n* increases. Thus, it is naturally enough to assume that for relatively small values of selection probability *p* or for big alphabet cardinality values *n* the delete leaf is placed at the first level of the optimal tree ($k=1$). This assumption corresponds to the fact that the need for the delete option grows with complexity of SI (cardinality of the alphabet) and with decrease of the selection probability. It is impossible to estimate in general the threshold value of the selection probability p_t such, that for any $p<p_t$ the delete leaf lays at the first level of the optimal tree, because there is a lot of factors involved – the structure of the frequencies distribution over the alphabet, the cardinality of the alphabet, the value of the selection probability. But this effect is easily predictable and can be illustrated in numerous examples. For instance, the figure 4-4 illustrates this fact for 3 different optimal spelling interfaces: over the alphabet *A* with frequencies of occurrence calculated as $f_i = 2^{-i}$ ($i=1, \ldots, n-1$), $f_n = 2^{-(n-1)}$, over the alphabet *B* with $f_i = 1/(i+1)^2$ ($i=1, , \ldots, n-1$), $f_n = 1 - \sum_{i=1}^{n-1} f_i$ and over the alphabet *C* with normalized frequencies of the first *n* English letters ascending ordered by their frequencies

$f_i^* = f_i(\sum_{i=1}^{n} f_i)^{-1}$. One can see that the threshold value for all alphabets increases with the alphabet cardinality and reaches values of about 0.87-0.91 for the cardinality n=14.

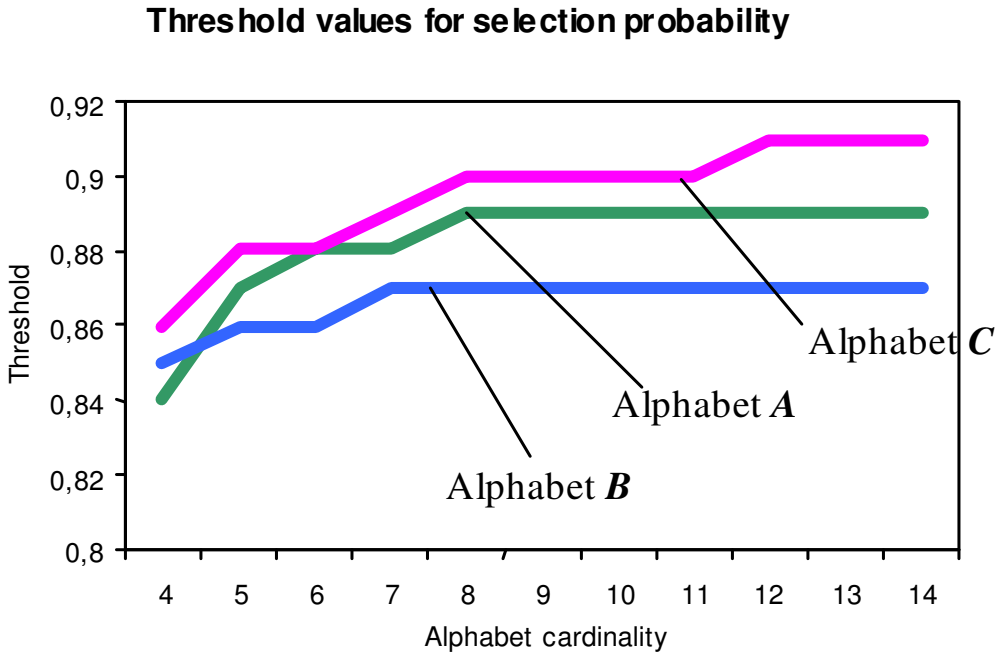

Figure 4-4: Threshold values for selection probability p_t such, that for any 0.5<p<p_t the delete leaf lays at the first level of the optimal tree

Thus, the optimization problem can be reduced to the building of the tree with the delete leaf on the top of it. Such a definite configuration of the tree gives us the possibility to consider trees without this delete leaf and to reduce the optimization problem to the optimization problem discussed below.

4.3.2. The auxiliary optimization problem

Let us consider now the following optimization problem. Given are an alphabet $A = \{a_i\}$, where i=1,.., n; and two nonnegative sequences $\{f_i = f(a_i)\}$ and $\{g_i = g(a_i)\}$. The auxiliary optimization problem is to build a full binary tree T with n leaves, in each leaf one letter from the alphabet A is placed, so that

$$B(T) = \sum_{i=1}^{n} (f_i d_T(a_i) + g_i / p^{d_T(a_i)}) \to \min, \qquad (4.4)$$

where $d_T(a_i)$ is the depth of the leaf with the letter a_i in the tree T and $p \in (0; 1)$. We will present an algorithm, which solves this problem in case when $\{f_i\}$ and $\{g_i\}$ possess a special property – they keep the same "merging order" by a merging operation relatively p.

The merging operation could be described as follows. Let both nonnegative sequences $\{ f_i \}$ and $\{ g_i \}$be non-increasing ($f_1 \geq f_2 \geq \ldots \geq f_n$ and $g_1 \geq g_2 \geq \ldots \geq g_n$). Then we can build two new sequences $\{f_i'\}$ and $\{g_i'\}$ where i=1,.., n-1, so that

$$
\begin{aligned}
& i \in [1, n-2] \quad \Rightarrow f_i' = f_i,\ g_i' = g_i; \\
& f_{n-1}' = f_{n-1} + f_n; \\
& g_{n-1}' = (g_{n-1} + g_n)/p.
\end{aligned}
$$

These formulas define the merging operation relatively $p \in (0; 1)$, which transforms two sequences of the length n into two new sequences consisting of n-1 elements. The next definition bases on this merging operation and will be used in this topic further on.

Definition

Two nonnegative sequences $\{f_i\}$ and $\{g_i\}$ ($i \in [1, n]$) keep the same merging order if $\exists\ i, j \in [1, n]$ so that $\forall\ k \neq i, j$ and $k \in [1, n]$: $f_i \leq f_k$, $g_i \leq g_k$, $f_j \leq f_k$, $g_j \leq g_k$.

If these sequences keep the same merging order after the merging operation, this operation can be applied to them once more, since in this case we can define two minimal elements in each sequence and these elements have the same number in both sequences.

4.3.3. The merging algorithm

The algorithm presented below is the some "variation on theme" of the famous Huffman's algorithm, but can be effectively applied only while corresponding sequences keep the merging order defined above. If the same merging order is kept after each merging operation, the algorithm also allows building the tree T corresponding to the optimal $B(T)$ in a bottom-up manner. Like the Huffman's algorithm, it begins with a set of n leaves and performs a sequence of merging operations to create the final tree. A priority queue can be used in order to identify the two objects (letters or sets of letters) with minimal values of f_i

and g_i to merge together. The result of the merger of two objects is a new object, which values are $f_{new} = f_i + f_j$ and $g_{new} = (g_i + g_j)/p$, where *i* and *j* correspond to the merged objects. Thus, the merging algorithm can be described as follows.

Algorithm 4-4 – The merging algorithm

1) Let *L* be a list of the values (f_i , g_i) of the letters corresponding to the leaves of the tree.
2) If the same merging order is kept, take the two smallest values in *L* (they always exist, when the same merging order is kept), make the corresponding nodes siblings and generate the intermediate node as their parent.
3) Replace the two values in *L* that are mentioned above with their sums $f_{new} = f_i + f_j$ and $g_{new} = (g_i + g_j)/p$, associated with the new intermediate node. If the new *L* contains only one element, then stop, otherwise return to step 2.

The proof of correctness for the algorithm is almost similar to that for the Huffman's algorithm and bases also on the similar ideas (the greedy-choice and the optimal substructure properties), which will be demonstrated in the following lemmas.

Lemma 4-2

Let in the alphabet *A* the sequences $\{f_i = f(a_i)\}$ and $\{g_i = g(a_i)\}$keep the same merging order and letters *x* and *y* be two letters having the lowest values ($f(x)$,$g(x)$) and ($f(y)$, $g(y)$). Then there exists an optimal tree *T,* in which the depths of *x* and *y* are the same, and *x* and *y* appear in the *T* as sibling leaves.

Proof. We assume that $b \in A$ and $c \in A$ are two letters that are sibling leaves of maximum depth in *T*. Since ($f(x)$, $g(x)$) and ($f(y)$, $g(y)$) are two pairs of the lowest values and ($f(b)$, $g(b)$) and ($f(c)$, $g(c)$) are two pairs of arbitrary values, then $f(x) \le f(b)$, $g(x) \le g(b)$, $f(y) \le f(c)$ and $g(y) \le g(c)$. We exchange the positions of *b* and *x* in *T* and create a tree T_1. Then we exchange the positions of *c* and *y* in T_1 and create a tree T_2. The difference in values of criteria between *T* and T_1 is:

$$B(T)-B(T_1)=\sum_{i=1}^{n}(f_i d_T(A_i)+g_i/p^{d_T(A_i)})-\sum_{i=1}^{n}(f_i d_{T_1}(A_i)+g_i/p^{d_{T_1}(A_i)})=$$
$$f(x)d_T(x)+g(x)/p^{d_T(x)}+f(b)d_T(b)+g(b)/p^{d_T(b)}-(f(x)d_{T_1}(x)+g(x)/p^{d_{T_1}(x)})-(f(b)d_{T_1}(b)+g(b)/p^{d_{T_1}(b)})=$$
$$f(x)d_T(x)+g(x)/p^{d_T(x)}+f(b)d_T(b)+g(b)/p^{d_T(b)}-(f(x)d_T(b)+g(x)/p^{d_T(b)})-(f(b)d_T(x)+g(b)/p^{d_T(x)})=$$
$$(f(x)-f(b))(d_T(x)-d_T(b))+(g(x)-g(b))(p^{-d_T(x)}-p^{-d_T(b)})\geq 0,$$

because all the terms are nonnegative: $f(x) \le f(b)$, $g(x) \le g(b)$ and $d_T(x) \le d_T(b)$, since we assumed that b is in the leaf of maximum depth in T. Similarly, because exchanging y and c does not increase the value of the criterion, $B(T_1)$-$B(T_2)$ is nonnegative. Therefore, $B(T_2) \le B(T)$, which implies $B(T_2) = B(T)$. Thus, T_2 is an optimal tree, in which x and y appear as sibling leaves of maximum depth, from which the lemma follows. This lemma implies that the process of building up an optimal tree by mergers can, without loss of generality, begin with the greedy choice of merging together two letters of the lowest values f and g.

Lemma 4-3

Let T be a full binary tree representing a solution of the optimization problem for B over an alphabet A with sequences $\{ f_i \}$ and $\{ g_i \}$ and $p \in (0; 1)$. Let us consider any two letters x and y that appear as sibling leaves in T, and let z be their parent. Then considering z as a letter with new values $f_z = f_x + f_y$ and $g_z = (g_x + g_y)/p$, the tree $T_1 = T - \{x, y\}$ represents an optimal solution of the optimization problem for B over an alphabet $A_1 = A - \{x, y\} \cup z$.

Proof. We show first that the criterion B(T) of the tree T can be expressed in terms of the criterion B(T1). For each letter $c \in A$ - {x, y}, we have $d_T(c) = d_{T_1}(c)$, and hence $f(c)d_T(c) = f(c)d_{T_1}(c)$ and $g(c)p^{-d_T(c)} = g(c)p^{-d_{T_1}(c)}$. Since $d_T(x) = d_{T_1}(z) + 1$, we have

$$f(x)d_T(x)+g(x)p^{-d_T(x)}+f(y)d_T(y)+g(y)p^{-d_T(y)}=(f(x)+f(y))(d_{T_1}(z)+1)+$$
$$(g(x)+g(y))/p^{d_{T_1}(z)+1}=f(z)d_{T_1}(z)+g(z)p^{-d_{T_1}(z)}+f(x)+f(y),$$

from which we can conclude that $B(T)=B(T_1)+f(x)+f(y)$. If T_1 represents a non-optimal tree for the alphabet A_1, then there exists a tree T_2, which leaves are letters from A_1, such that $B(T_2) < B(T_1)$. Since z is treated as a letter in A_1, it appears as a leaf in T_2. If we add x and y as children of z in T_2, then we obtain an optimal solution for A with the criterion value

$B(T_2)+f(x)+f(y) < B(T)$, which contradicts the optimality of T. Thus, T_1 must be optimal for the alphabet A_1. This lemma demonstrates that the optimization problem (4.4) has the optimal substructure property.

The presented merging algorithm can be illustrated with help of the following example.

Example 4-3

Let an alphabet A consist of five letter A={a, b, c, d, e} with frequencies of occurrence 0.35, 0.25, 0.20, 0.15, 0.05 respectively; the selection probability value is p=0.8. Table 4-5 illustrates the merging algorithm and shows both ordered sequences $\{f_i\}$ and $\{g_i\}$, and corresponding subsets of letters after each algorithm step (after each merging operation). The step 0 is the situation at the beginning of calculations.

Step 0	**a**	**b**	**c**	**d**	**e**
$\{f_i\}$	0.35	0.25	0.20	0.15	0.05
$\{g_i\}$	0.35	0.25	0.20	0.15	0.05
Step 1	**a**	**b**	**d, e**	**c**	
$\{f_i\}$	0.35	0.25	0.20	0.20	
$\{g_i\}$	0.35	0.25	0.25	0.20	
Step 2	**d, e, c**	**a**	**b**		
$\{f_i\}$	0.40	0.35	0.25		
$\{g_i\}$	0.562	0.35	0.25		
Step 3	**a, b**	**d, e, c**			
$\{f_i\}$	0.60	0.40			
$\{g_i\}$	0.75	0.562			

Table 4-5: All the steps of the merging algorithm for the example 4-3

As the result of these calculations we obtain the optimal tree T (see figure 4-6), which corresponds to the minimal criterion value $B(T)$=3.841. In this example, as could be seen from the table, the both sequences $\{f_i\}$ and $\{g_i\}$ keep the same merging order till the end of

the algorithm procedure. An attachment of the δ-leaf to the tree T (at the first level) leads to an optimal full binary SI (the tree T_1) over the alphabet A with the criterion-value $M(T_1)$=6,352.

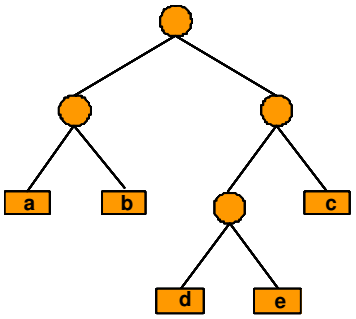

Figure 4-6: The optimal SI for the example 4-3

4.3.4. Comparison of the Full-Search algorithm and the merging algorithm (incl. example)

The auxiliary optimization problem discussed above and the corresponding merging algorithm can be used in some cases for our main optimization problem (4.1) and help to decrease its dimension. The criterion of optimization (4.1) – the average expectation of the number of trials, which are necessary to write one letter – in case of the homogeneous SI and when the delete leaf is placed at the first level of the tree, can be transformed as:

$$M(T)=\sum_{i=1}^{n} f_i\left[S_i+\frac{(1-p_i)p_\delta(S_\delta+R)}{p_i(2p_\delta-1)}\right]=\sum_{i=1}^{n}(f_i d_i+Kf_i/p^{d_i})-K\rightarrow\min, \qquad (4.5)$$

where d_i is the depth of the leaf with the letter a_i in the tree T and $K = p(1+R)/(2p-1)$. Thus, we can see that, considering $g_i = Kf_i$, the problem (4.1) of optimization of the criterion M possesses all the properties of the discussed auxiliary problem. The algorithm presented above can perform at least the first merging operation (because on account of $g_i = Kf_i$ the first merging operation is always possible) and works while the corresponding sequences keep the same merging-order relatively p. It allows us to use the presented auxiliary algorithm for the search for optimal sub-trees that, in turn, can be used in the Full-Search algorithm 4-2 as simple letters. A success of such acceleration depends on the circumstance, how long the sequences $\{f_i\}$ and $\{Kf_i\}$ would keep the same merging order. In

any case, this algorithm is useful, because even in the worst situation it accelerates the search by transforming an alphabet with *n* letters into an alphabet with *n*-1 letters.

For an alphabet, which consists of *n* letters, the merging algorithm processes firstly with sequences $\{f_i\}$ and $\{g_i\}$, where at the beginning $g_i = Kf_i$, because the delete leaf is assumed to lay at the first level of the optimal tree. We can also, equally, assume that $g_i \equiv f_i$, because the value of *K* does not matter for the merging algorithm itself and can be taken into account later on. When all possible sub-trees $T_1, T_2, \ldots, T_m$ $(m < n)$ are created and the merger order does not keep any longer, the Full-Search algorithm starts generating binary trees with *n*-*l*+*m* leaves, where *l* (*l* > *m*) is the number of letters in sub-trees $\{T_i\}$. For each tree generated in this way the value of the optimization criterion must be calculated and then compared with all other criterion values. While calculating the criterion values, one needs to place all *m* founded sub-trees $\{T_i\}$ in *n*-*l*+*m* leaves. For each such distribution of sub-trees the arrangement of *n*-*l* non-merged letters means essentially their distribution by frequencies of occurrence in the alphabet. So we have for each full binary tree the number of $(n-l+m)!((n-l)!)^{-1}$ criterion computations and comparisons instead of *n,* but instead of $\frac{1}{n+1}C_{2n}^{n} = \frac{(2n)!}{n!n!(n+1)}$ generated trees (the *n*-th Catalan number) we can generate only $\frac{1}{n-l+m}C_{2(n-l+m-1)}^{n-l+m-1} = \frac{(2(n-l+m-1))!}{((n-l+m-1)!)^2(n-l+m)}$ trees (the (*n*-*l*+*m*-1)-th Catalan number). It means that in this case we have achieved an acceleration in

$$\frac{(2n)!n((n-l+m-1)!)^2(n-l+m)(n-l)!}{(n!)^2(n+1)(2(n-l+m-1)!)^2(n-l+m)!} = \frac{(2n)!n(n-l+m-1)!\ (n-l)!}{(n!)^2(n+1)(2(n-l+m-1)!)^2}$$

times, calculated as a ratio of the number of criterion calculations and comparisons by the search for an optimal *n*-*l*+*m*-leaves tree with a following search for an optimal sub-trees distribution and the number of criterion calculations and comparisons by the full search for an optimal *n*-leaves tree. For instance, for *n*=15, *m*=2 and *l*=7 (see the example 4-4 below), we have the acceleration in 9694845*16/(4862*90) ≈ 354.5 times, where 9694845 – the number of generated 16-leave trees, 16 – the number of different placements for the delete option among 16 leaves, 4862 - the number of generated 10-leave trees, 90 – the number of different distributions for 2 sub-trees among 10 leaves.

Example 4-4

Let *A* be an alphabet, which consists of 15 letters with corresponding frequencies of occurrence (see table 4-7). Let us build an optimal spelling interface over this alphabet in a homogenous case, when both selection probabilities have the same value p=0.7.

Letter	Frequency	Letter	Frequency	Letter	Frequency
a	0.2	f	0.08	k	0.02
b	0.15	g	0.07	l	0.01
c	0.12	h	0.05	m	0.01
d	0.11	i	0.04	n	0.005
e	0.10	j	0.03	o	0.005

Table 4-7: Alphabet A (letters and their frequencies of occurrence) for the example 4-4

Since the selection probability value of 0.7 is relatively low, we can be sure that the delete leaf is situated at the first level of an optimal tree. Now we will demonstrate the above developed technique. Firstly, we will use the merging algorithm and try to find sub-trees of the optimal tree. Table 4-8 illustrates this algorithm from the beginning (Step 0) until the end (Step 5), when both sequences $\{f_i\}$ and $\{g_i\}$ lose the same merging order. We show in this table not all letters, but only the letters and sets of letters, which have 4 minimal values of f_i and g_i.

Step 0	**l**	**m**	**n**	**o**
$\{f_i\}$	0.01	0.01	0.005	0.005
$\{g_i\}$	0.01	0.01	0.005	0.005
Step 1	**k**	**n, o**	**l**	**m**
$\{f_i\}$	0.02	0.01	0.01	0.01
$\{g_i\}$	0.02	0.01429	0.01	0.01
Step 2	**j**	**l, m**	**k**	**n, o**
$\{f_i\}$	0.03	0.02	0.02	0.01
$\{g_i\}$	0.03	0.02857	0.02	0.01429
Step 3	**l**	**n, o, k**	**j**	**l, m**
$\{f_i\}$	0.04	0.03	0.03	0.02

$\{g_i\}$	0.04	0.04898	0.03	0.02857
Step 4	**l, m, j**	**h**	**i**	**n, o, k**
$\{f_i\}$	0.05	0.05	0.04	0.03
$\{g_i\}$	0.0837	0.05	0.04	0.04898
Step 5	**g**	**n, o, k, i**	**l, m, j**	**h**
$\{f_i\}$	0.07	0.07	0.05	0.05
$\{g_i\}$	0.07	0.12711	0.083673	0.05

Table 4-8: Steps of the merging algorithm for the alphabet A in the example 4-4

As one can see, having applied this merging algorithm we obtained two following sub-trees: T_1 with letters {l, m, j} and T_2 with letters {n, o, k, i} – see the figure 4-9 below:

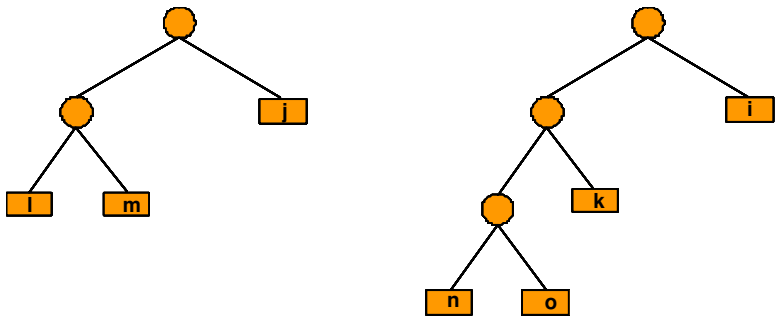

Figure 4-9: Sub-trees T_1 and T_2 for the example 4-4

These sub-trees are used as simple letters in the Full-Search algorithm in order to find an optimal tree with 15-7+2 = 10 leaves. The full binary tree for the optimal SI in this case is presented in the figure 4-10 (the criterion value is 23.327).

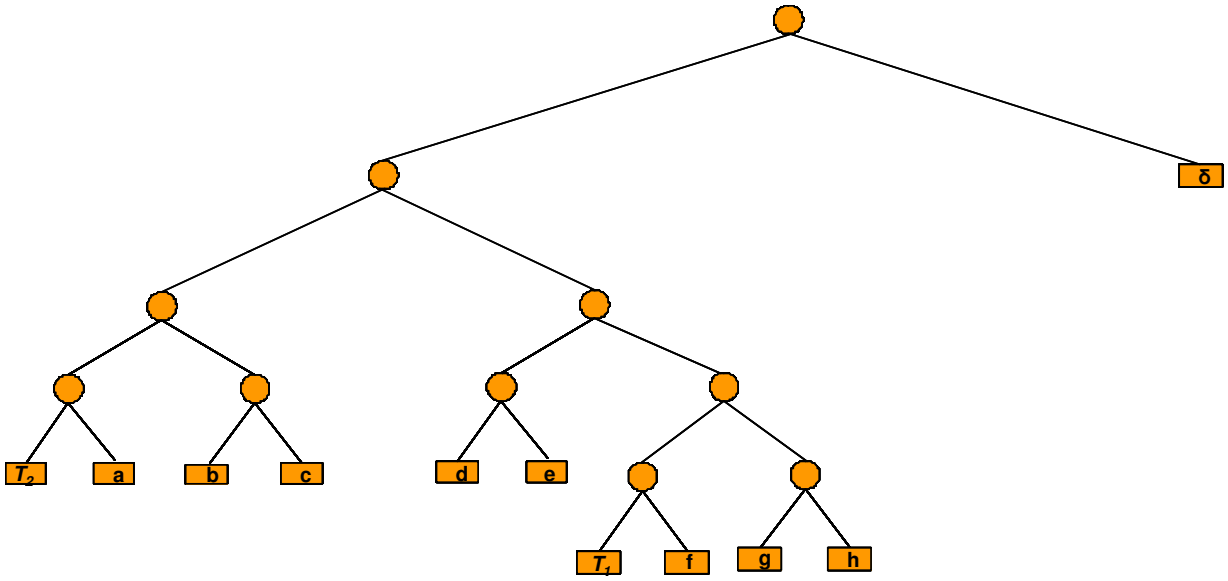

Figure 4-10: The final optimal full binary tree for the example 4-4

Being optimized as a SI-binary tree with all 16 leaves (15 letters and the delete-option) with the Full-Search algorithm, the optimal tree possesses the same criterion value and also belongs to optimal solutions of the optimization problem. Therefore, using two different algorithms we have obtained the same solution, but the first algorithm (merging and then – the Full-Search among founded sub-trees and left unmerging letters) in this case is essentially faster than the second one, since it needs essentially less computations (in about 354 times).

4.4. Computer implementation

In the sections above we have seen a number of developed algorithms and optimization methods that allow creation of optimal trees for spelling interfaces. In order to ensure the practical applicability of these algorithms and methods it was required to create a corresponding computer tool.

4.4.1. Requirements for integrated computer tool

As shown in chapters 2 and 3, the TTD represents the only one Brain-Computer Interface, which has been being practically used. The current version of the TTD bases, due to

historical reasons, upon a different optimization criterion - “the average probability to write one letter without errors” – that was described in the paper [37]. Hence, it would be very beneficial to consider this criterion as an additional criterion for the computer implementation, thus, developing an *integrated* computer tool. This tool should, therefore, cover all known methods and algorithms intended for creation of optimal binary SI’s, both newly developed and previously described.

In addition to that, an integrated computer tool should cover a comprehensive range of requirements and tasks linked to actual creation of ready-to-use Brain-Computer Interfaces, such as input of alphabet’s characteristics, consideration of predefined sub-trees etc. The complete list of these practical requirements can be found below.

1) *Optimization criteria*
    a. The average expectation of the number of trials, which are necessary to write one letter;
    b. The average probability to write one letter without errors.

2) *Optimization methods*[*]
    a. The Full-Search algorithm;
    b. The Huffman’s algorithm;
    c. The inhomogeneous algorithm;
    d. The merging algorithm;
    e. The Perelmouter-Birbaumer’s algorithm (also refer to [37]).

3) *Input data*
    a. Definition of a new alphabet (a list of letters or menu items and their frequencies of occurrence);
    b. Definition of predefined sub-trees;

[*] Clearly enough, not all optimization methods can be applied with each criterion in order to create an optimal binary SI for this particular criterion. Rather, specific methods fit to specific criteria. In case of the criterion “The average expectation of the number of trials, which are necessary to write one letter” four methods mentioned above would work – the full-search algorithm and the Huffman’s algorithm (for $p=q=1$) as universal methods, the inhomogeneous algorithm and the merging algorithm as methods developed for this particular criterion. In case of the criterion “The average probability to write one letter without errors” the full-search algorithm would work as a universal method, the inhomogeneous algorithm and the Perelmouter-Birbaumer’s algorithm – as methods developed for this particular criterion. All restrictions and limitations of algorithms described above should be taken into account accordingly.

c. Definition of predefined P-sequences;

d. Definition of individual selection probabilities *p* and *q.*

4) Additional requirements

a. Merging of P-sequences;

b. Merging of sub-trees.

4.4.2. General description of the integrated computer tool TBSIO; input and output data

Based upon the above requirements, the integrated computer tool called TBSIO (Tool for Binary Interface Optimization) has been created. It can be used for creation, evaluation, comparison and the final selection of the optimal Spelling Interface, which best matches both the individual requirements (alphabet) and the possibilities (selection probability values) of a particular patient. In the following figure 4-11 the main screen of the application is shown.

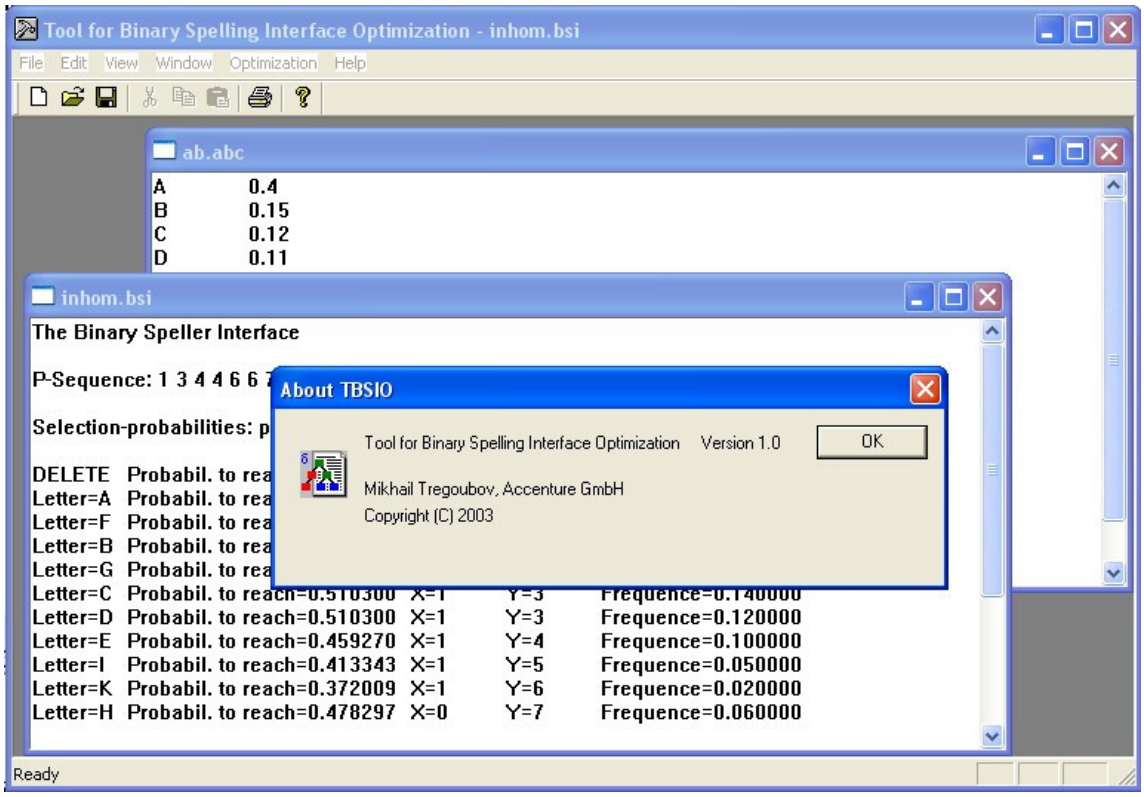

Figure 4-11: The TBSIO – the main screen

The tool is essentially a Windows-application that was built in the programming language C++ in the Microsoft Visual C++ 6.0 environment and shares the library Microsoft Foundation Classes. The application itself represents a so-called Windows multiple documents application (compare figure 4-11).

For input and output purposes text files are used. The application works with the following types of text files:

- *.abc (alphabet) – input files, which contain letters (or menu items) and the frequencies of their appearance;

- *.psq (P-sequence) – input files containing P-sequence elements;

- *.bst (binary sub-tree) – input files, which contain letters (or menu items) and the corresponding P-sequence and, therefore, describe a binary sub-tree. Such binary sub-trees contain a letter or a menu item in each of it's leaves and can be built taking into consideration some other aspects than the optimal criterion value (also see below);

- *.bsi (Binary Spelling Interface) – output files, which describe an obtained Spelling Interface and its main properties, such as the value of the corresponding criterion.

The necessity to have input files of the type *.bst is based upon the practical requirement to take into account sub-trees, which are built according to factors that are important or sometimes even crucial for a patient but don't comply with the optimal criterion value. Thus, in many cases, some menu items (or letters of the according alphabet) should be grouped in some particular way, which is useful for a patient. E.g., from the point of view of a patient, all menu items related to nutrition need to be located in the same sub-menu, where no other menu items should be located. Another example is a case when the subjective importance of some menu item for a patient is perceived as high and will not correspond to the effective frequency of choosing this particular menu item. For instance, a patient who is oversensitive to his environment would prefer to place the menu items "Open the window" and "Close the window" to the top of the spelling tree, even if their actual selection occurs rather seldom.

The application can deal with input files in a proper format regardless the way they were created. So, generally speaking, it is possible to create an input file of any type using a text editor and providing the created file with the required extension afterwards. However, the most convenient and safest way to create input files is to use dialogs for construction and control of input files, which can be found in the drop-down menu “File” of the TBSIO application (see figure 4-12).

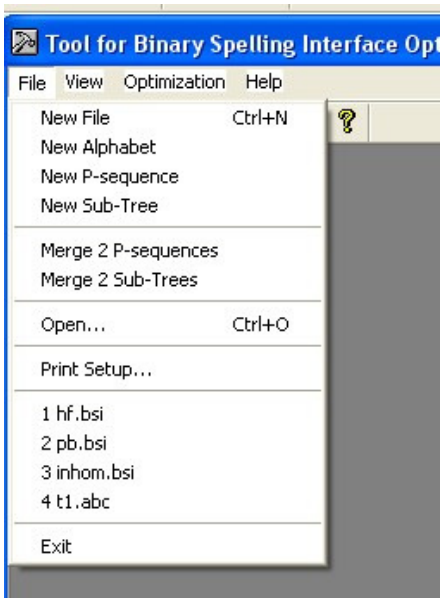

Figure 4-12: The TBSIO – the drop-down menu “File”

Using menu items „New Alphabet“, „New P-sequence“ and „New Sub-Tree“, one can open corresponding modal dialogs (for instance, the menu item „New Alphabet“, when being selected, opens the dialog shown in the figure 4-13; dialogs for creation of new P-sequences and new sub-trees look similar) and create desired input files in the dialog-based manner. During the creation of an input file and its consecutive recording the tool automatically conducts all necessary checks:

- each letter in the alphabet or in the external sub-tree must be unique;
- the sum of the frequencies of appearances for all letters must be equal to 1;
- elements of P-sequences must fulfill the requirements described in the theorem 4-1.

After an input file has been created it can be opened in the main window of the application and can be modified manually, if required.

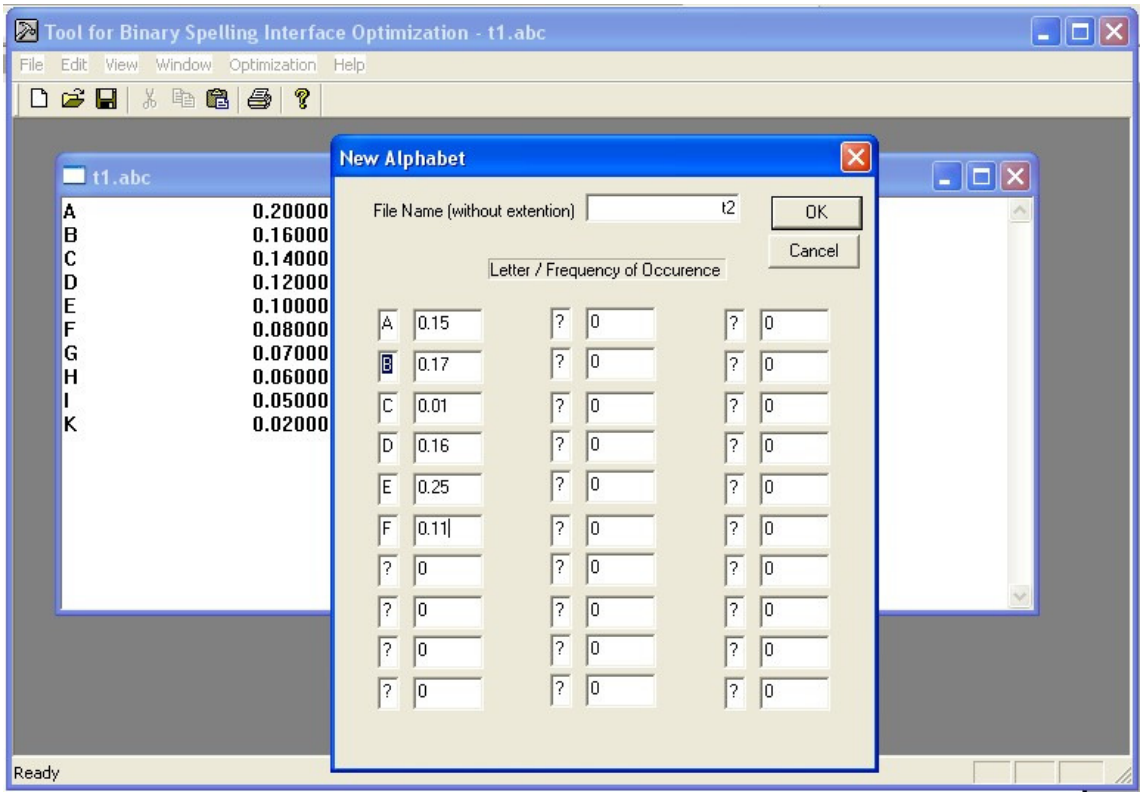

Figure 4-13: The TBSIO – the modal dialog "New Alphabet"

Additionally to the creation of a new alphabet and building of an optimal BCI upon it, a user can easily build desired predefined structures – P-sequences and sub-trees – through consecutive merging of simpler structures or even basic elements (compare lemma 4-1). This can be done using appropriate modal dialogs by selecting the menu items „Merge 2 P-sequences" or „Merge 2 Sub-Trees" and makes the application very suitable and convenient for creation of P-sequences and sub-trees. In the corresponding dialogs the user has to define two input files containing either valid P-sequences or valid binary sub-trees and the desired output file of the same type. Prior to execution of the merging process the application also checks the uniqueness of the letters of the alphabet contained in the substructures that need to be merged.

The menu item "Open" allows the user to browse through file folders and to find, select and open desired input files. The menu item "Print Setup" deals with setting up a printer.

The application shows four last used input and output files, thus, allowing the user to quickly access them. The menu item “Exit” closes the application.

4.4.3. Optimization criteria and methods

Input files described in the above section are used for the search for optimal Binary Spelling Interfaces by means of the optimization algorithms. Desired algorithms can be selected in the drop-down menu “Optimization”. The application supports the optimization based upon the both criteria described above: “the average expectation of the number of trials, which are necessary to write one letter” (see figure 4-14) and “the average probability to write one letter without errors” (see figure 4-15).

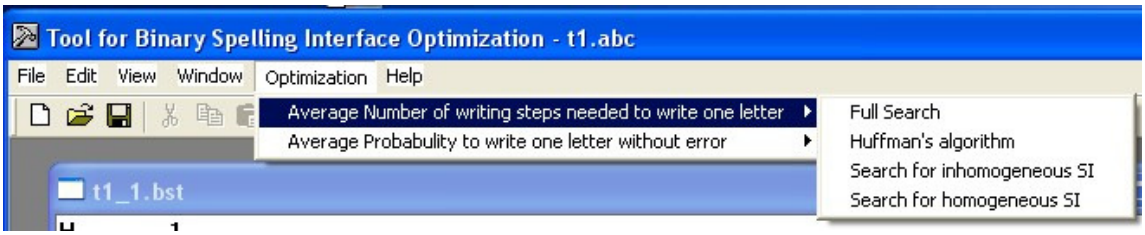

Figure 4-14: The TBSIO – the menu for selection of optimization algorithms based on the criterion “the average expectation of the number of trials, which are necessary to write one letter”

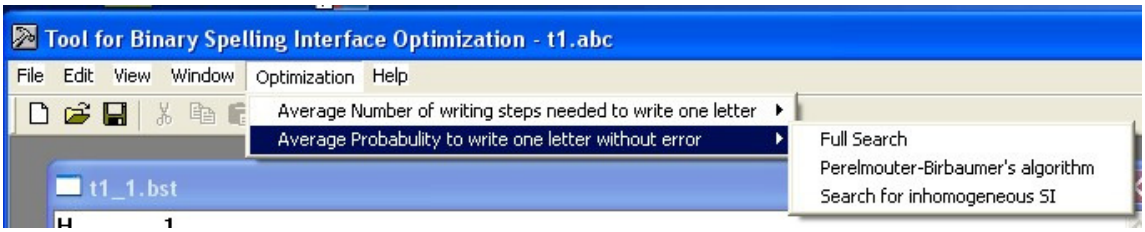

Figure 4-15: The TBSIO – the menu for selection of optimization algorithms based on the criterion “the average probability to write one letter without errors”

In order to carry out the optimization process it is required to define input and output files as well as other required data. This is done with help of correspondent modal dialogs. The figure 4-16 provides a good example for such modal dialogs and shows the dialog window

in case of the application of the Full-Search algorithm. Other modal dialogs are built in the similar way.

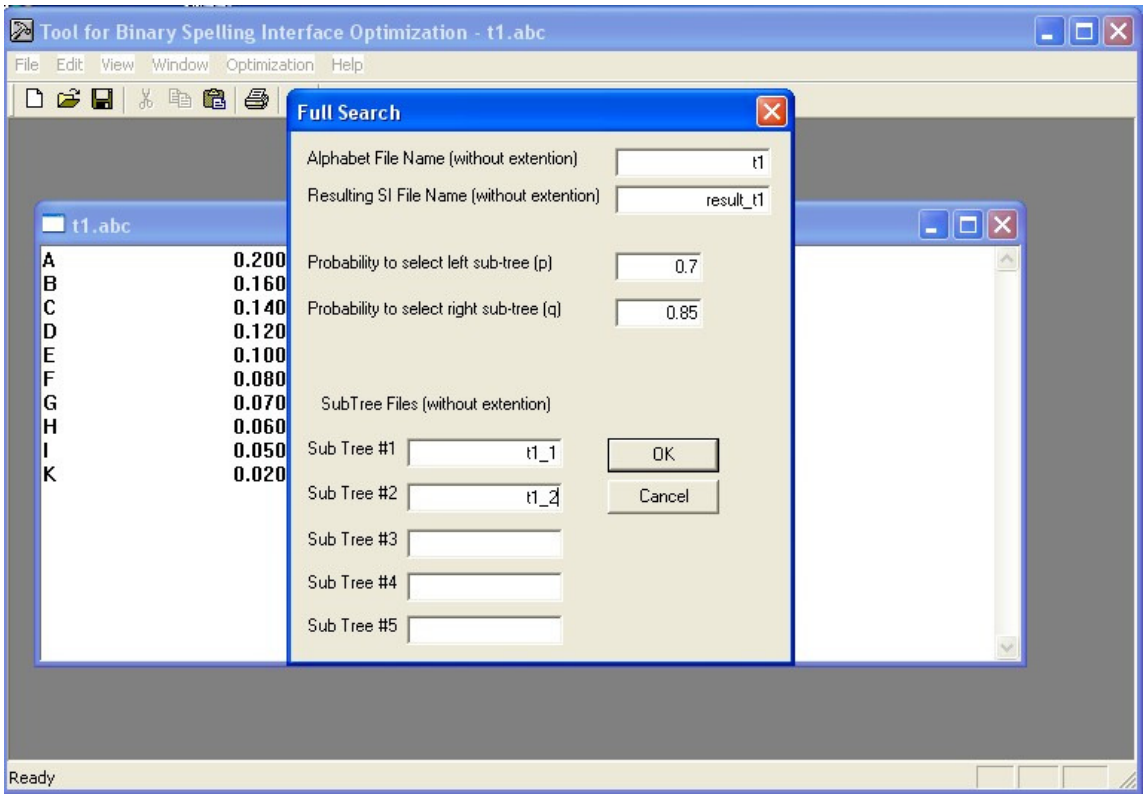

Figure 4-16: The TBSIO – the modal dialog "Full-Search" for definition and control of the input data in case of the application of the Full-Search algorithm

The text field "Alphabet File Name" defines the name of the input file that contains the desired alphabet. The text filed "Resulting SI File Name" defines the name of the output file the created optimal SI will be written into. Both text fields dealing with probabilities allow the user to enter values of individual selection and rejection probabilities of the patient the SI is meant for. Both values should stay in the interval *(0.5; 1)*. However, as explained above, due to practical applicability both values should amount to at least *0.65 – 0.7*.

In case the user decides to include predefined sub-trees into the SI he has to enter the names of input files into corresponding text fields. The user can include up to five sub-trees.

The button “OK” starts the calculation of the optimal SI. The button “”Cancel” closes the modal dialog.

In all algorithms based on the exhaustive search (the Full-Search algorithms and the inhomogeneous algorithm, represented through the menu item “Search for inhomogeneous SI”) it is possible to use simultaneously both single letters and completed sub-trees built in advance. The check-up prior to the optimization process ensures that the input data are consistent (e.g. a letter from one sub-tree must not be contained in other sub-trees and must belong to the selected alphabet etc.).

Contrary to this, in the algorithms based on the merging procedure (the Huffman’s algorithm, the merging algorithm, represented through the menu item “Search for homogeneous SI”, the Perelmouter-Birbaumer’s algorithm) only alphabets can be used as input data. Thereby in case of the merging algorithm, for instance, the application builds possible sub-trees using the merging algorithm 4-4 and in the next step informs a user of the dimension of the upcoming optimization problem (see figure 4-17). The user can either proceed with the optimization process through starting the Full-Search algorithm or has an option to change input data.

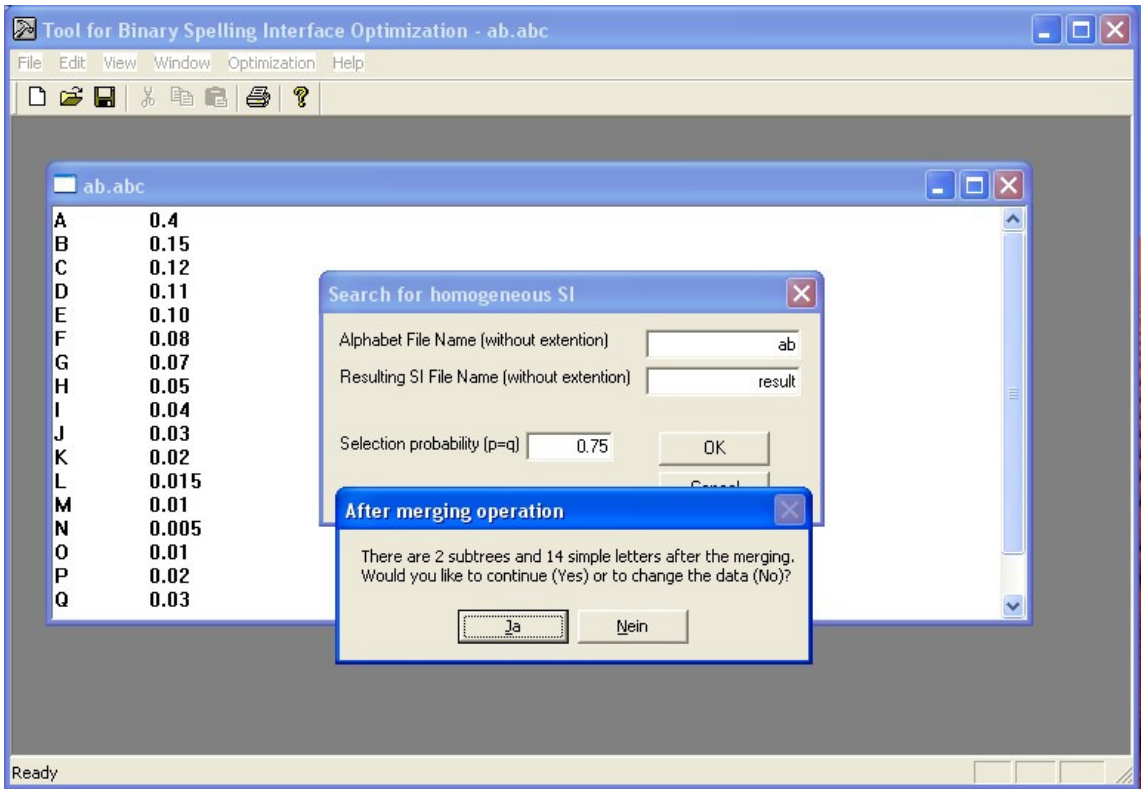

Figure 4-17: The TBSIO – the result of the merging algorithm as it is shown to a user (in this particular case an alphabet containing 19 letters consists after the merging of 2 sub-trees and 14 simple letters)

During the optimization process the current value of the optimization criterion and the number of conducted calculations are being computed and shown to a user on the screen (see figure 4-18). Optimization can be aborted by a user selecting the „Cancel“ button or continues until all binary graphs built by the correspondent algorithm have been analyzed and evaluated.

In both cases – upon abort or upon completion – the application creates an output file of the type *.bsi and opens it in the main application window. The outcome of the computation is indicated at the end of the file, where either the text “The optimization is completed” appears (in case the computation has been completed by the application) or the text “The optimization is not competed” (in case the computation has been aborted by a user; see figure 4-19).

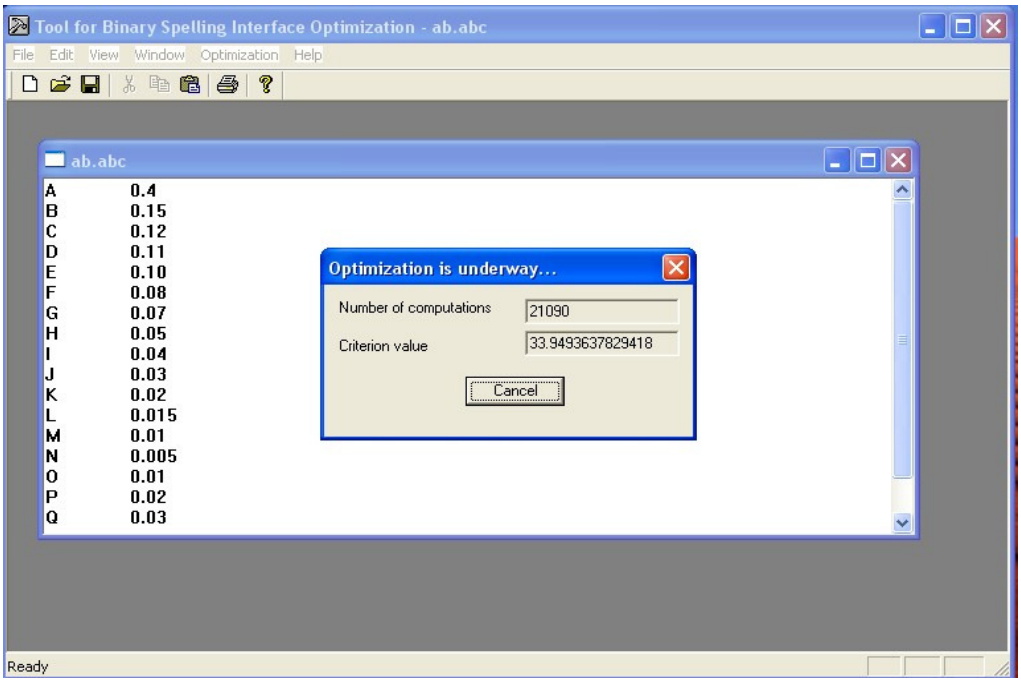

Figure 4-18: The TBSIO – the progress indicator during the optimization process

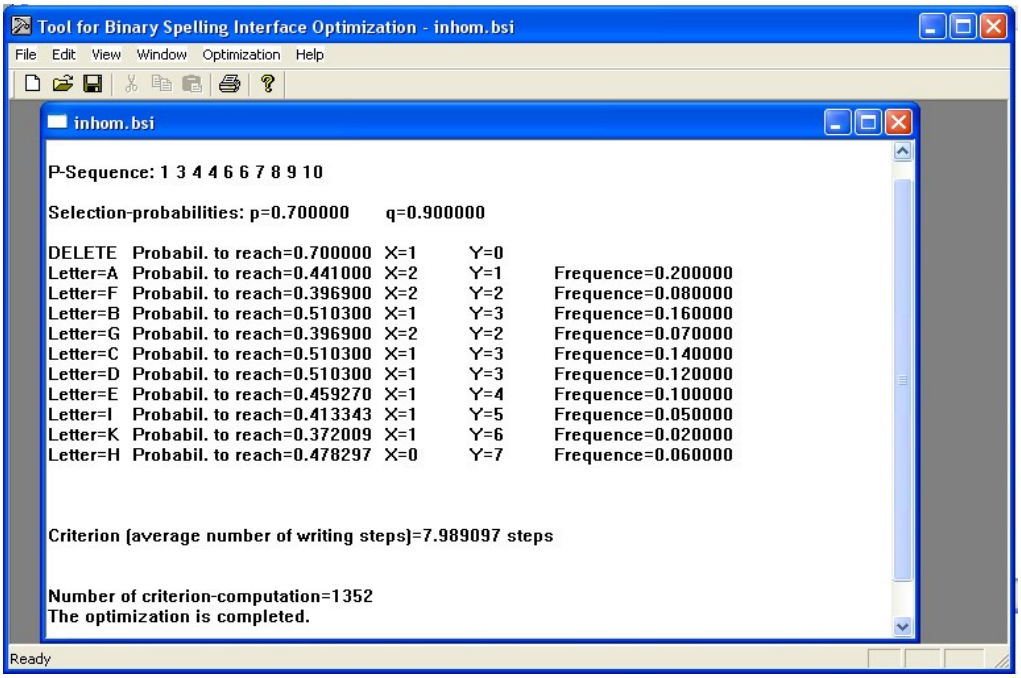

Figure 4-19: The TBSIO – the example of an output file in the main window (a Spelling Interface derived as a result of the optimization process)

While selecting the criterion “The average expectation of the number of trials, which are necessary to write one letter” (see formula 4.1) the following alternatives are possible. In case a patient is capable of the absolutely reliable selection (selection probabilities p=q=1) the user should use the Huffman’s algorithm that builds the Spelling Interface without a delete-option, which is not required. In case values of selection probabilities differ from each other a lot, one should apply the algorithm 4-3 – the inhomogeneous algorithm. In case values of selection probabilities are close to each other, one should use the method presented in the section 4.3 “Synthesis of optimal homogenous SI’s, the auxiliary optimization problem and the merging algorithm” – the merging algorithm 4-4 with the consecutive search among the created sub-trees and single letters left after the merging algorithm has been applied. In case of a low number of letters in an alphabet (low alphabet cardinality) the Full-Search algorithm 4-2 delivers the most consistent results.

For the criterion “The average probability to write one letter without any errors” that was introduced by Perelmouter and Birbaumer [37] the following alternatives are possible. In case the greedy-choice property (values of selection probabilities are close to each other or even equal) is fulfilled one can apply the Perelmouter-Birbaumer’s algorithm. Since the application automatically checks whether the greedy-choice property is fulfilled or not (as shown in the Figure 4-20), it makes sense to always start the optimization using the Perelmouter-Birbaumer’s algorithm. In case values of selection probabilities differ from each other a lot, one should apply the inhomogeneous algorithm 4-3. And, finally, similar to the first criterion the Full-Search algorithm 4-2 delivers the most consistent results.

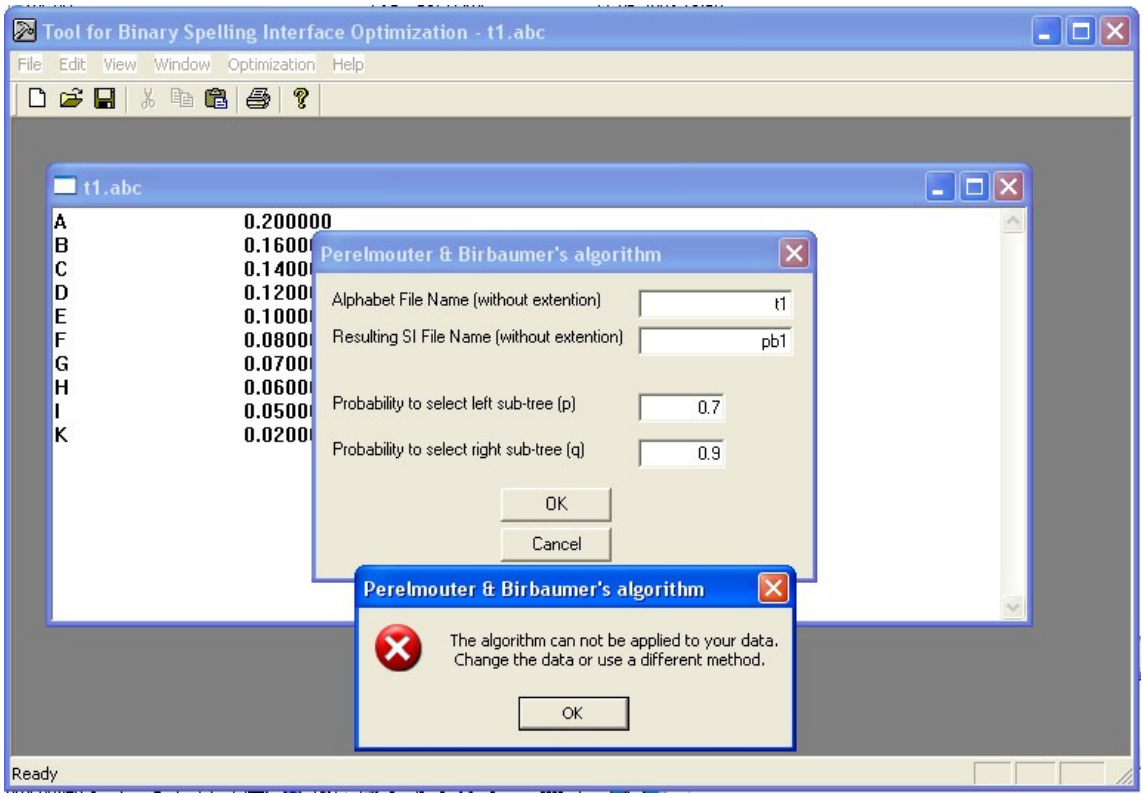

Figure 4-20: The TBSIO – an error message in case the greedy-choice property is not fulfilled for the input data (Perelmouter-Birbaumer's algorithm)

Thus, the application TBSIO completely covers and solves – at the current state of the art – the problem of creation of optimal Binary Spelling Interfaces based upon both criteria given above and represents an integrated computer tool covering all presented requirements.
It can be successfully used for building of optimal binary Spelling Interfaces for all kinds of binary communication.

5. Summary and outlook

5.1. Summary

The general problem of finding variable-length codes for non-zero communication has been described and formally defined in this thesis. In order to solve this problem, with respect to especially (but not only) Spelling Interfaces, an optimization criterion (or a corresponding cost function) needs to be defined, and an optimization method or methods have to be found.

Therefore, in the first step we have defined and presented the new optimization criterion for the SI-optimization. This criterion represents the most natural criterion for SI's from the point of view of usage by handicapped patients. In the second step, we have developed a number of new optimization methods, which help accelerate solving such an optimization problem. The introduced criterion corresponds exactly to the aim of SI building and can be used not only for binary SI, but also for *k*-ary problems, since it allows us not only to optimize but also to compare different given SI with each other and then to evaluate them from the unified point of view. This has been impossible so far. The presented criterion has an important advantage over other evaluation criteria described in [36], which need a simulation in order to obtain one particular criterion value. Contrary to such a comprehensive technique, values of the presented criterion can be directly calculated from the formula (4.1), thus, allowing quick and direct comparison and evaluation.

The developed optimization methods, which strive to minimize the value of the criterion, are built upon a number of invented algorithms. These algorithms are intended to be used, first and foremost, for the search for the binary SI, which is optimal in sense of the criterion (4.1). Besides that, the Full-Search algorithm and the algorithm for decoding of P-sequences can be used for an optimization procedure for any possible optimization criterion. Another developed algorithm covers the case of the optimization of inhomogeneous SI and can essentially accelerate a computation process in this case. The effectiveness of using asymmetrical SI properties depends only on difference in values of selection probabilities and is always achievable, thus, resulting in major search acceleration.

The merging algorithm, in turn, deals with the so-called homogeneous case and, together with the inhomogeneous algorithm and Huffman's algorithm, covers all cases of optimization problem for the criterion (4.1). It also can essentially accelerate a computation process. Whereat the Huffman's algorithm must be used in the case of $p=q=1$ and the inhomogeneous algorithm – in the case of essentially different values of $p<1$ and $q<1$, the merging algorithm is developed especially for the case $p=q<1$ and, together with the Full-Search algorithm applied to founded sub-trees, is usually highly efficient. The effectiveness of solving such homogeneous optimization problems with the presented merging method depends on the properties of the used alphabet – in particular, on the distribution of frequencies of occurrence – and can be easily estimated before the actual optimization process applying the presented merging algorithm. For example, for the English alphabet (26 letters with natural frequencies of occurrence) the merging leads in case of $p=0.9$ to 3 sub-trees (2, 2, and 6 letters) and 16 letters left unmerged, which decreases the dimension of the optimization problem from a 26-leaf tree to a 19-leaf tree. In the case of $0.75 \le p \le 0.85$ the algorithm gives us 1 sub-tree (5 letters) and 21 letters left unmerged, so decreasing the dimension to a 22-leaf tree and so on. As one can see, such an acceleration of a computation process is very valuable for the practical usage.

It is clear, that all these algorithms, being NP-hard, strongly depend on the cardinality of the alphabet (the computation in the case of n~20 needs some hours). But for binary SI's for German and English languages, which consist of 26 letters or menu items, the practical relevance of the proposed algorithms is given in the most cases, especially when the merging is possible. Moreover, the elaborated method makes possible merging of given binary sub-trees to a final tree, thus permitting to take into account predefined orders of letters or menu items. This is especially useful in case of menu-based BCI's, since it allows clustering of particular menu items based on their nature. For instance, all items dealing with nutrition – such as "hungry", "thirsty" etc. – can be grouped under the menu "Food and Drinks", all items dealing with environmental and comfort issues – such as "cold", "warm", "open window" etc. – can be grouped under the menu "Environment" and so on. Such groups of items can be built according to individual needs of a patient and then distributed with help of elaborated methods to create an optimal menu-based BCI.

Furthermore, the combination of developed methods represents the only known universal method, consisting of both criteria and algorithms, that allows the synthesis of individually

fitting Spelling Interfaces, since it considers individual selection and rejection probabilities of a patient. Therefore, it can be ensured that a Spelling Interface created with help of developed methods represents an optimal Spelling Interface for a particular patient, thus, making it possible for this certain patient to communicate at the maximal possible rate.

It should be noted again that the proposed algorithms and methods are useful not only for a binary Brain-Computer interface, but also for any type of menu-oriented communication system, which is controlled through a binary response and where appearance of errors is possible, thus, completely solving also the general problem of a non-zero error communication described in the section 3.2.

Additionally to invented optimization methods, a supporting computer tool called TBSIO has been developed. This computer tool covers the complete range of features required for synthesis of optimal binary Spelling Interfaces (e.g. definition of the alphabet, definition of predefined trees etc.). It allows the synthesis of binary SI's based upon two criteria: "average number of writing steps needed to write one letter" that has been identified in this thesis as the most natural criterion; and "average probability to write one letter without errors" that was described in the paper [37] and that currently builds a basis for the TTD Spelling Interface and, therefore, needs to be considered further from the point of view of practical applicability. In the computer tool all newly invented optimization methods as well as optimization methods described in the paper [37] were considered, whereat all corresponding limitations and checks (such as, a fulfillment of a greedy-choice property) were taken into account. Thus, the developed computer tool represents the only existing universal tool for creation of optimal binary SI's.

5.2. Outlook

Methods and algorithms developed in this research give for the very first time the opportunity to create individually fitting Spelling Interfaces that are optimal for particular patients. This leads to increasing speed and accuracy of communication, resulting, in turn, in increased motivation and quality of life. With respect to further upsurge and further possible improvements of communication with locked-in patients the following aspects can be mentioned.

First, the most optimization algorithms for SI's described in this thesis (either invented as a part of this thesis or developed by other authors before) are NP-hard in their nature. The calculation power of modern CPU's, however, definitely allows the synthesis of optimal SI's for alphabets consisting of twenty or more items even currently, thus, making its possible to create individual SI's for particular patients in English and German languages in many cases. On the other hand, the required calculation time corresponds with the Catalan number and increases exponentially with the cardinality of the used alphabet, thus, resulting in longer calculation times for a number of other languages (e.g. Russian, which alphabet consists of 32 letters). Although the increasing CPU power will help overcome this issue in the nearest future, any successful approach to further reduce calculation times will be very valuable. At the same time, it is extremely unlikely that other approaches will lead to development of algorithms that are not NP-hard.

Second, the successful training remains the major prerequisite for brain-computer communication. With help of training it is possible to increase the level of control of brain activity and, thus, increase values of selection and rejection probabilities. This will lead to reduction of the value of the optimization criterion and result in faster and more accurate communication. In order to achieve this, the development of new training concepts, supplementary to copy-spelling, is required.

Third, in case of menu item-based communication, as opposed to letter-based communication, final trees are created mostly out of pre-defined sub-trees. These sub-trees consist, in turn, of logically grouped menu items. The developed integrated computer tool provides a capability of merging pre-defined sub-trees to an optimal final tree. First of all, this kind of communication allows influencing cardinality of the used alphabet. Furthermore, it logically provides an opportunity for communication for the patients whose spelling skills are poor. Unfortunately, no comprehensive research has yet been accomplished on a subject, what core needs locked-in patients have in general and how these needs can be clustered in particular. Here, again, any successful attempt would be of a great value.

6. References

[1] Anderson C. W., Stolz E. A., and Shamsunder S. Multivariate autoregressive models for classification of spontaneous electroencephalogram during mental tasks. *IEEE Transactions on Biomedical Engineering*, 1998, 45: 277-286

[2] Beukelman D., Mirenda P. "Augmentative and Alternative Communication: Management of Severe Communication Disorders in Children and Adults," Baltimore: Paul H. Brooks Publishing Company, 1992

[3] Birbaumer N., Elbert T., Lutzenberger W., Rockstroh B., Schwarz, J. EEG and slow cortical potentials in anticipation of mental tasks with different hemispheric involvement. *Biol. Psychol.*, 1981, 13: 251-260

[4] Birbaumer N. Operant control of slow brain potentials: a tool in the investigation of the potential's meaning and its relation to attentional dysfunction. In T. Elbert, B. Rockstroh, W. Lutzenberger, & N. Birbaumer (Eds.), *Self-Regulation of the Brain and Behaviour* , 1984, 1 ed.: 227-239, Berlin: Springer-Verlag

[5] Birbaumer N., Lang P. J., Cook I.E., Elbert T., Lutzenberger W., Rockstroh B. Slow brain potentials, imagery and hemispheric differences. *International Journal of Neuroscience,* 1988, 39: 101-116

[6] Birbaumer N., Elbert T., Canavan A. G. M., Rockstroh B. Slow potentials of the cerebral cortex and behaviour. *Physiological Reviews,*1990, 70(1): 1-41

[7] Birbaumer N., Ghanayim N., Hinterberger T., Iversen I., Kotchoubey B., Kübler A., Perelmouter J., Taub E., Flor H. A spelling device for the paralysed. *Nature*, 1999, 398: 297-298

[8] Borasio G. D. Amyotrophe Lateralsklerose (ALS): Molekulare Pathogenesis und experimentelle Therapie (Molecular pathogenesis and expiremental therapy). *Neuroforum*, 1996, 4: 5-13

[9] Cormen T. H., Leiserson C. E., Rivest R. L. "Introduction to Algorithms," Cambrige, Massachusetts: The MIT Press, 1997, Chapt. 17, pp. 337-343

[10] Doncin E., Spencer K. M., and Wijesinghe R. The mental prothesis: Assessing the speed of a P300-based brain-computer interface. *IEEE Transactions on Rehabilitation Engineering*, 2000, 8: 174-179

[11] Durrleman S., and Alperovitch A. Increasing trend of ALS in France and elsewhere: Are the changes real? *Neurology*, 1989, 39: 768-773

[12] Farwell L. A., and Doncin E. Talking off the top of your head: Toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and Clinical Neurophysiology*, 1988, 70: 512-523

[13] Flor H., Birbaumer N., Roberts L. A., Feige B., Lutzenberger W., Hermann C., and Kopp B. Slow potentials, event-related potentials, „gamma-band“ activity, and motor responses during aversive conditioning in humans. *Experimental Brain Research*, 1996, 112: 298-312

[14] Hinterberger T., Kaiser J., Kübler A., Neumann N., Birbaumer N. The Thought Translation Device and its applications to the completely paralyzed. In H. H. Diebner, T. Druckrey, P. Weibel (Eds), *Science of the interface*, 2001, pp. 232-240, Genista, Tübingen

[15] Huffman D. A. A method for the construction of minimum redundancy codes, *Proc. IRE*, 1952, 40: 1098-1101

[16] Institute of Medical Psychology and Behavioural Neurobiology – Faculty of Medicine – University of Tübingen (Biofeedback) http://www.medizin.uni-tuebingen.de/medpsych/proekte/biofeedb.htm (TTD) http://www.medizin.uni-tuebingen.de/medpsych/proekte/als.htm

[17] Kaiser J., Perelmouter J., Iversen I. H., Neumann N., Ghanayim N., Hinterberger T., Kübler A., Kotchoubey B., Birbaumer N. Self-initiation of EEG-based communication in paralyzed patients. *Clinical Neurophysiology*, 2001, 112: 551-554

[18] Keirn Z. A., and Aunon J. I. Man-machine communications trough brain-wave processing. *IEEE Engineering in Medicine and Biology Magazine*, 1990, 9: 55-57

[19] Keirn Z. A., and Aunon J. I. A new mode of communication between man and his surroundings. *IEEE Transactions on Biomedical Engineering*, 1990, 37: 1209-1214

[20] Kennedy P. R., and Bakay R. A. E. Restoration of neural output from a paralyzed patient by a direct brain connection. *NeuroReport*, 1998, 9: 1707-1711

[21] Kennedy P. R., Bakay R. A. E., Moore M., Adams K., and Goldwaithe J. Direct control of a computer from the human central nervous system. *IEEE Transactions on Rehabilitation Engineering*, 2000, 8: 198-202

[22] Knuth D. E. “The Art of Computer Programming”, Vol 1: *Fundamental Algorithm*. Addison-Wesley, Reading, MA, 1968

[23] Knuth D. E. “The Art of Computer Programming”, Vol 3: *Sorting and Searching*. Addison-Wesley, Reading, MA, 1973

[24] Kotchoubey B., Schleichert H., Lutzenberger W., Birbaumer N. A new method for self-regulation of slow cortical potentials in a timed paradigm. *Applied Psychophysiology and Biofeedback,* 1997, 22(2): 77-93

[25] Kübler A., Kotchoubey B., Hinterberger T., Ghanayim N., Perelmouter J., Schauer M., Fritsch C., Taub E., Birbaumer N. The thought translation device: A neurophysiological approach to communication in total motor paralysis. *Experimental Brain Research*, 1999, 124: 223-232

[26] Kübler A., Kotchoubey B., Kaiser J., Wolpaw J. R., Birbaumer N. Brain-computer communication: unlock the locked-in. *Psychological Bulletin,* 2001, 127: 358-375

[27] Kuhlman W. N. EEG feedback training: Enhancement of somatosensory cortical activity. *Electroencephalography and Clinical Neurophysiology*, 1978, 44: 290-294

[28] Mäkinen E. A Survey on Binary Tree Codings. *The Computer Journal*, 1991, 34(5): 438-443

[29] McFarland D. J., McCane L. M., David S. V., and Wolpaw J. R. Spatial filter selection for EEG-based communication. *Electroencephalography and Clinical Neurophysiology*, 1997, 103: 386-394

[30] Middendorf M. S., McMillan G., Calhoun G., and Jones K. S. Brain-computer interfaces based on the steady-state visual evoked response. *IEEE Transactions on Rehabilitation Engineering*, 2000, 8: 211-214

[31] Mitzdorf U. Physiological sources of evoked potentials. *Electroencephalography and Clinical Neurophysiology, Suppl.* 1991, 42: 47-57

[32] Munro I. On random walks in binary trees, *Research Report* CS-76-33, University of Waterloo

[33] Norris F. N. Amyotrophic Lateral Sclerosis: The Clinical Disorder In: Smith R A, ed. Handbook of amyotrophic lateral sclerosis. New York: Marcel Dekker, Inc, 1992: 3 - 38

[34] Oksenberg A., Soroker N., Solzi P., Reider-Groswasser I. Polysomnography in locked-in syndrome. *Electroencephalography and clinical Neurophysiol.*, 1991; 78: 314-317

[35] Pallo J., Racca R. A Note on Generating Binary Trees in A-order and B-order. *Intern. J. Computer Math.* 1985, 18: 27-39

[36] Perelmouter J., Kotchoubey B., Kübler A., Taub E., Birbaumer N. A Language Support Program for Thought Translation Devices. *Automedica*, 1999, 18: 67-84

[37] Perelmouter J., Birbaumer N. A Binary Spelling Interface with Random Errors. *IEEE Transactions on Rehabilitation Engineering*, 2000, 8: 227-232

[38] Pfurtscheller G., Flotzinger D., Pregenzer M., Wolpau J. R., and McFarland D. J. EEG-based brain computer interface. *Medical Progress Through Technology*, 1996, 21, 111-121

[39] Pfurtscheller G., Neuper C., Guger C., Harkam W., Ramoser H., Schlögl A., Obermaier B., and Pregenzer, M. Current trends in Graz brain-computer interface (BCI) research. *IEEE Transactions on Rehabilitation Engineering*, 2000, 8, 216-219

[40] Polich J., McIsaak H. K., Comparison of auditory P300 habituation from active and passive conditions. *International Journal of Psychophysiology*, 1994, 17: 25-34

[41] Rockstroh B., Elbert T., Canavan A., Lutzenberger W., Birbaumer, N. (1989). *Slow cortical potentials and behaviour*. Baltimore: Urban & Schwarzenberg

[42] Rotem D., Varol Y. L. Generation of Binary Trees from Ballot Sequences. *Journ. Assoc. Comp. Mach.*, 1978, 25 (3): 396-404

[43] Rotem D. Stack sortable permutations. *Discrete Mathematics* 1981, 33: 185-196

[44] Speckmann E. J., Walden, J. Mechanisms underlying the generation of cortical field potentials. *Acta Otolaryngologica, Suppl.* 1991, 491: 17-24

[45] Sutter E. E. The brain response interface: Communication through visually-induced electrical brain responses. *Journal of Microcomputer Applications*, 1992, 15: 31-45

[46] Sutter E. E., and Tran D. Communication through visually induced electrical brain responses. *Schriftenreihe der Oesterreichischen Computer Gesellschaft*, 1990, 55: 279-288

[47] Trojanowski A. E. Ranking and listing algorithm for *k*-ary trees. SIAM J. Comput., 1978, 7(4): 492-510

[48] Vidal J. J. Toward direct brain computer communication. *Annual Reviews of Biophysics and Bioengineering*, 1973, 2: 157-180

[49] Wali G. M. "Fou rire prodromique" heralding a brainstem stroke. *Neurology, Neurosurgery and Psychiatry*, 1993; 56: 209 – 210

[50] Wolpaw J. R., Birbaumer N., Heetderks W. J., McFarland D. J., Peckham P. H., Schalk G., Donchin E., Quatrano L. A., Robinson C. J., Vaughan T. M. Brain-Computer Interface Technology: A Review of the First International Meeting. *IEEE Transactions on Rehabilitation Engineering*, 2000, 8: 164-173

[51] Woplau J. R., McFarland D. J., Neat G. W., and Forneris C. A. An EEG-based brain-computer interface for cursor control. *Electroencephalography and Clinical Neurophysiology*, 1991, 78: 252-259

[52] Wyrwicka W., and Sterman M. B. Instrumental conditioning of sensorimotor cortex EEG spindles in the waking cat. *Psychology and Behavior*, 1968, 3: 703-707

[53] Zaks S. Lexicographic generation of ordered trees. *Theor. Comp. Sci*. 1980, 10(1): 63-82