SCREEN KEYBOARD ARRANGEMENT OPTIMIZATION FOR POLISH LANGUAGE

Michał Wołosik, Marek Tabędzki

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

Abstract: The aim of this work was to find screen keyboard arrangement optimal for Polish language. This study adopted a standard shape and organization of the keyboard, the task is therefore only for identifying the best permutations of keys. Only the alphabet keys and five selected punctuation marks were permutated. In order to accomplish this task, machine learning methods were used: genetic algorithms and simulated annealing. Fitness function is based on two literary works and one technical document. The following criteria were used: of distance, the writing direction and row weights. The application prepared for the experiments was developed in Java. The paper describes used algorithms and obtained results. Best found arrangement would shorten the time to input sample texts by about 30% (assuming adequate accustom of the new layout by the writer).

Keywords: keyboard arrangement problem, genetic algorithms, simulated annealing

1. Introduction

The most popular input device, that among other things is used to enter the text, is a computer keyboard. Common alternative for a classic physical keyboard is its screen counterpart, that usually can be displayed on the monitor and managed with a mouse. Almost every country has its own key arrangement which is optimal for the local language. What's interesting, for some reason it's different in Poland so this subject is worth looking into and perhaps there is something that could be done about it.

The goal of this work is to develop keyboard arrangement that would be optimal for Polish language. The task is connected to minimizing lengths of moves that need to be performed by mouse pointer in order to rewrite given text. There are also other less important criteria that might be taken under consideration that define difficulty of typing the text. In this work keyboard that is being optimized is of standard rect-angular shape and the same set of keys that are present in QWERTY "programmer" version arrangement.

Advances in Computer Science Research, vol. 13, pp. 75-93, 2016.

In order to achieve mentioned goal it is necessary to use machine learning methods like evolutionary algorithms or simmulated annealing, which would train keyboard arrangement with sample text blocks that are well representants of given language characteristics. For instance, it could be some classic novels. Polish texts that have been used in this work are among the others: "With fire and sword" by Henryk Sienkiewicz (*Ogniem i mieczem*) and "Sir Thaddeus" by Adam Mickiewicz (*Pan Tadeusz*).

In the next section you can find a short review of typical approaches and algorithms that are usually used to solve keyboard arrangement problem (KAP). Then, the details about permutational representation of the chromosome are explained. Following section is dedicated to the topic of multicriterial evaluation function. Then, the optimization methods that were implemented and used for experiments are described. Final sections contain results of the tests, and present and describe optimized keyboard arrangement – achieved goal of this work. It is followed by a short summary of what have been done and some loose conclusions.

2. KAP

In Poland, the most popular arrangement is called by its first letters of upper row keys – QWERTY keyboard in so called "programmer" version, or QWERTZ in older devices. In French-speaking countries most commonly used layout is AZERTY [16] while in English-speaking countries many kinds of QWERTY and its rival DVORAC [2, 4] can be encountered.

Keyboard arrangement problem (KAP) for certain language is a task in which evolutionary algorithms are usually applied. The key issue here is to determine appropriate individual representation as chromosome. One way of doing this, what was proposed in [6], is permutation. Next thing, that has to be established is an evaluation function, which would measure optimization level of current keyboard arrangement. This requires answering the question what ergonomic factors should be considered. It mostly depends on input that keyboard operates on. Wheter all fingers of both hands can be used, only two thumbs like in [9] or maybe text can be typed with just one pointer.

In literature, in conjunction with problem that is being discussed in this work, beside usage of genetic algorithms, applications of other methods can also be found. Very common are simulated annealing [1, 3, 8] and ant colony algorithms [14].

2.1 Chromosome representation and typing simulation

Experiments were conducted for PC screen keyboard with standard rectangular shape and the same set of keys that can be found in QWERTY "programmer" arrangement. Similarly how it has been done by Julstrom, Goettl and Brugh in [6], chromosome were encoded as permutation of keys. Index of each element is connected to its position on keyboard. Permutation includes 26 keys that represents latin alphabet characters and 5 punctuation keys (in [6] there was only 4). Size of problem space to search is: $31! \approx 8.22 \times 10^{33}$

Application created for experiments, in order to rewrite exemplary text block coded in UTF-8, is able to use keys that are highlighted in Figure 1. Light grey ones are part of chromosome and the darker ones have fixed positions.

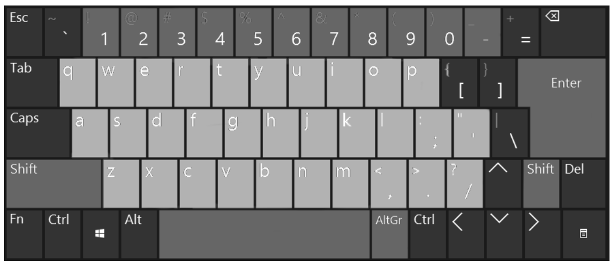
Esc ~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + =
Tab q w e r t y u i o p { [}] Enter
Caps a s d f g h j k l : ; " ' | \
Shift z x c v b n m < , > . ? / Shift Del
Fn Ctrl Alt AltGr Ctrl

Fig. 1. Movable and fixed keys

Each following recognizable character has to be mapped on a proper keys sequence. For instance for digit or small letter it will simply be one key that represents this digit or letter, and for capital letter it would be sequence of Shift key and this letter key.

3. Evaluation function for SFKL problem

Experiments were conducted for keyboard that is controlled with mouse pointer. In literature [3, 8] this approach is referred as single-finger keyboard layout problem (SFKL). This is important regarding cost calculation and ergonomics of typing. Chromosome evaluation function should estimate difficulty of rewriting given text block. Therefore, its global minimum has to be found.

3.1 Text blocks used for experiments

Texts that are going to be used to train keyboard arrangements should reflect characteristics of given language as well as possible. Set of three text blocks[1], that were used in this work contains following files: ogniem-i-mieczem-dwa-tomy.txt (With fire and sword), which includes 1719477 characters, pan-tadeusz.txt (Sir Thaddeus) with 503759 characters and wytyczne_informatyka-15.txt (Guidelines and advices on diploma work preparation) with 23850 characters. Chart in Figure 2 presents averaged probabilities of each key occurrence during rewriting of mentioned texts, and Figure 3 shows probabilities of pair of keys occurrences (one key after another). Charts do not include keys that are not part of chromosome.

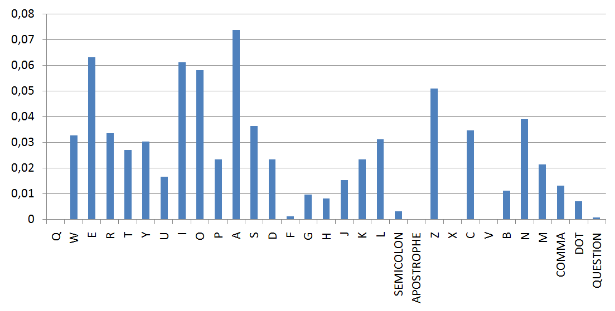

Fig. 2. Keys probabilities

Buttons which among the others refer to Polish diacritical characters have high frequency. For example key A has to be clicked each time when ‘a’ or ‘ą’ character occurs in the text. Therefore characteristics of keys typing frequency during writing Polish text with a keyboard should not be confused with characteristics of characters occurrences frequency in this language.

Four most probable pairs of keys that are clicked one after another are: $\langle I,E \rangle$, $\langle N,I \rangle$, $\langle O,W \rangle$, $\langle Z,E \rangle$ with probabilities of 0.01943, 0.01371, 0.00987, 0.0089 respectively.

[1] Texts “With fire and sword” by H. Sienkiewicz and “Sir Thaddeus” by A. Mickiewicz have been downloaded from `https://wolnelektury.pl/`. The third text block is a dokument that contains guidelines and advices on diploma work preparation which is available on `http://wi.pb.edu.pl/`.

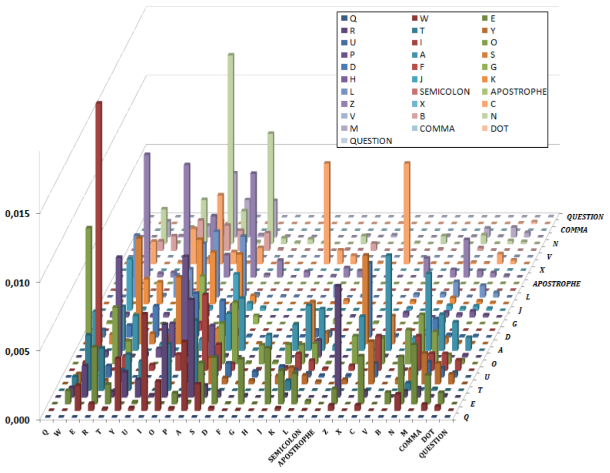

Fig. 3. Probabilities of pair of keys, first in row

For each of analyzed text blocks, proportions of concrete keys and pairs of keys frequencies are almost identical. In all cases key A is the most commonly clicked button and pair $\langle \mathrm{I}, \mathrm{E} \rangle$ is the most frequent.

3.2 Multicriterial evaluation function

Developed fitness function is composed of 3 different criteria, which are usually used for single-finger keyboard optimization problem. They are described below.

Distances criterion considers length of path that mouse pointer needs to travel by moving from previous to next keys. In [8] has pointed out, that "the s-finger keyboard layout problem can be modeled in terms of the Quadratic Assignment Problem (QAP)", which involves assigning n facilities to n locations in optimal way. In [3, 8] researchers mentioned two square matrixes: flow matrix, in which cell c_{ij} contains probability of flow between objects i and j, and also distances matrix in which d_{kl} is euclidean distance between positions k and l. Whole cost of rewriting the text for this criterion given in [8] and [3] is the sum of multiplications of cells corresponding to the same pairs of keys from both matrixes. Taking that n is the quantity of keys and $p(x)$ is the index of key x in permutation, formula looks like this:

$$D = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} d_{p(i)p(j)} \tag{1}$$

Criterion of hit direction – in [8] researchers concluded, that it is better if the direction of mouse pointer movement is corresponding to the direction of natural writing. For Polish language it is from left to right. The pair of keys I and E has the highest probability (0.01943) of occurrence one after another. Therefore, expected situation after optimization process for this criterion is arrangement where key corresponding to letter I is on the left from key corresponding to letter E. Formula below shows the penalty that is added:

$$S = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} v_{ij} \tag{2}$$

where: $v_{ij} = \begin{cases} 1, & \text{if } x_i - x_j > 0 \\ 0, & \text{otherwise} \end{cases}$

Row weights – the last criterion proposed in publication [8]. It is based on assumption that more frequent keys should be placed in rows which are more convenient to use. According to the most researchers opinion usage of the middle row is more comfortable than the other two, so the most frequent keys should be put in there. Nevertheless such solution would cause problems regarding distances criterion. Keys that correspond to Polish diacritical characters are the most frequent ones and putting them in the middle row their distances from fixed Alt key (which is below the bottom row) would be increased. Therefore it has been decided that penalties would be given for placing frequent keys in upper row only:

$$R = \sum_{i=1}^{n} f_i w_i \tag{3}$$

where:

- f_i – probability of pressing key i
- $w_i = \begin{cases} 2, & \text{if key } i \text{ is placed in the upper row} \\ 1, & \text{otherwise} \end{cases}$

The final form of the fitness function which is based on previously mentioned criteria:

$$F = aD + bS + cR \tag{4}$$

where: a, b and c are factors (weights of importance) of each criteria.

For cosmetic reason, these factors should sum up to 1. Optimal weights of criteria importance were found experimentally in the analogical way to what has been described in [8]. Test keyboard arrangement optimization algorithms has been run iteratively, each time for different set of discussed factors, starting from $a = 1$ after which factors b and c were increased by small values subtracted from a. Expected results for each criterion were met with $a = 0.8$, $b = 0.15$ and $c = 0.05$. Fitness function values will later on be presented in proportion to the fitness value of QWERTY programmer arrangement.

3.3 Typing time estimation, Fitts's law

Typing time is very important in keyboard optimization evaluation. Theoretically the distances criterion could be substituted with it. In [10] many different Fitts's law – based methods of mouse pointer travel time estimation (form one key to another) were described. Taking that A is the distance between centers of source and the target keys and W is the width of the target key, this is the formula that were used in this work:

$$T = \begin{cases} \frac{10}{49}\log_2\left(\frac{A}{W}+1\right), & \text{if } A > 0 \\ 0.127, & \text{if } A = 0 \end{cases} \tag{5}$$

4. Used optimization algorithms

Within this work, experiments were conducted with genetic algorithms with and without adaptive methods of mutation operator. Simulated annealing algorithm also has been used.

4.1 Genetic algorithms

Usage of evolutionary algorithms is very common in context of KAP. These methods operate on some population of chromosomes simultaneously. In each iteration (generation) to selected parents individuals from population, crossover and mutation operators are applied. Children population is being created in this way. Next step involves so called succession which is replacing some old individuals from actual population with the new ones. These steps are repeated till the specified stop condition is satisfied, with hope that the next generations will "produce" better specimens. Due to probabilistic nature of this algorithms they should be executed many times.

Selection and succession operators Some basic models of succession and selection operators are described by Wierzchoń in [15]. For experiments in this work selection SUS and rank succession were used.

Proportional selection, which extension is SUS [12], chooses individuals to parents population with probability $P_i = f_i/F$, where F is the sum fitness of whole population and f_i is the fitness of i-th individual. It should be remembered that according to chosen take on the problem, fitness function determines the cost, so before performing selection this function need to be transformed for each specimen: $f_i' = f_{AVG}/f_i$, where f_i' is the new fitness and f_{AVG} is average fitness.

SUS (Stochastic Universal Sampling) instead of using only one selection point on fitness sum space, it uses N equally spaced points like this, where N is a quantity of parents population. Method randomly chooses only one value from range $[0, 1/N]$, which becomes the first point. After that it generates next $N-1$ points of selection, starting from the first, incrementing value by interval $1/N$. Finally, N individuals addressed by these points being placed in their fitness range, are chosen to parents population. Figure 4 shows example situation where population size is 10 and $N = 6$.

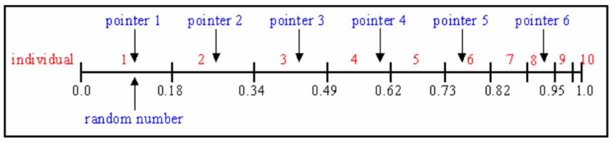

Fig. 4. Stochastic Universal Sampling, source: [12]

Thanks to this strategy there is lesser probability of premature convergence of evolutionary algorithm, because there is a big chance for weaker individuals to be chosen to parent population. In this way SUS maintains a large diversity of population.

Rank succession on the other hand, which was used for experiments, sorts population by individuals fitness values with descending order. For each specimen concrete ranks are being assigned – the first gets value 1, next one gets previous value increased by 1, etc. Ranks play the role of a new fitness function values. Following replacement goes on by the means of roulette wheel method. Individuals for replacement are chosen with probability $P_i = f_i/F$. The use of ranks makes it easier to distinguish better individuals from worse ones in population of low diversity.

Crossover and mutation operators The role of crossover is to exchange features between particular individuals – children are created by exchanging parents genes. Mutation, by contrast, makes small modifications of single individuals [15] and is usually performed with very small probability.

In experiments, for solutions coded as permutations, following crossover operators were used:

- UX (*Uniform crossover*) – at first, it iterates through both parents and copies the i-th element from the first or second one (randomly selected) on the i-th position of child chromosome. If on this position both elements from both parents are already in the child, an empty space is left. After that, all blank places are randomly filled with the rest available elements [6, 8, 11].
- PMX (*Partially mapped crossover*) – in detail, method was described in [11].
- OX (*Order crossover*) – at first a random segment is being copied from the first parent to a child. After that, missing elements are taken from second parent, starting right after the area of chosen segment and then continuing from the beginning of permutation [11].

In experiments, for chromosomes coded as permutations, following mutation operators were used:

- SWAP – it is based on swapping places of two randomly selected keys of permutation [8, 11]
- SCRAMBLE – elements on k randomly selected positions are randomly permuted (within selected positions only) [11]
- INSERTION – random element is put on a new randomly selected position [11]. In [8] this mutation has been used In slightly different form.
- SHIFT – random segment of permutation is moved to a different location [13].
- INVERSION – the order of random segment elements is reversed [11].

Diversity measurements of population The need of taking measurements of population diversity has many reasons of diagnostic nature. As [5] states, diversity measurements can be divided on: genotypic – which informs about differences between chromosomes, and phenotypic – that deals with differences in fitness values of population individuals. The first ones are usually more time consuming and the second ones are less accurate.

In this work, genotypic measure that were computed during the experiments is modified version of linear diversity measure for permutations that has $O(n)$ time complexity. Its original version was designed by Dudek M. and presented in [5].

$$LMRDP_{POP} = \frac{1}{n+1}\left(\frac{PZ_{dif}-1}{n-1} + \sum_{i=1}^{n}\frac{LP_i-2}{n-3}\right), \quad LMRDP_{POP} \in [0,1] \tag{6}$$

where:

- n – number of nodes – problem size (for example, number of keys in chromosome of coded keyboard)
- LP_i – number of different connections from node i, or 2 if all are identical
- PZ_{dif} – number of different first nodes of permutations within a population, possible values are in range $[1,n]$

Diversity in population is equal to 0 if all chromosomes are identical. On the other hand, the measure takes the value of 1 when starting nodes of permutations do not repeat and each node has connection to every other nodes within a population permutations. Maximal number of such connections for one node is $n-1$.

4.2 Adaptive methods for mutation

Publication [13] describes mechanisms, inter alia, AOC and MOS, which are based on usage of many mutation operators during one evolutionary algorithm. It suppose to increase the chance of leaving the local optimum.

AOC (*Adaptive operator cycling*) – mechanism of exchange is run periodically, after each execution of specific number of generations – so called adaptive period, and at least one mutation has been performed since previous period has finished. Probability of operator exchange is based on success rate factor r_s, which is the ratio of success mutations to all mutations of current adaptive period. Formula looks like this:

$$P_{OC} = \frac{1}{2}\left(1 + \cos\left(r_s \cdot \pi\right)\right) \tag{7}$$

AOC2 slightly differs from metod described above. Attempt to exchange mutation operator occurs when in actual adaptive period all mutation were not successful.

MOS (*Mutation operators statstics*) – at the end of each adaptive period, this metod recalculates success rates r_{si} for each i-th mutation operator. Calculation is based on previous adaptive period. In each generation, with some r_{si} based probability, a choice is being made which mutation operator should be use on given parent individual. In order to give a chance for mutation with r_{si} equal to 0 to be selected, in [13] usage of b_i parameter was proposed. It is calculated at the end of each adaptive period for each i-th mutation.

$$b_i = (\text{rand}(0;1) - 0.5) \cdot a \tag{8}$$

$$P_{chi} = \begin{cases} \frac{\max(0; r_{si}+b_i)}{\sum_{j=1}^{n} \max(0; r_{sj}+b_j)}, & \text{for } \sum_{j=1}^{n} \max(0; r_{sj}+b_j) \neq 0 \\ \frac{1}{n}, & \text{for } \sum_{j=1}^{n} \max(0; r_{sj}+b_j) = 0 \end{cases} \tag{9}$$

where:

- $\text{rand}(0;1)$ – random value from $[0,1]$ uniform distribution
- a – parameter which was found experimentally in [13] as 0.02
- P_{chi} – probability of i-th mutation selection
- n – mutations quantity

4.3 Simulated annealing algorithm

One of the approaches used in the keyboard arrangement problem, is simulated annealing algorithm and hybrid solutions combining it with genetic algorithms [1,3,8]. This algorithm operates on one individual, every step trying to replace it with a neighbor that is created using chosen mutation operator. The method introduces a parameter called the temperature, which is gradually reduced at a certain number of iterations (epoch). At higher temperature the probability that an individual will be replaced by a worse neighbor is higher.

Other parameters of simulated annealing approach are: α – thermal melting point, i.e. the rate at which the temperature decreases (usually in the range 0.95–0.98), L_{min} – initial length of the epoch, i.e. the number of iterations at the same temperature, L_{max} – maximum epoch length, β – the rate at which the length of the epoch grows.

This research is based on acceptance probability (the probability of substituting the current individual with the one obtained as a result of mutation) taken from [1] and [3]. Besides the temperature, it also takes into account the difference in the fitness values of the two individuals. Assuming that f is the fitness of the current individual, f' is the fitness of considered neighbor, and T is the current temperature, the formula looks as follows:

$$P_{subst} = \begin{cases} 1, & \text{if } f' < f \\ e^{-\frac{f'-f}{T}}, & \text{if } f' \leq f \end{cases} \tag{10}$$

5. Experiments and result analysis

Application for testing purposes was written in Java (JDK ver. 1.7) using NetBeans IDE (ver. 8.0.2). Some parts of the algorithms, including the fitness function calculation, has been parallelized. Tests were performed on a machine with quad-core processor Intel Core i7-2630QM 2.00GHz. Tests were conducted for a total of 15 optimization approaches: GA with 9 combinations of crossover and mutation operators, 3 adaptive mutation operators without crossover, and simulated annealing with 3 different mutation operators. Each of the approaches was run 10 times for each of the three text blocks, and the results were averaged.

5.1 Genetic algorithms

In experiments, the following parameters of GA were adopted: population size of 200, SUS selection with parental pool of 50, and rank-based selection. Crossover probability was 0.9 and mutation probability was 0.15. Termination condition: reaching 2,000 generations. The column "Operators" of the following table contains information about which crossover and mutation operators were used. For the SCRAMBLE operator, parameter *k* was equal to 3.

Table 1. Results for genetic algorithms

Operators	Best initial	Average initial	Best final (MBF)	Average final	Gain	Running time [s]
UX, SWAP	0.86457	0.99367	0.69951	0.70412	19%	24.26917
UX, SCRAMBLE	0.86611	0.99456	0.69882	0.69996	19%	24.80797
UX, INSERTION	0.86581	0.99394	0.69882	0.73335	19%	26.65170
PMX, SWAP	0.85778	0.99408	0.69500	0.70019	19%	25.11970
PMX, SCRAMBLE	0.86669	0.99438	0.69537	**0.69747**	20%	24.97160
PMX, INSERTION	0.86321	0.99415	**0.69460**	0.70439	20%	24.76830
OX, SWAP	0.86522	0.99287	0.70184	0.71601	19%	25.34267
OX, SCRAMBLE	0.86923	0.99431	0.70205	0.70547	19%	24.79710
OX, INSERTION	0.85807	0.99361	0.70567	0.72059	18%	25.32400

There were no major differences in average values of the reached fitting. Best MBF of 0.6946 was obtained using PMX crossover and INSERTION mutation. However, the top average of the final fitness (of 0.69747) was for the SCRAMBLE mutation with the same PMX crossover operator. The worst results were obtained with OX crossover, for which none of the final fitness drops below 0.7.

The sharp decline in the cost function took place in the initial phase of the algorithm – the first 250 generations, and then decreased gently. Population converged rapidly to the same time. After about 250 generations, the value LMRDP stabilized more or less on the value of 0.3.

5.2 Genetic algorithms with adaptive mutation

In experiments using adaptive mutation operator, an adaptation period was of 200 iterations, and the algorithm was stopped after $4,000$ iterations. Adaptations AOC and AOC2 were using only SWAP and INSERT mutation operators, while MOS – all of listed in section 4.1. For SCRAMBLE mutation value *k* was set to 3.

Table 2. Results for genetic algorithms with adaptive mutation

Adaptation method	Best initial	Average initial	Best final (MBF)	Average final	Gain	Running time [s]
AOC	0.86325	0.99440	0.69119	0.75307	20%	105.4112
AOC2	0.86239	0.99479	0.69061	**0.74204**	20%	104.0792
MOS	0.86009	0.99279	**0.69019**	0.76860	20%	103.3855

The final value of MBF are comparable for all types of adaptation. It is noteworthy that the results were somewhat better than in the case of not using adaptation algorithms. The best mean fitness of the best individuals was obtained for MOS adaptation (it equals 0.69019). Interestingly, this method also gave the worst mean fitness of average individuals (of 0.76860), which means poor stability. In this case, the best value was obtained for AOC2 approach (of 0.74204).

Success rate of the GA with the AOC adaptation, usually reached a low value already at the end of the first adaptive era, and in successive epochs further decreases and stabilizes around the value of 0.1. Hence the probability of substitution one mutation operator to another was high all the time, and it was swapped at the end of almost every epoch (ie. every 200 generations).

In the case of AOC2, in the initial phase of the algorithm, when there is a sharp decline in the cost function, the mutation operator was not changed even once. In the later stage of evolution, as the fit function decreased more gently, the operator was changed more often. Using this method, substitution of the mutation operator could be observed after the 2000 generation.

With the MOS adaptation approach, all mutation operators implemented in the application were used. In each generation of evolutionary algorithm, the decision which mutation operator to use was taken repeatedly. This results in that new individuals are produced in a less stable manner – diagrams were more irregular.

5.3 Simulated annealing

For simulated annealing approach the following parameters were adopted: initial temperature of 0.1 and the final temperature of 10^{-5}, whose achievement was a termination condition. The initial size of epoch was 10 iterations, and the maximum was 500. The cooling coefficient was $\alpha = 0.95$, and the growth factor was $\beta = 1.2$. This gives a total of $81,183$ iterations.

Table 3. Results for simulated annealing

Mutation	Initial	Final	Gain	Average time [s]
SWAP	**0.98105**	**0.68947**	**30%**	**295.9947**
SCRAMBLE $k = 3$	1.00439	0.69017	31%	315.0103
INSERTION	1.01540	0.71162	30%	294.1847

The best mean fitness of final individuals for all tests was obtained for SWAP mutation. It equals 0.68947. Simulated annealing algorithm yielded slightly better results than adaptive mutation approach.

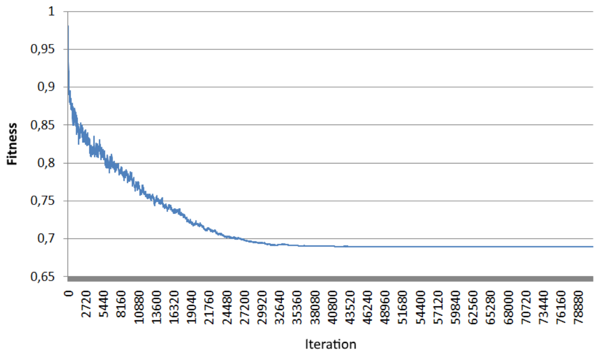

Fig. 5. Fitness for simulated annealing with SWAP mutation

As a result of the specificity of the simulated annealing algorithm, in the initial phase (at high temperatures) the probability to replace the current individual by worse one is high. This is reflected in the chart above – the beginning is more jagged.

5.4 The best keyboard arrangements found

The best keyboard arrangement was obtained using the simulated annealing algorithm. For the OM text ("With Fire and Sword"), the same arrangement was found using SA algorithm with SWAP and SCRAMBLE mutation operators. Its fitness value for this text was 0.68759101. Best keyboard for PT ("Sir Thaddeus") was obtained using SA with SWAP mutation (fitness of 0.69209257). The best fitness for the text WI (diploma guidelines) was 0.67895839. It was achieved by SA with SCRAMBLE mutation.

The following table shows the values of matching the best found keyboard arrangements for each of the test texts. Table rows correspond to the keyboards, and the column – blocks of text.

Table 4. Fitness of the top keyboards tested on different blocks of text

	OM	PT	WI	Average
OM keyboard	0.68759101	0.71119728	0.69419239	0.69766023
PT keyboard	**0.69542199**	**0.69209257**	**0.69207537**	**0.69319664**
WI keyboard	0.72413948	0.73202195	0.67895839	0.71170661

Keyboard trained on the WI text has poor fit for other blocks of text. Apparently it overfitted to the content of this particular document. The best average cost of rewriting all the texts was obtained with a keyboard optimized on the PT block. Its arrangement is shown on Fig. 6.

It is worth to compare the optimized keyboard with the other ones. Unfortunately, as noted by Herma in [7] there are no specific arrangements dedicated to the Polish language. Therefore, a comparison with to other popular arrangements was conducted – QWERTY and DVORAC. Table 5 summarizes the matching function (F_{FIT}) of said keyboards to that shown in Fig. 6 (PL_OPT). The table also includes time needed to rewrite each of the test blocks of text, estimated on the basis of the Fitts's law.

As can be seen, optimized keyboard has been rated as better than the QWERTY layout by about 30% from the perspective of the text of the novel "With Fire and

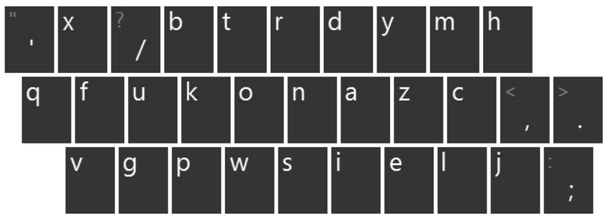

Fig. 6. The best keyboard arrangement for Polish language found

Table 5. Comparison results for different keyboard arrangements

	OM		PT		WI	
	F_{FIT}	time [h]	F_{FIT}	time [h]	F_{FIT}	time [h]
QWERTY	1.0000	197:29	1.0000	58:01	1.0000	2:47
DVORAC	1.0748	208:03	1.0923	61:11	1.1208	3:00
PL_OPT	0.6954	158:52	0.6921	46:50	0.6921	2:14

Sword" and about 31% from the perspective of the poem "Sir Thaddeus" and the document WI (diploma guidelines). Using keyboard arrangement tuned to the Polish language, the average time gain (in relation to the QWERTY keyboard) is 19.52%.

6. Conslusions and final thoughts

The goal of this work was to develop keyboard arrangement optimal for Polish language. Taking the standard, rectangular layout and set of keys of QWERTY keyboard (Polish programmer variant), optimal permutation was searched for, taking into account criteria of distance and position. For this purpose, authors used methods of machine learning such as evolutionary algorithms and simulated annealing.

This goal was fully achieved. Among numerous arrangements, found in experiments with different algorithms and sets of text, one was selected, featuring the best mean fitness. In other words, selected arrangement is well suited to the specifics of the Polish language, and is not overfitted to a specific block of text. The multicriterial function, used to evaluate keyboard arrangement, showed that the found keyboard gives more that 30% better fit over the standard QWERTY arrangement. Basing on the estimation of the typing time, calculated according to the Fitts's law, was shown that the time cost of writing text in is smaller than in the case of QWERTY and DVORAC keyboards. The results are very promising and encourage to the practical realization of such a keyboard arrangement.

There are many possibilities of future research. Authors consider conducting experiments using other algorithms or different sets of text. The most interesting research direction is to give up standard keyboard layout and try to find a different one, of custom shape.

It should be noted however, that regardless of the calculated costs and fitness, a very important factor is the habits of the target users. If you are not accustomed to the new arrangement, you will have to spend some time relearning the keyboard layout to achieve greater typing speed than with QWERTY. Is the final profit from using the optimized keyboard is worth the time spent on getting used to it? It depends on the individual needs and expectations of each of us.

References

[1] Navid Samimi Behbahan. Optimization of farsi letter arrangement on keyboard by simulated annealing and genetic algorithms. *Majlesi Journal of Multimedia Processing*, 2012.

[2] Randy Cassingham. The dvorak keyboard. `http://www.dvorak-keyboard.com`. Retrived: 2016-01-23.

[3] Mauro Dell'Amico, José Carlos Díaz Díaz, Manuel Iori, and Roberto Montanari. The single-finger keyboard layout problem. *Computers & Operations Research*, 36(11):3002–3012, 2009.

[4] Richard Dickenson. Did sholes and densmore know what they were doing when they designed their keyboard? *ETCetera – Journal of the Early Typewriter Collectors Association*, 1989.

[5] Bogusław Filipowicz, Wojciech Chmiel, Maciej Dudek, and Piotr Kadłuczka. Efektywność wielopopulacyjnego algorytmu ewolucyjnego dla zagadnień permutacyjnych. *Automatyka/Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie*, 15:147–158, 2011.

[6] Jeffrey S Goettl, Alexander W Brugh, and Bryant A Julstrom. Call me e-mail: arranging the keyboard with a permutation-coded genetic algorithm. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 947–951. ACM, 2005.

[7] Mariusz Herma. Na tropie polskiej klawiatury. `http://www.polityka.pl`. Retrived: 2016-09-05.

[8] Manar I Hosny, Nourah Alswaidan, and Abir Benabid Najjar. An optimized single-finger arabic keyboard layout. In *Science and Information Conference (SAI), 2014*, pages 321–328. IEEE, 2014.

[9] Wojciech Kulik. Klawiatura ekranowa kalq przyspieszy o 34% pisanie na tabletach i smartfonach. `http://www.benchmark.pl/`. Retrived: 2016-09-01.

[10] Yanzhi Li, Lijuan Chen, and Ravindra S Goonetilleke. A heuristic-based approach to optimize keyboard design for single-finger keying applications. *International Journal of Industrial Ergonomics*, 36(8):695–704, 2006.

[11] Alberto Moraglio and Riccardo Poli. Geometric crossover for the permutation representation. *Intelligenza Artificiale*, 5(1):49–63, 2011.

[12] Tania Pencheva, Krassimir Atanassov, and Anthony Shannon. Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets. In *Proceedings of the Tenth International Workshop on Generalized Nets, Sofia*, pages 1–7, 2009.

[13] Aleksandar Prokopec and Marin Golub. Adaptive mutation operator cycling. In *Applications of Digital Information and Web Technologies, 2009. ICADIWT'09. Second International Conference on the*, pages 634–639. IEEE, 2009.

[14] Marc Oliver Wagner, Bernard Yannou, Steffen Kehl, Dominique Feillet, and Jan Eggers. Ergonomic modelling and optimization of the keyboard arrangement with an ant colony algorithm. *Journal of Engineering Design*, 14(2):187–208, 2003.

[15] Sławomir Tadeusz Wierzchoń. *Sztuczne systemy immunologiczne: teoria i zastosowania*. Akademicka Oficyna Wydawnicza EXIT, 2001.

[16] Wikipedia. Keyboard layout. `https://en.wikipedia.org/wiki/Keyboard_layout`. Retrived: 2016-08-23.

List of Abbreviations

AOC	Adaptive operator cycling
GA	Genetic algorithm
KAP	Keyboard arrangement problem
MBF	Mean best fitness
MOS	Mutation operators statistics
OX	Order crossover
PMX	Partially mapped crossover
QAP	Quadratic assignment problem
SA	Simulated annealing
SFKL	Single finger keyboard layout
SUS	Stochastic universal sampling
UX	Uniform crossover

OPTYMALIZACJA UKŁADU KLAWIATURY EKRANOWEJ DLA JĘZYKA POLSKIEGO

Streszczenie Celem niniejszej pracy było opracowanie układu klawiatury ekranowej przeznaczonej dla języka polskiego. Przyjęto standardowy kształt i organizację klawiatury, zatem jest to zadanie wskazania najlepszej permutacji klawiszy, przy czym permutacji podlegały jedynie klawisze znaków alfabetu oraz pięć wybranych znaków interpunkcyjnych. W celu realizacji tak określonego zadania, posłużono się metodami uczenia maszynowego: algorytmami genetycznymi oraz algorytmem symulowanego wyżarzania. Funkcja dopasowania opiera się na dwóch utworach literackich oraz jednym dokumencie technicznym. Zastosowano kryteria odległości oraz lokalizacji klawiszy (biorąc pod uwagę kierunek pisania oraz wagi rzędów). Aplikację przygotowaną w celu wykonania badań eksperymentalnych opracowano w języku Java. W pracy opisano zastosowane algorytmy oraz przedstawiono wyniki uzyskane na drodze eksperymentów. Najlepsze znalezione układy pozwoliłyby skrócić czas wprowadzania przykładowych tekstów o około 30% (zakładając odpowiednie opanowanie nowego układu przez piszącego).

Słowa kluczowe: optymalizacja układu klawiatury, algorytmy genetyczne, symulowane wyżarzanie

Artykuł zrealizowano w ramach pracy badawczej S/WI/2/2013 i sfinansowano ze środków na naukę MNiSW.