Advances in Computer Science Research

Volume 10

Oficyna Wydawnicza Politechniki Białostockiej

Białystok 2013

EDITOR-IN-CHIEF / REDAKTOR NACZELNY:
Marek Krętowski (m.kretowski@pb.edu.pl, 85 746 90 95)

SCIENTIFIC EDITORS / SEKRETARZE NAUKOWI:
Magdalena Topczewska (m.topczewska@pb.edu.pl, 85 746 90 86)
Marek Parfieniuk (m.parfieniuk@pb.edu.pl, 85 746 91 08)

STATISTICAL EDITOR / REDAKTOR STATYSTYCZNY:
Dorota Mozyrska (d.mozyrska@pb.edu.pl, 85 746 90 82)

TECHNICAL EDITOR / SEKRETARZ TECHNICZNY:
Tomasz Łukaszuk (t.lukaszuk@pb.edu.pl, 85 746 90 86)

ISSN 2300-715X

This journal is the continuation of *Zeszyty Naukowe Politechniki Białostockiej. Informatyka*
issued in the years 2002-2012, ISSN 1644-0331

THE ADRESS FOR THE CORRESPONDENCE / ADRES DO KORESPONDENCJI:

„*Advances in Computer Science Research*"
Faculty of Computer Science / Wydział Informatyki
Bialystok University of Technology / Politechnika Białostocka
ul. Wiejska 45a, 15-351 Białystok
tel. 85 746 90 50, fax 85 746 97 22
e-mail: znpb@irys.wi.pb.edu.pl
http://irys.wi.pb.edu.pl/znpb

Printing / Druk: Oficyna Wydawnicza Politechniki Białostockiej
Issue / Nakład: 100 copies / egzemplarzy

CONTENTS

THE SET OF FORMULAS OF PrAL$^+$ VALID IN A FINITE STRUCTURE IS UNDECIDABLE

Anna Borowska

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: We consider a probabilistic logic of programs. In [6] it is proved that the set of formulas of the logic PrAL, valid in a finite structure, is decidable with respect to the diagram of the structure. We add to the language L_P of PrAL a sign $\bigcup$ and a functor lg. Next we justify that the set of formulas of extended logic, valid in a finite at least 2-element structure (for L_P^+) is undecidable.

Keywords: Probabilistic Algorithmic Logic, existential iteration quantifier

1. Introduction

In [6] the Probabilistic Algorithmic Logic PrAL is considered, constructed for expressing properties of probabilistic algorithms understood as iterative programs with two probabilistic constructions $x :=$ **random** and **either**$_p$ ... **or** ... **ro**. In order to describe probabilities of behaviours of programs a sort of variables (interpreted as real numbers) and symbols $+, -, *, 0, 1, <$ (interpreted in the standard way in the ordered field of real numbers) was added to the language L_P of PrAL.

In the paper [5] the changes of information which depend on realizations of probabilistic program was considered. That's why the language L_P was extended by adding the sign $\bigcup$ (called the existential iteration quantifier) and the functor lg (for the one-argument operation of a logarithm with a base 2 interpreted in the real ordered field). The new language was denoted by L_P^+.

The paper [6] contains an effective method of determining probabilities for probabilistic programs interpreted in a finite structure. The effectiveness of the method leads to the decidability of the set of formulas of L_P, valid in a fixed finite structure (provided that we have at our disposal a suitable finite part of the diagram of the structure). Here we shall justify that the set of probabilistic algorithmic formulas of L_P^+, valid in an arbitrary, finite at least 2-element structure, is undecidable with respect to its diagram.

We shall start from a presentation of the syntax and the semantics of the language L_P^+. We use the syntax and the semantics of L_P proposed by W. Danko in [6].

2. Syntax and Semantics of L_P^+

A language L_P is an extension of a first-order language L and includes three kinds of well-formed expressions: terms, formulas and programs. As mentioned above, the alphabet of L_P^+ contains two additional elements: the arithmetic one-argument functor lg and the sign $\bigcup$ (the existential iteration quantifier). An interpretation of L_P^+ relies on an interpretation of the first-order language L in a structure $\mathfrak{I}$ (We take into consideration only finite structures. By finite structure we mean a structure with a finite, at least 2-element set A.) and on the standard interpretation of the language $\mathrm{L}_{\mathfrak{R}}$ in the ordered field of real numbers (cf. [6]).

The alphabet of the language L_P^+ contains

- a set of constants C_P, which consists of a finite subset $C = \{c_1, \ldots, c_u\}$ of symbols for each element of the set $A = \{a_1, \ldots, a_u\}$, a subset $C_{\mathfrak{R}}$ of real constant symbols and a subset C_L of logical constant symbols,
- an enumerable set $V_P = \{V \cup V_{\mathfrak{R}} \bigcup V_0\}$ of variables, where a subset $V = \{v_0, v_1, \ldots\}$ consists of non-arithmetic individual variables, a subset $V_{\mathfrak{R}} = \{x_0, x_1, \ldots\}$ contains real variables and a subset $V_0 = \{q_0, q_1, \ldots\}$ contains propositional variables,
- a set of signs of relations $\Psi_P = \{\Psi \cup \Psi_{\mathfrak{R}}\}$, where the subset Ψ consists of non-arithmetic predicates and the subset $\Psi_{\mathfrak{R}} = \{<_{\mathfrak{R}}, =_{\mathfrak{R}}\}$ contains arithmetic predicates,
- an enumerable set of functors $\Phi_P = \{\Phi \cup \Phi_{\mathfrak{R}}\}$, which consists of the subset $\Phi_{\mathfrak{R}} = \{+, -, *, \mathrm{lg}\}$ of symbols for arithmetic operations and the subset Φ of symbols for non-arithmetic operations,
- the set $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ of logical connectives,
- the set {**if**, **then**, **else**, **fi**, **while**, **do**, **od**, **either**, **or**, **ro**, **random**$_l$[1]} of symbols for program constructions,
- the set $\{\exists, \forall\}$ of symbols for classical quantifiers (for real variables only),
- the existential iteration quantifier $\bigcup$,

[1] For each probability distribution defined on a set A we generate a different random assignment. We use a number l to distinguish them.

– the set $\{(,)\}$ of auxiliary symbols.

In the language L_P^+ we distinguish two kinds of terms (arithmetic and non-arithmetic), formulas (classical and algorithmic) and programs.

The set of terms $T_P = \{T \cup T_{\Re}\}$ of L_P^+ consists of a subset of non-arithmetic terms T and a subset $T_{\Re}$ of arithmetic terms.

Definition 2.1 The set T of *non-arithmetic terms* is defined as the smallest set of expressions satisfying the following conditions:
– each constant of C and each variable of V belongs to T,
– if $\phi_i \in \Phi$ (ϕ_i – an n_i-argument functor $(n_i \geq 0)$) and $\tau_1, \ldots, \tau_{n_i} \in T$ then an expression $\phi_i(\tau_1, \ldots, \tau_{n_i})$ belongs to T.

Definition 2.2 The set $T_{\Re}$ of *arithmetic terms* is the smallest set such that:
– each constant of $C_{\Re}$ and each real variable of $V_{\Re}$ belongs to $T_{\Re}$,
– if $t_1, t_2 \in T_{\Re}$ then expressions $t_1 + t_2$, $t_1 - t_2$, $t_1 * t_2$, $\lg t_1$ belong to $T_{\Re}$,
– if α is a formula of L then $P(\alpha)$ belongs to $T_{\Re}$. (We read the symbol P as follows "probability that".)

Definition 2.3 The set F_O of *open formulas* is the smallest set such that:
– if $\tau_1, \ldots, \tau_{m_j} \in T$ and $\psi_j \in \Psi$ (ψ_j – an m_j-argument predicate) then $\psi_j(\tau_1, \ldots, \tau_{m_j}) \in F_O$,
– if $\alpha, \beta \in F_O$ then expressions $\neg\alpha$, $\alpha \vee \beta$, $\alpha \wedge \beta$, $\alpha \Rightarrow \beta$, $\alpha \Leftrightarrow \beta$ belong to F_O.

Definition 2.4 The set Π of all *programs* is defined as the smallest set of expressions satisfying the following conditions:
– each expression of the form $v := \tau$ or $v :=$**random**$_l$, where $v \in V$, $\tau \in T$ is a program,
– if $\gamma \in F_O$ and $M_1, M_2 \in \Pi$ then expressions $M_1; M_2$, **if** γ **then** M_1 **else** M_2 **fi**, **while** γ **do** M_1 **od**, **either**$_p$ M_1 **or** M_2 **ro** (p is a real number) are programs.

We establish that in an expression $\bigcup K\alpha$ (where K is a program) the letter α denotes a formula which does not contain any iteration quantifiers.

Definition 2.5 The set F_P of all *formulas* of the language L_P^+ is the smallest extension of the set F_O such that:
– if $t_1, t_2 \in T_{\Re}$ then $t_1 =_{\Re} t_2$, $t_1 <_{\Re} t_2$ belong to F_P,
– if $\alpha, \beta \in F_P$ then the expressions $\neg\alpha$, $\alpha \vee \beta$, $\alpha \wedge \beta$, $\alpha \Rightarrow \beta$, $\alpha \Leftrightarrow \beta$ belong to F_P,

– if $\alpha \in F_P$ and $x \in V_{\mathfrak{R}}$ is a free variable in α then $\exists x\alpha$, $\forall x\alpha$ belong to F_P,
– if $K \in \Pi$ and $\alpha \in F_P$ then $K\alpha$ is a formula of F_P,
– if $K \in \Pi$ and $\alpha \in F_P$ then $\bigcup K\alpha$ belongs to F_P.

A variable x is *free* in a formula α if x is not bounded by any quantifier.

Let L_P^+ be a fixed algorithmic language of the type $< \{n_k\}_{\phi_k \in \Phi_P}, \{m_l\}_{\psi_l \in \Psi_P} >$ and let a relational system $\Im =< A \cup R;\ \{\phi_{k\Im}\}_{\phi_k \in \Phi_P},\ \{\psi_{l\Im}\}_{\psi_l \in \Psi_P} >$ (which consists of the fixed, finite, at least 2-element set A, the set R of real numbers, operations and relations) be a fixed data structure for L_P^+.

We interpret non-arithmetic individual variables of L_P^+ as elements of A. Real variables are interpreted as elements of the set R of real numbers.

Let's denote the set of possible valuations w of non-arithmetic variables by W.

Definition 2.6 By *the interpretation of a non-aritmetic term* τ of L_P in the structure $\Im$ we mean a function $\tau_\Im : W \mapsto A$ which is defined recursively.
– If τ is a variable $v \in V$ then $v_\Im(w) \overset{df}{=} w(v)$.
– If τ is of the form $\phi(\tau_1, \ldots, \tau_n)$, where $\tau_1, \ldots, \tau_n \in T$ and $\phi \in \Phi$ is an n-argument functor then $\phi(\tau_1, \ldots, \tau_n)_\Im(w) \overset{df}{=} \phi_\Im(\tau_{1\Im}(w), \ldots, \tau_{n\Im}(w))$, where $\tau_{1\Im}(w), \ldots, \tau_{n\Im}(w)$ are defined earlier.

To interpret random assignments (i.e. constructions of the form $v :=$**random**$_l$) in a probabilistic way we assume that there exists a fixed probability distribution defined on A

$$\rho_l : A \mapsto [0,1], \qquad \sum_{i=1}^{u} \rho_l(a_i) = 1.$$

Definition 2.7 (cf. [6]) A pair $< \Im, \rho >$, where ρ is a set of fixed probability distributions ρ_l defined on A and $\Im$ is a structure for L_P^+, is called *a probabilistic structure*. In this structure we interpret probabilistic programs.

By $\mathcal{M}$ we denote the set of all probability distributions defined on the set W of valuations of non-arithmetic variables such that

$$\mu : W \mapsto [0,1], \qquad \sum_{w_i \in W} \mu(w_i) \leq 1.$$

By S we mean the set of all *states*, i.e. all pairs $s =< \mu, w_{\mathfrak{R}} >$, where μ is a probability distribution of valuations of non-arithmetic variables and $w_{\mathfrak{R}}$ is a valuation of real variables of $V_{\mathfrak{R}}$.

Definition 2.8 (cf. [6]) A probabilistic program K is interpreted in the structure $< \Im, \rho >$ as a partial function transforming the set of states into the set of states

$$K_{<\Im,\rho>} : S \mapsto S.$$

Let $K(v_1, \ldots, v_h)$ represent a fixed program in L_P^+. An arbitrary program K contains only a finite number of non-arithmetic variables. We denote this set of variables by $V = \{v_1, \ldots, v_h\}$. Since $A = \{a_1, \ldots, a_u\}$ is also a finite set, then a set of all possible valuations of program variables will be also finite. We denote it by $\{w_1, \ldots, w_n\}$, where $n = u^h$.

Let's notice that programs do not operate on variables of $V_{\Re}$. Thus we can interpret an arbitrary program K as partial functions transforming probability distributions defined on the set of valuations of program variables (cf. [6])

$$K_{<\Im,\rho>} : \mathcal{M} \mapsto \mathcal{M}.$$

If μ is the input probability distribution of valuations of program variables (input probability distribution for short) then a realization of a program K leads to a new output probability distribution $\mu^{'}$ of valuations of program variables (output probability distribution for short). A distribution μ ($\mu^{'}$) associates with each valuation w of program variables a corresponding probability of its appearance.

The interpretation of program constructions (used in this paper) can be found in the Appendix.

An arithmetic term of the form $\mathrm{P}(\alpha)$ denotes the probability, that the formula α of L is satisfied at a distribution μ (cf. [6])

$[\mathrm{P}(\alpha)]_{\Im}(s) = \sum_{w \in W^{\alpha}} \mu(w)$, where $W^{\alpha} = \{w \in W : \Im, w \models \alpha\}$.

Let $s =< \mu, w_{\Re} >$ be a state and let $s^{'} =< \mu^{'}, w_{\Re} >$ represent the state $s^{'} = K_{<\Im,\rho>}(s)$.

Given below is the interpretation of a formula $K\alpha$ ($\alpha \in F_P$ and $K \in \Pi$).

$$(K\alpha)_{<\Im,\rho>}(s) = \begin{cases} \alpha_{<\Im,\rho>}(s^{'}) & \text{if } K_{<\Im,\rho>}(s) \text{ is defined and } s^{'} = K_{<\Im,\rho>}(s) \\ \text{is not defined otherwise} \end{cases}$$

The satisfiability of a formula $K\alpha$, where $\alpha \in F_P$ and $K \in \Pi$, is defined in the following way (cf. [6])

$< \Im, \rho >, s \models K\alpha$ iff $< \Im, \rho >, s^{'} \models \alpha$, where $s^{'} = K_{<\Im,\rho>}(s)$.

The next definition establishes the meaning of the existential iteration quantifier ($K \in \Pi$, $\alpha \in F_P$).

$$(\bigcup K\alpha)_{<\Im,\rho>}(s) \overset{df}{=} \underset{i\in N}{\text{l.u.b.}} (K^i\alpha)_{<\Im,\rho>}(s).$$

We can informally express the formula $\bigcup K\alpha$ in the following way $\alpha \vee K\alpha \vee K^2\alpha \vee \ldots$

The satisfiability of a formula $\bigcup K\alpha$ ($K \in \Pi$, $\alpha \in F_P$) is defined as an infinite alternative of formulas $(K^i\alpha)$ for $i \in N$.

Example 2.10 Now we shall present a formula which contains the iteration quantifier. Let's consider the formula β : $K_0 \bigcup K\alpha$ such that

K_0: $v_1 := 0$;
K : **if** $(v_1 = 0)$ **then** v_1 :=**random**$_1$; $v_2 := 0$; **else** $v_2 := 1$; **fi**
α: $x = \mathrm{P}(v_1 = 1 \vee v_2 = 0)$

where K_0 and K are programs interpreted in the structure $< \Im, \rho >$ with a 2-element set $A = \{0,1\}$. For a random assignment v_1 :=**random**$_1$ we define the probability distribution $\rho_1 = [0.5, 0.5]$. The set of possible valuations of program variables contains 4 elements: $w_1 = (0,0)$, $w_2 = (0,1)$, $w_3 = (1,0)$, $w_4 = (1,1)$. We carry out computations for the input probability distribution $\mu = [0.25, 0.25, 0.25, 0.25]$. $\mathrm{P}(\gamma)$ denotes the probability that γ is satisfied (at a distribution μ). Let's notice, that formula β describes the following fact

$(x = 0) \vee (x = 0.5) \vee (x = 0.5 * 0.5) \vee (x = 0.5 * 0.5 * 0.5) \vee \ldots$

3. The proof of the main lemma

As we have mentioned (it is proved in [6]), the set of probabilistic algorithmic formulas of PrAL valid in a finite structure for L_P is decidable with respect to the diagram of the structure. By the diagram $D(\Im)$ of the structure $\Im$ we understand the set of all atomic or negated atomic formulas $\phi(c_{i_1}, \ldots, c_{i_m}) = c_{i_0}$ (ϕ is a functor of L) and $\psi(c_{i_1}, \ldots, c_{i_m})$ (ψ is a predicate symbol of L), which are valid in $\Im$.

The proof of decidability of PrAL essentially uses the Lemma which reduces the problem of validity of sentences of L_P to the (decidable) problem of the validity of sentences of the first-order arithmetic of real numbers. Finally, it appears that the set of formulas of PrAL, valid in all at most u-element structures for L_P, is decidable.

We shall show that if the language L_P^+ contains additionally the sign $\bigcup$ and the functor lg (for the operation of a logarithm) we can define natural numbers and operations of addition and multiplication for natural numbers.

Let's assume that 0.5^i abbreviates the expression $\underbrace{0.5 * 0.5 * \ldots * 0.5}_{i\ times}$.

Lemma 3.1 Let $< \Im, \rho >$ be an arbitrary fixed probabilistic structure (for L_P^+) with a finite set $A = \{a_1, a_2, \ldots, a_u\}$, where $u > 1$. Let K_0 and K be as follows

```
K_0:  $v_1 := a_u$;
K :   **if** $(v_1 = a_u)$ **then**
          **either**$_{0.5}$ $v_1 := a_u$; $v_2 := a_u$; **or** $v_1 := a_{u-1}$; $v_2 := a_u$; **ro**
      **else** $v_1 := a_1$; $v_2 := a_u$; **fi**
```

For an arbitrary natural number $i > 0$, if $\mu = [\mu_1, \mu_2, \ldots, \mu_{u^2}]$ is an input probability distribution then as a result of realization of program $K_0; K^i$ we obtain the following output probability distribution

$$\mu' = K_0 K^i_{<\Im,\rho>}(\mu) = [\underbrace{0,\ldots,0}_{u-1\ times}, 1 - 0.5^{(i-1)}, \underbrace{0,\ldots,0}_{u^2-2u-1\ times}, 0.5^i, \underbrace{0,\ldots,0}_{u-1\ times}, 0.5^i].$$

Proof. Let us assume that $< \Im, \rho >$ is a fixed probabilistic structure (for L_P^+) with a finite at least 2-element set $A = \{a_1, a_2, \ldots, a_u\}$. Let's consider an arbitrary program $K_0; K^i$ ($i \in N_+$). The set of possible valuations of program variables contains u^2 elements: $w_1 = (a_1, a_1)$, $w_2 = (a_1, a_2)$, ..., $w_u = (a_1, a_u)$, $w_{u+1} = (a_2, a_1)$, $w_{u+2} = (a_2, a_2)$, ..., $w_{2u} = (a_2, a_u)$, ...,$w_{u^2-u+1} = (a_u, a_1)$, $w_{u^2-u+2} = (a_u, a_2)$, ..., $w_{u^2} = (a_u, a_u)$. We carry out computations for the input probability distribution $\mu = [\mu_1, \mu_2, \ldots, \mu_{u^2}]$. The proof of the Lemma 3.1 will proceed by induction on the length of programs.

(A) The base of induction.

First we shall justify that the realization of the program $K_0; K$ leads to the probability distribution

$$\mu' = K_0 K_{<\Im,\rho>}(\mu) = [\underbrace{0,\ldots,0}_{u^2-u-1\ times}, 0.5, \underbrace{0,\ldots,0}_{u-1\ times}, 0.5].$$

We shall determine the necessary probability distributions (cf. the Appendix).

$$[v_1 := a_1]_{<\Im,\rho>}(\mu) = [\mu_1 + \mu_{u+1} + \ldots + \mu_{u^2-u+1}, \mu_2 + \mu_{u+2} + \ldots + \mu_{u^2-u+2}, \ldots, \mu_u + \mu_{2u} + \ldots + \mu_{u^2}, \underbrace{0,\ldots,0}_{u^2-u\ times}]$$

$$[v_1 := a_{u-1}]_{<\Im,\rho>}(\mu) = [\underbrace{0,\ldots,0}_{u^2-2u\ times}, \mu_1 + \mu_{u+1} + \ldots + \mu_{u^2-u+1}, \mu_2 + \mu_{u+2} + \ldots + \mu_{u^2-u+2}, \ldots, \mu_u + \mu_{2u} + \ldots + \mu_{u^2}, \underbrace{0,\ldots,0}_{u\ times}]$$

$[v_1 := a_u]_{<\Im,\rho>}(\mu) = [\underbrace{0,\ldots,0}_{u^2-u\ times},\mu_1 + \mu_{u+1} + \ldots + \mu_{u^2-u+1},\mu_2 + \mu_{u+2} + \ldots + \mu_{u^2-u+2},\ldots,\mu_u + \mu_{2u} + \ldots + \mu_{u^2}]$

$[v_2 := a_u]_{<\Im,\rho>}(\mu) = [\underbrace{0,\ldots,0}_{u-1\ times},\mu_1 + \mu_2 + \ldots + \mu_u,\underbrace{0,\ldots,0}_{u-1\ times},\mu_{u+1} + \mu_{u+2} + \ldots + \mu_{2u},\underbrace{0,\ldots,0}_{u-1\ times},\ldots,\underbrace{0,\ldots,0}_{u-1\ times},\mu_{u^2-u+1} + \mu_{u^2-u+2} + \ldots + \mu_{u^2}]$

Let's denote the subprogram $v_1 := a_u$; $v_2 := a_u$; by N_1.

$N_{1<\Im,\rho>}(\mu) = [v_2 := a_u]_{<\Im,\rho>}([v_1 := a_u]_{<\Im,\rho>}(\mu)) =$
$= [\underbrace{0,\ldots,0}_{u^2-1\ times},(\mu_1 + \mu_{u+1} + \ldots + \mu_{u^2-u+1}) + (\mu_2 + \mu_{u+2} + \ldots + \mu_{u^2-u+2}) + \ldots + (\mu_u + \mu_{2u} + \ldots + \mu_{u^2})] =$
$= [\underbrace{0,\ldots,0}_{u^2-1\ times},\mu_1 + \mu_2 + \ldots + \mu_{u^2}] = [\underbrace{0,\ldots,0}_{u^2-1\ times},1]$

By N_2 we denote the subprogram $v_1 := a_{u-1}$; $v_2 := a_u$;.

$N_{2<\Im,\rho>}(\mu) = [v_2 := a_u]_{<\Im,\rho>}([v_1 := a_{u-1}]_{<\Im,\rho>}(\mu)) =$
$= [\underbrace{0,\ldots,0}_{u^2-u-1\ times},(\mu_1 + \mu_{u+1} + \ldots + \mu_{u^2-u+1}) + (\mu_2 + \mu_{u+2} + \ldots + \mu_{u^2-u+2}) + \ldots + (\mu_u + \mu_{2u} + \ldots + \mu_{u^2}),\underbrace{0,\ldots,0}_{u\ times}] =$
$= [\underbrace{0,\ldots,0}_{u^2-u-1\ times},\mu_1 + \mu_2 + \ldots + \mu_{u^2},\underbrace{0,\ldots,0}_{u\ times}] = [\underbrace{0,\ldots,0}_{u^2-u-1\ times},1,\underbrace{0,\ldots,0}_{u\ times}]$

The subprogram $v_1 := a_1$; $v_2 := a_u$; we denote by N_3.

$N_{3<\Im,\rho>}(\mu) = [v_2 := a_u]_{<\Im,\rho>}([v_1 := a_1]_{<\Im,\rho>}(\mu)) =$
$= [\underbrace{0,\ldots,0}_{u-1\ times},(\mu_1 + \mu_{u+1} + \ldots + \mu_{u^2-u+1}) + (\mu_2 + \mu_{u+2} + \ldots + \mu_{u^2-u+2}) + \ldots + (\mu_u + \mu_{2u} + \ldots + \mu_{u^2}),\underbrace{0,\ldots,0}_{u^2-u\ times}] =$
$= [\underbrace{0,\ldots,0}_{u-1\ times},\mu_1 + \mu_2 + \ldots + \mu_{u^2},\underbrace{0,\ldots,0}_{u^2-u\ times}] = [\underbrace{0,\ldots,0}_{u-1\ times},1,\underbrace{0,\ldots,0}_{u^2-u\ times}]$

Let's denote the subprogram **either**$_{0.5}$ N_1 **or** N_2 **ro** by E.

$E_{<\Im,\rho>}(\mu) = 0.5 * (N_{1<\Im,\rho>}(\mu)) + 0.5 * (N_{2<\Im,\rho>}(\mu)) =$
$= 0.5 * [\underbrace{0,\ldots,0}_{u^2-1\ times},\mu_1 + \mu_2 + \ldots + \mu_{u^2}] + 0.5 * [\underbrace{0,\ldots,0}_{u^2-u-1\ times},\mu_1 + \mu_2 + \ldots + \mu_{u^2},\underbrace{0,\ldots,0}_{u\ times}] =$

$$= [\underbrace{0,\dots,0}_{u^2-u-1\ times}, 0.5*(\mu_1+\dots+\mu_{u^2}), \underbrace{0,\dots,0}_{u-1\ times}, 0.5*(\mu_1+\dots+\mu_{u^2})] =$$
$$= [\underbrace{0,\dots,0}_{u^2-u-1\ times}, 0.5, \underbrace{0,\dots,0}_{u-1\ times}, 0.5]$$

$$[(v_1=a_u)?]_{<\Im,\rho>}(\mu) = [\underbrace{0,\dots,0}_{u^2-u\ times}, \mu_{u^2-u+1}, \mu_{u^2-u+2}, \dots, \mu_{u^2}]$$
$$[\neg(v_1=a_u)?]_{<\Im,\rho>}(\mu) = [\mu_1,\mu_2,\dots,\mu_{u^2-u}, \underbrace{0,\dots,0}_{u\ times}]$$

$$K_{<\Im,\rho>}(\mu) = E_{<\Im,\rho>}([(v_1=a_u)?]_{<\Im,\rho>}(\mu)) + N_{3<\Im,\rho>}([\neg(v_1=a_u)?]_{<\Im,\rho>}(\mu)) =$$
$$= [\underbrace{0,\dots,0}_{u^2-u-1\ times}, 0.5*(\mu_{u^2-u+1}+\mu_{u^2-u+2}+\dots+\mu_{u^2}), \underbrace{0,\dots,0}_{u-1\ times}, 0.5*(\mu_{u^2-u+1} +$$
$$\mu_{u^2-u+2}+\dots+\mu_{u^2})] + [\underbrace{0,\dots,0}_{u-1\ times}, (\mu_1+\mu_2+\dots+\mu_{u^2-u}), \underbrace{0,\dots,0}_{u^2-u\ times}] =$$
$$= [\underbrace{0,\dots,0}_{u-1\ times}, (\mu_1+\mu_2+\dots+\mu_{u^2-u}), \underbrace{0,\dots,0}_{u^2-2u-1\ times}, 0.5*(\mu_{u^2-u+1}+\mu_{u^2-u+2}+\dots+$$
$$\mu_{u^2}), \underbrace{0,\dots,0}_{u-1\ times}, 0.5*(\mu_{u^2-u+1}+\mu_{u^2-u+2}+\dots+\mu_{u^2})]$$

Finally

$$K_{<\Im,\rho>}(K_{0<\Im,\rho>}(\mu)) = K_{<\Im,\rho>}([v_1 := a_u]_{<\Im,\rho>}(\mu)) =$$
$$= [\underbrace{0,\dots,0}_{u^2-u-1\ times}, 0.5*((\mu_1+\mu_{u+1}+\dots+\mu_{u^2-u+1})+(\mu_2+\mu_{u+2}+\dots+\mu_{u^2-u+2})+\dots+$$
$$(\mu_u+\mu_{2u}+\dots+\mu_{u^2})), \underbrace{0,\dots,0}_{u-1\ times}, 0.5*((\mu_1+\mu_{u+1}+\dots+\mu_{u^2-u+1})+(\mu_2+\mu_{u+2}+$$
$$\dots+\mu_{u^2-u+2})+\dots+(\mu_u+\mu_{2u}+\dots+\mu_{u^2}))] =$$
$$= [\underbrace{0,\dots,0}_{u^2-u-1\ times}, 0.5*(\mu_1+\mu_2+\dots+\mu_{u^2}), \underbrace{0,\dots,0}_{u-1\ times}, 0.5*(\mu_1+\mu_2+\dots+\mu_{u^2})] =$$
$$= [\underbrace{0,\dots,0}_{u^2-u-1\ times}, 0.5, \underbrace{0,\dots,0}_{u-1\ times}, 0.5].$$

(B) The inductive step.

The inductive assumption. For a certain natural number k, if $\mu = [\mu_1,\mu_2,\dots,\mu_{u^2}]$ is an input probability distribution then as a result of realization of the program $K_0;K^k$ we obtain the following output probability distribution

$K_0K^k\!<\Im,\rho>(\mu) =$
$= [\underbrace{0,\ldots,0}_{u-1\ times},(1-0.5^{(k-1)})*(\mu_1+\mu_2+\ldots+\mu_{u^2}),\ \underbrace{0,\ldots,0}_{u^2-2u-1\ times}\ ,0.5^k*(\mu_1+\mu_2+\ldots+$
$\mu_{u^2}),\underbrace{0,\ldots,0}_{u-1\ times},0.5^k*(\mu_1+\mu_2+\ldots+\mu_{u^2})] =$
$= [\underbrace{0,\ldots,0}_{u-1\ times},(1-0.5^{(k-1)}),\ \underbrace{0,\ldots,0}_{u^2-2u-1\ times}\ ,0.5^k,\underbrace{0,\ldots,0}_{u-1\ times},0.5^k]$

We shall apply the inductive assumption to show that if we take $\mu = [\mu_1,\mu_2,\ldots,\mu_{u^2}]$ as the input probability distribution then after the execution of the program $K_0;K^{k+1}$ we obtain the following output probability distribution
$K_0K^{k+1}_{<\Im,\rho>}(\mu) = [\underbrace{0,\ldots,0}_{u-1\ times},(1-0.5^k)*(\mu_1+\mu_2+\ldots+\mu_{u^2}),\ \underbrace{0,\ldots,0}_{u^2-2u-1\ times}\ ,$
$0.5^{(k+1)}*(\mu_1+\mu_2+\ldots+\mu_{u^2}),\underbrace{0,\ldots,0}_{u-1\ times},0.5^{(k+1)}*(\mu_1+\mu_2+\ldots+\mu_{u^2})] =$
$= [\underbrace{0,\ldots,0}_{u-1\ times},(1-0.5^k),\ \underbrace{0,\ldots,0}_{u^2-2u-1\ times}\ ,0.5^{(k+1)},\underbrace{0,\ldots,0}_{u-1\ times},0.5^{(k+1)}]$

We can express a composition of programs in the following way (cf. the Appendix)
$K_0K^{k+1}_{<\Im,\rho>}(\mu) = K_{<\Im,\rho>}(K_0K^k_{<\Im,\rho>}(\mu))$

Hence by the inductive assumption
$K_{<\Im,\rho>}(K_0K^k_{<\Im,\rho>}(\mu)) = K_{<\Im,\rho>}([\underbrace{0,\ldots,0}_{u-1\ times},(1-0.5^{(k-1)})*(\mu_1+\mu_2+\ldots+\mu_{u^2}),$
$\underbrace{0,\ldots,0}_{u^2-2u-1\ times}\ ,0.5^k*(\mu_1+\mu_2+\ldots+\mu_{u^2}),\underbrace{0,\ldots,0}_{u-1\ times},0.5^k*(\mu_1+\mu_2+\ldots+\mu_{u^2})]) =$
$= [\underbrace{0,\ldots,0}_{u-1\ times},(1-0.5^{(k-1)}+0.5^k),\ \underbrace{0,\ldots,0}_{u^2-2u-1\ times}\ ,0.5*0.5^k,\underbrace{0,\ldots,0}_{u-1\ times},0.5*0.5^k] =$
$= [\underbrace{0,\ldots,0}_{u-1\ times},(1-0.5^k),\ \underbrace{0,\ldots,0}_{u^2-2u-1\ times}\ ,0.5^{(k+1)},\underbrace{0,\ldots,0}_{u-1\ times},0.5^{(k+1)}]$

which accomplishes the inductive proof.

□

Lemma 3.2 Let $<\Im,\rho>$ be an arbitrary fixed structure (for L_P^+) with a finite set $A=\{a_1,a_2,\ldots,a_u\}$, where $u>1$. The set of formulas of PrAL$^+$ valid in $<\Im,\rho>$ is undecidable.

Proof. Let $<\Im,\rho>$ be an arbitrary fixed structure (for L_P^+) with a finite at least 2-element set $A=\{a_1,\dots,a_u\}$. Let's consider the formula β of the form $K_0\bigcup K\alpha$, where K_0, K are the programs considered in the Lemma 3.1 and α is as follows

α: $x=P(v_1=a_{u-1}\wedge v_2=a_u)$.

The computations are carried out for the input probability distribution $\mu=[\mu_1,\mu_2,\dots,\mu_{u^2}]$ and for programs K_0 and $K_0;K^i$, where $i\in N_+$. Let's denote $K_{0<\Im,\rho>}(\mu)$ by η. We know that

$\eta=K_{0<\Im,\rho>}(\mu)=[v_1:=a_u]_{<\Im,\rho>}(\mu)=[\underbrace{0,\dots,0}_{u^2-u\ times},\mu_1+\mu_{u+1}+\dots+\mu_{u^2-u+1},\mu_2+\mu_{u+2}+\dots+\mu_{u^2-u+2},\dots,\mu_u+\mu_{2u}+\dots+\mu_{u^2}]$.

By the Lemma 3.1 we obtain that for an arbitrary number $i>0$

$\mu'=K_0K^i_{<\Im,\rho>}(\mu)=[\underbrace{0,\dots,0}_{u-1\ times},(1-0.5^{(i-1)}),\ \underbrace{0,\dots,0}_{u^2-2u-1\ times}\ ,\underline{0.5^i},\underbrace{0,\dots,0}_{u-1\ times},0.5^i]$.

We recall, that $P(v_1=a_{u-1}\wedge v_2=a_u)=\mu'(w_{u^2-u})$, where $w_{u^2-u}=(a_{u-1},a_u)$. We can notice that for $i\in N_+$ we have $\mu'(w_{u^2-u})=\underline{0.5^i}$ and additionally $\eta(w_{u^2-u})=\underline{0}$. Therefore the formula β: $K_0\bigcup K\alpha$ describes the following fact

$(x=0)\vee(x=0.5)\vee(x=0.25)\vee(x=0.125)\vee\dots\vee(x=0.5^i)\vee\dots$

Let's notice, that we can define an arbitrary natural number k in the following way. Let k be a real number

$N(k)$ iff $<\Im,\rho>\models\ (k=0\vee\exists x((k=-\lg x)\wedge K_0\bigcup K\alpha))$.

Since the natural numbers were generated among real numbers and operations of addition and multiplication exist in the structure $\Re=<R;+,-,*,0,1,<>$, we can define these operations for constructed natural numbers. For arbitrary x_0, x_1, x_2

$x_0\underline{+}x_1=x_2$ iff $<\Im,\rho>\models N(x_0)\wedge N(x_1)\wedge x_2=x_0+x_1$,
$x_0\underline{*}\,x_1=x_2$ iff $<\Im,\rho>\models N(x_0)\wedge N(x_1)\wedge x_2=x_0*x_1$.

Since $Th(<N;\underline{+},\underline{*},0,1>)$ is undecidable (cf. [2,11,7]), the set of formulas of considered algorithmic logic, valid in a fixed, finite at least 2-element structure (for L_P^+) is also undecidable.

□

4. Appendix (cf. [6])

By the interpretation of a program K of L_P^+ in the structure $<\Im,\rho>$ we mean a function $K_{<\Im,\rho>}: \mathcal{M} \mapsto \mathcal{M}$ which is defined recursively.

- If K is *an assignment instruction* of the form $v_r := \tau$ (for $v_r \in V$, $r = 1,\dots,h$ and $\tau \in T$) then
  $[v_r := \tau]_{<\Im,\rho>}(\mu) = \mu'$,where
  $\mu'(w_j) = \sum_{w \in W^{r,\tau}} \mu(w)$ for $j = 1,\dots,n$ and
  $W^{r,\tau} = \{w \in W : w(v_r) = \tau_\Im(w_{in}) \wedge \forall_{v \in V \setminus \{v_r\}}\, w(v) = w_{in}(v)\}$.
  w_{in} denotes an input valuation of program variables.

- If K is *a random assignment* of the form $v_r :=$**random**$_l$ (for $v_r \in V$, $r = 1,\dots,h$ and ρ_l being a probability distribution defined on A) then
  $[v_r :=\textbf{random}_l]_{<\Im,\rho>}(\mu) = \mu'$, where
  $\mu'(w_j) = \rho_l(w_j(v_r)) * \sum_{w \in W^r} \mu(w)$ and
  $W^r = \{w \in W : \forall_{v \in V \setminus \{v_r\}}\, w(v) = w_{in}(v)\}$.

- We interpret the program **while** $\neg\gamma$ **do** $v := v$ **od** (for $v \in V$ and $\gamma \in F_O$) in the following way
  $[\gamma?]_{<\Im,\rho>}(\mu) = [\textbf{while}\ \neg\gamma\ \textbf{do}\ v := v\ \textbf{od}]_{<\Im,\rho>}(\mu) = \mu'$, where
  $$\mu'(w_j) = \begin{cases} \mu(w_i) \text{ for } w_i = w_j \ \wedge\ \Im, w_i \models \gamma \\ 0 \quad\ \text{otherwise} \end{cases}$$
  We denote this program construction by $[\gamma?]$.

- If K is *a composition of programs* M_1, M_2 and $M_{1<\Im,\rho>}(\mu)$, $M_{2<\Im,\rho>}(\mu)$ are defined then
  $[M_1;\ M_2]_{<\Im,\rho>}(\mu) = M_{2<\Im,\rho>}(M_{1<\Im,\rho>}(\mu))$.

- If K is *a branching between the two programs* M_1, M_2 and $M_{1<\Im,\rho>}(\mu)$, $M_{2<\Im,\rho>}(\mu)$ are defined then
  $[\textbf{if}\ \gamma\ \textbf{then}\ M_1\ \textbf{else}\ M_2\ \textbf{fi}]_{<\Im,\rho>}(\mu) =$
  $= M_{1<\Im,\rho>}([\gamma?]_{<\Im,\rho>}(\mu)) + M_{2<\Im,\rho>}([\neg\gamma?]_{<\Im,\rho>}(\mu))$.

- If K is *a probabilistic branching*, $p \in R$, $0 < p < 1$ and $M_{1<\Im,\rho>}(\mu)$, $M_{2<\Im,\rho>}(\mu)$ are defined then
  $[\textbf{either}_p\ M_1\ \textbf{or}\ M_2\ \textbf{ro}]_{<\Im,\rho>}(\mu) = p * M_{1<\Im,\rho>}(\mu) + (1-p) * M_{2<\Im,\rho>}(\mu)$.

References

[1] L. Banachowski, Investigations of Properties of Programs by Means of the Extended Algorithmic Logic, Fundamenta Informatica, 1 (1977), p. 167–193.

[2] J. Barwise, Handbook of Mathematical Logic, Elsevier Science Publishers B.V., 1983.

[3] A. Borowska, Algorithmic Information Theory, Computer Information Systems and Industrial Management Applications, 2003, p. 5–31.

[4] A. Borowska, Algebraic Methods of Determining of Transition Probabilities in Probabilistic Algorithms, Conference Materials of XV FIT, 2004, p. 148–157.

[5] A. Borowska, Selected Problems of the Probabilistic Algorithms and the Markov Chains. Reduction of Valuations, Ph.D. Thesis, Technical University of Bialystok, 2010.

[6] W. Danko, The set of Probabilistic Algorithmic Formulas Valid in a Finite Structure is Decidable with Respect to its diagram, Fundamenta Informatica, Vol. 19 (1993), p. 417–431.

[7] A. Grzegorczyk, An Outline of Theoretical Arithmetics, PWN, Warsaw 1971.

[8] J. Koszelew, The Methods of Analysis of Properties of Probabilistic Programs Interpreted in Finite Fields, Ph.D. Thesis, PAN, 2000.

[9] A. Kreczmar, Effectivity problems of algorithmic logic, Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 14 (1974), Publisher Springer Berlin Heidelberg, p. 584–600.

[10] A. Kreczmar, The Set of all Tautologies of Algorithmic Logic is Hyperarithmetical, Bull. Acad. Polon. Sci., Ser. Math. Astron. Phys., 21 (1971), p. 781–783.

[11] R. C. London, About Mathematical Logic, PWN, Warsaw, 1968.

[12] G. Mirkowska, A. Salwicki, Algorithmic Logic, PWN-Polish Scientific Publishers, 1987.

[13] G. Mirkowska, A. Salwicki, Algorithmic Logic for Programmers, WN-T, Warsaw, 1992.

[14] A. Rutkowski, Elements of Mathematical Logic, WSiP, Warsaw, 1978.

Anna Borowska

ZBIÓR FORMUŁ LOGIKI PrAL^{+} PRAWDZIWYCH W SKOŃCZONEJ STRUKTURZE JEST NIEROZSTRZYGALNY

Streszczenie Rozważamy probabilistyczną logikę algorytmiczną. W pracy [6] znajduje się uzasadnienie, że zbiór formuł logiki PrAL, prawdziwych w skończonej strukturze, jest rozstrzygalny ze względu na diagram struktury. Dodajemy do języka L_P logiki PrAL znak $\bigcup$ i funktor lg. Następnie uzasadniamy, że zbiór formuł rozszerzonej logiki, prawdziwych w skończonej co najmniej 2-elementowej strukturze (dla L_P^{+}), nie jest już rozstrzygalny.

Słowa kluczowe: probabilistyczna logika algorytmiczna, egzystencjalny kwantyfikator iteracji

THE CRYPTANALYSIS OF THE ENIGMA CIPHER

Anna Borowska

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: In this paper we study cryptanalysis of the military Enigma machine which was used by the German Army during the Second World War. We are interested in the problem of decoding secret messages transmitted after 15 September 1938. We give a complete algorithm which can be used to: generate the ring settings, guess what kinds of drums are chosen and determine the order of the drums on a shared shaft. The proposed algorithm is an optimization of the Zygalski's sheets method. Before we present it, we will describe the mystery which is hidden in the sheets (author's observations). In order to read the encrypted messages we need the plugboard settings. Connections of the plugboard influence neither Zygalski's method (which is a well-known fact) nor the presented algorithm. The missing (original) algorithm solving the problem of the plugboard along with an algebraic description will appear very soon.

Keywords: Zygalski's sheets method, Enigma machine, ring settings, message settings

1. Introduction

The military Enigma machine was a portable electro-mechanical rotor encrypting machine used during the Second World War mainly by the German military and government services. The first intercepted German text, encrypted by using the Enigma, was broken by M. Rejewski in 1932. Since then the Polish cryptologists (M. Rejewski, J. Rozycki and H. Zygalski) systematically worked on: decoding ciphers, constantly modified manners of generating secret messages and modernized constructions of Enigma machines.

The algorithm presented below can be used to decode messages transmitted after 15 September 1938. That day the Germans withdrew the *initial drum settings* from tables of *daily key settings*. We assume (according to a knowledge of that time) that we know connections of all kinds of drums.

The algorithm relies on Zygalski's sheets. We propose an implementation of this method in cpp language. But we stress the point that no source (mainly historical

Advances in Computer Science Research, vol. 10, pp. 19-38, 2013.

sources) describes this method precisely. Historians make a lot of mistakes and they do not describe all the elements of this method. In this case accuracy is essential. Therefore, the given algorithm is a result of many tests in order to get the *initial ring settings*. The cryptologists might have done it in a different way during World War II.

The proposed algorithm was assembled on the basis of different facts which had been found in literature. These facts were completed with author's observations and ideas. The facts that are described clearly in literature are denoted in this paper by ($|_{[1!]}$). The other facts are presented in various ways in different books. To get the total algorithm we tested different possibilities and we chose the ones, which give a proper result. We denoted this ambiguously described information by ($|_{[2!]}$). We also used the facts which are described very vaguely in literature (non-concrete facts) and we had to complete them with our ideas (these ideas are denoted by ($|_{[3!]}$). Historians do not describe all the elements of the Zygalski's method. Thus, some facts are presented in this paper as author's observations. These facts and the author's own ideas are denoted by ($|_{[4!]}$).

We optimize the Zygalski's method. We browse fields which represent permutations on a first sheet only, more precisely, on its specified fragment. The fields which represent permutations with 1-cycles are remembered in a list. Next they are processed until the program returns a proper result. In section 7.6 we propose an additional improvement concerning fields which lie in an interfered area. We suppose, these fields gave the cryptologists a hard time when they were guessing the initial ring settings.

The Germans used different kinds of Enigma machines (also commercial), modified their constructions and changed the manner of generating secret messages. Therefore we describe the construction and the parameters of the machine for which we do cryptanalysis. We are interested in the M3 Enigma machine. Sections 2, 3 and 4 make up a brief survey of information taken from publications [4], [8], [5], [3], [6]. In section 5 we give some facts about 1-cycles which are essential to understand a presented method. Section 6 contains a description of sheets (author's observations). We received these observations with the help of a lot of tests. In section 7 we give an actual algorithm. By means of this algorithm we can generate the initial ring settings and guess the order of drums on the basis of a given set of messages intercepted after 15 September 1938.

A lot of facts which we present here must have been known by cryptologists during World War II. Since they did not disclose them, we had to study the secret of the Enigma machine from the beginning.

2. The construction of the military Enigma machine

The M3 Enigma machine is a combination of electrical and mechanical subsystems. This machine consists of an alphabetical 26-letter keyboard, a lampboard (set of 26 lights), a set of three rotating, encrypting disks (called *drums*) placed on a shared shaft, two fixed wheels: an *entry wheel* and a *reflector*, a *plugboard*, a battery, and a turning mechanism (used to turn one, two or three drums after pressing any key).

Figure 1 shows a simplified diagram of how the Enigma works. Pressing any key (e.g. the key `E`) causes the closure of an electric circuit. Then current flows through the various components in the present configuration of the circuit. It starts from the battery (3) and flows through a connection under the pressed key (2) to the plugboard (1). Next, it flows through the entry wheel (E), via three drums (R, M and L) to the reflector (B). The reflector inverts the signal (but using an entirely different route). From the reflector the current passes through drums L, M and R to the entry wheel (E), to the plugboard (1) and finally to an appropriate lamp (4) (which represents a letter different from `E`), causing it to light up (cf. [5], p.236).

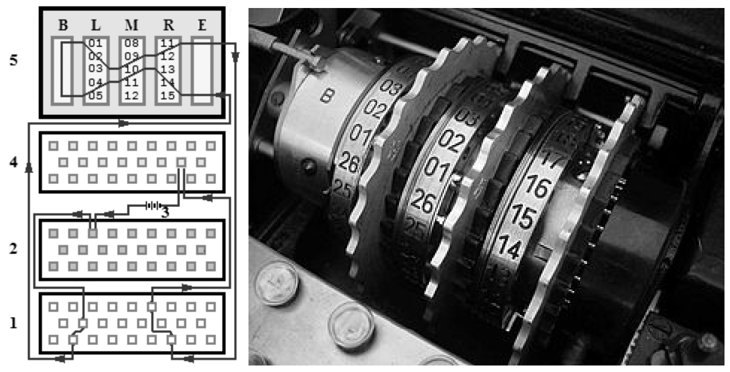


Fig. 1. I. The diagram presents how the military Enigma machine works (a hypothetical electric circuit). (1) the plugboard, (2) the keyboard, (3) the battery, (4) the lampboard, (5) disks: three drums (L, M, R), the entry wheel (E) and the reflector (B) (cf. [9]).
II. Assembled set of disks (three movable drums between the entry wheel and the reflector) (cf. [9]).

Each *drum* is in the shape of a wheel. Inside the drum there is a disk (called a *rotor*). On one side of each rotor there are 26 brass spring pins (arranged in a circle near the edge of the rotor). On the other side there is a corresponding number of flat electrical contacts. 26 contacts (and 26 pins), on either side, represent 26 letters of the alphabet. Each rotor hides 26 insulated wires. The wires connect the pins on one side to the contacts on the other in an established manner (different for various kinds of drums) (cf. [6], [4]). Since the drums are mounted side-by-side on the shaft, the pins of one drum touch the contacts of the neighbouring one, forming 26 fragments of an electrical circuit (cf. [6]). Each drum has a metal rotating *ring* applied to the rotor. Engraved numbers (on this ring) correspond to the 26 letters of the alphabet. On the edge of the rotor there is one distinguished place. The letter on the ring which is engraved opposite this position is treated as the *ring setting* (cf. [4]). Individual kinds of cipher drums differ not only by connections of pins and contacts but also by the so-called *turnover positions*. The turnover positions of five kinds of drums (denoted by I, II, III, IV, V) used by the German Land Forces and Air Force were as follows I - `Q`, II - `E`, III - `V`, IV - `J`, V - `Z` (cf. [4]).

Pressing any key of the Enigma causes one, two or three cipher drums to turn one twenty-sixth of a full rotation (before the electrical circuit closes). Thus the signal path changes constantly. More precisely, after pressing the key the right drum turns $1/26$ of a full rotation. When this drum reaches the turnover position the middle drum turns $1/26$ of a full rotation, too. When the second drum reaches the turnover position, the left drum turns $1/26$ of a full rotation (cf. [3], [6]). In this way each letter is coded with the drums in different positions.

The position of each movable drum is characterized by a number (engraved on a ring) which we can see through a window in the lid of the machine. The *rotor position* is described as the difference between the ring setting and the position of the drum.

Connections of the *entry wheel* in the military Enigma are represented by the identity permutation (cf. [4]). The *reflector* (reversal drum) connects the outputs of the last rotor into pairs, redirecting the current back through the drums using a different path. The reflector does not move (cf. [6]).

The *plugboard* contains 26 plugs and sockets (one pair for each letter of the alphabet). If the operator connects, by means of a cable, a plug of one pair with a socket of the second one (and inversely), then two letters are swapped both before and after the signal flows through the rotors. If a plug and a socket of a given letter are not connected with a cable to a plug and a secket of another letter, they are internally connected with each other (cf. [3], [4]).

In order to decode a text encrypted with the Enigma, the receiver had to set up his Enigma in the same way as the sender set up his during ciphering. The Enigma machine's initial settings were delivered to each military unit which used the Enigma, in the form of tables of the *daily key settings*. Daily key settings (until 15 September 1938) consisted of the *wheel order* (i.e. the choice of drums and the order in which they were fitted), the *ring settings*, plug connections (i.e. connections of the plugs in the plugboard) and initial positions of the drums (cf. [3]). [*Since 15 September 1938 the Germans, neither changed nor added anything to the machine, but changed the manner of announcing message settings. Starting with this date, the operator was* *forced to choose his own arbitrary three letters, which he placed in the headline of the message without ciphering* (these letters played a role of the initial positions of the drums). *Next, he set the drums to these letters and chose three other letters as the message settings. These letters, as before, after two-time coding, were placed at the beginning of the message, and then the drums were set to the message settings and the actual ciphering of the message began.*] (cf. [8]).

3. Mathematical analysis

Let **P** = {A, B, C,..., Z} be a set of possible plaintexts, and let **C** = **P** be a set of possible ciphertexts. The military Enigma machine codes text T by using a poly-alphabetic substitution cipher. Each letter $p \in \mathbf{P}$ of the message is transformed according to the following permutation (cf. [8])

$$\Lambda = SH(Q^zRQ^{-z})(Q^yMQ^{-y})(Q^xLQ^{-x})B(Q^xL^{-1}Q^{-x})(Q^yM^{-1}Q^{-y})(Q^zR^{-1}Q^{-z})H^{-1}S^{-1} \tag{1}$$

S – is a permutation describing the plugboard transformation,
B – is a permutation describing the reflector transformation, ($B = B^{-1}$),
L, M, R – are permutations describing transformations of the three cipher drums,
H – is a transformation of the entry wheel (H is an identity permutation),
Q – is a cyclic permutation mapping A to B, B to C to D, ..., Z to A,
x, y, z – are the positions of rotors before pressing any key (values from set $\mathbf{IP} = \{0,1,\dots,25\}$),
$x = (n(\alpha) - n(\delta))\%26$ for the left rotor,
$y = (n(\beta) - n(\varepsilon))\%26$ for the middle rotor,
$z = (n(\gamma) - n(\zeta))\%26$ for the right rotor (cf. [4]),
α, β, γ – positions of drums (left, middle, right) before pressing any key ($\alpha, \beta, \gamma \in \mathbf{P}$)
δ, ε, ζ – positions of rings (left, middle and right) (values from set **P**)
$n(\alpha)$ – the number of a letter α (value from set **IP**)

The Enigma codes the first 6 letters (meaning the double coded message settings) using the following permutations (cf. [8])

$$A = SH(Q^{z+1}RQ^{-(z+1)}Q^yMQ^{-y}Q^xLQ^{-x})B(Q^xL^{-1}Q^{-x}Q^yM^{-1}Q^{-y}Q^{z+1}R^{-1}Q^{-(z+1)})H^{-1}S^{-1}$$
$$B = SH(Q^{z+2}RQ^{-(z+2)}Q^yMQ^{-y}Q^xLQ^{-x})B(Q^xL^{-1}Q^{-x}Q^yM^{-1}Q^{-y}Q^{z+2}R^{-1}Q^{-(z+2)})H^{-1}S^{-1}$$
$$C = SH(Q^{z+3}RQ^{-(z+3)}Q^yMQ^{-y}Q^xLQ^{-x})B(Q^xL^{-1}Q^{-x}Q^yM^{-1}Q^{-y}Q^{z+3}R^{-1}Q^{-(z+3)})H^{-1}S^{-1}$$
$$D = SH(Q^{z+4}RQ^{-(z+4)}Q^yMQ^{-y}Q^xLQ^{-x})B(Q^xL^{-1}Q^{-x}Q^yM^{-1}Q^{-y}Q^{z+4}R^{-1}Q^{-(z+4)})H^{-1}S^{-1}$$
$$E = SH(Q^{z+5}RQ^{-(z+5)}Q^yMQ^{-y}Q^xLQ^{-x})B(Q^xL^{-1}Q^{-x}Q^yM^{-1}Q^{-y}Q^{z+5}R^{-1}Q^{-(z+5)})H^{-1}S^{-1}$$
$$F = SH(Q^{z+6}RQ^{-(z+6)}Q^yMQ^{-y}Q^xLQ^{-x})B(Q^xL^{-1}Q^{-x}Q^yM^{-1}Q^{-y}Q^{z+6}R^{-1}Q^{-(z+6)})H^{-1}S^{-1}$$

We calculate a product AD in the same way as M. Rejewski in [8].

4. Cryptanalysis (Zygalski's sheets)

Zygalski's sheets were a cryptologic technique used to decrypt messages ciphered on the German Enigma machines after 15 September 1938. This method was applied in order to guess which (three out of five) drums were put on the shaft, to calculate the order of the drums and to determine the ring settings (cf. [5]). Three three-letter fragments of the message were used for analysis, i.e. initial positions of the drums and double ciphered message settings (cf. [6]).

(1) `ZXV ZBB GIL` (2) `AFV AFB LFF` (3) `FUA RUF CHJ` (4) `QDV LXU WQH`

Cryptologists took advantage of the fact that identical letters on positions one and four (or two and five or three and six) in the double ciphered message settings mean that a permutation AD, (or BE or CF) contains 1-cycles (named *females*). They created a catalogue of all products AD (BE, CF) in which 1-cycles appear. Next they compared these products with products (containing 1-cycles) which were generated for initial drum positions of messages eavesdropped the same day (cf. [6]).

For each possible order[1] of drums they made a set of 3×26 sheets (one sheet for each letter of the alphabet, for each product AD, BE, CF) (cf. [5]). Each sheet contained a square divided into 51×51 small fields. On the top (and on the bottom) of the square the letters `A`$-$`Z`, `A`$-$`Y` were written down. These letters meant possible positions of the middle drum (cf. [4] $|_{[2!]}$). Similarly the sides of the square were described. These letters meant possible positions of the right drum (cf. [4] $|_{[2!]}$). Each field $[c][r]$ of the square represented a product AD (BE or CF). If the product contained 1-cycles, the field was perforated. [*When the sheets were superposed and moved in the proper sequence and the proper manner with respect to each other, in accordance with a strictly defined program, the number of visible apertures gradually decreased. And, if a sufficient quantity of data was available, then finally remained a single aperture, probably corresponding to the right case, that is, to the solution. . . . From the position of the aperture one could calculate the order of the drums and the setting of their rings*] (cf. [8]).

5. 1-cycles

Let us assume that we have at our disposal the following set of messages eavesdropped during the same day (i.e. received for the same daily key settings).

(1) `IHE BHX BZR` (2) `ZHH OAR OGI` (3) `ADU UXM JXS` (4) `EDL PCW ZMW`

[1] At the beginning Germans applied 3 kinds of drums. We have $3 * 2 * 1 = 6$ ordered sequences of 3 distinct elements of the set $\mathbf{D} = \{\text{I, II, III}\}$. Then they added two extra kinds of drums and a number of orders increased to $5 * 4 * 3 = 60$ (cf. [5]).

We can obtain each of the permutations *A*, *B*, *C*, *D*, *E*, *F* (with which the three-letter message settings are double ciphered) in the following way. We set up our Enigma machine in an identical way as the coder set his machine up during ciphering. Next we press in order all the letters of the alphabet without using the turning mechanism (cf. [6]). In case of the message (1) we receive

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A:	B	A	H	U	L	W	R	C	Y	N	Z	E	O	J	M	T	V	G	X	P	D	Q	F	S	I	K
D:	B	A	W	X	U	Q	V	O	K	P	I	T	Z	Y	H	J	F	S	R	L	E	G	C	D	N	M
AD:	A	B	O	E	T	C	S	W	N	Y	M	U	H	P	Z	L	G	V	D	J	X	F	Q	R	K	I

Permutations *A*, *D* consist of 13 transpositions. On the base of the double ciphered message settings BHX BZR we can conclude that both *A* and *D* permutations contain a transposition $(x, \mathtt{B})$, where x is a ciphered letter (different from B). In the case of the message (1) that is the transposition $(\mathtt{A}, \mathtt{B})$. Therefore the product *AD* assigns the letter B to the letter B and the letter A to the letter A. Thus there are females (1-cycles) in the permutation *AD*. Let us write down the permutation *AD* as a product of disjoint cycles

AD: (A)(B)(COZINPLUXRVF)(DETJYKMHWQGS)

We can see that the first letter of the message settings was the letter A. For the message (3) we calculate the product *BE*, because the repeated letter X occurs on the positions two and five. Below we present the permutation *BE* as a product of disjoint cycles

BE: (AGPQWUT)(BNK)(CJI)(DFESROL)(HY)(M)(VZ)(X)

We can guess that the second letter of the message settings was the letter M.

6. The sheets (author's observations)

Observations presented below we received for real connections of both drums and plugboard, which were used by Wehrmacht. Given examples were executed for drums: L = I, M = II, R = III, for the reflector B = UKW B and for the plugboard connections: $S = (\mathtt{A,G})(\mathtt{C,F})(\mathtt{K,O})(\mathtt{L,Y})(\mathtt{R,W})(\mathtt{S,Z})$. But one can generalize obtained facts. The sheets were generated for the products *AD*, but we shall show that one can observe similar results for permutations *BE* and *CF* (author's observations). Let us fix, that we shall treat the letters A, B, ..., Z of the Latin alphabet as the numbers from the set **IP** = $\{0, 1, \ldots, 25\}$.

The output contacts of cipher drums I, II, and III: (cf. [4])

I:	E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J
II:	A	J	D	K	S	I	R	U	X	B	L	H	W	T	M	C	Q	G	Z	N	P	Y	F	V	O	E
III:	B	D	F	H	J	L	C	P	R	T	X	V	Z	N	Y	E	I	W	G	A	K	M	U	S	Q	O

The turnover positions for selected drums: I - Q, II - E , III - V (cf. [4])

The reflector connections (cf. [4]):
UKW B: (AY)(BR)(CU)(DH)(EQ)(FS)(GL)(IP)(JX)(KN)(MO)(TZ)(VW)

In order to present the mystery hidden in the sheets, we represent them by means of two-dimensional matrices. We number them horizontally and vertically from 0 to 25 (for simplification). In each of the 60 sets (we can place 5 drums on 3 positions in $60 = 5 * 4 * 3$ manners) there are 26 sheets (one sheet for each letter). Let us denote by $[s][c][r]$ a square field $[c][r]$ on a sheet number s, where c is a number of a column and r is a number of a row $(|_{[2!]})$. The value s represents a number (or a letter) on the left drum, c is a number (or a letter) on the middle drum and r - a number (or a letter) on the right drum. Each field on any sheet corresponds to a product of permutations A and D and is coloured in black (for permutation AD with 1-cycles) or white (for permutation AD without females).

Example 6.1 Let us assume that we have the ring settings BBB. The field [B][A][A] represents a product AD for the drum settings BAA. This field is coloured in white, because a permutation AD does not contain any females. However, the field [B][B][A] represents a product AD for the drum settings BBA and it is coloured in black because AD contains 1-cycles.
AD: (ABZLQNIUHXY)(CEKFDPTGWJV)(MR)(OS) (for the drum settings BAA)
AD: (AZBFDNMXR)(CHG)(EYPILWOKJ)(Q)(STU)(V) (for the drum settings BBA)

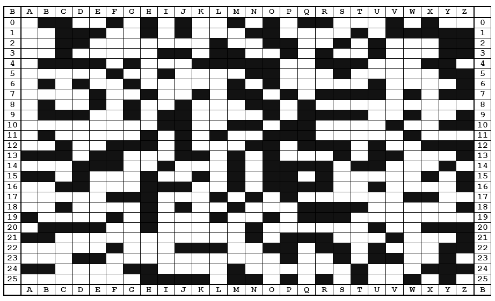


Fig. 2. The sheet B for the ring settings BBB.

Let us fix the meanings of the two phrases. By the *initial ring settings* (in short: IRS) we understand these ring settings for which the set of messages was ciphered. By the *starting ring settings* (in short: SRS) we define these ring settings $(|_{[3!]})$ for which we do calculations in order to obtain the initial ring settings.

The tests confirmed that it does not matter what set of sheets (i.e. for what SRS) we use to determine the initial ring settings $(|_{[3!]})$. In each set there are sheets which contain identical (with some exceptions defined below) appropriately moved representations of permutations AD with females ($|_{[4!]}$). For instance the sheet `B` from the set for SRS = `BBB` corresponds to the sheet `A` from the set for SRS = `AAA`.

Figure 3 shows that if we move the sheet `A` down one row and one column to the right then we obtain the sheet `B`. The discrepancies on each sheet (marked with grey colour) are caused by the turnover position (in short tp) of a right drum ($|_{[4!]}$). Since the order of drums was the same the whole day, the discrepancies on each sheet appeared in the same rows (i.e. in $(r-i)$-th rows, where $i = 0,1,2,3$, $tp_R = r$ denotes a turnover position of the right drum) ($|_{[4!]}$). Because for the drum III tp = `V`, so in the picture 3 we can see discrepancies in the rows `V`$-\{0,1,2,3\}$.

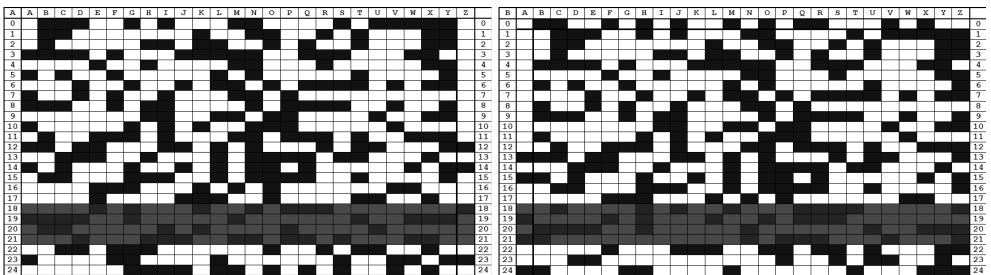


Fig. 3. The sheets `A` and `B` for SRS = `AAA` and SRS = `BBB` (appropriately).

Below we give three methods of the `Sheets` class. Let x be any sheet (from the set for SRS = $X_1X_2X_3$). The method `compareSh()` determines a sheet y (from the set for SRS = $Y_1Y_2Y_3$) which corresponds to x (cf. lines 13 – 15). We used the following formulas

$y = (x + Y_1 - X_1)\%26$ – the sheet (from the set for SRS = $Y_1Y_2Y_3$) which corresponds to the sheet x (from the set for SRS = $X_1X_2X_3$) ($|_{[4!]}$),

$c = (X_2 - Y_2)\%26$ – it is necessary to move the sheet y to the right by c columns relative to the sheet x ($|_{[4!]}$),

$r = (X_3 - Y_3)\%26$ – it is necessary to move the sheet y down by r rows relative to the sheet x ($|_{[4!]}$).

The method `compareSh()` additionally checks (for confirmation) that fields with females agree (after suitable shift) on both sheets. This method omits the fields from the interfered areas. Checking that the field $[j][i]$ on a sheet x or y (a sheet y is moved right c columns and down r rows relative to a sheet x) is in the interfered area takes place in lines (20 – 22). The method `pushSh()` generates a string `"(j,i)"` for a product AD with 1-cycles and an empty string `""` for a product without females. The method `ITC()` exchanges a number from the set **IP** $= \{0,1,\ldots,25\}$ for a suitable letter. The method `CTI()` does an opposite operation. `HE` is an object of the `Enigma` class. By tpR we signify the turnover position for the right drum. `moveD()` shifts drum settings `dr` for k presses of keys. `createAorD()` generates a permutation Λ (cf. the formula (1), section 3) for the ring settings SRS = `rs` and for the drum settings `dr`. If a permutation *AD* contains any females the method `Fem()` returns `true`. `codeLetter()` ciphers a letter described by a first parameter for the rotor settings determined by three next parameters.

```
String Sheets::pushSh(String rs, char x, int i, int j){
Perm *A1, *D1, *AD;
String dr="", s="", crr="";
crr=IntToStr(j)+","+IntToStr(i);
dr=x; dr+=ITC(j%26); dr+=ITC(i%26);
A1 = createAorD(rs,HE->moveD(dr,1));
D1 = createAorD(rs,HE->moveD(dr,4));
AD = A1->Prod(D1);
if(AD->Fem()){PF.push_back(crr); s="("+crr+")";}
delete A1; delete D1; delete AD;
return s;}
//----
bool Sheets::compareSh(String rs1, String rs2, char x){
char y = ITC((26+CTI(x)+CTI(rs2[1])-CTI(rs1[1]))%26);
int c = (26+CTI(rs1[2])-CTI(rs2[2]))%26;
int r = (26+CTI(rs1[3])-CTI(rs2[3]))%26;
cout « rs1 « "," « rs2 « "," « x « "," «y « "," « IntToStr(c) « "," «IntToStr(r);
String s1="", s2=""; bool b=true, b1, b2;
for(int i=0; i<=25; i++)
  for(int j=0; j<=25; j++){
     b1=cM(i,r,0)||cM(i,r,1)||cM(i,r,2)||cM(i,r,3);
     b2=cM(i,0,0)||cM(i,0,1)||cM(i,0,2)||cM(i,0,3);
     if(!(b1||b2)){
        s1=pushSh(rs1,x,(26+i+r)%26,(26+j+c)%26);
        s2=pushSh(rs2,y,i,j);
        if((s1==""&&s2!="")||(s1!=""&&s2==""))b=false;}}
return b;}
//----
Perm* Sheets::createAorD(String rs, String dr){
HE->setEnigma(rs,dr);
char* A = new char[27]; A[0]=26;
for(int i=1; i<=26; i++){
  A[i]=HE->codeLetter(pH[i],HE->Rt[1],HE->Rt[2],HE->Rt[3]);}
return new Perm(A);}
//----
String Enigma::moveD(String dr, int k){
String DR = dr, DRH="";
for(int i=1; i<=k; i++){DRH=DR;
  DR[3]=ITC((CTI(DR[3])+1)%26);
  if(DRH[3]==tpR){
     DR[2]=ITC((CTI(DR[2])+1)%26);
     if(DRH[2]==tpM)DR[1]=ITC((CTI(DR[1])+1)%26);}}
return DR;}
```

Example 6.2 Below we present a more complicated example. Let $X_1X_2X_3$ = `ZAE`, $Y_1Y_2Y_3$ = `CIW`, x = `A`. For the products *AD* the algorithm determined the following values y = `D`, $c = 18$, $r = 8$. It means that the sheet `D` (for SRS = `CIW`) is moved right 18 columns and down 8 rows relative to the sheet `A` (for SRS = `ZAE`) (cf. Fig. 4).

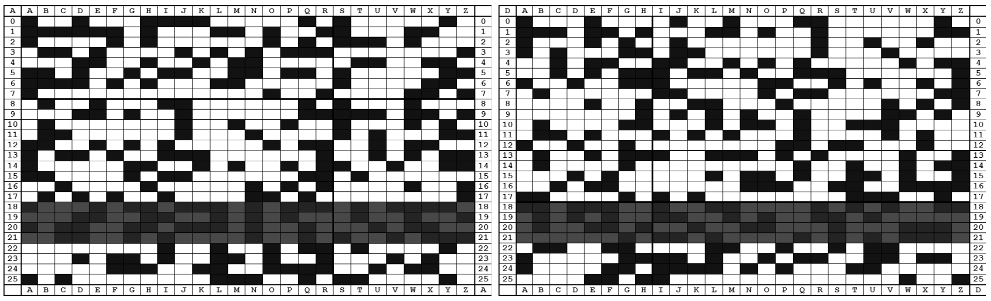


Fig. 4. The sheet A (for SRS = ZAE) and the sheet D (for SRS = CIW).

The author observed that in the case of the products *BE* and *CF* the discrepancies on sheets (caused by the turnover position of the right drum) take place in $(r-i)$-th rows, where $i = 0,1,2,3,4$ (for the products *BE*) and $i = 0,1,2,3,4,5$ (for the products *CF*) ($|_{[4!]}$). We can observe this fact in Figures 5, 6.

Example 6.3 Let $X_1X_2X_3$ = `ZAE`, $Y_1Y_2Y_3$ = `CIW`, x = `A`.

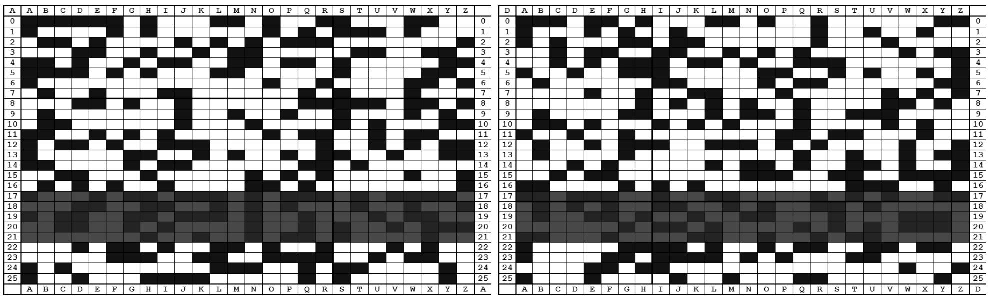


Fig. 5. For the products *BE* the algorithm determined the following values y = `D`, $c = 18$, $r = 8$.

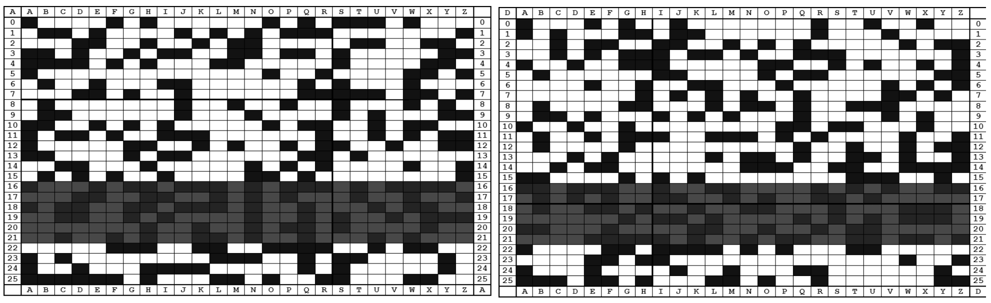


Fig. 6. For the products *CF* the algorithm determined the following values y = `D`, $c = 18$, $r = 8$.

7. The cryptanalysis (the optimized algorithm)

The algorithm presented below relies on the Zygalski's sheets method.

7.1 Initial activities

A cryptologist chooses 3 out of 5 drums and places them on the shaft in a selected order. We used the drums L = I, M = II, R = III for obtaining results presented in this paper. Next he establishes his own ring settings ($|_{[2!]}$), for example SRS = AAA. He eliminates messages in which initial positions of the drums cause a shift of the middle drum ($|_{[2!]}$) and he loads to the list M maximum 26^2 messages (i.e. 3 three-letter groups: the initial drum settings and double ciphered message settings). It is not necessary to load them in alphabetical order. It is possible to use several messages with the same first letter in the initial drum settings^3 ($|_{[4!]}$). We do not need to analyse all 26 messages. We usually get a result after calculations for $10-20$ messages.

7.2 The manner of moving the sheets (optimization - author's ideas)

The manner of moving the sheets given in literature (mainly by historians) is not precise enough. Moreover, we can find several different versions for this operation. Thus, we are not sure if the algorithm presented below does calculations in the exact same way as the cryptologists did it during World War II. We obtained the manner as a result of many checks of different hints which we had found.

Table 1. The set I (messages which we used in experiments)

(0) ABH YHJ YEU	(6) GBH QAC QIS	(12) MDH XLI XQI	(18) SDH UDK UTO	(24) YHB UPH UPN
(1) BJW ESX EQV	(7) HBE VNS VSZ	(13) NCF HKE HUL	(19) TDD QHQ QYN	(25) ZHH DMH DIM
(2) CDA LNZ LBJ	(8) IHE BHX BZR	(14) OFE RFI RVG	(20) UII RUN RHA	
(3) DBD YSM YKG	(9) JIA LFK LQI	(15) PEE UTI UZN	(21) VAH AWL AIR	
(4) EEE DNQ DDK	(10) KFF HCM HZS	(16) QHC DRH DOL	(22) WCF OFG OFV	
(5) FEC WMU WLZ	(11) LIC UBL UUW	(17) RGA VVP VFL	(23) XID DON DCP	

First we choose a sheet for a message (0). Let Z be this sheet. Next we put the sheet A (this sheet corresponds to the message (1)) on the sheet Z. We move the sheet A right c columns and down r rows relative to the sheet Z, where (cf. [6] $|_{[2!]}$)

$c = (\mathtt{B}-\mathtt{J})\%26 = 25, \qquad r = (\mathtt{H}-\mathtt{W})\%26 = 24.$

Next we move remaining sheets relative to the sheet (0) in an analogous manner, i.e., according to the following formulas ($|_{[2!]}$)

$c = (D_M^{(0)} - D_M^{(i)})\%26$ // $D_M^{(i)}$ - a setting of a drum M for a message (i),

$r = (D_R^{(0)} - D_R^{(i)})\%26$ // $D_R^{(i)}$ - a setting of a drum R for a message (i).

^{2} The value 26 was established for the sake of historical facts. In every set, during World War II, there was only one sheet for each letter. The program will accept any number of messages.

^{3} Cryptologists were not able to do it because they had only one sheet for each letter in each set.

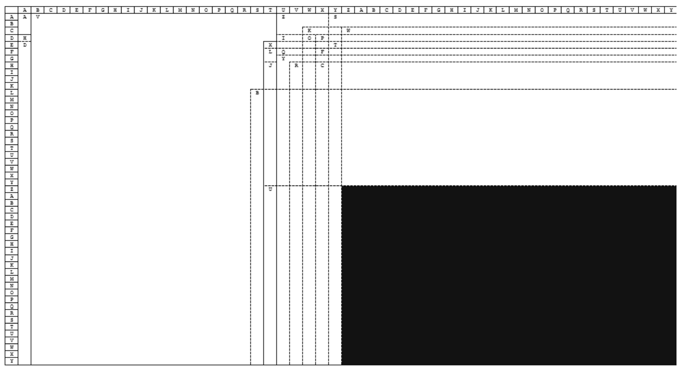


Fig. 7. The manner of moving the sheets.

In the above picture each sheet is signified by means of the first letter of the initial drum settings. We cannot see any of them because they are under other sheets. The rectangle coloured in black shows the area which is common for all used sheets. Within this rectangle we look for the representations of permutations *AD* with females. We are not interested in the remaining fragments of the sheets, that is we do not do any calculations for fields outside the rectangle ($|_{[4!]}$). We avoid the time-consuming calculations. The optimization of Zygalski's method relies on this idea. The method `rectangle()` of the `Sheets` class determines an area of a rectangle and writes down a result in the table `Rec[]` as follows. `Rec[1]` - the first column, `Rec[2]` - the last column, `Rec[3]` - the first row, `Rec[4]` - the last row. String `s1` is a field of an object of the `Message` class and represents the initial drum settings of a message (for the message (0) `s1` = `ABH`).

```
void Sheets::rectangle(){
Rec = new int[5];
int maxc=0, maxr=0, mc=0, mr=0;
for(unsigned int i=1; i<M.size(); i++){
  mc=(26+CTI(M[0]->s1[2])-CTI(M[i]->s1[2]))%26;
  mr=(26+CTI(M[0]->s1[3])-CTI(M[i]->s1[3]))%26;
  maxc=(mc>maxc)?mc:maxc;
  maxr=(mr>maxr)?mr:maxr;}
Rec[1]=maxc; Rec[2]=50; Rec[3]=maxr; Rec[4]=50;}
```

For messages from the set I the above algorithm determined the following rectangle `Rec[1]=25` , `Rec[2]=50` , `Rec[3]=25` , `Rec[4]=50` .

7.3 Determining products *AD* with 1-cycles (optimization - author's ideas)

Another optimization of Zygalski's method realizes the following idea. We check whether products *AD* contain 1-cycles or not only for permutations *AD* whose representations are within the determined rectangle and only for a message (0) ($|_{[4!]}$).

In the method `createSh()` we assign a sheet x to a message (0). Next we determine a product AD for each field $[x][j][i]$ which is situated in the rectangle. If AD contains 1-cycles, we place a pair `"(j,i)"` at the end of the list `PF` (look: the method `pushSh()`, section 6). Next (in the method `updateSh()`) we analyse remaining messages as follows. Let `M[i]` be a current message. We take from the list `PF` in turn each pair `"(c,r)"` (pairs `"(c,r)"` represent products AD with 1-cycles on a sheet (0)). And we check whether a field $[(c+D_M^{(i)}-D_M^{(0)})\%26][(r+D_R^{(i)}-D_R^{(0)})\%26]$ (on a sheet (i)), which covers a field $[x][c][r]$ (on a sheet (0)), also represents a product AD with 1-cycles. If this AD contains females, a pair `"(c,r)"` remains in the list `PF`, otherwise we remove it from `PF`.

If at the beginning we assigned a correct sheet to a message (0), then after analysing all the sheets, the list `PF` is empty or contains several fields. If the list is empty, it means we used improper drums or an order of drums is wrong. If the list `PF` contains several pairs, then these pairs usually represent the same product AD, that is we can obtain pairs (a,b), $(a+26,b)$, $(a,b+26)$, $(a+26,b+26)$.

```
void Sheets::updateSh(char* sh){
String crr="", dr="", s=""; char x;
int c, r, c1, c2, hlp, pf; Perm *A1, *D1, *AD;
for(unsigned int i=1; i<M.size(); i++){
  x = sh[i]; s="SHEET "; s+=x;
  pf=PF.size();
  for(int j=0; j<pf; j++){
      crr=PF[0]; PF.erase(&PF[0]);
      hlp = crr.Pos(',');
      c = StrToInt(crr.SubString(1,hlp-1));
      r = StrToInt(crr.SubString(hlp+1,crr.Length()-1));
      c1 = (26+c+CTI(M[i]->s1[2])-CTI(M[0]->s1[2]))%26;
      r1 = (26+r+CTI(M[i]->s1[3])-CTI(M[0]->s1[3]))%26;
      dr=x; dr+=ITC(c1); dr+=ITC(r1);
      A1 = createAorD(HE->Rs,HE->moveD(dr,1));
      D1 = createAorD(HE->Rs,HE->moveD(dr,4));
      AD = A1->Prod(D1);
      if(AD->Fem()){PF.push_back(crr); s+="("+crr+")";}
      delete A1; delete D1; delete AD;}
  cout « s;}
PF.clear();}
//----
void Sheets::createSh(char x){
String s="SHEET "; s+=x;
for(int i=Rec[3]; i<=Rec[4]; i++)
  for(int j=Rec[1]; j<=Rec[2]; j++)
      s+=pushSh(HE->Rs,x,i,j);
cout « s;}
```

Below we present a result of the calculations for the set I of messages.

```
SHEET Z(27,25)(31,25)(34,25)(35,25)(37,25)(40,25)(41,25)(42,25)(49,25)(50,25)(27,26)(28,26)(32,26)(39,26)(40,26)(43,26)
```

```
(45,26) (46,26) (47,26) (48,26) (50,26) (28,27) (35,27) (36,27) (37,27) (38,27) (43,27) (44,27) (28,28) (30,28) (33,28) (34,28) (36,28)
(37,28) (40,28) (43,28) (44,28) (45,28) (47,28) (48,28) (25,29) (27,29) (28,29) (32,29) (33,29) (34,29) (35,29) (36,29) (42,29) (48,29)
(49,29) (26,30) (35,30) (38,30) (43,30) (48,30) (49,30) (50,30) (29,31) (30,31) (31,31) (37,31) (39,31) (40,31) (44,31) (45,31) (46,31)
(48,31) (50,31) (25,32) (26,32) (29,32) (33,32) (34,32) (35,32) (37,32) (39,32) (40,32) (42,32) (45,32) (48,32) (30,33) (31,33) (34,33)
(35,33) (38,33) (39,33) (44,33) (45,33) (30,34) (31,34) (39,34) (40,34) (41,34) (46,34) (48,34) (49,34) (26,35) (29,35) (31,35) (32,35)
(33,35) (35,35) (37,35) (39,35) (45,35) (28,36) (29,36) (32,36) (36,36) (38,36) (42,36) (44,36) (50,36) (28,37) (30,37) (35,37) (36,37)
(40,37) (41,37) (42,37) (43,37) (44,37) (47,37) (25,38) (27,38) (41,38) (43,38) (49,38) (26,39) (27,39) (29,39) (33,39) (34,39) (38,39)
(39,39) (40,39) (41,39) (43,39) (44,39) (47,39) (49,39) (25,40) (26,40) (27,40) (31,40) (33,40) (34,40) (47,40) (49,40) (50,40) (29,41)
(30,41) (31,41) (32,41) (33,41) (34,41) (40,41) (42,41) (49,41) (29,42) (33,42) (34,42) (38,42) (40,42) (41,42) (43,42) (44,42) (49,42)
(50,42) (25,43) (28,43) (30,43) (31,43) (32,43) (36,43) (40,43) (42,43) (50,43) (28,44) (33,44) (34,44) (35,44) (36,44) (37,44) (40,44)
(41,44) (42,44) (43,44) (25,45) (29,45) (31,45) (34,45) (36,45) (40,45) (46,45) (47,45) (48,45) (50,45) (25,46) (26,46) (27,46) (29,46)
(34,46) (36,46) (37,46) (38,46) (43,46) (44,46) (45,46) (48,46) (50,46) (26,47) (31,47) (34,47) (36,47) (38,47) (42,47) (43,47) (45,47)
(46,47) (47,47) (48,47) (49,47) (26,48) (31,48) (34,48) (35,48) (38,48) (39,48) (43,48) (49,48) (25,49) (29,49) (39,49) (40,49) (42,49)
(44,49) (46,49) (47,49) (26,50) (27,50) (28,50) (39,50) (40,50) (41,50) (47,50) (50,50)
SHEET A(27,25) (31,25) (34,25) (40,25) (42,25) (50,25) (32,26) (45,26) (46,26) (50,26) (28,27) (28,28) (37,28) (48,28) (28,29) (33,29)
(34,29) (35,29) (48,29) (50,30) (45,31) (46,31) (25,32) (26,32) (33,32) (34,32) (40,32) (35,33) (38,33) (39,34) (40,34) (49,34) (26,35)
(29,35) (32,35) (39,35) (29,36) (38,36) (44,36) (50,36) (30,37) (36,37) (40,37) (41,37) (42,37) (47,37) (41,38) (26,39) (29,39) (34,39)
(41,39) (27,40) (31,40) (34,40) (47,40) (49,40) (30,41) (31,41) (42,41) (29,42) (44,42) (49,42) (30,43) (32,43) (36,43) (42,43) (50,43)
(33,44) (25,45) (31,45) (34,45) (36,45) (46,45) (25,46) (26,46) (29,46) (38,46) (26,47) (31,47) (26,48) (31,48) (34,48) (35,48) (49,48)
(29,49) (42,49) (44,49) (47,49) (41,50) (47,50)
SHEET B(34,25) (42,25) (50,25) (50,26) (28,27) (33,29) (34,29) (48,29) (50,30) (46,31) (25,32) (26,32) (33,32) (40,32) (39,34) (40,34)
(49,34) (26,35) (29,35) (38,36) (50,36) (40,37) (42,37) (41,38) (49,40) (30,41) (29,42) (44,42) (49,42) (30,43) (36,43) (42,43) (50,43)
(33,44) (31,45) (46,45) (25,46) (29,46) (38,46) (26,48) (42,49) (41,50)
SHEET C(50,26) (33,29) (48,29) (50,30) (26,32) (49,34) (50,36) (42,37) (36,43) (46,45) (25,46) (38,46) (26,48)
SHEET D(33,29) (26,32) (50,36) (42,37) (26,48)
SHEET E(26,32) (50,36)
SHEET F(26,32)
SHEET G(26,32)
SHEET H(26,32)  SHEET I(26,32)  SHEET J(26,32)  SHEET K(26,32)  SHEET L(26,32)  SHEET M(26,32)  SHEET N(26,32)  SHEET O(26,32)
SHEET P(26,32)  SHEET Q(26,32)  SHEET R(26,32)  SHEET S(26,32)  SHEET T(26,32)  SHEET U(26,32)  SHEET V(26,32)  SHEET W(26,32)
SHEET X(26,32)  SHEET Y(26,32)
```

7.4 Determining the initial ring settings

Let us assume that we assigned the sheet Z to a message (0) and as a result we obtained the pair $(c,r) = (26,32)$. This means that each field (on each sheet) which covers the field [Z][26][32] = [Z][0][6] = [Z][A][G] (on the sheet Z) represents a product AD with females. Then we determine the initial ring settings using formulas given below (cf. [4] $|_{[3!]}$)

$$\mathrm{IRS}_L = (\mathrm{SRS}_L + (D_L^{(0)} - s))\%26,$$
$$\mathrm{IRS}_M = (\mathrm{SRS}_M + (D_M^{(0)} - c))\%26,$$
$$\mathrm{IRS}_R = (\mathrm{SRS}_R + (D_R^{(0)} - r))\%26.$$

We can treat $[D_L^{(0)} - s, D_M^{(0)} - c, D_R^{(0)} - r]$ as a vector where SRS is an initial point and IRS is a terminal point ($|_{[3!]}$). For the given set I of messages and for SRS = AAA we calculate the initial ring settings as follows.

$$\mathrm{IRS}_L = (\mathtt{A}+(\mathtt{A}-\mathtt{Z}))\%26 = 1 = \mathtt{B},$$
$$\mathrm{IRS}_M = (\mathtt{A}+(\mathtt{B}-0))\%26 = 1 = \mathtt{B},$$
$$\mathrm{IRS}_R = (\mathtt{A}+(\mathtt{H}-6))\%26 = 1 = \mathtt{B}.$$

We obtained IRS = BBB and in fact we generated the set I of messages for the initial ring settings BBB .

7.5 The sequence of hypotheses concerning a sheet (0)

Next problem lies in the fact that we do not know which sheet we ought to assign to a message (0). Historians write very generally about this problem. For instance [*When the sheets were superposed and moved in the proper sequence and the proper manner* ...] (cf. [8]). In the program we used a very general suggestion that cryptologists made hypotheses concerning sheets (cf. [5]). Next we observed, on the basis of many tests, that we can determine *i*-th sheet (for a message (*i*)) from the formula $(|_{[4!]})$.

$$\texttt{sheet[i]} = (\mathrm{SRS}_L + D_L^{(i)} + j)\%26 \quad \text{for } i = 0, 1, \ldots, \texttt{numberOfMessages} - 1$$

j $(j = 0, 1, \ldots, 25)$ denotes the number of the realized hypothesis. That is, in the first step we superpose sheets on a `sheet[0]` for $j = 0$, next we superpose sheets on a `sheet[0]` for $j = 1$, etc. The method `IRS()` solves the problem of the initial ring settings. We make all possible hypotheses (26 hypotheses) concerning assigning a first sheet to a message (0).

```
void Sheets::IRS(){
    // loading messages to the list M
char* sheet = new char[27];
rectangle();
for(int j=0; j<=25; j++){
  for(unsigned int i=0;i<M.size(); i++)
     sheet[i]=ITC((CTI(HE->Rs[1])+CTI(M[i]->s1[1])+j)%26);
  createSh(sheet[0]);
  updateSh(sheet);}}
```

For the given set I of messages we obtained a proper result for a hypothesis number $j = 25$. Then the program assigned the sheet `Z` to a message (0).

7.6 The discrepancies (optimization - author's ideas)

In the case of the set I of messages we received a proper result (i.e. the pair $(26, 32)$) when we assigned the sheet `Z` to the message (0). Let us notice, that each field (on each sheet with the exception of the sheet `A`) which covers the field $[\mathtt{Z}][26][32]$ is outside an interfered area. In the case of the sheet `A` (for a message (1)) the field $[\mathtt{Z}][26][32]$ is covered by the field $[\mathtt{A}][8][21]$. The field $[\mathtt{A}][8][21]$ corresponds to the drum settings `AIV`. Let us notice that the turnover position of the right drum is $tpr = V = 21$. Thus $[\mathtt{A}][8][21]$ is located inside the interfered area. But this field represents a product with females (it is coloured in black). The algorithm given above returned a proper solution, because the discrepancies did not influence the result.

Case one. Let us assume that a field $[s][c][r]$ (on a sheet s) covers the field $[\mathtt{Z}][26][32]$, $[c][r]$ is inside an interfered area and does not represent a product AD with 1-cycles.

Then a correct result will be removed from the list `PF` during analysing a sheet s and finally the list `PF` will be empty. We shall interpret this fact as an improper order of drums.

How did cryptologists solve this problem? Historians write that cryptologists before beginning of the analysis of messages, eliminated some of them in order to avoid the situation described above. For each order of drums they had to eliminate another set of messages. We can find different formulas describing which messages ought to be removed.

The improvement of Zygalski's method is a realization of the following idea. Let us assume that we have case one. Before determining a product *AD* for a field $[s][c][r]$ we check whether this field is inside an interfered area or not. If it lies outside, we check a product *AD* like previously, otherwise we leave a pair `"(c,r)"` in the list `PF` (we treat this field as a representation of a product *AD* with 1-cycles) ($|_{[4!]}$). After analysing all messages we can get additional, wrong results, but we shall avoid a situation we omit a proper order of drums. If we obtain some results, it is not difficult to verify that a given one is wrong. We can analyse an additional message, for instance.

```
void Sheets::updateSh(char* sh){
String crr="", dr="", s=""; char x;
int c, r, c1, r1, pf, hlp; Perm *A1, *D1, *AD;
for(unsigned int i=1; i<M.size(); i++){
  x =sh[i]; s="SHEET "; s+=x;
  pf=PF.size();
  for(int j=0; j<pf; j++){
      crr=PF[0]; PF.erase(&PF[0]);
      hlp = crr.Pos(',');
      c = StrToInt(crr.SubString(1,hlp-1));
      r = StrToInt(crr.SubString(hlp+1,crr.Length()-1));
      c1 = (26+c+CTI(M[i]->s1[2])-CTI(M[0]->s1[2]))%26;
      r1 = (26+r+CTI(M[i]->s1[3])-CTI(M[0]->s1[3]))%26;
      dr=x; dr+=ITC(c1); dr+=ITC(r1);
      if(cM(r1,0,0)||cM(r1,0,1)||cM(r1,0,2)||cM(r1,0,3)){
        PF.push_back(crr); s+="("+crr+")";}
      else{
        A1 = createAorD(HE->Rs,HE->moveD(dr,1));
        D1 = createAorD(HE->Rs,HE->moveD(dr,4));
        AD = A1->Prod(D1);
        if(AD->Fem()){PF.push_back(crr); s+="("+crr+")";}
        delete A1; delete D1; delete AD;}}
  cout « s;}
PF.clear();}
//----
String Sheets::pushSh(String rs, char x, int i, int j){
Perm *A1, *D1, *AD;
String dr="", s="", crr="";
crr=IntToStr(j)+","+IntToStr(i);
(29) dr=x; dr+=ITC(j%26); dr+=ITC(i%26);
(30) if(cM(i,0,0)||cM(i,0,1)||cM(i,0,2)||cM(i,0,3)){
(31)   PF.push_back(crr); s="("+crr+")";}
(32) else{
(33)   A1 = createAorD(rs,HE->moveD(dr,1));
(34)   D1 = createAorD(rs,HE->moveD(dr,4));
(35)   AD = A1->Prod(D1);
(36)   if(AD->Fem()){PF.push_back(crr); s="("+crr+")";}
(37)   delete A1; delete D1; delete AD;}
(38) return s;}
```

We do not change a body of the method `createSh()` . In the method `pushSh()` we added lines $(30-31)$ in which we check whether a field $[x][j][i]$ (on a sheet (0)) is located in an interfered area or not. If it is, we add a pair `"(j,i)"` to the list `PF`. In the other case we proceed like in section 6, that is we add a pair `"(j,i)"` if only *AD* contains any females. With the other messages we proceed in the method `updateSh()` in a similar way. We added lines $(15-16)$ in which we check whether a field $[x][c1][r1]$ (on a sheet (i)) is located in an interfered area or not. If it is, we leave a pair `"(c,r)"` in the list `PF`. Otherwise we proceed like in section 6, that is we leave a pair `"(c,r)"` in the list `PF` if only *AD* contains any females. Otherwise, we remove it.

Table 2. The set II (messages obtained for IRS = EHM)

(0) EFE QML QTU	(6) DRE WKD WWY	(12) UBA BGS BOF	(18) OBE FOE FDG	(24) VCM KWY KXG
(1) HOP PXP PBW	(7) NMD WLU WMR	(13) GOZ IRX IOP	(19) QRZ YTJ YVW	(25) XAB HIH HWQ
(2) LRM ZAX ZVV	(8) CEW MTY MGZ	(14) IRB UUT USU	(20) SCH YQU YGW	
(3) MAX ERG EKR	(9) PBW WYY WHM	(15) JAD FOB FFT	(21) WBC ZQW ZVD	
(4) BTX FIJ FOI	(10) RCI XCZ XSE	(16) QDZ QQH QBG	(22) YDF XZZ XXA	
(5) AAY VDN VQW	(11) TCE XZB XCA	(17) KBW VHT VEB	(23) ZDL IQO IOM	

Example 7.1 Let us analyse the set II of messages by means of the improved algorithm (for SRS = AAA).

The program assigned the following sheets to the given above messages:
A, D, H, I, X, W, Z, J, Y, L, N, P, Q, C, E, F, M, G, K, M, O, S, U, V, R, T.
Finally, as a result we obtained two pairs $(24,44)$, $(50,44)$. This means that each field (on each sheet) which covers the field [A][24][44] (or [A][50][44]) (on the sheet A) either represents a product *AD* with females or it is located inside the interfered area. Let us calculate IRS.

$\mathrm{IRS}_L = (\mathtt{A}+(\mathtt{E}-\mathtt{A}))\%26 = 4 = \mathtt{E}$,
$\mathrm{IRS}_M = (\mathtt{A}+(\mathtt{F}-24))\%26 = 7 = \mathtt{H}$,
$\mathrm{IRS}_R = (\mathtt{A}+(\mathtt{E}-18))\%26 = 12 = \mathtt{M}$.

Thus IRS = EHM. Let us notice, we analysed messages (16) and (19). The first letter of the initial drum settings in both cases is the same (the letter Q). We assign the same sheet M to these messages . The cryptologists did not have this possibility. They had only one sheet for each letter.

8. The plugboard settings

We know the initial ring settings. But it does not mean that we can read messages. We still have to find the plugboard settings. It is a well-known fact that the connections of the plugboard do not influence the shapes of products AD, BE, CF (cf. [6]). That is, if a product AD contains 1-cycles for the plugboard settings $S1$, then it contains 1-cycles for the plugboard settings $S2$. Therefore, the plugboard settings do not have any influence on the work of the above algorithm.

Example 8.1 We generated sets I and II of messages and executed all the above calculations for connections of the plugboard $S = $ (A,G)(C,F)(K,O)(L,Y)(R,W)(S,Z). Next we set up the plugboard as follows $S1 = $ (C,O)(D,I)(F,R)(H,U)(J,W)(L,S)(T,X) and we analysed the set II of messages once again. As the reader can guess, we obtained the same result.

References

[1] Baginski, C.: *Introduction to Group Theory*, SCRIPT, Warsaw, 2012.
[2] Brynski, M.: *Elements of the Galois Theory*, Alfa Publishing House, Warsaw, 1985.
[3] Garlinski J.: *Enigma. Mystery of the Second World War*, University of Maria Curie-Sklodowska Publishing House, Lublin, 1989.
[4] Gay K.: *The Enigma Cypher. The Method of Breaking*, Communication and Connection Publishing House, Warsaw, 1989.
[5] Grajek M.: *Enigma. Closer to the truth*, REBIS Publishing House, Poznan, 2007.
[6] Gralewski L.: *Breaking of Enigma. History of Marian Rejewski*, Adam Marszalek Publishing House, Torun, 2005.
[7] Mostowski A., Stark M.: *Elements of Higher Algebra*, PWN, Warsaw, 1970.
[8] Rejewski M.: How did Polish Mathematicians Decipher the Enigma, *Polish Mathematics Association Yearbooks*, Series 2nd: Mathematical News XXIII (1980).
[9] http://pl.wikipedia.org/wiki/Enigma.

Anna Borowska

KRYPTOANALIZA SZYFRU ENIGMY

Streszczenie Tematem pracy jest kryptoanaliza Enigmy wojskowej używanej przez siły zbrojne oraz inne służby państwowe Niemiec podczas II wojny światowej. Jesteśmy zainteresowani problemem dekodowania zaszyfrowanych depesz transmitowanych po 15 września 1983 roku. Prezentujemy pełny algorytm służący do generowania ustawień pierścieni, odgadywania które rodzaje bębenków zostały wybrane i wyznaczania ich porządku na wspólnej osi. Proponowany algorytm jest uzupełnieniem i optymalizacją metody płacht Zygalskiego. W pracy opisane są istotne własności płacht wynikające z konstrukcji maszyny i teorii permutacji (spostrzeźenia autora). Do czytania depesz zaszyfrowanych za pomocą Enigmy potrzebne są jeszcze ustawienia łącznicy wtyczkowej. Połączenia łącznicy nie mają wpływu ani na metodę Zygalskiego (znany fakt) ani na przedstawiony algorytm. Brakujący (oryginalny) algorytm rozwiązujący problem łącznicy (wraz z algebraicznym opisem) pojawi się niebawem.

Słowa kluczowe: metoda płacht Zygalskiego, ustawienia pierścieni, ustawienia depeszy

APPLICATION OF THE RECURSIVE FEATURE ELIMINATION AND THE RELAXED LINEAR SEPARABILITY FEATURE SELECTION ALGORITHMS TO GENE EXPRESSION DATA ANALYSIS

Joanna Gościk, Tomasz Łukaszuk

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Most of the commonly known feature selection methods focus on selecting appropriate predictors for image recognition or generally on data mining issues. In this paper we present a comparison between widely used Recursive Feature Elimination (RFE) with resampling method and the Relaxed Linear Separability (RLS) approach with application to the analysis of the data sets resulting from gene expression experiments. Different types of classification algorithms such as K-Nearest Neighbours (KNN), Support Vector Machines (SVM) and Random Forests (RF) are exploited and compared in terms of classification accuracy with optimal set of genes treated as predictors selected by either the RFE or the RLS approaches. Ten-fold cross-validation was used to determine classification accuracy.

Keywords: gene expression analysis, feature selection, classification

1. Introduction

Gene expression data analysis has become a very important topic nowadays due to the need of understanding mechanisms underlying disease development, phenotypic differences between groups of subjects/patients or the influence of environmental factors on functioning of a particular organism. The existence of variety of vendors (e.g. Affymetrix, Agilent, Illumina), either providing standard or custom-made platforms, enabling measuring gene expression causes a variety of data formats produced with the use of different techniques. As a result, many tools have been created to deal with data preprocessing (e.g. normalization across arrays, background signal correction) – a fist and an inevitable step that must be performed in order to analyse gene expression data. Incorporating a specificity of an experiment's design into the analysis is also a fundamental issue when one wants to conduct the analysis. This specificity includes the occurrence of technical and biological replicates since those cannot be treated the same way. An example of a software making possible data preprocessing and discovery of genes significantly differentially expressed in groups under investigation is Limma package [1] freely available as a part of the Bioconductor project [2]. The approach proposed in this package along with many other solutions enables identifying a set of under or over-expressed genes by performing multiple testing (one test for each gene) but gives no information about the discriminative power of this set – whether the set of chosen genes can be treated as a set of features providing good classification accuracy. In this paper we present a comparison between two methods of feature selection: the Recursive Feature Elimination (RFE) with resampling [3] technique and the Relaxed Linear Separability (RLS) [4] approach. Obtained discriminative set of genes is validated using 10-fold cross validation with the use of the following classifiers: K-Nearest Neighbours (KNN), Support Vector Machines (SVM) and Random Forests (RF) and the classification accuracy is reported.

2. Data acquisition and characteristic

All samples composing an example data set used for all calculations were downloaded from Gene Expression Omnibus database [5]. It is a public functional data repository which contains detailed information about platforms used for the experiment, series of experiments conducted with the use of a specific platform and samples gathered within a concrete study (series). GEO database provides users with an advanced search engine allowing finding e.g. experiments carried out with a desired technique as gene expression profiling by array (microarray experiments), experiments carried out with a desired platform - especially useful when one wants to compare own results with another obtained with the same equipment or experiments relating to a specific organism.

Forty eight samples of series GSE27144 available online since June 15th, 2012 were exploited in our study. Technique used in this experiment was Real-Time PCR (qPCR) and samples concerned homo sapiens. Available genetic data, extended by clinical features as BMI was analysed and published [6]. Authors investigated whole saliva as a source of biomarkers to discriminate subjects who have and have not undergone severe and life-threating difficulties. Objective of the research was to evaluate an influence of severe life events and difficulties on: (1) clinical characteristics, (2) salivary analyte metrics and (3) salivary gene expression. Genetic part of the data set related to genes that have previously revealed differential expression in genome-wide analysis of adults experiencing various levels of a chronic stress and only this part of the data is taken under consideration in our study. Data was gathered within the framework of Oregon Youth Substance Use Project (OYSUP) [7] and divided into two separate groups: low level of stress (L) and high level of stress (H), each consisting of 24 subjects. Every sample contained expression levels for 38 genes represented by normalized C_T value using $\Delta\Delta C_T$ method [8].

2.1 Brief introduction to Real-Time PCR technology

Polymerase chain reaction (PCR) is a method that allows exponential amplification of short DNA sequences within a longer double stranded DNA molecule which consist of subsequent cycles theoretically resulting in doubling the DNA amount when compared to the previous cycle - an exponential reaction. The reaction finally tails off and reaches a plateau. For the most part, this method was developed to enable measuring differences in gene expression. To introduce basic terms related to RT-PCR let us assume that we have two kinds of cells: drug treated (experimental lane) and non-treated (control lane) and two signal intensities for a specific gene - one for each sample. If the amount of signal in the experimental sample was 100 times greater when compared to the control sample we could say that expression of the gene has increased 100-fold in the experimental cells but it could also mean that we have 100 times more RNA in the experimental lane - in other words we have a loading artefact. To prevent this kind of artefacts housekeeping genes were introduced. Housekeeping gene is a kind of gene, that its expression does not change depending on the conditions (e.g. it remains the same in different kinds of tissues). We could call these genes loading control. Let us say that the experiment showed that for our housekeeping gene there is twice as much DNA in the experimental lane comparing to control sample. This means that the real change in gene under investigation expression is equal to 100/2=50 fold. The actual fold change is given by the Equation 1.

$$\text{Fold change} = \frac{\text{Fold change of a target gene}}{\text{Fold change of a reference gene}} \tag{1}$$

The goal of an RT-PCR experiment is to measure level of expression of a certain gene from a specific sample. This measurement is expressed in Cycled to Threshold (C_T) - a relative value representing the cycle number at which the amount of amplified DNA reaches the threshold level - exceeds background intensity. An important property of C_T is that it is reversely proportional to the DNA quantity present in a sample expressed on a logarithmic scale (the amount of DNA doubles with every cycle). In order to make different C_T values comparable across multiple samples normalization is applied. There are many methods of normalization available, but in this article we will focus on $\Delta\Delta C_T$ method [8]. A detailed description of the process can be found in the cited article so we will just mention the main formula which is given below.

$$\Delta\Delta C_T = \Delta C_T\,(\text{experimental}) - \Delta C_T\,(\text{reference}) \tag{2}$$

Where ΔC_T (experimental) refers to the C_T value for the gene in an experimental lane normalized against housekeeping gene and ΔC_T (reference) refers to the C_T value for that gene in an control lane again normalized against housekeeping gene. The amount of target normalized against housekeeping gene and relative to the control is given by the Equation 3.

$$\text{Amount of target} = 2^{-\Delta\Delta C_T} \tag{3}$$

Quantities obtained with the use of Equation 3 can then be compared across samples.

2.2 Missing values imputation

Source data set, after combining all samples, contained missing values. Missing values characteristics for each class are given in Table 1. where columns represent genes and rows represent appropriate metrics.

Table 1. Missing values occurence in classes with low (L) and high (H) levels of stress.

		ADAR	**ADORA2A**	**BTN3A3**	**C7orf68**	**CD9**	**CSF1R**	**CX3CR1**	**CYLD**	**DPYD**	**FGL2**	**FOLR3**
Class L	N observed	24	24	22	23	24	23	23	23	23	24	22
	Missing	0%	0%	8%	4%	0%	4%	4%	4%	4%	0%	8%
Class H	N observed	23	24	16	24	24	17	18	21	22	20	15
	Missing	4%	0%	33%	0%	0%	29%	25%	12%	8%	17%	37%
		FOSB	**GADD45B**	**GALC**	**GBP1**	**GNA15**	**GORASP2**	**HLA-DQB1**	**HSPA1B**	**IL8**	**MAFF**	**NAIP**
Class L	N observed	24	24	23	23	24	23	12	24	24	24	22
	Missing	0%	0%	4%	4%	0%	4%	50%	0%	0%	0%	8%
Class H	N observed	21	24	16	17	24	22	10	24	24	24	17
	Missing	12%	0%	33%	29%	0%	8%	58%	0%	0%	0%	29%
		NDUFB7	**NGLY1**	**NRG1**	**NSF**	**PUM2**	**RAB27A**	**RPA1**	**SEC24A**	**SERPINB2**	**SLC35A1**	**SLC7A5**
Class L	N observed	24	23	23	22	23	24	23	24	24	22	24
	Missing	0%	4%	4%	8%	4%	0%	4%	0%	0%	8%	0%
Class H	N observed	23	21	16	18	22	19	19	23	19	16	23
	Missing	4%	12%	33%	25%	8%	21%	21%	4%	21%	33%	4%
		STAT1	**STX7**	**THBS1**	**VEGFA**	**WDR7**						
Class L	N observed	24	24	23	24	23						
	Missing	0%	0%	4%	0%	4%						
Class H	N observed	23	23	24	23	18						
	Missing	4%	4%	0%	4%	25%						

Package impute [9] from the Bioconductor project [2] was used to complete the data set. Method implemented in this package uses K-Nearest Neighbours approach to impute missing expression data provided that the data set exploited in the computation is in matrix form with rows corresponding to genes and columns corresponding to samples. For each gene with missing expression levels K nearest neighbours are found using the Euclidean metric with restriction to samples for which expression level for that gene is defined. Descriptive statistics including mean and its standard error before and after missing values imputation are presented in Table 2 and Table 3 respectively.

Table 2. Descriptive statistics for both Low (L) and High (H) levels of stress for data set before imputing missing values.

		ADAR	**ADORA2A**	**BTN3A3**	**C7orf68**	**CD9**	**CSF1R**	**CX3CR1**	**CYLD**	**DPYD**	**FGL2**	**FOLR3**
Class L	Mean	1.03	1.13	1.33	1.21	1.04	1.49	1.27	1.12	1.22	1.05	1.31
	Std. Err.	0.01	0.01	0.01	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.02
Class H	Mean	1.02	1.12	1.34	1.18	1.08	1.50	1.27	1.15	1.18	1.07	1.33
	Std. Err.	0.01	0.02	0.01	0.02	0.02	0.02	0.01	0.01	0.01	0.02	0.01
		FOSB	**GADD45B**	**GALC**	**GBP1**	**GNA15**	**GORASP2**	**HLA-DQB1**	**HSPA1B**	**IL8**	**MAFF**	**NAIP**
Class L	Mean	1.17	0.87	1.25	0.98	1.12	1.26	1.29	0.80	0.70	0.98	1.05
	Std. Err.	0.01	0.01	0.01	0.02	0.01	0.01	0.11	0.01	0.02	0.01	0.01
Class H	Mean	1.19	0.86	1.33	1.05	1.09	1.22	1.12	0.79	0.84	0.97	1.10
	Std. Err.	0.01	0.01	0.02	0.03	0.02	0.02	0.07	0.01	0.04	0.01	0.02
		NDUFB7	**NGLY1**	**NRG1**	**NSF**	**PUM2**	**RAB27A**	**RPA1**	**SEC24A**	**SERPINB2**	**SLC35A1**	**SLC7A5**
Class L	Mean	1.18	1.23	1.52	1.21	1.26	1.13	1.26	1.23	1.26	1.27	1.10
	Std. Err.	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01
Class H	Mean	1.18	1.22	1.52	1.25	1.26	1.17	1.25	1.24	1.30	1.31	1.13
	Std. Err.	0.01	0.01	0.03	0.01	0.01	0.01	0.02	0.01	0.02	0.02	0.02
		STAT1	**STX7**	**THBS1**	**VEGFA**	**WDR7**						
Class L	Mean	1.00	1.08	1.09	1.04	1.31						
	Std. Err.	0.01	0.01	0.02	0.01	0.01						
Class H	Mean	1.00	1.11	1.05	1.03	1.32						
	Std. Err.	0.01	0.01	0.02	0.01	0.02						

Table 3. Descriptive statistics for both Low (L) and High (H) levels of stress for data set after imputing missing values.

		ADAR	**ADORA2A**	**BTN3A3**	**C7orf68**	**CD9**	**CSF1R**	**CX3CR1**	**CYLD**	**DPYD**	**FGL2**	**FOLR3**
Class L	Mean	1.03	1.13	1.33	1.21	1.04	1.49	1.27	1.12	1.21	1.05	1.31
	Std. Err.	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.02
Class H	Mean	1.01	1.11	1.33	1.18	1.08	1.49	1.26	1.15	1.18	1.07	1.33
	Std. Err.	0.01	0.02	0.01	0.02	0.02	0.01	0.01	0.01	0.01	0.02	0.01
		FOSB	**GADD45B**	**GALC**	**GBP1**	**GNA15**	**GORASP2**	**HLA-DQB1**	**HSPA1B**	**IL8**	**MAFF**	**NAIP**
Class L	Mean	1.17	0.87	1.25	0.97	1.12	1.26	1.34	0.80	0.70	0.98	1.05
	Std. Err.	0.01	0.01	0.01	0.02	0.01	0.01	0.07	0.01	0.02	0.01	0.01
Class H	Mean	1.20	0.86	1.34	1.05	1.09	1.22	1.17	0.79	0.84	0.97	1.11
	Std. Err.	0.02	0.01	0.02	0.02	0.02	0.02	0.04	0.01	0.04	0.01	0.01
		NDUFB7	**NGLY1**	**NRG1**	**NSF**	**PUM2**	**RAB27A**	**RPA1**	**SEC24A**	**SERPINB2**	**SLC35A1**	**SLC7A5**
Class L	Mean	1.18	1.23	1.52	1.21	1.26	1.13	1.26	1.23	1.26	1.27	1.10
	Std. Err.	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01
Class H	Mean	1.18	1.22	1.50	1.25	1.26	1.18	1.24	1.24	1.30	1.30	1.13
	Std. Err.	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01
		STAT1	**STX7**	**THBS1**	**VEGFA**	**WDR7**						
Class L	Mean	1.00	1.08	1.08	1.04	1.31						
	Std. Err.	0.01	0.01	0.02	0.01	0.01						
Class H	Mean	1.00	1.11	1.05	1.03	1.31						
	Std. Err.	0.01	0.01	0.02	0.01	0.01						

3. Feature selection for classification

Feature selection process can be perceived as a technique of choosing a subset of variables from available feature set with the evaluation of accuracy for each choice. Nowadays, when the amount of genomic data is growing rapidly, feature selection has become very important issue because of the specificity of the data. Data sets containing genetic information are usually in a form of a matrix where genes are considered as features which expression levels are measured for each sample (rows of a matrix). One distinctive feature characterizing these kind of data sets is the disequilibrium between the number of samples and the number of features - the number of features is generally 10^3 greater (or more) than the number of samples as the modern genomic technology (e.g. microarrays, Next Generation Sequencing - NGS) enables quantifying expression of enormous number of different kinds of genetic attributes (e.g. genes, microRNA). This disequilibrium is associated with the experiment's design: it is focused on determining as many genetic attributes as possible (e.g. whole genome sequencing where one can discover individual differences comparing to the reference genome of as small variations as Single Nucleotide Polymorphism - SNP). Moreover, performing these kinds of experiments is quite expensive and time-consuming, so usually very few replications, either technical or biological, are produced. Due to just outlined matters feature selection appears to be very important task in genomic data analysis.

As it was previously mentioned our data set contained 38 features (genes) and 48 samples divided into two separate groups determined by the level of stress. In this work we present a comparison of two feature selection methods and their influence on the classification accuracy. First method we adapted for the purpose of feature subset selection is the Recursive Feature Elimination (RFE) incorporating resampling implemented in the caret package [10]. The second method is based on the Relaxed Linear Separability (RLS) approach [4]. Subsets of features providing the best classification accuracy generated either with the use of the RFE method or the RLS approach were generated for three classifiers: K-Nearest Neighbours (KNN), Support Vector Machines (SVM) and Random Forests (RF). Ten-fold cross validation was used to determine the classification accuracy.

3.1 Recursive Feature Elimination approach

Basic Backward Selection (a.k.a. Recursive Feature Elimination) algorithm firstly fits the model to all of the features. Each feature is then ranked according to its importance to the model. Let S be a sequence of ordered numbers representing the number of features to be kept ($S_1 > S_2 > S_3$...). At each iteration of feature selection algorithm, the S_i top raked features are kept, the model is refit and the accuracy is assessed. The value of S_i with the best accuracy is assessed and the top S_i features are used to fit the final model. Algorithm 1 gives a description of subsequent steps of this procedure.

Algorithm 1 Basic Recursive Feature Elimination

```
 1: Train the model using all features
 2: Determine model's accuracy
 3: Determine feature's importance to the model for each feature
 4: for Each subset size $S_i$, $i = 1 \ldots N$ do
 5:     Keep the $S_i$ most important features
 6:     Train the model using $S_i$ features
 7:     Determine model's accuracy
 8: end for
 9: Calculate the accuracy profile over the $S_i$
10: Determine the appropriate number of features
11: Use the model corresponding to the optimal $S_i$
```

Model building process is composed of few successive steps and feature selection is one of these. Due to that, resampling methods (e.g. cross-validation) should contribute to this process when calculating model's accuracy. It has been showed that improper use of resampling when measuring accuracy can result in model's poor performance on new samples [11] [12]. A modification of Algorithm 1 including resampling was suggested and is outlined in Algorithm 2.

3.2 Relaxed Linear Separability approach

Relaxed Linear Separability (RLS) approach to feature selection problem refers to the concept of linear separability of the learning sets [14]. The term "relaxation" means here deterioration of the linear separability due to the gradual neglect of selected features. The considered approach to feature selection is based on repetitive minimization of the convex and piecewise-linear (CPL) criterion functions. These CPL criterion functions, which have origins in the theory of neural networks, include the cost of various features [13]. Increasing the cost of individual features makes these features falling out of the feature subspace. Quality the reduced feature subspaces is assessed by the accuracy of the CPL optimal classifiers built in this subspace.

RLS feature selection method consists of three stages. The first stage is to determine an optimal hyperplane $H(\mathbf{w}, \theta)$ separating objects $\mathbf{x}_j$ ($\mathbf{x}_j = [x_{j1}, ..., x_{jn}]$) from

Algorithm 2 Recursive Feature Elimination with resampling

```
 1: for Each Resampling Iteration do
 2:     Partition data into training and testing data sets via resampling
 3:     Train the model on the training set using all features
 4:     Predict the held-back samples
 5:     Determine feature's importance to the model for each feature
 6:     for Each subset size $S_i$, i = 1...N do
 7:         Keep the $S_i$ most important features
 8:         Train the model using $S_i$ features
 9:         Predict the held-back samples
10:     end for
11: end for
12: Calculate the accuracy profile over the $S_i$ using held-back samples
13: Determine the appropriate number of features
14: Estimate the final list of features to keep in the final model
15: Fit the final model based on the optimal $S_i$ using the original training set
```

two learning sets $G^+ = \{\mathbf{x}_j; j \in J^+\}$ and $G^- = \{\mathbf{x}_j; j \in J^-\}$ (J^+ and J^- are disjoined sets of indices j ($J^+ \cap J^- = \emptyset$)).

$$H(\mathbf{w},\theta) = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = \theta\} \tag{4}$$

The hyperplane $H(\mathbf{w},\theta)$ is calculated by the minimization of the criterion function $\Psi_\lambda(\mathbf{w},\theta)$.

$$\Phi_\lambda(\mathbf{w},\theta) = \sum_{\mathbf{x}_j \in G^+} \varphi_j^+(\mathbf{w},\theta) + \sum_{\mathbf{x}_j \in G^-} \varphi_j^-(\mathbf{w},\theta) + \lambda \sum_{i \in \{1,\dots,n\}} \phi_i(\mathbf{w}) \tag{5}$$

where $\lambda \geq 0$.

The function $\Phi_\lambda(\mathbf{w},\theta)$ is the sum of the penalty functions $\varphi_j^+(\mathbf{w},\theta)$ or $\varphi_j^-(\mathbf{w},\theta)$ and $\phi_i(\mathbf{w},\theta)$. The functions $\varphi_j^+(\mathbf{w},\theta)$ are defined on the feature vectors $\mathbf{x}_j$ from the set G^+. Similarly $\varphi_j^-(\mathbf{w},\theta)$ are based on the elements $\mathbf{x}_j$ of the set G^-.

$$(\forall \mathbf{x}_j \in G^+) \quad \varphi_j^+(\mathbf{w},\theta) = \begin{cases} 1+\theta-\mathbf{w}^T\mathbf{x}_j & if\ \mathbf{w}^T\mathbf{x}_j < 1+\theta \\ 0 & if\ \ \mathbf{w}^T\mathbf{x}_j \geq 1+\theta \end{cases} \tag{6}$$

and

$$(\forall \mathbf{x}_j \in G^-) \quad \varphi_j^-(\mathbf{w},\theta) = \begin{cases} 1+\theta+\mathbf{w}^T\mathbf{x}_j & if\ \mathbf{w}^T\mathbf{x}_j > -1+\theta \\ 0 & if\ \ \mathbf{w}^T\mathbf{x}_j \leq -1+\theta \end{cases} \tag{7}$$

The penalty functions $\phi_i(\mathbf{w},\theta)$ are related to particular features x_i.

$$\phi_i(\mathbf{w}) = |w_i| \tag{8}$$

The criterion function $\Phi_\lambda(\mathbf{w},\theta)$ (5) is the convex and piecewise linear (*CPL*) function as the sum of the *CPL* penalty functions $\varphi_j^+(\mathbf{w},\theta)$ (6), $\varphi_j^-(\mathbf{w},\theta)$ (7) and $\phi_i(\mathbf{w})$ (8). The basis exchange algorithm allows to find the minimum efficiently, even in the case of large multidimensional data sets G^+ and G^- [15].

The vector of parameters $\mathbf{w} = [w_1, ..., w_n]^T$ is used in the feature reduction rule [13]:

$$(w_i = 0) \Rightarrow (the\ feature\ i\ is\ omitted) \tag{9}$$

In the result of the first stage of RLS method we obtain optimal hyperplane $H(\mathbf{w},\theta)$ and initial feature set F_k composed of k features not subject to the reduction rule (9).

In the second stage the value of the cost level λ in the criterion function $\Psi_\lambda(\mathbf{w},\theta)$ is successive increased. It causes reduction of some features x_i. It is possible to determine the value of Δ_k ($\Delta_k > 0$) by which to enlarge λ in order to reduce only one feature from feature set F_k. In the result of the second stage we obtain the descended sequence of feature subsets F_k with decreased dimensionality ($F_k \supset F_{k-1}$):

$$F_k \rightarrow F_{k-1} \rightarrow ... \rightarrow F_1 \tag{10}$$

The last step is the calculation of classifier accuracy in each reduced data set corresponding to the subsets of features in sequence (10). The accuracy is usually determined by the use of the CPL [13] classifier and in cross-validation procedure. In this work, different than usual, accuracy is determined by the use of KNN, SVM or RF classifier. As the result of the third stage and the whole RLS method we obtain feature set F^*. It is the feature set characterized by the highest value of classifier accuracy.

4. Results

The classification accuracy determined for the set of genes treated as predictors and obtained with the use of either the Recursive Feature Elimination with resampling (RFE) or the Relaxed Linear Separability (RLS) approach feature selection method is presented for each of the three tested classificators (KNN, SVM, RF). Figures 1 and 2 show the dependence between the number of features selected and the classification accuracy for the KNN classification obtained with the set of features chosen by the two methods of feature selection: RFE and RLS respectively. Figures 3 and 4, 5 and 6 show the same relationship for the remaining two classifiers: SVM and RF also created for the two methods of feature selection. The best classification accuracy and related number of predictors are marked. The correlation coefficient for the linear dependency between the number of features and classification accuracy is also presented on each figure.

As it can be noticed there are two main facts differentiating the two feature selection methods under investigation and their influence on the classification accuracy. First aspect is the existence of the statistically significant linear dependency between number of features and classification accuracy in case of the RFE method – accuracy rises in conjunction with an increasing number of features and reaches its best value with almost all features included in the predictors set. When considering the RLS method this linear relationship is not observed for the KNN and the SVM classification methods and only in the case of the RF classification statistically significant linear relationship was found, however this relation is somehow disputable – it is probably caused by relatively big decrease in the classification accuracy when considering number of features varying from three to eight. It is also worth to mention that the characteristic of the linear relationship is different for those two methods of feature selection: in the event of RFE we have found directly proportional relationship between the number of features selected and the classification accuracy as opposed to the RLS method where this relationship was rather inversely proportional. Second aspect distinguishing the RFE and the RLS methods is the number of features chosen to acquire the best classification accuracy – approximately a magnitude higher in case of the RFE method.

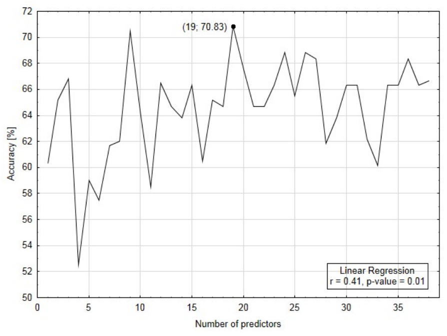


Fig. 1. KNN classification accuracy obtained with an RFE approach.

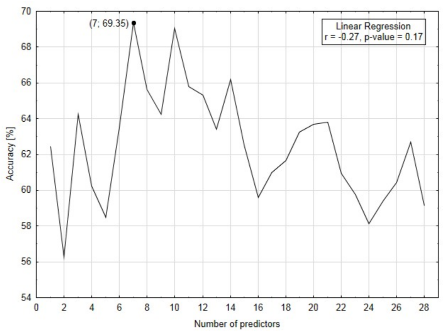


Fig. 2. KNN classification accuracy obtained with an RLS approach.

Table 4 gives a desctiption of genes selected by the RLS method. Given superscripts along with the gene name have the following meaning: 1 - gene was selected for the KNN classifier, 2 - gene was selected for the SVM classifier, 3 - gene was selected

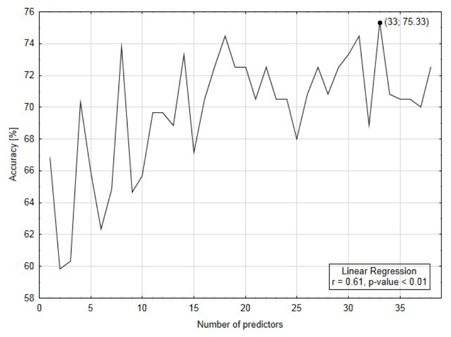


Fig. 3. SVM classification accuracy obtained with an RFE approach.

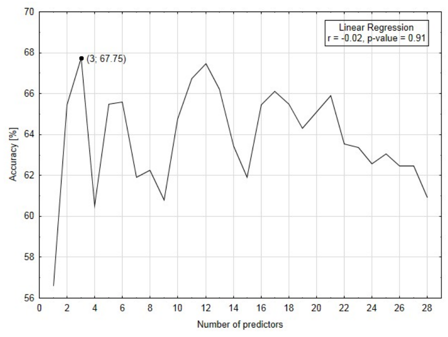


Fig. 4. SVM classification accuracy obtained with an RLS approach.

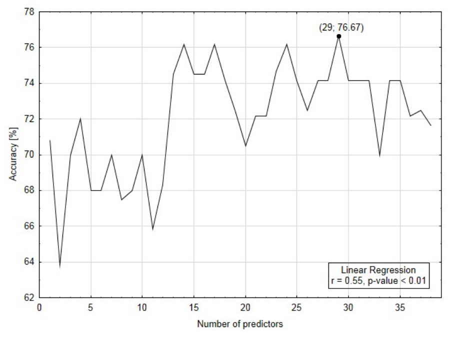


Fig. 5. RF classification accuracy obtained with an RFE approach.

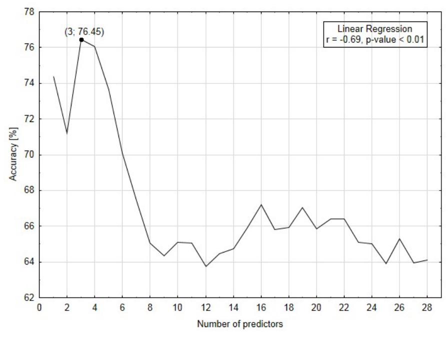


Fig. 6. RF classification accuracy obtained with an RLS approach.

for the RF classifier. As it can be seen: the same set of genes was determined for the SVM and the RF classification methods. The selected set of genes can be characterized by two features: (1) they play an important role in the immune system, (2) they are connected with the nervous system. Those two attributes might be considered as related to the response to stress exposure which is the case in our data set. Lists of genes selected by the RFE method for each classifier are not presented here because they contains almost all of the available features.

5. Conclusions

Classification accuracy obtained with the use of features/genes selected by the RFE method was slightly higher than the one calculated for the set of features/genes se-

Table 4. Genes selected by the RLS method and their description.

Gene name	Description
HLA-DQB1[1, 2, 3]	The HLA-DQB1 gene provides instructions for making a protein that plays a critical role in the immune system. The HLA-DQB1 gene is part of a family of genes called the human leukocyte antigen (HLA) complex. The HLA complex helps the immune system distinguish the body's own proteins from proteins made by foreign invaders such as viruses and bacteria.
IL8[1, 2, 3]	Is one of the major mediators of the inflammatory response. It is released from several cell types in response to an inflammatory stimulus.
NAIP[1, 2, 3]	Acts as a mediator of neuronal survival in pathological conditions. Prevents motor-neuron apoptosis (a type of cell death) induced by a variety of signals.
VEGFA[1]	Is implicated in every type of angiogenic disorder, including those associated with cancer, ischemia, and inflammation. It is also involved in neurodegeneration.
FGL2[1]	May play a role in physiologic lymphocyte functions at mucosal sites.
SLC7A5[1]	Plays a role in neuronal cell proliferation (neurogenesis) in brain.
CSF1R[1]	Plays an important role in innate immunity and in inflammatory processes. Plays an important role in the regulation of osteoclast proliferation and differentiation, the regulation of bone resorption, and is required for normal bone and tooth development.

lected by the RLS method for all three classifiers: KNN, SVM and RF although this measure did not differ substantially. One reason that might have caused this is the specificity of the data set: the RLS method was designed to perform well on sets characterized by considerably great number of features in comparison to the number of samples (few orders of magnitude greater) as presented in [4] and in the data set used in our work the number of samples was almost the same as the number of features. Another issue that is worth to mention is the lack of linear dependency between the number of selected features and the classification accuracy in case of the RLS method – it suggests that the method tends to find the optimal set of predictors providing the best classification accuracy independently from the number of available features which is especially important when one wants to analyse data sets with great amount of variables such as genomic data sets. One disadvantage of the RFE feature selection method that must be intimated is its time-consumingness, which is a common characteristic of all methods exploiting recurrence. In the event of the data set used in our experiment this issue was not so troublesome, although noticeable, but in cases when the number of features is in order of thousands or millions (what is common when analysing genomic data) it might turn out that the trade-off between the classification accuracy and the time spent on computation is not cost-effective.

References

[1] G.K. Smyth, Limma: linear models for microarray data., Bioinformatics and Computational Biology Solutions using R and Bioconductor., R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds), Springer, New York, pp. 397-420.

[2] The Bioconductor project. [http://www.bioconductor.org]

[3] F. De Martino, G. Valente, N. Staeren, J. Ashburner, R. Goebel, E. Formisano, Combining multivariate voxel selection and support vector machines for mapping and classification of fMRI spatial patterns., NeuroImage, 43, pp. 44-48, (2008).

[4] L. Bobrowski, T. Łukaszuk, Feature selection based on relaxed linear separability., In: Biocybernetical and Biomedical Engineering, vol.29, nr 2, pp. 43-58, (2009).

[5] Gene Expression Omnibus. [http://www.ncbi.nlm.nih.gov/geo]

[6] A.W. Bergen, A. Mallick, D. Nishita, X. Wei et al., Chronic psychosocial stressors and salivary biomarkers in emerging adults., Psychoneuroendocrinology 2012 Aug; 37(8):1158-70.

[7] J. Andrews, Oregon Youth Substance Use Project (OYSUP), 1998-2010. ICPSR34263-v1., Ann Arbor, MI: Inter-university Consortium for Political and Social Research, 2013-03-29. doi:10.3886/ICPSR34263.v1

[8] K.J. Livak, T.D. Schmittgen, Analysis of Relative Gene Expression Data Using Real-Time Quantitative PCR and the $2^{-\Delta\Delta C_T}$ Method., Methods 25, 402-408 (2001).

[9] T. Hastie, R. Tibshirani, B. Narasimhan, G. Chu, impute: Imputation for microarray data. R package version 1.32.0.

[10] M. Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt and Tony Cooper, caret: Classification and Regression Training (2013). R package version 7.17-7.

[11] C. Ambroise, G.J. McLachlan, Selection bias in gene extraction on the basis of microarray gene-expression data., PNAS vol. 90 (10), pp. 6562-6566, 2002.

[12] V. Svetnik, A. Liaw, C. Tong, T. Wang, Application of Breiman's Random Forest to Modeling Structure-Activity Relationships of Pharmaceutical Molecules., Lecture Notes in Computer Science vol. 3077, pp. 334-343, 2004.

[13] L. Bobrowski, Eksploracja danych oparta na wypukłych i odcinkowo-liniowych funkcjach kryterialnych., Wyd. Politechniki Białostockiej, Białystok, (2005).

[14] L. Bobrowski Feature subsets selection based on linear separbilty, In: Lecture Notes of the VII-th ICB Seminar: Statistics and Clinical Practice, ed. by H. Bacelar-Nicolau, L. Bobrowski, J. Doroszewski, C. Kulikowski, N. Victor, June 2008, Warsaw, 2008.

[15] L. Bobrowski, Design of Piecewise Linear Classifiers from Formal Neurons by Some Basis Exchange Technique, pp. 863–870 in: Pattern Recognition, 24(9), 1991.

REKURENCYJNA ELIMINACJA CECH Z WALIDACJĄ ORAZ RELAKSACJA LINIOWEJ SEPAROWALNOŚCI JAKO METODY SELEKCJI CECH DO ANALIZY ZBIORÓW DANYCH ZAWIERAJĄCYCH WARTOŚCI EKSPRESJI GENÓW

Streszczenie Zdecydowana większość znanych metod selekcji cech skupia się na wyborze odpowiednich predyktorów dla takich zagadnień jak rozpoznawanie obrazów czy też ogólnie eksploracji danych. W publikacji prezentujemy porównanie pomiędzy powszechnie stosowaną metodą Rekurencyjnej Eliminacji Cech z walidacją (ang. Recursive Feature Elimination - RFE) a metodą stosującą podejście Relaksacji Liniowej Separowalności (ang. Relaxed Linear Separability - RLS) z zastosowaniem do analizy zbiorów danych zawierających wartości ekspresji genów. W artykule wykorzystano różne algorytmy klasyfikacji, takie jak K-Najbliższych Sąsiadów (ang. K-Nearest Neighbours - KNN), Maszynę Wektorów Wspierających (ang. Support Vector Machines - SVM) oraz Lasy Losowe (ang. Random Forests - RF). Porównana została jakość klasyfikacji uzyskana przy pomocy tych algorytmów z optymalnym zestawem cech wygenerowanym z wykorzystaniem metody selekcji cech RFE bądź RLS. W celu wyznaczenia jakości klasyfikacji wykorzystano 10-krotną walidację krzyżową.

Słowa kluczowe: analiza ekspresji genów, selekcja cech, klasyfikacja

Artykuł zrealizowano w ramach projektu badawczego N N519 657940 finansowanego przez Ministerstwo Nauki i Szkolnictwa Wyższego oraz pracy badawczej S/WI/2/2013 Wydziału Informatyki PB.

LAZY EVALUATION METHOD IN THE COMPONENT ENVIRONMENTS

Michał Kniotek

AGH University of Science and Technology, Kraków, Poland

Abstract: This paper describes the manually use of the lazy evaluation code optimization method in the component environments such as Java VM, MS .NET, Mono. Despite the implemented solutions in optimizers, there are occurrences when manual code optimization can accelerate execution of programs. In component environments, due to the optimization performed during JIT (Just In Time) compilation, the code cannot be fully optimized because of the short time available. JIT optimization takes place during execution of the currently used part of the code. That is the reason why the time spent on searching the best optimization methods must be balanced between the program response time and the choice of optimal optimization. This article presents optimization method ending with conclusion to answer in which component environment is recommended to use a given method manually. The presented method is called lazy evaluation.

Keywords: code optimization, component environments, lazy evaluation

1. Introduction

This article was written to prove that the use of manual code optimization allows to achieve additional speed as opposed to using only optimizers incorporated in the component environments. Similar research can be found in the article about loop optimization [16].

At present, the component environments are widely used for coding various programs, because of their main advantages: code portability between different operating systems and computer architectures; the standardized notation of the code, which allows for using universal optimization methods [1].

Currently used digital machines, reached its limit of computing due to the achievement of maximum technological potential capabilities, hence multiple programs may have long execution time. That is why, complicated calculations are performed in the schema of cloud computing. Even on a single machine, as well as on

Advances in Computer Science Research, vol. 10, pp. 53-65, 2013.

multiple machines, it is natural to search for the appropriate code optimizations that reduce the computation time.

There are two cases where optimization should be applied: programs whose running time is long, such as used in the geophysics (magnetotellurics) [13], or in programs where data must be processed quickly, such as in multimedia [17].

Optimization methods are currently looked for in many areas of science, such as: video reconstruction [7], the problem of resource block allocation for downlink Long Term Evolution (LTE) [10], acceleration of generalized minimum aberration designs of hadamard matrices [4], acceleration of element subroutines in finite element method [5], stereo images processing [8].

Optimization method presented in this article was examined. Method was tested on a different computer architecture and in different operating systems. In each case, program execution was timed, intermediate code examined (reverse engineering), and results summarized. The results allowed to answer when, why and in which component environments it is recommended to use a given method.

2. Compilation and optimizatfion methods

Compilation in all the environments mentioned in this article is performed in the same way. In the beginning, the source code is compiled through compilers provided with the package for developers into the intermediate code. The obtained files are then compiled once again by a virtual machine at the program's startup. Virtual machines compile the intermediate code to machine code which is executed by the processor. Compilation of the intermediate code to the machine code is performed by the JIT compiler; it takes place in stages and is subject to ad hoc optimizations at intervals set by the machine (thus the name: Just-in-Time).

JIT compilation occurs at the start of the application. It is performed at every launch of the program, because it could happen that the intermediate code has been moved to a different hardware platform or operating system. The purpose of the aforementioned compilation is to translate the intermediate code into the machine code of the currently used platform. Each code method or function is compiled only when there is need for it. Thus, the program can run without being fully compiled, because some methods may be unused. During the execution of the program, once the compiled parts are not lost and can be reused, they are loaded into the cache as ready to use the machine code [6,12,15].

JIT compilation is limited only by one factor: time. This is due to the fact of its execution during the launch of the application. Therefore, the analysis and code optimization cannot last as long as they could at the AOT (Ahead-of-Time) compilation. [14] However, its advantages are code portability and application optimization suited to the currently used hardware and operating system. To compare, applications compiled using the JIT techniques run faster than scripts which are executed by interpreters [2].

The program can be implemented in many different ways and in all cases the result will be correct. However, certain approaches can be [3]:

- easier (the solution of the problem itself thanks to which the implementation can be easier and more understandable);
- cleaner (they use less memory);
- easier to maintain (to adapt the code to frequent changes and improvements);
- faster (time to obtain the result is shorter).

3. Methodology and test performance characteristics

In order to distinguish test environments, later in the paper they are named by shortcuts E64 and E32. Numbers designate operating systems (32 and 64 bits) installed in the respective test environments.

Hardware specification (E64): Intel® Core^{TM} 2 E6400 @ 2.46GHz (CPU), 1.5GB DDR2-667 (640MHz) (RAM). Operating systems (E64): Microsoft Windows 7 Professional SP1 (64bit), Fedora 16 (64bit).

Hardware specification (E32): Intel® Celeron® M520 @ 1.60GHz (CPU), 512MB DDR2-667 (532MHz) (RAM). Operating systems (E32): Microsoft Windows XP Professional SP2 (32bit), Fedora 11 (32bit).

Virtual machines installed in an E32 and E64 environments: Microsoft .NET 4.0 (Windows), Microsoft .NET 2.0 (Windows), Java Development Kit 7u4 (Windows and Linux), Mono 2.10.8 (Windows), Mono 2.4.3 (Fedora 11 - E32), Mono 2.10.5 (Fedora 16 - E64).

In order to reach objective test results, all testing was performed in the same way, in accordance with the principles set out below.

- Tested instructions were carried out in a loop; the total time of their operation was measured. The loop had a predefined amount of iterations.
- If the test required sample data, they were randomized for each iteration of the loop. Randomizing took place before the measurement of time and the values were stored in arrays. Data were randomized in order to keep the conditions close to real application run. Thanks to randomization, the data were both pessimistic and optimistic (those that can make to return result faster or slower).The randomizer was initiated using current time.

- The measurements were performed for each test ten times. It is always the shortest time of all that was chosen. This approach is burdened with the smallest error, due to the applications that run in the background in a test environment [9].
- To measure time with nanosecond accuracy, methods provided together with executing environments were used. In Mono and .NET, the Stopwatch class from System.Diagnostics package was used, while in Java, it was the System.nanoTime() method.
- The use of the source code compilers to bytecode:
  - .NET, compiler csc with /o flag (optimization launch),
  - Mono, compiler mcs with -optimize+ flag (optimization launch),
  - Java, compiler javac (by default, optimization is turned on).
- The use of JIT compilers:
  - .NET, lack of interference in the applied optimization,
  - Mono, compiler mono with flag -O=all (turn on all possible optimization),
  - Java, compiler java with flag -XX:+AggressiveOpts (optimizations foreseen for the next release of JVM), also separately launched in mode -client and -server.

4. Performance of the lazy evaluation

The test is performed in order to compare the time spans after the use of optimization of the lazy evaluation. [11] The test cases provide for using optimization manually and the lack of the optimization. In addition, tests were carried out for different ranges of the variables that are permitted by conditional statements. This is performed to check whether there is a correlation between the range of the permitted variables in conditional statements and the time of the program execution. In addition, the test is designed to draw attention to the memory usage and the intermediate code after disassembling.

The test was performed as described by pseudo-code shown in Alg. 1. During the randomization of sample data vector (variable vector) it is worth noting that the numbers are randomized from the closed interval <-100;100>. Four variants have been provided for the test; the fundamental difference is shown by Tab. 1. In general, in each variant timeConsumingFunction() (Alg. 2) have been executed and random number from the first element of the vector variable have been checked. The content of the Tab. 1 with the appropriate variant should be inserted in the comment's place in the pseudo-code shown in Alg. 1. The timeConsumingFunction() is designed to perform several calculations (addition, multiplication, division with remainder) that perform longer than it takes to compare numbers. The condition upon which the function printOnScreen(count) is executed is noteworthy. In testing, the condition was never fulfilled, but using it ensured that the optimizer does not recognize the count variable as unused in the code. Optimizers encountering unused variables in the code often use additional optimizations, which had an impact on the test. Typically, fragments of the code which handle such variables are treated as redundant, and this fragment of code is the main part of the test. Lack of that condition caused shortening of the times measured.

Algorithm 1 pseudo-code for the test of the lazy evaluation

```
Ensure: time
 1: for i=0 to i < 1 000 000; i++ do
 2:     for j = 0 to j < 10; j++ do
 3:         vector[i][j] = randomNumber()% 201 - 100;
 4:     end for
 5: end for
 6: startTime();
 7: for i=0 to i < 1 000 000; i++ do
 8:     if /* Different cases from Tab. 2 */ then
 9:         count++;
10:     end if
11: end for
12: stopTime();
13: if count > 500 000 then
14:     printOnScreen(count);
15: end if
```

Algorithm 2 timeConsumingFunction

```
Require: vector
Ensure: boolean
 1: sum = 0;
 2: var = 1;
 3: for i = 0 to i < 10; i++ do
 4:     sum += vector[i];
 5:     var *= vector[i];
 6: end for
 7: return ((sum + var) % 2 == 0) ? true : false;
```

Table 1. Variants for the test of the lazy evaluation

Variant	Code to insert
B1 – before optimization, permitted range (0;100), half of the range of values which have been randomized into the vector variable	timeConsumingFunction(vector[i]) && vector[i][0]<100 && vector[i][0]>0
B2 – after optimization, permitted range (0;100) , half of the range of values which have been randomized into the vector variable	vector[i][0]<100 && vector[i][0]>0 && timeConsumingFunction(vector [i])
B3 - before optimization, permitted range (50;100) , one fourth of the range of values which have been randomized into the vector variable	timeConsumingFunction(vector[i]) && vector[i][0]<100 && vector[i][0]>50
B4 - after optimization, permitted range (50;100), one fourth of the range of values which have been randomized into the vector variable	vector[i][0]<100 && vector[i][0]>50 && timeConsumingFunction(vector[i])

In Tab. 2 all the times measured during the test have been gathered. In addition, the acceleration of using various optimization variants with different order of conditions and ranges of permitted variables has been calculated.

The measured times in the test environment E64 and E32.

Execution time in variants B1 and B3 in every case is almost equal. Using optimization of the lazy evaluation (B2, B4) always reduced the duration of the program execution. More restrictive limit range of permitted values (variant B4) additionally reduces the program's execution time.

The measured times in the test environment E64.

The fastest is the program executed in .NET Framework and in the JVM, client version in Windows 7 in variant B4. The slowest is the program executed in variants B1 and B3 in the JVM, client version in Windows 7 and in Mono in Fedora 16.

The measured times in the test environment E32.

The fastest is the program executed in .NET Framework and in the JVM, client version on both operating systems in variant B4. The slowest is the program executed in variants B1 and B3 in the JVM, client version in both operating systems. Note the variant B4 in JVM client version on Fedora 11, where the execution time is almost equal to the same variant in E64 test environment on Fedora 16.

Table 2. Test results of the lazy evaluation

Environment			Time [ms]				Acceleration [%]	
			B1	**B2**	**B3**	**B4**	**(B1-B2)/B1**	**(B3-B4)/B3**
E64	**Windows 7**	**.NET 2.0**	30.918	21.645	30.398	16.446	29.99	45.90
		.NET 4.0	33.233	22.120	32.427	16.701	33.44	48.50
		Mono	41.313	28.161	41.305	20.334	31.84	50.77
		Java Client	50.554	20.104	49.915	16.061	60.23	67.82
		Java Server	32.990	22.159	32.781	19.872	32.83	39.38
	Fedora 16	**Mono**	50.927	33.221	50.068	25.772	34.77	48.53
		Java Client	37.411	28.111	37.797	23.177	24.86	38.68
		Java Server	39.251	26.705	35.841	24.076	31.96	32.83
E32	**Windows XP**	**.NET 2.0**	50.233	34.898	49.396	25.617	30.53	48.14
		.NET 4.0	53.881	35.941	52.697	26.166	33.30	50.35
		Mono	69.747	45.637	69.621	32.336	34.57	53.55
		Java Client	80.212	32.340	78.951	25.004	59.68	68.33
		Java Server	58.671	37.979	58.322	35.147	35.27	39.74
	Fedora 11	**Mono**	66.111	43.688	66.202	30.745	33.92	53.56
		Java Client	79.162	31.932	77.929	24.543	59.66	68.51
		Java Server	60.916	40.046	60.212	37.219	34.26	38.19

The acceleration in test environment E64 and E32.

The first acceleration was achieved after using the optimization of the lazy evaluation on the code from the B1 variant, while the second was obtained after using the optimization of the lazy evaluation on the code from the B3 variant. The main difference is the range limit of the first number from the vector variable in the conditional statement. In the mentioned cases, the acceleration of the program execution has been noted. There was a higher acceleration when the limiting range was from the interval (50;100). However, there is no linear relationship between the various ranges and acceleration.

The acceleration in test environment E64.

The smallest impact between the permitted range and acceleration is observed in the JVM, server version in Fedora 16. The highest acceleration occurs in the JVM, client version in the Windows 7 operating system. By optimizing the program in the JVM, client version in Windows 7, it has grown from the slowest to the fastest. The smallest acceleration was recorded in the JVM, client version in Fedora 16 with range (0;100).

The acceleration in test environment E32.

The smallest impact between the permitted range and acceleration is observed in the JVM, server version in both operating systems. The highest acceleration occurs in the JVM, client version in both operating systems. By optimizing the program in the JVM, client version in both operating systems, it has grown from the slowest to the fastest. The smallest acceleration was recorded in the JVM server version (both operating systems), .NET Framework and Mono (both operating systems) with range (0;100).

Below are presented the Java bytecodes of conditional statement after disassembling, which is the critical part of the test. Conditional statement in the B3 variant is presented on Tab. 3, while in the B4 variant on Tab. 4. It should be noted that during generation of the intermediate code by the compiler, it did not changed the sequence of conditions. In the B3 variant, the sequence of execution is: function timeConsumingFunction() (number 83:), comparison with the number 100 (number 92: and 94:), and finally a comparison with the number 50 (number 100: and 102:). While in the B4 variant the sequence of execution is (by changing the order obtained the acceleration in test): comparison with the number 100 (number 84: and 86:), comparison with the number 50 (number 92: and 94:), and finally execution of the function timeConsumingFunction() (number 99:), if the previous conditions were met.

Table 3. Disassembly of Java bytecode – lazy evaluation before optimization in the variant B3

```
81: aload_1
82: aload_2
83: invokevirtual #12; //Method timeConsumingFunction:([I)Z
86: ifeq   108
89: aload_2
90: iconst_0  91: iaload
92: bipush 100
94: if_icmpge     108
97: aload_2
98: iconst_0
99: iaload
100: bipush 50
102: if_icmple    108
```

Table 4. Disassembly of the Java bytecode – lazy evaluation after optimization in the variant B4

```
81: aload_2
82: iconst_0
83: iaload
84: bipush 100
86: if_icmpge     108
89: aload_2
90: iconst_0
91: iaload
92: bipush 50
94: if_icmple     108
97: aload_1
98: aload_2
99: invokevirtual #12; //Method timeConsumingFunction:([I)Z
102: ifeq  108
```

Below are presented managed code compiled using a csc compiler provided with .NET Framework disassembled by the Ildasm tool. It shows the managed code of conditional statement in the variants B3 (Tab. 5) and B4 (Tab. 6). It is worth noting that the instructions after compiling with mcs compiler provided with Mono environment are identical after disassemblation by the Ildasm tool. In the B3 variant the sequence of execution is: function timeConsumingFunction() (lines IL_0060 and IL_0065), comparison with the number 100 (line IL_006c), finally a comparison with the number 50 (line IL_0073). While in the B4 variant, the sequence of execution is (by changing the order obtained the acceleration in test): comparison with the number 100 (line IL_0063), comparison with the number 50 (line IL_006a), and finally execution of the function timeConsumingFunction() (lines IL_006e and IL_0073), if the previous conditions were met.

In summary, the best solution is to use optimization of the lazy evaluation in all environments. Its effectiveness depends on the permitted values in conditional statement. More restrictive conditions must be applied at the beginning of a conditional statement, because they allow eliminating most cases at the beginning and probably further check will not be necessary. Due to this, greater acceleration could be achieved, but it also still depends on the specific runtime environment. But we should not forget that more restrictive conditions may be much more expensive computationally. In this case, times should by measured again, because less expensive computationally conditions maybe should be checked earlier. There should be found

Table 5. Disassembly of thr managed code compiled with csc compiler and used Ildasm tool – lazy evaluation before optimization in the variant B3

```
IL_005e: ldloc.0
IL_005f: ldloc.1
IL_0060: callvirt  instance bool SprawdzanieWarunkowPRZED.
    SprawdzanieWarunkowPRZED:: timeConsumingFunction (int32[])
IL_0065: brfalse.s IL_0079
IL_0067: ldloc.1
IL_0068: ldc.i4.0
IL_0069: ldelem.i4
IL_006a: ldc.i4.s  100
IL_006c: bge.s     IL_0079
IL_006e: ldloc.1
IL_006f: ldc.i4.0
IL_0070: ldelem.i4
IL_0071: ldc.i4.s  50
IL_0073: ble.s     IL_0079
```

Table 6. Disassembly of the managed code compiled with csc compiler and used Ildasm tool – lazy evaluation after optimization in the variant B4

```
IL_005e: ldloc.1
IL_005f: ldc.i4.0
IL_0060: ldelem.i4
IL_0061: ldc.i4.s  100
IL_0063: bge.s     IL_0079
IL_0065: ldloc.1
IL_0066: ldc.i4.0
IL_0067: ldelem.i4
IL_0068: ldc.i4.s  50
IL_006a: ble.s     IL_0079
IL_006c: ldloc.0
IL_006d: ldloc.1
IL_006e: callvirt  instance bool SprawdzanieWarunkowPO.
    SprawdzanieWarunkowPO:: timeConsumingFunction (int32[])
IL_0073: brfalse.s IL_0079
```

the middle ground solution between the time of checking and the range of permissible values. The most important thing in this case is the programmer's knowledge about the problem. Particular attention should be paid to the fact that when the optimization is not applied, the effect of limiting the range is negligible. The use of optimization does not affect the additional memory consumption, which is at a constant level. *Garbage Collector* and the optimizations implemented in all environments can easily handle memory management.

5. Conclusions

The authorial contribution is the analysis of acceleration in the component environments after using various optimization with conclusions about it. Benchmarks presented in this paper may help developers to write their own code. On this basis, they can determine that in their case, the optimization would be effective or not. The analysis specifies opportunities to optimize the code in the various stages of the work with it. The entire testing process, allow to explain and understand what the optimizations do. Testing is also a time-consuming task, so they will not have to undergo the same testing process as shown in article, thanks to the results and conclusions to every test presented here. In the test, also the factors on which effectiveness of optimization method may depend have been pointed, such as the limiting range in the optimization of the lazy evaluation. In this article, the intermediate code was analyzed by using reverse engineering methods. The analysis of the intermediate code which provides instructions similar to those that would be executed by the processor, allows to understand the essence of the optimizations.

References

[1] Aho A. V., Lam M. S., Sethi R., Ullman. J. D. (2006), Compilers. Principles, Techniques, and Tools (second edition), Prentice Hall, Upper Saddle River.

[2] Aycock J. (2003), A brief history of just-in-time., ACM Computing Surveys, 35(2), 97–113.

[3] Gray J. (2003), Writing Faster Managed Code: Know What Things Cost. MSDN Library, 06.2003.

[4] Calhoun, J., Graham, J., Hong Zhou, Hai Jiang (2012), Acceleration of Generalized Minimum Aberration Designs of Hadamard Matrices on Graphics Processing Units., 2012 IEEE 9th International Conference on High Performance Computing and Communication, Liverpool, 25-27 June 2012.

[5] Filipovic, J., Fousek, J., Lakomy, B., Madzin, M. (2012), Automatically Optimized GPU Acceleration of Element Subroutines in Finite Element Method., 2012 Symposium on Application Accelerators in High Performance Computing (SAAHPC), Chicago IL, 10-11 July 2012.

[6] Hind M. (2006), Dynamic Compilation and Adaptive Optimization in Virtual Machines, `http://www.research.ibm.com/people/h/hind/ACACES06.pdf`, available 07.07.2012.

[7] Jones, D.R., Schlick, R.O., Marcia, R.F. (2012), Compressive video recovery with upper and lower bound constraints., 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Kyoto, 25-30 March 2012.

[8] Kaaniche, M., Pesquet-Popescu, B., Pesquet, J.-C. (2012), l1-adapted non separable vector lifting schemes for stereo image coding., Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European, Bucharest, 27-31 Aug. 2012.

[9] Kalibera T., Tuma P. (2006), Precise regression benchmarking with random effects: Improving Mono benchmark results., Formal Methods and Stochastic Models for Performance Evaluation, LNCS, 4054, 63–77.

[10] Lin S., Ping W., Fuqiang L. (2012), Particle swarm optimization based resource block allocation algorithm for downlink LTE systems., 2012 18th Asia-Pacific Conference on Communications (APCC), Jeju Island, 15-17 Oct. 2012.

[11] McCarthy J., Abrahams P. W., Edwards D. J., Hart T. P., Levin M. I. (1962), LISP 1.5 PROGRAMMER'S MANUAL, `http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD0406138&Location=U2&doc=GetTRDoc.pdf`, available 05.12.2013.

[12] Microsoft (2004), CLR (Common Language Runtime), `http://msdn.microsoft.com/pl-pl/netframework/cc511286`, available 05.12.2013.

[13] Mudge, J.C., Chandrasekhar, P., Heinson, G.S., Thiel, S. (2011), Evolving Inversion Methods in Geophysics with Cloud Computing - A Case Study of an eScience Collaboration. E-Science (e-Science), 2011 IEEE 7th International Conference on 5-8 Dec..

[14] Noriskin G. (2003), Writing High-Performance Managed Applications: A Primer., MSDN Library, 06.2003.

[15] Oracle Documentation (2009), Understanding JIT Compilation and Optimizations, `http://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/underst_jit.html`, available 05.12.2013.

[16] Piórkowski A., Żupnik M. (2010), Loop optimization in manager code environments with expressions evaluated only once., TASK QUARTERLY, 14(4), 397–404.

[17] Tsai, M.-H., Sung, J.-T., Huang, Y.-M. (2010), Resource management to increase connection capacity of real-time streaming in mobile WiMAX., Communications, IET 4(9), 1108 - 1115.

METODA „LAZY EVALUATION” W ŚRODOWISKACH KOMPONENTOWYCH

Streszczenie Artykuł opisuje użycie metody optymalizacji kodu “lazy evaluation” w środowiskach komponentowych (Java VM, MS .NET, Mono). Pomimo zaimplementowanych rozwiązań w optymalizatorach, występują przypadki, gdy doraźne zoptymalizowanie kodu skutkuje przyspieszeniem pracy programu. Optymalizacja kodu jest przeprowadzana podczas kompilacji JIT (Just In Time) w środowiskach komponentowych, dlatego kod nie może zostać w pełni zoptymalizowany. Optymalizacja i kompilacja następuje w momencie wywołania danej części kodu przez aplikację. Skutkuje to ograniczonym czasem, który jest dostępny na poszukiwanie najlepszej optymalizacji. Dostępny czas musi zostać zbalansowany pomiędzy czas odpowiedzi programu, a wybór optymalnej metody optymalizacji. Artykuł zakończono wnioskami, które pozwalają odpowiedzieć na pytanie, kiedy użycie metody “lazy evaluation” jest zalecane.

Słowa kluczowe: optymalizacja kodu, środowiska komponentowe, lazy evaluation

COLLABORATIVE FILTERING RECOMMENDER SYSTEMS IN MUSIC RECOMMENDATION

Urszula Kużelewska[1], Rafał Ducki[2]

[1] Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

[2] Student of Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Nowadays, the primary place of information exchange is the internet. Its features, such as: availability, unlimited capacity and diversity of information influenced its unrivalled popularity, making the internet a powerful platform for storage, dissemination and retrieval of information. On the other hand, the internet data are highly dynamic and unstructured. As a result, the internet users face the problem of data overload. Recommender systems help the users to find the products, services or information they are looking for.
The article presents a recommender system for music artist recommendation. It is composed of user-based as well as item-based procedures, which can be selected dynamically during a user's session. This also includes different similarity measures. The following measures are used to assess the recommendations and adapt the appropriate procedure: RMSE, MAE, Precision and Recall. Finally, the generated recommendations and calculated similarities among artists are compared with the results from LastFM service.

Keywords: collaborative filtering, music recommendations, recommender systems

1. Introduction

Recommender systems (RS) are methods approaching the problem of information filtering. Their task is to register and analyse a user's preferences and generate a personalised list of items. In other words, the systems filter the information that may be presented to the user based on their interest. As input data, they register products' ratings, views of Web sites, purchases of items, as well as specific characteristics or descriptions of the products [11].

Recommendation concerns, among the others, news, music, video, content of e-learning courses, books and subject of web sites or web site navigation.

Music is regarded as particularly difficult domain for recommender systems application [3]. It combines the fields of music information retrieval (MIR) and recommendations [14]. There are several approaches addressed this problem. The easiest solution is to gather ratings from users, however this type of data is difficult to obtain and can contain, sometimes intended, outliers and noise. The other approach is to count tracks played by users and process them to form ratings, e.g. LastFM (http://www.lastfm.com) . Finally, input data can be users' playlists composed of their favourite songs and artists. There are also methods, which process music streams extracting fundamental complex features from the records, e.g. Mufin (http://www.mufin.com), Pandora (http://www.pandora.com).

The article presents a recommender system for music artist recommendation. It uses track play counts as input data. Different RS approaches, including similarity measures, have been implemented and evaluated using efficiency coefficients and compared to LastFM service results. The paper is organised as follows: the next section introduces recommender system domain: classification, problems, similarity measures and evaluation. The following part presents selected music recommendation solutions. The last two sections concern experiments as well as analysis of the results and the final conclusions.

2. Introduction to recommender systems

Recommender systems help customers to find interesting and valuable resources in the internet services. Their priority is to create and examine users individual profiles, which contain their preferences, then update the service content to finally increase the user's satisfaction. This section introduces recommender systems: their classification and main problems. It presents selected similarity measures and lists the most common approaches to recommendations evaluation.

2.1 Classification and problems in recommender systems

Considering a type of input data as well used methods, recommendation systems are divided into content-based, collaborative filtering (CF), knowledge-based and hybrid [9].

Content-based recommendations (called content-based filtering) base on attribute (characteristic) vectors of items created from text connected with the items, e.g. their description, genre, etc [11]. As an example, in case of books, the item characteristics include its genre, topic or author. The content-based algorithms recommend items, which are similar to highly rated by the user other items in past. As an example, if a user liked (rated or bought) X movie, a recommender system searched other movies, which were similar to X with regard to its genre, title, director's name or description of the story. The main advantages of content-based systems are: relatively simple implementation and independence of users. The disadvantages are: a problem of "cold start" for users and the requirement of items' features analysis.

Knowledge-based approach is better for one-time users stores, e.g. selling cameras (people do not buy cameras often) [1]. The approach bases on technical attributes of the items and user preferences, also weighted, related to the attributes. Knowledge acquirement is often realised by interaction with users. This is an approach, where the "cold start" problem does not appear and users' data are not required to store for long time, however they have to use specific techniques to gather the knowledge.

Collaborative filtering techniques search similarities among users or items, however only archives of users behaviour are analysed [1]. As an example, similar users have mostly the same products in their baskets and similar items are bought by the same customers. This is the most preferred technique in recommender systems. They based on the assumption, that if two users have the same opinion on the particular item, it is very likely they like similarly other items. The most important advantages of this kind of systems are: high precision, simple implementation, no additional knowledge about a domain or objects. The long list of advantages is supplemented with the following disadvantages: a problem of "cold start" for users and objects and poor scalability.

Hybrid approach combines at least two different methods: problems in each of them are solved by strengths of the other one.

The most often issue in RS domain is cold-start problem [9]. It concerns a new user case, when there is no information about their preferences, and a new item, when a new object is added to the offer. Due to the fact, that the new object is not assigned to any user, it can't be recommended to anyone. Content-based approach solves this issue by calculating similarity between the new and already stored items basing on their features.

In arbitrary recommender system application, the number of offered items is large, whereas a user during one session visits a few to tens of them. It results in sparsity of input data and lower reliability in terms of measuring the similarity between customers [4].

Finally, however vitally important challenge in the field of on-line recommendations is scalability. RS deal with large amount of dynamic data, however the time of results generation should be reasonable to apply them in real-time applications. A user reading news expects to see next proposition for him/her in seconds, whereas millions of archived news have to be analysed [4].

2.2 Methods of similarity calculation in collaborative filtering systems

The final content of a recommendation list significantly depends on the similarity measure chosen for the recommendation system. To measure closeness between points $x = [x_1, x_2, \ldots, x_m]$ and $y = [y_1, y_2, \ldots, y_m]$ one can use measures, which have been adapted from relationship or distance among objects coefficients or coefficients created especially for recommendations. For all similarity measures, their higher values indicate higher degree of similarity.

The most popular measure is cosine based one, which calculates cosine of the angle between objects (see Equation 1). When two users or items are similar they have comparable ratings, therefore they are close in space and have the same direction from the origin. The value for the closest points is equal 1, whereas for the farthest: -1.

$$s_{cosine}(x,y) = \frac{\sum_{i=1}^{m} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{m} x_i^2} \cdot \sqrt{\sum_{i=1}^{m} y_i^2}} \tag{1}$$

Another example of similarity measure is Pearson correlation, which calculates the tendency of two series of paired values to move together proportionally. The correlation is described by Equation 2 and has value from the interval [-1,1].

$$s_{Pearson}(x,y) = \frac{\sum_{i=1}^{m} (x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m} (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{m} (y_i - \bar{y})^2}} \tag{2}$$

Pearson correlation, although simple and often used in early research papers, suffers from several disadvantages. First of all, the value does not consider relationship between overlap values and the size of vectors. The other one is an undefined value if a user has the same preference for all items.

Another variant of correlation based similarity is Spearman's rank correlation coefficient. It also measures the relation between variables, however instead of the preference values their relative ranks are taken for computation. The ranks are based on order of ratings, therefore the lowest score of a user have a rank equal 1, the following one - a rank equal 2, etc. Equation 3 describes the coefficient; the rank vectors x' and y' correspond to the preferences vectors, respectively x and y.

$$s_{Spearman}(x,y) = 1 - \frac{6 \cdot \sum_{i=1}^{m} (x'_i - y'_i)}{m \cdot (m^2 - 1)} \tag{3}$$

Tanimoto coefficient is a measure, which ignores preference values taking into account sets of objects appearing in both vectors (see Equation 4).

$$s_{Tanimoto}(x,y) = \frac{|x \cap y|}{|x \cup y|} \tag{4}$$

Distance based measures are commonly used for measure similarity among objects. To adapt their values for increasing according to their closeness rising one can use Equation 5, in which $d(x,y)$ denotes distance between x and y.

$$s(x,y) = \frac{1}{1 - d(x,y)} \tag{5}$$

Distance based similarity's values are from interval (0,1]. The most popular distance measures in recommender systems are Euclidean and Manhattan metrics.

2.3 Evaluation of recommender systems

Evaluating recommender systems and their algorithms is a difficult task [8]. The main reason is their data dependent performance. The approach should be adjusted to different values of ratings as well as to predominant number of users or items. The second reason is changing data in time. Users' tastes vary in time as new products or items appear. Finally, many recommender systems use different metrics as well as new measures are proposed, which are also dependent on a specific dataset.

However, there are measures, which are often used to assess prediction accuracy. Predictive accuracy metrics measure how close the recommender system's predicted ratings are to the true user ratings [8].

The most common example is RMSE, which has been popularised in Netflix prize (www.netflixprize.com). It is described by Equation 6, in which the difference between predicted p and real r rating is calculated. Similar measure is MAE (see Equation 7), however it is more tolerant for high rating divergences. Both metrics should have the lowest values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{m}(p_i - r_i)^2} \tag{6}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{m}|p_i - r_i| \tag{7}$$

Another approach to recommendations evaluation uses metrics from information retrieval domain: *Precision* and *Recall*. They were applied for RS by Sarwar [12]. Items can appear in recommendation list and be relevant (N_{rr}) or irrelevant (N_{ri}). They can be not recommended, as well, but be in fact relevant (N_{nr}) or irrelevant

(N_{ni}). *Precision* is defined as the ratio of relevant items recommended to number of all items in the recommendation list (Equation 8), whereas *Recall* is defined as the ratio of recommended relevant items to total number of relevant items available (Equation 9).

$$Precision = \frac{N_{rr}}{N_{rr} + N_{ri}} \tag{8}$$

$$Recall = \frac{N_{rr}}{N_{rr} + N_{nr}} \tag{9}$$

Precision represents the probability that a recommended item is relevant, whereas *Recall* calculates the probability that a relevant item is recommended. It is desirable to have the highest values of these metrics.

3. Music recommendation

Music is a part of people's everyday life. We can listen to the music on the radio, on the internet or buy albums in stores. However, only promoted or the most popular music is easy to find. Recommender systems are a good tool to address the problem.

The most popular approaches to music recommendation are: collaborative filtering, content-based information retrieval, emotion-based and context-based models [14].

The collaborative filtering music recommenders base on history of track plays or direct music ratings. Interesting solution is automated playlist generation [3], where collocated artists are identified basing on their occurrence in the playlists.

Content based procedures analyse songs' description, features or acoustic characteristics [5]. Based on extracted features, data mining algorithms, such as clustering or kNN classification is applied.

Similar to content-based approach, the emotion-based models base on patterns of acoustic characteristic, however prefer perceptual features such as energy, rhythm, temporal, spectral, and harmony [2].

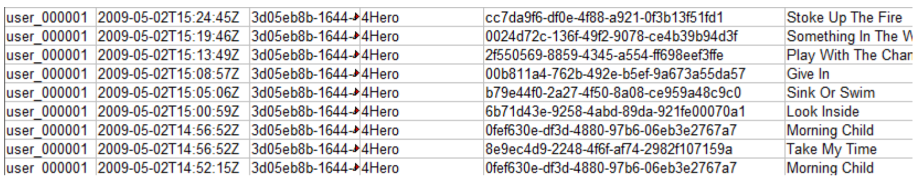
Context-based approach uses public opinion to discover and recommend music [10]. Popular social networks websites provide rich human knowledge such as comments, music reviews, tags and friendship relations. The context-based techniques collect the information to identify artist similarity, genre classification, emotion detection or semantic space.

Music database is an example of extremely huge size source of data. There is a large number of music artists, however there is far more of music fans. Although there are popular music discovery websites such as LastFM, Allmusic (http://www.allmusic.com) or Pandora (http://www.pandora.com), many new methods in scientific articles appear. The most often proposed are the hybrid recommender systems, which are a good solution to cope with the size of data [6].

4. Experiments

This section presents results of experiments with a hybrid system of music recommendation. The system was created as a part of master thesis in Bialystok University of Technology [7]. It is a Web application using Apache Mahout library [16].

Training data was extracted from LastFM music service in form of text files presented in Figure 1. The set contained ratings of 500 users, who listened 13680 tracks of 4436 artists. On average, one user listened to 27.36 songs, whereas one artist was played 3.08 times. Each row of the file contains a user id, date and time of the listening, an id and a name of the artist, an id and a name of the track played by the user.



user_000001	2009-05-02T15:24:45Z	3d05eb8b-1644	4Hero	cc7da9f6-df0e-4f88-a921-0f3b13f51fd1	Stoke Up The Fire
user_000001	2009-05-02T15:19:46Z	3d05eb8b-1644	4Hero	0024d72c-136f-49f2-9078-ce4b39b94d3f	Something In The W
user_000001	2009-05-02T15:13:49Z	3d05eb8b-1644	4Hero	2f550569-8859-4345-a554-ff698eef3ffe	Play With The Char
user_000001	2009-05-02T15:08:57Z	3d05eb8b-1644	4Hero	00b811a4-762b-492e-b5ef-9a673a55da57	Give In
user_000001	2009-05-02T15:05:06Z	3d05eb8b-1644	4Hero	b79e44f0-2a27-4f50-8a08-ce959a48c9c0	Sink Or Swim
user_000001	2009-05-02T15:00:59Z	3d05eb8b-1644	4Hero	6b71d43e-9258-4abd-89da-921fe00070a1	Look Inside
user_000001	2009-05-02T14:56:52Z	3d05eb8b-1644	4Hero	0fef630e-df3d-4880-97b6-06eb3e2767a7	Morning Child
user_000001	2009-05-02T14:56:52Z	3d05eb8b-1644	4Hero	8e9ec4d9-2248-4f6f-af74-2982f107159a	Take My Time
user_000001	2009-05-02T14:52:15Z	3d05eb8b-1644	4Hero	0fef630e-df3d-4880-97b6-06eb3e2767a7	Morning Child

Fig. 1. Data extracted from LastFM service

The aim of the practical part of the article was to construct and evaluate a recommender system in real environment (see Figure 2). It concerns the source of data, deployment of the application on the server as well as the way of evaluation, which was comparison to recommendation lists from LastFM. To make its recommendations effective, the system had to adapt the procedure of recommendations to an active user basing on implemented measures of errors.

One of the main problem was to preprocess the data assigning users to tracks with a rating value. It was performed using the count of track plays. The first step was to normalize the number of plays (see Equation 10) and rank the results with integers from the interval [1,2,3,4,5]. The symbols used in the equation are the following: $r(u_i,t_j)$ - means the rating value, $|u_i(t_j)|$ is a number of plays of track t_j of user u_i, $|u_i|$ is a total number of plays listened by the user u_i and the remaining component denotes maximal number of plays of a particular track listened by one of users from input data.

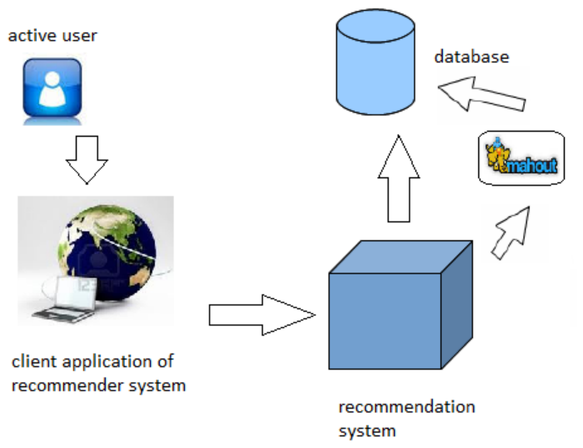


Fig. 2. The architecture of the created recommender system

$$r(u_i, t_j) = \frac{|u_i(t_j)|}{|u_i| \cdot \max_{x=1,y=1}^{x=m,y=n}\{u_x(t_y)\}} \tag{10}$$

The normalisation operation does not affects the data relationship; the graphs presenting the most often played artists and their popularity for randomly selected user are similar (see Figure 3).

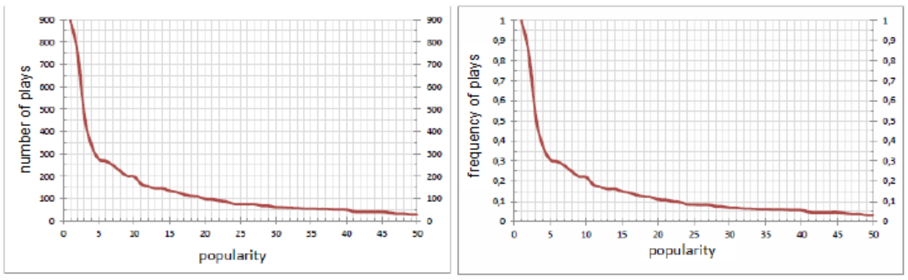


Fig. 3. The most often played 50 artists popularity for randomly selected user before (left) and after (right) data processing

Density of the prepared users' rating matrix calculated using Equation 11 (p is a number of ratings, m - a number of users and n - a number of artists) was 0.62%, which is high enough to apply collaborative filtering procedures and not influencing negatively on time of a generated list of recommendations.

$$\rho = \frac{p}{m+n} \tag{11}$$

The value 1 was the most frequent rating in the processed data (47.41%), followed by 2 (25.58%), 3 (11.24%), 5 (9.9%)and 4 (5.86%). It is worth mentioning that the choice of the ratings range was not only dictated by its popularity. The experiments results performed for the range [1,..,10] using RMSE value were worse.

The algorithms taken for the experiments were user-based as well as item-based collaborative filtering methods. The similarity measures were: cosine measure, Pearson and Spearman correlation, Tanimoto coefficient, Manhattan and Euclidean distance based measures.

In user-based approach it is necessary to determine neighbourhood of an active user. The most popular approach is to identify its k nearest neighbours (*kNN* method). A number of the neighbours is important and affects precision of recommendations. Table 1 contains the results of RMSE for various number of k.

Table 1. RMSE in user-based approach for various number of neighbours and different similarity measures.

k	Manhattan dist. based	Euclidean dist. based	Cosine similarity	Pearson correlation	Spearman rank correlation	Tanimoto coefficient
5	1.35	1.41	1.70	1.53	1.63	1.33
10	1.56	1.31	1.56	1.51	1.61	1.31
15	1.67	1.36	1.59	1.54	1.62	1.33
20	1.76	1.36	1.60	1.53	1.65	1.34
25	1.79	1.35	1.63	1.54	1.65	1.34
30	1.79	1.35	1.60	1.53	1.65	1.34
35	1.79	1.35	1.60	1.53	1.63	1.33
40	1.81	1.33	1.59	1.55	1.61	1.33
45	1.78	1.31	1.56	1.56	1.60	1.33
50	1.80	1.32	1.54	1.57	1.61	1.33
75	1.75	1.30	1.45	1.60	1.58	1.33
100	1.72	1.30	1.40	1.61	1.62	1.33
250	1.42	1.29	1.34	1.61	1.67	1.32
500	1.33	1.29	1.34	1.61	1.67	1.32

In most of the similarity measures cases the value of RMSE decreases when the size of neighbourhood rises. The exception are both correlation coefficients. The greater number of neighbour users requires more time to identify and process them. Taking the mentioned information into account, the optimal results is ~ 1.30 generated for 100 and 250 size of neighbourhood and Euclidean based as well as Tanimoto coefficients.

The best values of user-based results were compared to item-based approach (see Table 2). In this case the best similarity is Manhattan distance based measure. In case of Spearman correlation no results were generated. In all cases, the item-based solution generates recommendations much faster that user-based methods.

Table 2. Comparison of user-based and item-based approach for different similarity measures.

Similarity	user-based			item-based		
	MAE	RMSE	time [s]	MAE	RMSE	time [s]
Manhattan	0.97	1.35	0.704	0.65	0.99	0.007
Euclidean	0.96	1.30	0.303	0.82	1.15	0.009
Cosine	1.08	1.40	0.320	0.85	1.17	0.008
Pearson	1.13	1.53	0.302	1.06	1.51	0.010
Spearman	1.18	1.60	0.322	-	-	-
Tanimoto	0.98	1.31	0.312	0.84	1.16	0.005

The results presented above show, that there is no procedure of recommender system, which is able to be the most effective during its work. It depends on the ratio of the number of users and items as well as on density of the rating matrix. One of the solutions is to combine various approaches and to change the systems according to their performance. In the system presented in this paper the method of recommendation generation is changed in case of efficiency deterioration.

Finally, the system composed of the selected RS methods was compared with LastFM results. The highest coherency of recommendation lists was equal 35%. The similarity between artists was also compared as follows. For every singer or band from training data the system generated 100 most similar the other artists. The list was compared to 100 most similar artists from LastFM service generated for the same singer or band. Figure 4 presents the percentage measure values of coherence for each of the artists. One can notice similarity about 30-40%, however there are many values approximately equal 90%.

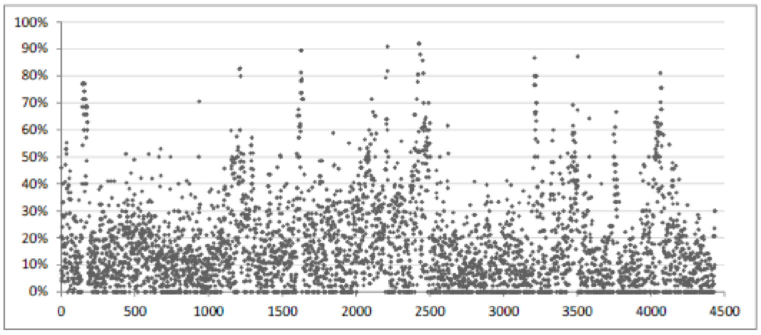


Fig. 4. Comparison of artists' similarity coherence calculated by the proposed system and LastFM service

5. Conclusions

This article presents an approach to music recommendation based on predictive efficiency, which was tested in real environment. The recommendations proposed in the created system are evaluated using error based coefficients as well as compared to real results from the LastFM service. The system combined different approaches: user-based and item-based to generate the recommendations effectively. Despite small size of training data, the obtained results were better than the authors expected. Thereby, they considered them as satisfactory.

Future research can be connected with attachment content based part for similarity calculation among artists as well as tracks. Description of artists or tags for tracks can be taken as input data. Due to limitations of predictive approach of evaluation, the other effectiveness measures can be used. An example is Intra-List Similarity Measure [15], which increases diversification of recommendations with regard to their sort or topic. Another approach is to track users' clickstream. If an active user selects none of the proposed tracks, the procedure of recommendation can be changed and a new list generated.

References

[1] S.S. Anand, B. Mobasher, Intelligent techniques for web personalization, Lecture Notes in Computer Science, vol.3169 (2005), pp. 1-36.

[2] K. Bischoff et all, Music mood and theme classification - a hybrid approach, 10th International Society for Music Information Retrieval Conference, 2009, pp. 657-662.

[3] G. Bonnin, D. Jannach, Evaluating the quality of playlists based on hand-crafted samples, 14th International Society for Music Information Retrieval Conference, 2013, pp. 263-268.

[4] D. Bridge, J. Kelleher, Experiments in Sparsity Reduction: Using Clustering in Collaborative Recommenders, Lecture Notes in Computer Science, vol. 2464 (2002), pp. 144-149.

[5] M.A. Casey et all, Content-based music information retrieval: current directions and future challenges, Proceedings of IEEE, vol.96, no. 4 (2008), pp. 668-696.

[6] O. Celma, Music Recommendation and discovery - the long tail, long fail, and long play in the digital music space, Springer, 2010

[7] R. Ducki, System rekomendujący wykonawców muzycznych (in Polish), Master Thesis supervised by U. Kużelewska, Faculty of Computer Science, Bialystok University of Technology, 2013.

[8] J.L. Herlocker, Evaluating Collaborative Filtering Recommender Systems, ACM Transactions on Information Systems, vol. 22, no. 1 (2004), pp. 5–53.

[9] D. Jannach et all, Recommender Systems: An Introduction, Cambridge University Press, 2010

[10] S. Panagiotis et all, Ternary Semantic Analysis of Social Tags for Personalized Music Recommendation, 9th International Society for Music Information Retrieval Conference, 2008, pp. 219-224.

[11] F. Ricci et all, Recommender Systems Handbook, Springer, 2010.

[12] B. Sarwar et all, Analysis of recommendation algorithms for E-commerce, 2nd ACM Conference on Electronic Commerce, 2000, pp. 285–295.

[13] B. Sarwar et all, Recommender Systems for Large-Scale E-Commerce: Scalable Neighborhood Formation Using Clustering, 5th International Conference on Computer and Information Technology, 2002.

[14] Y. Song,S. Dixon, M. Pearce, A Survey of Music Recommendation Systems and Future Perspectives, 9th International Symposium on Computer Music Modelling and Retrieval (CMMR 2012), 2012, pp. 395-410.

[15] C.N. Ziegler et all, Improving Recommendation Lists through Topic Diversification, 14th International Conference on WWW, ACM Press, 2005, pp. 22-32.

[16] Apache Mahout, Open-source data mining library, `[http://www.mahout.apache.org]`, accessed 20.10.2013.

SYSTEMY TYPU *COLLABORATIVE FILTERING* W REKOMENDACJI MUZYKI

Streszczenie W obecnych czasach głównym miejscem wymiany informacji jest internet. Jego cechy, takie jak: wysoka dostępność, nieograniczona pojemność i różnorodność informacji wpłynęły na jego niezrównaną popularność. W ten sposób internet stał się potężną platformą do przechowywania, rozpowszechniania i udostępniania informacji. Z drugiej strony, dane internetowe są bardzo dynamiczne i niestrukturalizowane. W rezultacie, użytkownicy internetu muszą radzić sobie z problemem przeładowania danych. Systemy rekomendujące służą pomocą użytkownikom w celu znalezienia poszukiwanych produktów, usług lub informacji.

W artykule przedstawiono system rekomendujący artystów muzycznych. Składa się on z procedur typu user-based oraz item-based oraz różnych sposobów szacowania podobieństwa, które mogą być zmieniane dynamicznie podczas sesji użytkownika. Do oceny list rekomendacji wykorzystano następujące miary: RMSE, MAE, Precision i Recall. Dodatkowo, wygenerowane rekomendacje i obliczone podobieństwa między artystami są porównywane z wynikami z serwisu LastFM .

Słowa kluczowe: systemy rekomendujące, rekomendacja muzyki, wspólna filtracja

Artykuł zrealizowano w ramach pracy badawczej S/WI/3/2013.

AUTOMATIC RECOGNITION OF MULTIPLE BRANDS IN IMAGES ON MOBILE DEVICES

Marcin Skoczylas

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Visibility of the product on a shelf is an important task of modern marketing principles. Very often companies have agreements with merchants that particular product will be visible and cover defined percentage of the shelf. This trivial task of counting the amount of products or branding logos that are visible within a certain range, is performed manually by an auditor that checks if the agreement is fulfilled. Up till now there does not exist an easy, mobile mechanism that allows to easily capture, recognise and count defined, multiple objects that are visible in the surroundings of the user. Such scenario however, can be achieved using modern mobile phones and their cameras to capture surroundings, and then use their computing power to perform the recognition and counting. For this purpose, feature detectors (such as SIFT, SURF or BRISK) are utilised to create a database of products box images and extracted keypoints are stored. In a new image keypoints are found using the same feature detector, but to avoid problem of multiple identical keypoints, the image is divided and analysed using a sliding window. Keypoints from a window are extracted and are considered as candidates for keypoints that correspond to training images. When enough points are found then perspective transform is calculated. If detected corners are correctly shaped then product is marked with recognised class. In this paper preliminary results of a mobile framework that allows recognition and counting of visible products in surroundings of the user will be presented.

Keywords: visual shelf monitoring product cover logo recognition keypoints detection mobile devices

1. Introduction

Visual shelf monitoring is a known method of checking products availability. There exist commercial applications that are able to monitor goods on the shelf and raise alarms when certain product is not available[1]. Such applications are relying on a simple detection mechanisms, such as comparing two adjacent images captured in two

[1] for example http://mathecsys.com/visual-shelf-monitoring/

different moments of time, and raise alarms when an empty space is visible instead of a package or branding logo. This technology however, requires a static camera attached that is monitoring space only to a limited extent. Any angle changes or camera movements require image fetching calibration.

In a modern world there exists a requirement to perform image analysis in a fast and mobile way. As an example let's consider a typical check of the visibility of the product availability in shops, especially counting of products visibility percentage.

Visibility of the product on a shelf is an important task of modern marketing principles. Very often companies have agreements with merchants that particular product will be visible and cover defined percentage of the shelf. To confirm that the agreement is fulfilled the auditor is checking several shops and shelves every day. To check that the product is visible within a set range, this trivial task of counting the amount of products visible is performed manually by a person that checks if the agreement is fulfilled. Recent mobile phones are gaining computing power and can perform such tasks automatically.

Until now there does not exist an easy, mobile mechanism that allows a user to easily capture, recognise and count defined objects that are visible in the surroundings. Such scenario however, can be achieved using modern mobile phones and their cameras to capture surroundings, and then use computing power to perform the recognition and counting. In this paper preliminary results of a framework that allows recognition and counting of visible products in surroundings of the user will be presented.

2. Requirements

In this section main purposes of the presented approach, as well as differences in existing applications will be delineated.

The main purpose of the solution is a possibility of proper recognition and counting products or visible branding logos in user surroundings. Shapes, representing branding logos, are defined before capturing images on-site and are reused in following runs of the algorithm. The idea of this approach is that the auditor will use only a recent mobile phone (such as Android or iOS powered devices) with the recognition software installed. The auditor should be able to capture surroundings using his mobile phone camera, and as a result of the procedure, the algorithm should give to the user information about detected branding logos and visibility percentage. These values can be stored for further processing by the auditor or merchant. In overall the algorithm has to be robust with low computational complexity, but also with high accuracy. It's important to note, that packages and branding can (and certainly will) be visible in user's surroundings in different scales and rotations. The solution sho-uld handle this situation gracefully, giving to the user possibility of different capture angles and light intensities, possibly without complicated calibration.

There exist different approaches to image recognition problems, using texture features classifiers [1], image processing filters, etc. Existing solutions can perform correct image detection and recognition in captured images as static [2], but also mo-ving images [3]. None of existing commercial solutions of visual shelf monitoring or products visibility counting can capture and perform brand logo recognition in mul-tiple images of user surroundings. Also the algorithms are aimed towards detection and recognition of only one object visible in the image. The problem described in this paper requires detection of multiple objects visible on multiple images.

3. Related work

Recognition of defined images on other images is a well-known problem and many solutions are existing. Most popular approach is to extract from training images and the captured image keypoints that are defining certain properties of the area. Different keypoint extracting algorithms were invented by authors over last many years, these include for example Scale-Invariant Feature Transform (SIFT [4]) or Speeded Up Robust Features (SURF [5]), but also recently presented Binary Robust Invariant Scalable Keypoints (BRISK [6]) and many others.

Topic of images recognition using keypoints descriptors became recently very popular. For example, SURF algorithm was successfuly utilised for face recognition [7] and road obstacle recognition [8]. The BRISK algorithm was implemened into traffic sign recognition as presented in [9]. These objects are recognised using a key-points detector and descriptors of the points from training images are used to find si-milar descriptors in a captured image. One of such examples is also implementation of SIFT object recogniser in FPGA presented recently [10]. Authors implemented their algorithm using some of the concepts that are also a base for this publication, however the difference is that the algorithm from [10] can correctly identify only one object in the image. It's not suitable for counting amount of the same objects visi-ble on captured image, but the possibility of real-time calculations thanks to robust FPGA implementation are promising.

There exist also different approaches to logo recognition as for example recen-tly presented in [11]. Authors are classifying vehicle logos using SVM based on a Bag-of-Words (BoW) approach. The algorithm extracts SIFT features and quantizes features into visual words by 'Soft-assignment'. The database is then used for new objects recognition.

All the algorithms above are mainly searching for one particular object in the image. The purpose of this publication is definition of an algorithm, that can recognise multiple objects in the image, also repeating ones.

4. Recognition algorithm idea

4.1 Feature descriptors

Image feature descriptors are becoming a standard in current state of the art of image recognition algorithms. Their application is mainly for detection and recognition purposes, however there are additional tasks such as medical image registration [12]. For this study, author selected most common and popular feature detectors: SIFT, SURF, but also recently presented BRISK. Main principles of these three algorithms are delineated in the following paragraph.

- Scale-Invariant Feature Transform (SIFT)

The SIFT algorithm is quite computational extensive algorithm that detects features in images. Best thing about SIFT is that it is invariant to scale changes, but also rotations and light intensity changes. The main algorithm can be boiled down to these steps:

1. Scale space is constructed. The original image is incrementally convolved with gaussian operator: $L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$ multiple times, with different σ value[2] and then the resulting images are scaled down by a half. On scaled images, again the gaussian convolution is applied and so on. This operation generates multiple scale pyramids, with different σ parameters.
2. Local extrema are detected. Algorithm iterates for each pixel, that is compared to a current image, one above and one below in the pyramid scale. A key point candidate is detected only if it is larger than all neighbors or smaller than all others.
3. Key point candidates are evaluated. Low-contrast features and edges (such as corners) are removed.
4. Orientation is assigned to each key point. Gaussian smoothed image is selected that has the same scale as the key point. For each image pixel $L(x,y)$ at this scale, the gradient magnitude, $m(x,y) = \sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2}$ and orientation

[2] suggested amount of octaves by SIFT author is 5

$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$ are calculated around the key point. Orientation histogram is created with 36 bins (divided orientations in 360°). Then, from the histogram a main gradient orientation is identified by finding two preminent peaks. This key point is marked with that found gradient orientation (thus orientation invariance can be applied by removing that relative orientation).

5. Descriptor of the key point is generated. Image gradient magnitudes and orientations are sampled around the keypoint location, a Gaussian weighting function with σ equal to one half the width of the descriptor window is used to assign a weight to the magnitude of each sample point, orientation histograms over 4x4 regions are created and a descriptor vector is created by concatenating all orientation histogram entries. This descriptor vector is normalised. Finally, a SIFT key point descriptor vector is obtained that contains 128 values.

- Speeded Up Robust Features (SURF)

SURF algorithm is also known as an approximate version of SIFT. Main idea of the algorithm is similar, however in SURF authors drew attention to the performance and applied algorithm optimalisations. As presented by authors, the algorithm outperforms SIFT in terms of the quality and performance. Scale pyramid is not constructed as in SIFT, instead different filter sizes (octaves) are used to achieve the same purpose (the scale space is analysed by up-scaling the filter size rather than iteratively reducing the image size[13]). Main algorithm steps are:

1. 2 Aproximate Laplacian of Gaussian images are created by using a box filter and integral images: $I_{int}(x,y) = \sum_{i}^{x}\sum_{j}^{y} I(i,j)$. Integral images are optimisations for computations of intensities of any rectangle from the original image.
2. Key points are detected by applying Hessian determinants. Hessian matrix eigenvectors construct an orthogonal basis that shows direction of the curve of the image, local extremum can be detected easily when product of eigenvalues is positive. Maxima are interpolated in scale and image space.
3. To get rotation invariant key point descriptor, a Haar wavelet responses are calculated in x and y direction (invariant to light intensity changes), in a circular neighborhood of radius $6s$ (s is the scale of the detected key point). Points are weighted using a Gaussian of $2.5s$ and dominant orientation (vector) is found: sum of all responses within a sliding orientation window that covers angle of $\frac{\pi}{3}$ is calculated, and a new vector for the sliding window is obtained by summing horizontal and vertical responses.

4. Descriptor of the key point is generated. A region is constructed from a square of size $20s$ that is centered on each key point, orientation is the same as previous step vector. Region is divided into 4x4 square sub-regions, a feature vector is calculated for each region: haar wavelets are calculated for sub-region, and wavelet responses are summed: $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$ with respect to the detected orientation. Finally, a SURF key point descriptor vector is obtained that contains 64 values.

- Binary Robust Invariant Scalable Keypoints (BRISK)

BRISK is a novel algorithm presented recently. As authors state the quality of key point detection is compared to top state-of-the-art key detector SURF, but needs less computation time. This is achieved by detecting key points in octave layers of image scale pyramid, but also in layers in-between in continuous domain via quadratic function fitting. Brief description of the BRISK method [6]:

1. Octaves in scale pyramid are created by half-sampling original image, in addition intra-octaves located in-between layers are created by downsampling original image by a factor of 1.5 and successive half-sampling.
2. A FAST[14] corner detector with a mask of 9 consecutive pixels in a 16-pixel circle is applied on every layer and sub-layer (intra-octave) with the same threshold T and ROIs are detected.
3. Within detected ROIs all points are tested to meet maximum condition against its 8 neighboring FAST scores in the same layer (a corner is detected). Then, scores in layer above and below are checked and need to be lower than this detected maximum.
4. To limit complexity, a 2D quadratic function is fit to the 3x3 patch surrounding the pixel and three sub-pixel maxima are determined (from layer of the keypoint, one above and one below). Maxima are interpolated using 1D quadratic function along the scale space and local maximum for score and scale is found. Image coordinates are interpolated between patches in layers to the scale found.
5. Descriptor of the key point is generated. A sampling pattern is defined: these are N locations equally spaced on circles concentric with the keypoint[6]. A Gaussian smoothing is applied with σ_i proportional to distance between points and respective circle. Two subsets are created: short-distance pairs and long-distance pairs. A local gradient is calculated for long-distance pairs, gradients are summed and feature orientation α is found, then a descriptor is formed from short-distance pairs that are taken from a sampling pattern rotated using that orientation: a bit-vector is constructed by performing comparison of all short-distance pairs:

$$\forall (p_i^\alpha, p_j^\alpha) \in S,\ b = \begin{cases} 1, I(p_j^\alpha, \sigma_j) > I(p_i^\alpha, \sigma_i) \\ 0, otherwise \end{cases}$$

4.2 Product detection and recognition

First, original product images database is created. From all images from the products database keypoints are detected using one of features detector described in section 4.1. Keypoints feature descriptors associated to each product image from the database are stored, these are reference vectors of descriptors used for products recognition. Thus, for each product or brand logo image, a vector P_k of features descriptors is created. Note, that this vector can have various sizes, depending on complexity of the image.

When a new image is acquired by a mobile camera, first the keypoints are searched using the same detector as used for the database creation. When a set of image keypoints descriptors W_{all} is obtained, the image is scanned using a sliding window, which size is selected iteratively, starting from a small size and iteratively doubled. Image is divided into d equal-sized areas. Starting value of the divider parameter d can be tweaked based on the expected accuracy of the recognition[3]. It is important to note, that image division is needed to correctly associate training image keypoints with these from a window. If a window overlaps two identical images that have the same keypoints, but in different places, then the calculation of proper homography is problematic. Thus, that parameter has a key role in a definition of a minimal size of the training image visible in the captured image.

For each window size the image is scanned using a sliding window: keypoints inside that window are extracted, detection is performed and finally recognised keypoints are removed from W_{all}. The sliding window is moved from top-left position to bottom-right in steps. The step size is set to a half of the window size (independently: half-vertical and half-horizontal). When sliding is finished the window size is doubled to include bigger area and so on. Process is stopped when previous window size encompassed the original image.

During scanning of sliding window only a part of keypoints from the whole image are taken into consideration, such that coordinates of the keypoints are inside current area of the window. Detection of products and recognition is performed using a technique called keypoint matching [15], but only keypoints detected in a current

[3] for example 1/4 of original image size, but also one can get good results with 1/8 or even 1/2.

area of the sliding window are considered (a set $W_{i,j}$). A nearest neighbour kNN search is performed on each product keypoints descriptors vector (P_k) and window's keypoints set ($W_{i,j}$) with[4] $K = 2$. Found pairs are filtered to find good matches using technique described in [17]: first, the minimum distance (*min*) is found from all matches, and then all distances that are bigger than[5] $2 * min$ are discarded. If the amount of keypoints in a set containing found matches is less than 4 (thus, at least four corners), then operation is repeated for next product vector (P_{k+1}), in other case the product is detected, marked by product code and added to result set R. Finally a homography is found using a well-known RANSAC [18] algorithm using pairs of keypoint matches and then perspective matrix transformation of vectors is performed. If the transformed polygon is not convex, then the object is not added to the result set R. The image is scanned using a window to correctly find all copies of the product packages. If the kNN-2 would be performed on the whole image, only one copy of the product package would be detected[6]. When product is detected in the original image, all the keypoints that are inside a product's box in the transformed perspective are removed from the original set W_{all} of keypoints descriptors. Still, this could lead to over-detection. Detected areas have to be checked for such situation and repeated detections filtered out, this is described in section 4.3.

Keypoint matching can be improved for example by supervised learning [19]. In this study it was not necessary, as the results for the presented purpose are satisfactory.

4.3 Over-detection

With the procedure described above it is possible that some products recognised in a resulting set R will be repeated. To solve that, product box areas that are overlapping by more than 75% are examined. Note, that the resulting transformed box is in fact a trapeze (a polygon). Combinations of two polygons from the set R are considered for removal. To find overlapping parts of two polygons a Yu-Peng Clipper [20] algorithm is used to calculate intersection of the polygons. If the intersection area size is bigger than a mean area size of two candidate polygons, then the smaller polygon is removed from the resulting set R. Thus, at the end all over-detected polygons are

[4] The k=2 in kNN is suggested by J. Beis and D. Lowe in their Best-Bin-First (BBF) algorithm [16]

[5] This parameter was set empirically. In the literature different parameters for this distance can be found.

[6] Actually it's possible to construct an algorithm that will avoid a sliding window. Author is working on a solution of this problem

removed. Yu-Peng Clipper algorithm is not perfect and in very rare scenarios it can fail [20], but the speed gained in a whole procedure is crucial and that can be accepted in a mobile environment.

A flowchart diagram of the whole algorithm is presented in Figure 1 for one image class variant. Different training image classes are reocognised iteratively, for each training image a movement of sliding window is repeated.

5. Selecting the keypoint detector algorithm

It is very important to select proper keypoint detector algorithm depending on run time and image conditions. Considering, that the whole procedure will be run in a mobile environment with limited resources, the selected algorithm must be robust. Thus, the speed and run time of the algorithm is favored over the accuracy, but from the other hand high accuracy also must be achieved. The keypoint detector is most crucial part of the whole procedure, as it's the most computation exhaustive part. Other crucial factor is keypoints descriptors comparator. For SIFT and SURF algorithms it's a kNN algorithm (see section 4.2), however for the BRISK algorithm the simple brute-force with hamming distance algorithm was used.

5.1 Methods and results

To select the keypoint detector algorithm, a database of images was created. Author captured 20 good defined covers and logos of different medicines using a mobile camera, that was marked as a learning set. In addition real-life images were captured (test images). These images presented many different medicines in different conditions (rotated, scaled and in different lighting conditions). On one test image up to 20 different medicine packages were visible, some packages also repeated. In addition, some images were not consisting any image from the learning set. Such database was evaluated using algorithms described in previous sections, but in addition tests with additional image filtering were performed (in particular, histogram normalisation).

The first test was performed as follows: on the original learning image a keypoint detector algorithm was run, keypoints descriptors were stored in a resulting set R_{learn} together with their position (x,y) on the image. That image was further processed and altered: scale was changed to a factor of $(0.1, 10.0)$ with a step of 0.1, image was rotated and additional black frame was added around the image (rotated by a step of $15°$ from $0°$ to $360°$), a gaussian random noise was applied ($I_{test}(x,y) = I_{orig}(x,y) + random$) from 10 to 300 with a step of 10, and lightness was altered ($I_{test}(x,y) = I_{orig}(x,y) + lightness$) from -50 to 50 with a step of 5. After applying changes to

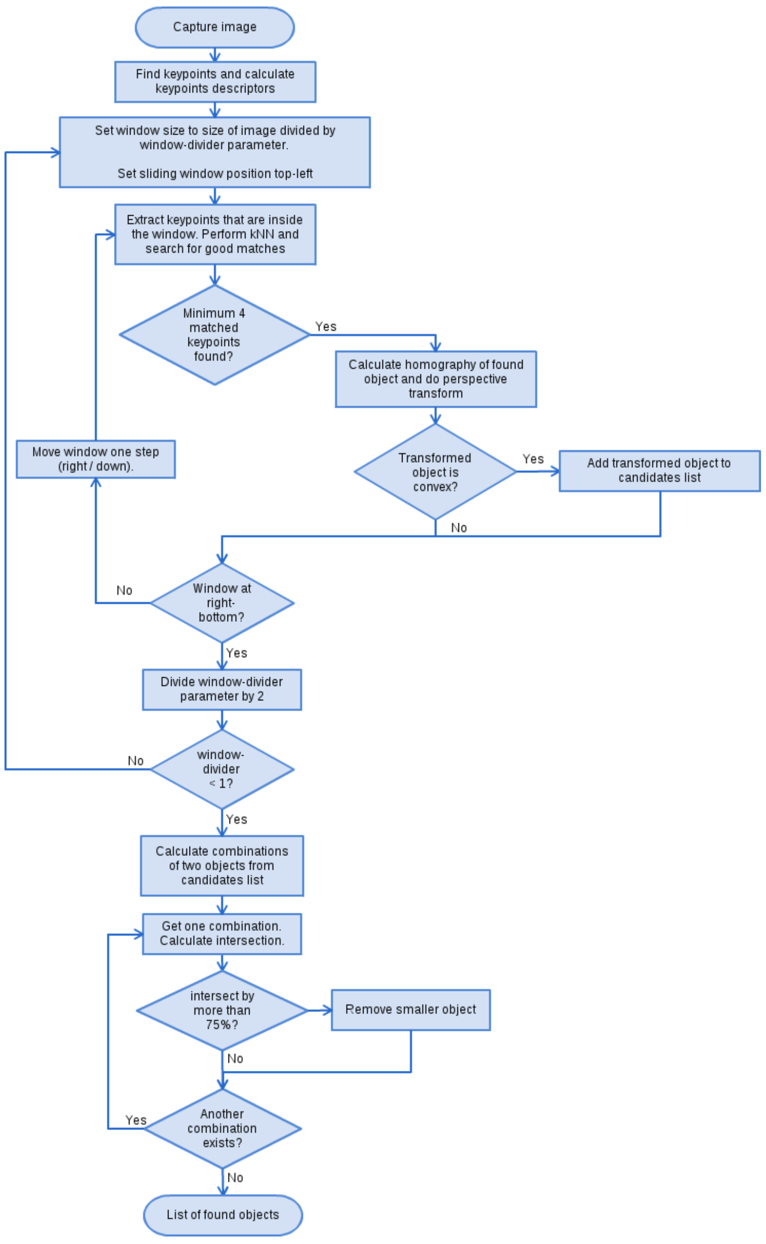

Capture image
Find keypoints and calculate keypoints descriptors
Set window size to size of image divided by window-divider parameter.
Set sliding window position top-left
Extract keypoints that are inside the window. Perform kNN and search for good matches
Minimum 4 matched keypoints found?
Yes
Calculate homography of found object and do perspective transform
Transformed object is convex?
Yes
Add transformed object to candidates list
No
Move window one step (right / down).
No
Window at right-bottom?
Yes
Divide window-divider parameter by 2
No
window-divider < 1?
Yes
Calculate combinations of two objects from candidates list
Get one combination. Calculate intersection.
intersect by more than 75%?
Remove smaller object
No
Yes
Another combination exists?
No
List of found objects

Figure 1. A flowchart diagram presenting the whole procedure from sections 4.2 and 4.3 for one image class.

the image, whole procedure described in section 4.2 was performed: on a new image keypoints were detected, keypoints pairs were found and homography was calculated. Having a new homography, the image was transformed back to origins: perspective of altered image was transformed by inverse of homography calculated. In overall that procedure moved all keypoints from altered image to positions that correspond with original learning image. For a perfect keypoints detector, all keypoints (K_{all}) from original image should have exactly the same positions as the transformed-altered keypoints. Thus, one can calculate a $ratio$ how many pairs of keypoints (from original image and altered image) are exactly positioned. If distance of two keypoints from one pair was higher than 5 pixels, then such pair was discarded (keypoints do not „fit”), number of correctly positioned keypoints K_{fit} was increased otherwise. To calculate the $ratio$, a number of keypoints K_{fit} is divided by a number of all keypoints K_{all}.

On each image from original learning data set a test procedure was run, result $ratios$ and $times$ needed to perform the test were recorded. Then, a mean, standard deviation, max and min was calculated from recorded ratios of all images for each parameter change.

Results are shown on Figures 2, 3, 4 and 5. Lines are added to increase readability, the ratio was tested only in certain data points.

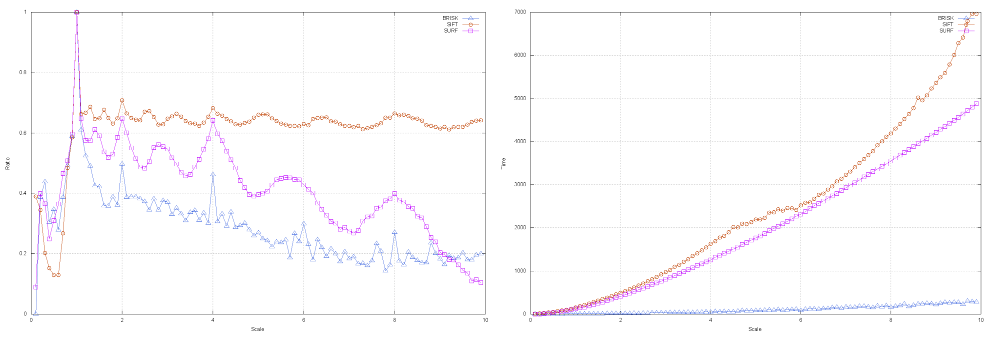


Figure 2. Mean Correct Keypoints Ratio (on the left) and algorithm run time in milliseconds (right) after changing scale (from 0.2 to 10, step of 0.1).

Another, global check was performed. In this test, all parameters (ratio, scale, noise, intensity) were changed iteratively, and all combinations of these parameters values for all images were checked. For each combination a result ratio and time was obtained. Comparison of the algorithms are presented in Table 1.

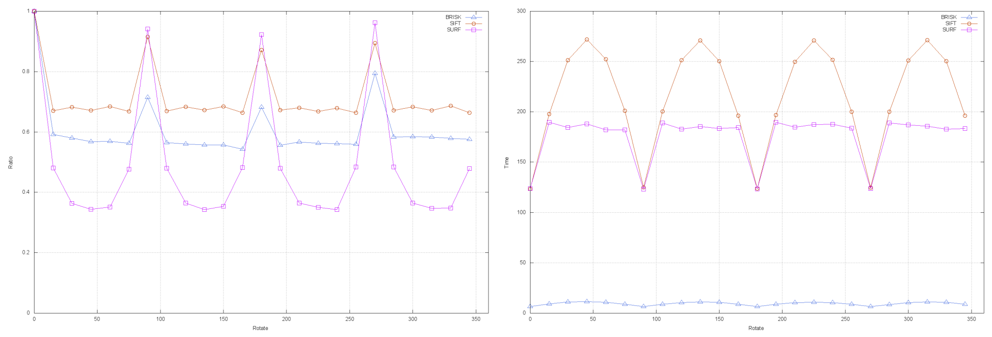


Figure 3. Mean Correct Keypoints Ratio (on the left) and algorithm run time in milliseconds (right) after applying rotation.

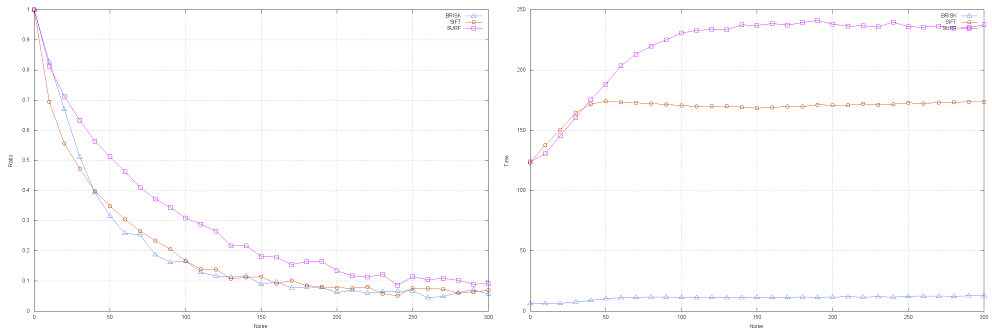


Figure 4. Mean Correct Keypoints Ratio (on the left) and algorithm run time in milliseconds (right) after applying gaussian noise.

The mean ratios are very small due to the fact that it's a comprehensive test that includes all possible image amendments, that also included a huge noise and almost zero-lightness for hundreds of tests - thus giving a small result mean ratios (that amended images are not readable also by humans). These are not real-life scenarios and were provided here only for artificial comparison of the algorithms, but surprisingly some algorithms overperformed others even in such low-quality images. The key in this table is comparison of algorithms performance in terms of speed, but also subtle difference in recognition ratios can be spotted.

From the calculated results the BRISK algorithm outperforms SIFT and SURF in terms of computational speed. The difference is huge, however the cost of computations has been replaced by lowest detection ratio. Nevertheless, the ratio is still satisfactory, and it is lower only by a fraction from the best algorithm in this field

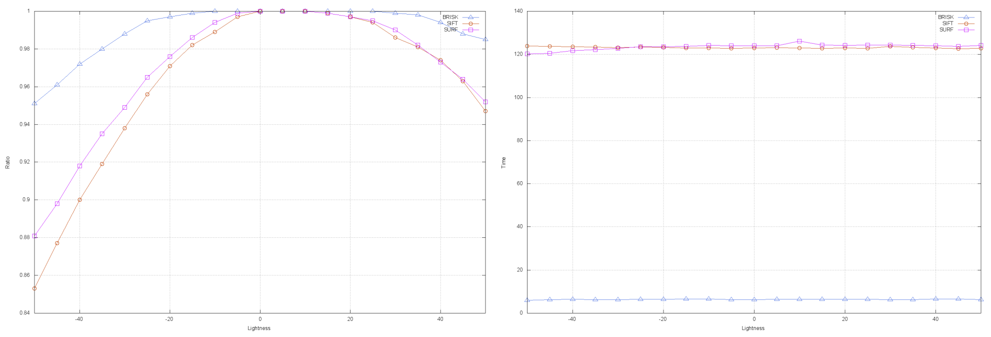


Figure 5. Mean Correct Keypoints Ratio (on the left) and algorithm run time in milliseconds (right) after amending pixels intensity.

(SIFT). Considering, that the overall procedure matches hundreds keypoints at once, and in fact only few of them are necessary to properly recognise the product, then it is obvious that the BRISK algorithm is a choice for the problem described in this paper. Not only, the robustness of BRISK allows it to be run in a mobile environment, but also good keypoints matching ratios are satisfactory. If more accuracy is needed, the SURF algorithm also performs in acceptable speed.

Table 1. Comparison of keypoint detector and matching algorithms. Table shows ratio of correct key points (the higher the better), and calculations time (the lower the better).

Algorithm	Ratio				Time (in ms)			
	mean	σ	*min*	*max*	*mean*	σ	*min*	*max*
BRISK	0.1904	0.1386	0.02	1.0	22.4	39.0	1.1	239.7
SIFT	0.2297	0.1709	0.02	1.0	744.2	1188.9	5.9	4856.3
SURF	0.2173	0.1332	0.02	1.0	429.9	639.6	3.2	2710.5

Examples of the final result of whole recognition procedure is shown in Figure 6. The reference, learning image is shown on the picture below the testing image. The lower detected box in testing image is slightly different, it's height is bigger than original learning image.

Complexity of the whole recognition procedure mainly depends on the amount of detected keypoints in the captured image. If the image is complex, the more key-points are detected. Bigger amounts of keypoints cause a reduction of recognition speed, due to the fact that all these points have to be compared with all keypoints from training images. Also, the amount of training images is a key factor of the recognition time for the same reason. Other parameter that counts up to the recognition time is image divider (d) value. Example values of the recognition times of real-life images, running the whole procedure on iPad[7] mobile device and SURF keypoints detector, are presented in Table 2.

The memory footprint however is not an issue in this scenario. Amount of keypoints is not big, for real-life images this does not exceed several kilobytes in size, thus considering size of nowadays mobile phones memory it is just a fraction that can be omitted.

Nevertheless, further study is needed to correctly define the impact of these factors on the whole recognition procedure times.

Table 2. Example real-life images recognition times in *ms* depending on image complexity (# of keypoints detected) and startup image-divider d, using SURF keypoint detector, running on iPad device. The amount of images in training database is 7.

image resolution	# of keypoints	$d = 1$	$d = 2$	$d = 4$	$d = 8$
1000x800	1,487	1,074	3,961	9,966	23,987
1000x865	1,662	2,009	4,006	8,993	22,929
750x759	1,832	1,991	4,003	10,035	27,288
1000x800	1,868	2,025	4,963	10,987	28,044

6. Conclusions

The problem of proper recognition and counting products or visible branding logos in user surroundings can be solved by the algorithm from section 4.2. Shapes, representing branding logos, are detected using SURF or BRISK algorithm with very good rate, it's low computation time allows the user to run it on a mobile device, such as mobile phone. Proper and fast keypoints matching adds for the user possibility to capture the image in different angles and light intensities, without any calibration. However, it is still necessary to perform further study of the performance of this algorithm in a real-life scenarios and images.

Acknowledgements

This work was supported by the MB/WI/3/2012 and S/WI/1/2013. Author would like to thank QBurst Poland for help in the problem definition.

[7] machine string is *iPad3,4*

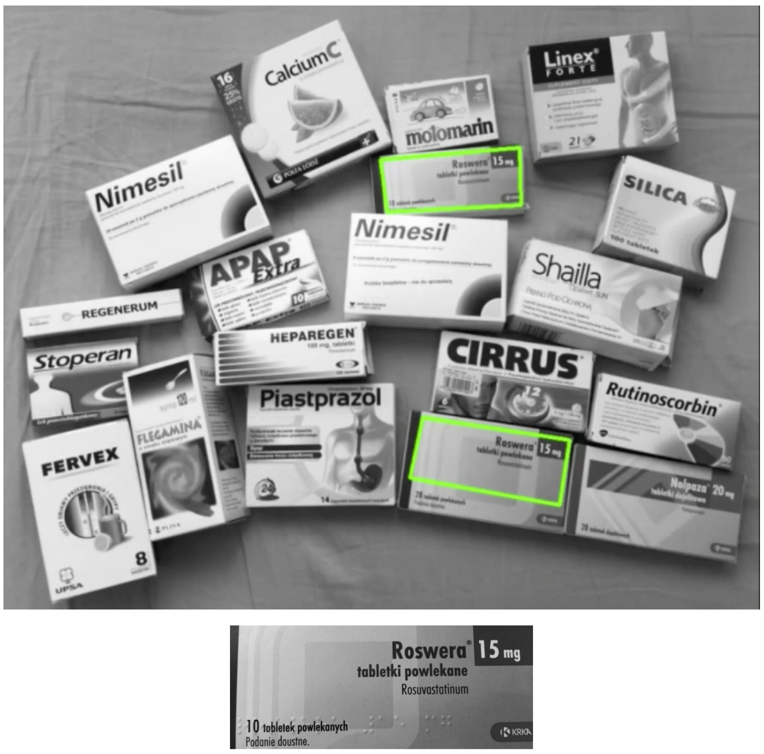


Figure 6. Example recognition marked by gray lines (top) of the reference image (bottom). Both two boxes were detected on the testing image.

References

[1] M. Skoczylas, W. Rakowski, R. Cherubini, and S. Gerardi. Unstained viable cell recognition in phase-contrast microscopy. *Opto-Electronics Review*, 19(3):307–319, 2011.

[2] D. Ding, J. Yoon, and C. Lee. Traffic sign detection and identification using SURF algorithm and GPGPU. In *SoC Design Conference (ISOCC), 2012 International*, pages 506–508, 2012.

[3] J. Pan, W. Chen, and W. Peng. A new moving objects detection method based on improved SURF algorithm. In *Control and Decision Conference (CCDC), 2013 25th Chinese*, pages 901–906, 2013.

[4] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[6] S. Leutenegger, M. Chli, and R. Siegwart. BRISK: Binary Robust invariant scalable keypoints. *Computer Vision, IEEE International Conference on*, 0:2548–2555, 2011.

[7] H. Li, T. Xu, J. Li, and L. Zhang. Face recognition based on improved surf. *2013 Third International Conference on Intelligent System Design and Engineering Applications (ISDEA)*, pages 755 – 758, 2013.

[8] B. Besbes, A. Apatean, A. Rogozan, and A. Bensrhair. Combining surf-based local and global features for road obstacle recognition in far infrared images. *2010 13th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1869 – 1874, 2010.

[9] Z. Zheng, H. Zhang, B. Wang, and Z. Gao. Robust traffic sign recognition and tracking for advanced driver assistance systems. *2012 15th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 704 – 709, 2012.

[10] Z. Wang, H. Xiao, W. He, F. Wen, and K. Yuan. Real-time sift-based object recognition system. *2013 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1361 – 1366, 2013.

[11] S. Yu, S. Zheng, H. Yang, and L. Liang. Vehicle logo recognition based on bag-of-words. *2013 10th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 353 – 358, 2013.

[12] P. Lukashevich, B. Zalesky, and S. Ablameyko. Medical image registration based on surf detector. *Pattern Recognit. Image Anal.*, 21(3):519–521, September 2011.

[13] E. Oyallon and J. Rabin. An analysis and implementation of the SURF method, and its comparison to SIFT. *Image Processing On Line*, 2013.

[14] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *In European Conference on Computer Vision*, pages 430–443, 2006.

[15] C. Huang and C. Chen. Keypoint matching technique and its applications, 2009.

[16] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. *In Conference on Computer Vision and Pattern Recognition, Puerto Rico*, pages 1000 – 1006, 1997.

[17] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[18] M. Fischler and C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[19] H. Sergieh, E. Egyed-Zsigmond, M. Doller, D. Coquil, J. Pinon, and H. Kosch. Improving SURF Image Matching Using Supervised Learning. In *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*, pages 230–237, 2012.

[20] Y. Peng, J. Yong, W. Dong, H. Zhang, and J. Sun. Technical section: A new algorithm for boolean operations on general polygons. *Comput. Graph.*, 29(1):57–70, February 2005.

AUTOMATYCZNE ROZPOZNAWANIE WIELU MAREK W OBRAZACH NA URZĄDZENIACH MOBILNYCH

Streszczenie: Widoczność produktu na półce jest ważnym zadaniem nowoczesnych zasad marketingu. Bardzo często firmy mają umowy ze sprzedawcami, że konkretny produkt będzie widoczny na półce w określonym procencie w stosunku do innych widocznych produktów. To banalne zadanie sprawdzenia liczby produktów lub widocznych logo marki, jest wykonywane ręcznie przez biegłego rewidenta, który sprawdza, czy warunki umowy są spełnione. Nie istnieje łatwy, mobilny mechanizm pozwalający w prosty sposób policzyć zdefiniowane produkty, które są widoczne w otoczeniu użytkownika. Taki scenariusz może jednak zostać osiągnięty przy użyciu nowoczesnych telefonów komórkowych. Za pomocą kamery można uchwycić obrazy otoczenia, a następnie wykorzystać moc obliczeniową urządzenia mobilnego do wykrywania produktów na obrazach, by finalnie obliczyć ilość widocznych produktów. W tym celu detektory punktów kluczowych w obrazach (np. algorytmy SIFT, SURF lub BRISK) są wykorzystywane do tworzenia bazy danych obrazów produktów, a wyodrębnione deskryptory punktów kluczowych są przechowywane. W nowym obrazie punkty kluczowe znajdowane są przy użyciu tego samego detektora, ale aby uniknąć problem wykrywania wielu identycznych punktów kluczowych, obraz jest podzielony i analizowany stosując przesuwne okno. Punkty kluczowe znajdujące się wewnątrz okna są wyodrębniane i są rozważane jako kandydaci występujące na obrazach treningowych. Przy wystarczającej ilości potwierdzonych punktów obliczane jest przekształcenie perspektywy i jeśli wykryte rogi są prawidłowo ukształtowane to produkt jest oznaczony jako rozpoznany. W tej pracy zostanie zaprezentowany algorytm, który umożliwia w środowisku mobilnym rozpoznawanie oraz liczenie widocznych produktów w otoczeniu użytkownika.

Słowa kluczowe: monitoring produktów opakowań logo rozpoznawanie detekcja punktów kluczowych urządzenia mobilne

Artykuł zrealizowano w ramach pracy badawczej MB/WI/3/2012 oraz S/WI/1/2013.

LISTA RECENZENTÓW / THE LIST OF REVIEWERS (2013)