A Deep Genetic Method for Keyboard Layout Optimization

Keren Nivasch
Computer Science Department, Ariel University
Ariel, Israel
kerenni@ariel.ac.il

Amos Azaria
Computer Science Department, Ariel University
Ariel, Israel
amos.azaria@ariel.ac.il

***Abstract*—The QWERTY keyboard layout that is commonly used today was designed, over 100 years ago, for typewriters rather than for modern keyboards. Over the decades, many people have tried manually to come up with better layout designs. Recently, researchers have also attempted to automatically find a better keyboard layout by using advanced algorithms. In this paper we propose the use of deep learning with a genetic algorithm for finding improved keyboard layouts. We also show that using an appropriate crossover routine, instead of the crossover routine previously used in the literature, significantly improves the performance of the genetic algorithm. Our method, which we call *MKLOGA*, produces a keyboard layout that outperforms previous layouts, including those found by other algorithms, according to the realistic *typing effort model* of carpalx. We provide an installation of our keyboard layout. MKLOGA might also be useful for developing good layouts for languages other than English, and possibly for other domains in which objects must be placed in predefined locations.**

***Index Terms*—Keyboard Layout, Genetic Algorithm, Neural Network**

I. Introduction

Despite recent advances in automatic speech recognition, keyboard input remains the most common method of text communication. While people use keyboards very often, most people do not pay much attention to the keyboard layout. This is unfortunate, since the keyboard layout has an impact not only on the typing speed, but also on wrist pain and repetitive strain injury (RSI) [1].

The modern *QWERTY* keyboard layout used today was introduced in the 1870's by Christopher Latham Sholes [2]. One common belief is that the QWERTY layout was designed to minimize type-bar jams by placing common letters far away from each-other [3]. Nevertheless, the QWERTY layout was designed for typewriters rather than keyboards; therefore, it is very likely that it is suboptimal for modern use.

In the early 1930's, August Dvorak introduced the keyboard layout known today as *Dvorak* [4], which he hoped would be more ergonomic and lead to faster typing. Even though the QWERTY layout is still the most common layout in use, most major operating systems offer the option of switching to the Dvorak layout. Nevertheless, Dvorak has not gained much popularity, probably because QWERTY is already so entrenched.

Since then, there have been other attempts at creating better keyboard layouts. A popular alternative to QWERTY and Dvorak is the *Colemak* layout [5], introduced by Shai Coleman in 2006. It maintains the position of 17 keys of QWERTY, including many keys commonly used for keyboard shortcuts, with the hope of making it easier to learn for people accustomed to the QWERTY layout. While Colemak is not officially supported by Windows, creating and installing a custom layout in Windows can be easily done with the Microsoft Keyboard Layout Creator. Once a layout is installed, the characters appearing on the physical keys will not match the virtual characters, but this is a very minor issue; on the contrary, it encourages early adoption of touch typing.

Subsequently there were several attempts to find better keyboard layouts by automating the process [6], [7]. While a brute-force search over all possible arrangements is not feasible, due to the astronomically large number of different arrangements, there are many optimization algorithms that can be used instead. One example of a commonly used and efficient class of optimization algorithms is the genetic algorithm.

Genetic algorithms belong to the larger class of evolutionary algorithms. They are a technique inspired by the process of natural selection, which are commonly used to generate high-quality solutions to optimization and search problems by relying on the biologically inspired operations of mutation, crossover and selection. In a genetic algorithm there is a "population" of candidate solutions, each of which has a set of characteristics that can be altered. There is an objective function that assigns a "fitness" value to each solution. One typically starts with an initial random population, which will probably have very low fitness. The algorithm proceeds in "generations"; each generation is obtained from the previous one by selecting the most fit candidates and generating new candidates by a process of crossover. In addition, random mutations are performed on the selected candidates before being added to the next generation. While most of the crossovers and mutations are likely to reduce the fitness of the candidates, a small fraction of them will yield more-fit candidates, and the improved traits will gradually spread throughout the population. Hence, as the generations progress, the overall fitness of the population will increase.

In this paper we present a method for optimizing keyboard layouts using a hybrid approach of deep learning and genetic algorithms. Our method is fast and therefore allows the use

of large corpora for training, as well as the use of complex fitness functions. One of the features of our method is the use of the *cycle crossover* routine [8], which greatly enhances the performance of the genetic part of the algorithm. We show that our method outperforms the state-of-the-art methods from the literature even when using their own metrics[1].

As mentioned in [7], keyboard layout optimization techniques might be useful for a wider class of problems in which there are objects that must be placed in predefined locations, the objects will be accessed one after the other in some order, and the goal is to optimize the placement of the objects. Real-life examples of this scenario include books in a library and products in a vending machine.

To summarize, the contribution of this paper is two-fold.

1) We propose the use of deep learning to assist in a genetic algorithm process for finding an improved keyboard layout.
2) We show that the cycle crossover routine significantly outperforms the crossover routine that was previously used in the literature.

II. Related Work

Genetic algorithms have been used for keyboard design optimization. Yin and Su [6] considered several scenarios for the general keyboard arrangement problem, such as single-character and multi-character keyboards, single-finger and multi-finger typing, and optimization according to different criteria, such as typing ergonomics, word disambiguation, and prediction effectiveness. They offered an evolutionary approach using a cyber swarm method and showed that it produces keyboard layouts that are better than existing ones. Other works that use genetic algorithms for keyboard optimization are [7], [9]–[11].

In particular, in their recent work, Fadel et al. [7] developed a genetic-based algorithm that is used to find better layouts than QWERTY and Dvorak. Their algorithm works by iteratively performing the operations of Selection, Crossover and Mutation, on a population of candidate layouts. They measure the fitness of a layout using a simple objective function that sums the Euclidean distances between every single character in the training corpus and the nearest finger to it. Using their method they find layouts that are better than QWERTY and Dvorak according to their objective function. They call the best keyboard layout they found "}.?BZQ".

Krzywinski [12] introduced *carpalx*, which includes a more realistic and complex objective function for evaluating layouts. The carpalx typing effort model is based on *triads*, which are three-character substrings formed from the training text. The effort associated with typing a triad has two components: effort to hit a key (independently of preceding and successive strokes) and effort to hit the group of keys. Independent effort is based on finger distance and includes hand, finger, and row penalties associated with that key. The effort associated with the group of keys considers their *stroke path*, which is evaluated by taking into account hand-alternation, row-alternation, and finger-alternation.

The carpalx model is highly parameterized, as the formula for the effort involves many weights whose value can be adjusted. Hence, the user can decide what is important to her layout, so the model can be made highly subjective. For more details on the computation of the carpax effrt model see [12]. For this project we left all the carpalx parameters with their default values and did not change them.

Due to its high complexity, the carpalx objective function requires excessive computing power (approximately 0.6 seconds on a computer with Intel Core i7 CPU).

The carpalx project also includes an implementation of a simulated annealing based method for finding good keyboard layouts. Carpalx has been used to construct layouts optimized for the Filipino [13] and Latvian languages [14].

A problem related to the keyboard layout optimization problem is the Quadratic Assignment Problem (QAP). In this problem there are n facilities and n locations, and there is a distance between each pair of facilities, as well as a flow between each pair of locations. The objective is to assign the facilities to different locations in order to minimize the sum of the distances multiplied by the corresponding flows. This problem is somewhat similar to the keyboard layout optimization problem: If the function we wish to minimize is the total movement of the fingers, then the keys and the finger base positions correspond to the facilities. There are several works that tackle the QAP problem with genetic algorithms [15]–[17].

There are also several previous works that combine genetic algorithms with deep learning. Sehgal et al. [18] use a genetic algorithm to find the values of parameters used in a reinforcement learning task related to robotic manipulation. Potapov and Rodionov [19] implement a genetic algorithm with a crossover operator that uses a deep neural network. Hu et al. [20] combine a genetic algorithm and deep neural network models to construct property diagrams for grain boundaries.

III. The MKLOGA Model

In this paper, we present our *Method for Keyboard Layout Optimization using a deep Genetic Algorithm (MKLOGA)*. The method improves the one described by Fadel et al. [7] in several aspects. First, MKLOGA uses a better crossover routine for generating a new layout from its parents, as explained in section III-A below. In addition, MKLOGA uses the complex and more realistic objective function of carpalx [12] for evaluating layouts. Due to the excessive computing power required by the carpalx objective function, MKLOGA includes several improvements to the genetic algorithm process, one of which is the use of deep learning. All MKLOGA software is available at `https://github.com/kerenivasch/MKLOGA`.

A. The Cycle Crossover Routine

As mentioned above, MKLOGA uses the *cycle crossover* routine of [8] for generating a new keyboard layout K_3 out of

[1] A link for installing the keyboard layout generated by our method on Windows is available at: `https://github.com/kerenivasch/MKLOGA`

two given keyboard layouts K_1, K_2. The crucial property of this routine (as opposed to the crossover routine of [7]) is that each key placement in K_3 is copied from either K_1 or K_2. As we show in Section V, the cycle crossover routine alone provides a significant improvement to the performance of the algorithm of [7].

We proceed to explain the cycle crossover routine for the sake of completeness. Let S be the set of symbols whose placement is allowed to change, and let P be the set of keys that can take symbols. We first pick a random parent K from among K_1, K_2, we call the other parent K'. Then we pick a random key $p_1 \in P$ and copy its symbol s_1 from K to K_3. Now we check in which key p_2, the symbol s_1 is located in K'. We copy the symbol s_2 of the key p_2 in K. Then we check in which key p_3, the symbol s_2 is located in K'. We continue this way until we return to p_1 and thus close a cycle. Hence, all the symbols in this cycle were copied from K.

If additional keys are left, we make another random choice for K, K' between K_1, K_2, and pick another random available key and repeat the process. This way every placement in K_3 has been copied from either K_1 or K_2.

Figure 1 shows an example of the cycle crossover routine. Here, the routine first picked parent 1, picked from it the letter `n`, and copied the letter `n` to the child. The routine checked the location of the letter `n` in parent 2; in that location, parent 1 has the letter `l`. The routine copied the letter `l` to the child, and checked its location in parent 2. In that location, parent 1 has the letter `z`. Continuing this way, the routine copied the letter `z` and then the letter `d` to the child, and then came back to the letter `n`, which was the initial letter copied from parent 1. This finished one cycle of the crossover routine. The routine then picked parent 2, and picked from it the letter `b`. Continuing as described before, the routine copied from parent 2 to the child the letters `b`, `q`, `a`, `h`, `f`, `y`, and `v`, and then came back to `b` and closed another cycle. This process continued until the child layout was complete.

B. Using The Carpalx Objective Function

As mentioned, in order to obtain an improved keyboard layout, MKLOGA uses the complex and more realistic keyboard effort model of carpalx [12] for evaluating layouts. Since the keyboard effort model requires excessive computing power to evaluate, MKLOGA also includes a neural network for fast estimation of the effort. The neural network is initially trained on randomly generated layouts. After the training, the model is saved in order to be used as the initial model for the genetic part of the algorithm. During the genetic algorithm process, the neural network is fine-tuned by retraining it with some of the best layouts found in the current generation using their true effort value. The input layouts for the neural network are represented using a one-hot representation, as a square $0/1$-matrix whose size corresponds to the number of key positions that are allowed to change. The use of the neural network allows us to evaluate the expensive effort function only on a small number of layouts, leading to a significant speedup of the running time.

The genetic algorithm of MKLOGA proceeds as a sequence of generations. Each generation consists of a population of n layouts. The first generation is generated randomly. In each generation the layouts are evaluated and sorted according to the neural network's estimation. In order to construct the new generation, r layouts are first generated randomly. The best m layouts of the previous generation pass automatically to the new generation, and they are also evaluated according to the true effort function. The best m' of these are used to generate $n - r - m$ new layouts using the cycle crossover routine. Each new generated layouts also undergoes a random number between 0 and t of random mutations. Each mutation consists of selecting 2 random keys and swapping them. Figure 2 shows a flowchart of the MKLOGA algorithm, and Figure 3 shows how a generation is constructed from the previous one.

IV. Experiments

We first evaluated the effect of using the cycle crossover routine. For this, we took the code of [7], and replaced their crossover routine with the one described in Section III-A. We carried out the two types of experiments that were made by [7]: changing only the positions of the letters of the standard layout (called "Letters Only" in [7]), and changing also the positions of the punctuation symbols ("Letters and Punctuation").

We then proceeded to implement MKLOGA. The first step of the implementation was to train a neural network on a data-set of 4800 randomly generated layouts labeled with their effort values. The neural network had a hidden layer of size 64 with ReLU activation. For the genetic part, we used a population size of $n = 5000$, the number of random layouts added in each generation was $r = 1000$, the parameter m was 250, and m' was 100. The maximum number of mutations was $t = 5$. We ran the genetic algorithm for 30 generations.

We allowed to change only the positions of the letters, except for one difference: Following the lead of some previous keyboard designs (carpalx [12], colemak [5]) we moved one letter from the top row of letters to the middle row, so that the top row contains 9 letters and the middle row contains 10 letters. The effort value was calculated using a corpus provided by [12] of size 267KB.

V. Results

A. The Cycle Crossover Routine

The cycle crossover routine led to a significant improvement in the performance of the genetic algorithm. As depicted in Figure 4, with the cycle crossover the objective function decreased much faster. Furthermore, for the case of Letters Only, with the old crossover routine, the genetic algorithm reached the final objective function value at generation 88, whereas with the cycle crossover routine this same value was achieved at generation 16. For the case of Letters and Punctuation, the cycle crossover yielded a lower final objective function value and it was also achieved in a much earlier generation. See Table I.

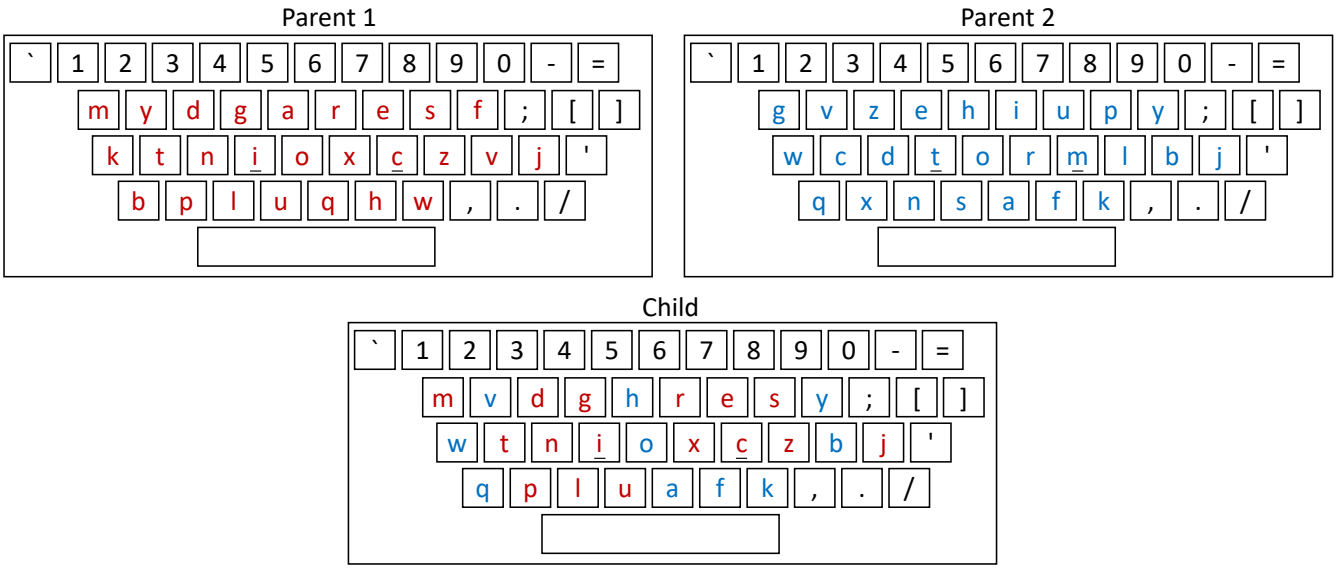

Fig. 1. Example of the cycle crossover routine.

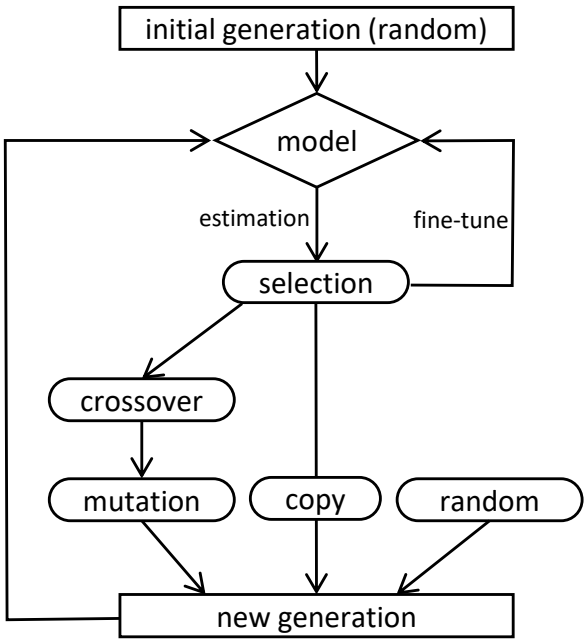

Fig. 2. MKLOGA flowchart.

TABLE I
RESULTS WITH THE CYCLE CROSSOVER

	Letters Only		Letters & Punct.	
	effort	gens	effort	gens
Fadel et al.	1394663.75	88	1312948.02	97
MKLOGA	1394663.75	16	1311932.84	27

B. *MKLOGA*

As mentioned above, MKLOGA fine-tunes the neural network model during the course of the genetic algorithm. In our experiment, the loss of the model decreased from 0.048 to 0.0028. This indicates that the model's prediction accuracy improved during the course of the execution.

When we ran MKLOGA, it found a layout with an effort value of 1.625, and it did so already at generation 19. See Figure 5 (left). For comparison, the layout found by Fadel et al. [7] has an effort value of 2.508 (though, as mentioned

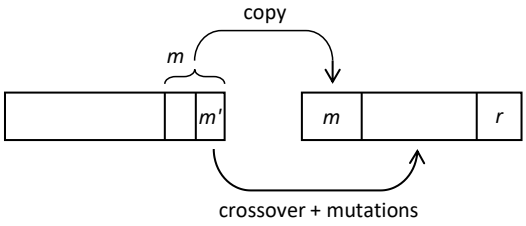

Fig. 3. The construction of the new generation from the previous one. First, r layouts are generated randomly. The best m layouts of the previous generation (according to the model estimate) are directly copied to the new generation, and the best m' of them (according to the true effort value) are used to generate new layouts through crossover and mutations.

TABLE II
THE CARPALX EFFORT VALUE OF DIFFERENT KEYBOARDS.

keyboard	effort
Qwerty	2.962
Dvorak	2.046
Colemak	1.796
Carpalx Sim. Ann.	2.038
}.?BZQ (Fadel et al.)	2.508
MKLOGA	**1.625**

above, they optimized for a different objective function). We also ran the simulated annealing code of [12] using its default parameters. We did so 10 times and took the average effort value of the produced layouts. See Table II, which also shows the effort value of a few other well known layouts, for comparison. Moreover, the neural network model of MKLOGA takes approximately only one millisecond to estimate the effort value of a layout, which is much faster than calculating the true effort.

VI. Discussion

As mentioned above, the keyboard layout found by MKLOGA achieves an effort value of 1.625. In comparison, the best layout offered by the carpalx project [12] (the one they refer to as "qgmlwb") achieves an effort value of 1.629 (in their website they give a value of 1.668, the difference being due to the use of a larger corpus). However, the layout that they recommend (which they refer to as "qgmlwy") leaves the keys Z, X, C, V, in the classical QWERTY positions due to their frequent use in keyboard shortcuts, and it achieves an effort value of 1.635 (1.670 according to their website). Interestingly, in one run, MKLOGA produced a keyboard with the keys Z, X, C, V, in place, with an effort value of 1.633. See Figure 5 (right). Finally, it is worth noting that the above-mentioned carpalx layouts were probably built with human assistance and not purely with a computer algorithm.

MKLOGA may be useful in developing good keyboard layouts for languages other than English. Also, as mentioned in [7], there is a need for left-hand only and right-hand only layouts for handicapped people. There is an increasing need for good layouts for smartphone keyboards, in which people type with only one finger.

VII. Future Work

In current form MKLOGA calculates the effort value with a relatively small corpus; we plan to switch to a larger corpus in order to get more accurate results. Furthermore, we plan to give MKLOGA more freedom by allowing it to change the positions of punctuation symbols as well. This will entail reconstructing and retraining the neural network from scratch, since its keyboard representation includes only the symbols that can change positions.

In addition, we plan to implement an even more realistic objective function that takes into account other relevant factors in typing. Also, we suspect that the carpalx effort model is not realistic enough, because it assigns a much better score to the Dvorak keyboard than to QWERTY, despite the fact that research shows that among experienced typewriters, it does not make much of a difference whether they use the Dvorak or the QWERTY layout. We would like a more realistic function that will explain this counter-intuitive fact so we can construct a keyboard layout that is truly better.

VIII. Conclusion

Over the years, people have tried to come up with better keyboard layout designs, both manually and with computer search programs. In this paper we proposed MKLOGA, which combines deep learning with a genetic algorithm for finding improved keyboard layouts. It also makes use of the cycle crossover routine, that significantly outperforms the crossover routine that was previously used in the literature. As we showed, MKLOGA produced a better keyboard layout than previous algorithms, according to the realistic typing effort model of carpalx [12]. MKLOGA might be useful for developing good layouts for languages other than English, and for other situations in which objects must be placed in predefined locations.

Acknowledgment

This research was supported in part by the Ministry of Science, Technology & Space, Israel.

References

[1] The Age, "Wrist pain? try the Dvorak keyboard," 2004, https://www.theage.com.au/technology/wrist-pain-try-the-dvorak-keyboard-20041210-gdka9g.html.
[2] J. Stamp, "Fact of fiction? the legend of the QWERTY keyboard," 2013, https://www.smithsonianmag.com/arts-culture/fact-of-fiction-the-legend-of-the-qwerty-keyboard-49863249/.
[3] J. Noyes, "The qwerty keyboard: A review," *International Journal of Man-Machine Studies*, vol. 18, no. 3, pp. 265–281, 1983.
[4] N. Baker, "Why do we all use qwerty keyboards?" 2010, https://www.bbc.com/news/technology-10925456.
[5] S. Coleman, "Colemak," 2006, http://colemak.com.
[6] P.-Y. Yin and E.-P. Su, "Cyber swarm optimization for general keyboard arrangement problem," *International Journal of Industrial Ergonomics*, vol. 41, no. 1, pp. 43–52, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169814110001058
[7] A. Fadel, I. Tuffaha, M. Al-Ayyoub, and Y. Jararwch, "Qwerty keyboard? .?BZQ is better!" in *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. IEEE, 2020, pp. 81–86.

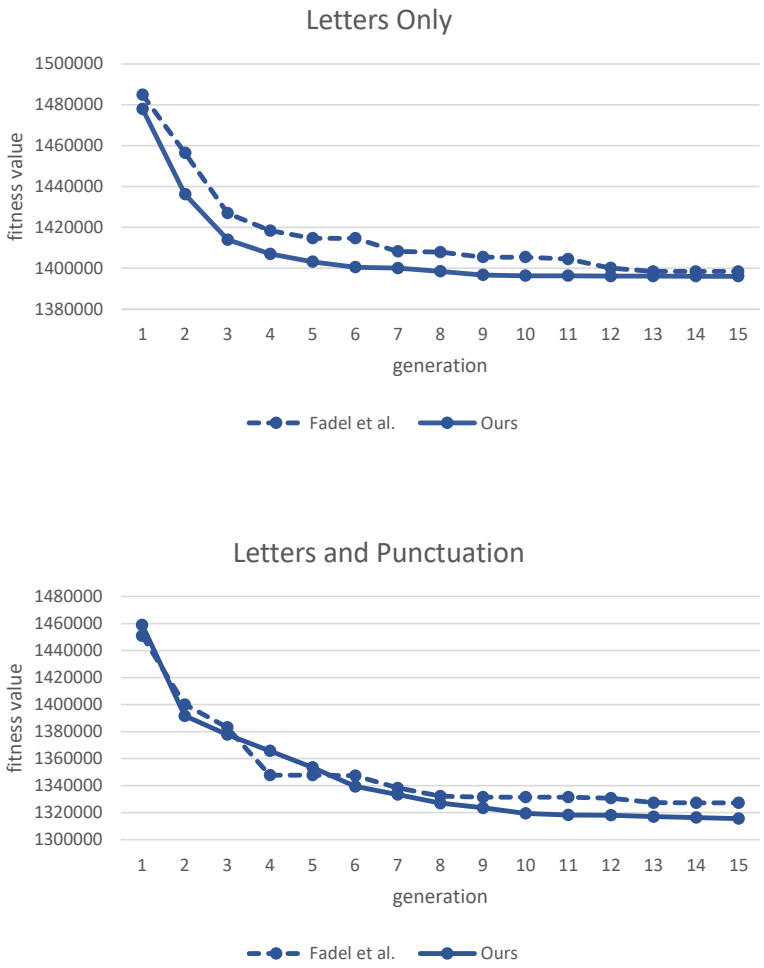

Fig. 4. A comparison between the performance of the genetic algorithm using the crossover method proposed by [7] and the cycle crossover method. The performance is measured according to the objective function defined in [7] (lower is better).

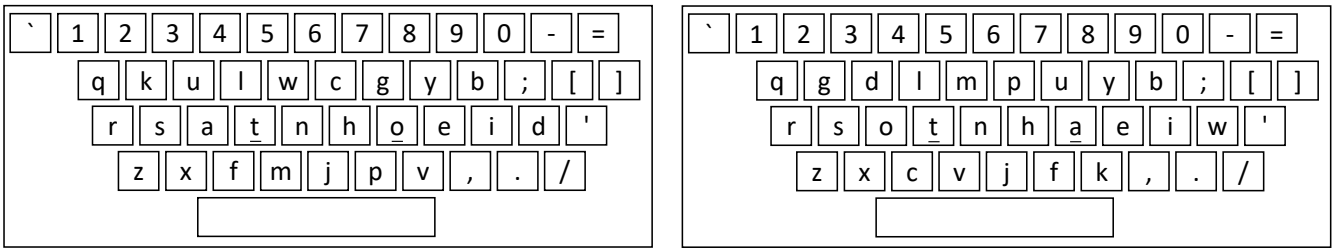

Fig. 5. Left: Best keyboard layout found by MKLOGA. Right: Another keyboard layout found by MKLOGA, with ZXCV in place.

[8] I. Oliver, D. Smith, and J. R. Holland, "Study of permutation crossover operators on the traveling salesman problem," in *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[9] C. Liao and P. Choe, "Chinese keyboard layout design based on polyphone disambiguation and a genetic algorithm," *International Journal of Human–Computer Interaction*, vol. 29, no. 6, pp. 391–403, 2013. [Online]. Available: https://doi.org/10.1080/10447318.2013.777827

[10] M. Govind and V. V. Panicker, "Optimization of a single finger keyboard layout using genetic algorithm and topsis," *International Journal of Scientific & Engineering Research*, vol. 7, no. 2, pp. 102–105, 2016.

[11] A. H. H. Onsorodi and O. Korhan, "Application of a genetic algorithm to the keyboard layout problem," *PLOS ONE*, vol. 15, no. 1, pp. 1–11, 01 2020. [Online]. Available: https://doi.org/10.1371/journal.pone.0226611

[12] M. Krzywinski, "Carpalx keyboard layout optimizer," 2005, http://mkweb.bcgsc.ca/carpalx/.

[13] J. M. R. Salvo, C. J. B. Raagas, M. T. C. M. Medina, and A. J. A. Portus, "Ergonomic keyboard layout designed for the filipino language," in *Advances in Physical Ergonomics and Human Factors*. Springer, 2016, pp. 407–416.

[14] V. Vitolins, "Modernized latvian ergonomic keyboard," *arXiv preprint arXiv:1707.03753*, 2017.

[15] H. Azarbonyad and R. Babazadeh, "A genetic algorithm for solving

quadratic assignment problem (qap)," in *Proceeding of 5th International Conference of Iranian Operations Research Society (ICIORS), Tabriz, Iran*, 2012. [Online]. Available: https://arxiv.org/abs/1405.5050

[16] R. K. Ahuja, J. B. Orlin, and A. Tiwari, "A greedy genetic algorithm for the quadratic assignment problem," *Computers & Operations Research*, vol. 27, no. 10, pp. 917–934, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054899000672

[17] A. Hameed, B. Aboobaider, M. Mutar, and N. Choon, "A new hybrid approach based on discrete differential evolution algorithm to enhancement solutions of quadratic assignment problem," *International Journal of Industrial Engineering Computations*, vol. 11, no. 1, pp. 51–72, 2020.

[18] A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 596–601.

[19] A. Potapov and S. Rodionov, "Genetic algorithms with dnn-based trainable crossover as an example of partial specialization of general search," in *International Conference on Artificial General Intelligence*. Springer, 2017, pp. 101–111.

[20] C. Hu, Y. Zuo, C. Chen, S. P. Ong, and J. Luo, "Genetic algorithm-guided deep learning of grain boundary diagrams: addressing the challenge of five degrees of freedom," *Materials Today*, vol. 38, pp. 49–57, 2020.