Air traffic controller keyboard optimization by artificial evolution

Daniel Delahaye, Stéphane Puechmorel

Air Traffic Controller Keyboard Optimization by Artificial Evolution

Daniel Delahaye[1] and Stephane Puechmorel[2]

[1] CENA ⋆⋆⋆
[2] ENAC†
delahaye@tls.cena.fr
puechmor@recherche.enac.fr
CENA-ENAC
7, Avenue Edouard Belin
31055 Toulouse, France

Abstract. The annual number of daily flights in France has increased from about 3500 in 1982 to about in 8000 in 2000. The number of flights simultaneously present on the radar screen of the controller has also increased. Usually controllers manage about 15 aircraft on their position and sometime this number reach a maximum of 20. On the radar screen, aircraft are represented by spots (with some previous positions and their speed vector) and the associated label which give the flight ID, the speed and the altitude of the aircraft. The controller in charge of the controlled area, has to be able to select any aircraft in order to manipulate some parameters of the flight such as heading, speed, altitude etc. Aircraft selection is done by the mean of a virtual keyboard where the controller pressed the keys of the flight ID. This ID is composed by a sequence of three letters (maximum) which represents the airline code, followed by the flight number. When such a selection is done, the associated flight is made highlighting on the radar screen. Depending of the flight ID distribution on a control position, the virtual keyboard can be optimized in order to speed up the aircraft selections and to improve the work of the controllers mainly when the sectors are overloaded. This keyboard optimization problem may be addressed like a pure assignment problem which is NP_Hard. This paper shows how artificial evolution has been used for solving such a problem with very good results on real instance associated to the Roissy departure sector.

1 Introduction

When an aircraft flies from a city A to a city B, it has to be managed by air traffic controllers in order to avoid collisions with others aircraft. Everyday, about 8000 aircraft fly in the French airspace, inducing a huge amount of control workload. Such a workload, is then spread by the mean of the airspace sectoring.

⋆⋆⋆ Centre d'Études de la Navigation Aérienne
† École Nationale de l'Aviation Civile

The airspace is divided into geometrical sectors, each of them being assigned to a controller team. When a conflict between two (or more) aircraft is detected, the controller changes their routes (heading, speed, altitude) in order to keep a minimum distance between them during the crossing. All flying aircraft are then monitored during their navigation and so from the departure till the destination. This monitoring is helped by mean of the flight ID. This ID is a code associated to the flight composed by several letters related to the airline, followed by a number. For instance, the ID TWA810 is associated to the flight Boston-Paris operated by the TWA airline. All airlines have a normalized code given by the ICAO[1] authority. This code has a maximum length of 3 letters. Air France has the code "AFR", British Airways "BA" and so on. In order to proceed a flight, the pilot has first to produce a flight plan which is a kind of summary of the preferred navigation route. This flight plan gives the flight ID and a list of navigation segments connected by normalized way points (points in the airspace extracted from an official set produced by the civil aviation authority). For all those points (2D), the pilot must produce the associated altitude or the flight level[2] of the aircraft when it will be above this point.

The flight ID selection is done by mean of a numerical keyboard for which letters are associated to numbers like in a telephone keyboard. The fact that two aircraft belonging to the same airline being in the same controlled area at the same time, is very rare and the flight ID selection is done only on the letter associated to the airline. When such uncommon event happen, the selection is extended with the number.

Nowadays, the controller enters all letters of the flight ID. Instead of using a physical keyboard, we propose to use a virtual one for which the alphabetic association to the numeric keyboard will be optimized in order to speed up the selection process.

For all controlled areas, it is easy to determine the number of Air France flights, the number of TWA flights etc. and to build statistics about the airline codes. This counting is done on the year base. An example of such a statistic is given in the following table:

Number of flights	Airline ID	Airline	Percentage
44	AFR	Air France	23
24	BA	British Airways	12
21	LF	Lufthansa	11
...	...	...	...

Our problem is to synthesize a dedicated virtual keyboard to any controlled area in order to minimize the average selection time of aircraft IDs. This keyboard

[1] International Civil Aviation Organization

[2] A flight level is a measure of altitude, given in hundred of feet by an altimeter referenced to the 1013 hpa altisurface (average pressure at the sea level). For instance if an altimeter measures a pressure of 164 hpa, there is a difference of 164-1013=-848 hpa which gives an altitude difference of 848∗23=19504ft ($1hpa \Rightarrow 23ft$); then the associated flight level is 195.

is based on the numerical keys like the telephone keyboard but the distribution of the letters on those keys have to be optimized. The numerical keys has to be kept because such a keyboard is also used for others numerical tasks of the controller work.

In a first part, a refined description of the problem is given for which the performance and the constraints of the virtual keyboard are given. The second part proposes an associated mathematical modeling of the problem. The third and the fourth parts show how artificial evolution has been applied to such a problem. Finally the fifth part presents results on some real traffic sets.

2 Problem Description

Based on the airline statistics associated to the considered controlled area, one has to find the optimal letter assignment on a numerical keyboard. The numerical distribution of the keys on a telephone keyboard is given Figure 1.

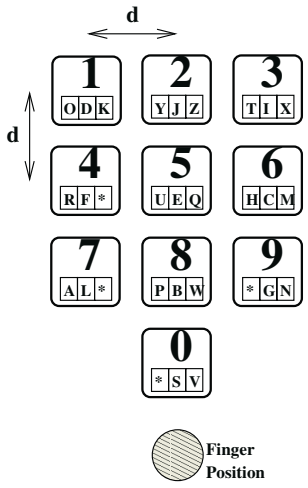


Fig. 1. Distribution of the letter on a telephone keyboard

The three boxes under each number are the potential positions of the letters; so there are 30 positions where the airline ID letters can be included (a random association has been represented in this example). Such a keyboard will generate a translation between the alphabetic code (airline descriptor) and the resulting numerical code. Having more boxes (30) than alphabetic letters (26), 4 null characters will be included in such alphabet for completion. This association of letters to the numerical keyboard has to optimize the following two criteria:

1. *the total number of pressed keys;*
2. *the total distance on the keyboard.*

Depending on the letter distribution, it is possible to produce automatic completion without ambiguity. As a matter of fact, suppose the following unrealistic

airline code set is given :{AAA 50%;GGG 50%}. It says that 50% of the aircraft has the airline code "AAA" and the other 50% has the code "GGG". It is easy to understand that the first letter is enough to remove the ambiguity between the two possible airline codes subject that the two letters are located on two different keys. In such example, the number of pressed key reductions is given by : (50%.2 + 50%.2).total number of aircraft. When the system has not ambiguity anymore, it can highlight the associated aircraft.

For the key sequence which have to be pressed on the keyboard, the associated distance has also to be minimized. The keyboard will try to put as close as possible the letter associated to the most frequent letter codes.

Such optimization has to avoid also collisions between the resulting numerical codes. As a matter of fact, different sequences of letters have to be coded by different sequences of numbers in order to avoid ambiguities. For instance, in the previous example, if the two letters "A" and "G" are coded by the same number, the system will not be able to distinguish between the two airlines.

Some previous related works may be found in the reference [16] but the objective is to compare the performances of several virtual keyboards without optimization.

The problem that has to be solved is very closed to the one of information compression as it appears in the information theory [6]. In such compression process, a file composed of different characters has to be reduced in size in order to be transmitted or stored on any data support. Initially, all the characters are coded with the same binary digit sequence length. The idea consists in coding with short binary digit sequences the most frequent characters and by long ones the less frequent ones (Huffman algorithm [12]). Such approach is not adapted for our problem for the two following reasons :

1. to applied the Huffman algorithm to such a problem, every sequence of letters has to be considered as a "super-letter" of an alphabet composed of $27 \times 27 \times 27 = 19683$ characters (26 alphabetic letters plus the null character). The Huffman algorithm will then produce a transcoding without any meaning for the controller. For instance all the Air France (AFR) flights would be coded by the sequence 24, the British Airways (BA) by the sequence 39. The two initial alphabetic codes have a common letter, but the produced numerical codes have no intersection;
2. the less frequent sequences could be coded with longer numerical sequences compared to the maximum length of three in the alphabetic sequence which is also very disturbing for the controller.

Instead of using such information theory tools which are not adapted enough to our problem, we propose to address this problem from the combinatoric optimization [13] point of view. Before presenting the method which has been used, a mathematical model is first presented.

3 Mathematical Modeling

The problem that has to be solved consists in finding an optimal assignment of the alphabetic letters to the numerical keyboard with a maximum of three letters per key (see Figure 1). This assignment represents the state space of our problem.

If the initial alphabet is extended with 4 null letters (stars on Figure 1), a point in our state domain can be considered as a permutation of those 30 letters (26 alphabetics plus 4 nulls). If numbers are assigned to all alphabetic letter (A $\rightarrow$ 0; B $\rightarrow$ 1; C $\rightarrow$ 2; ...; Z $\rightarrow$ 25; Null $\rightarrow$ 26; ...; Null $\rightarrow$ 29), state points are pure permutations among those 30 integers. In order to evaluate such a point in the state space, an objective function has to be defined. As it has been mentioned in the introduction, the optimization process must minimize the number of pressed keys and the distance covered on the numerical keyboard during code entering process. To compute this objective, the first step consists in computing the list of numerical codes ($\tilde{N}$) builded from the alphabetic sequences ($\tilde{L}$) and the given permutation (P). Based on this initial list, each numerical sequence is checked with the other ones in order to determine the last superfluous digits which can be removed in the final numerical sequences. Those final sequences will be the ones which will be effectively pressed on the keyboard. The system will then produce the automatic completion and will highlight the associated flight. Those final sequences will be also called *compressed sequences* in the following. In order to compute the distance related to the pressed keys, a distance matrix has been introduced. To build such a matrix, it has been supposed that only one finger presses the keys on the numerical keyboard and the initial position of this finger is under the "0" key as it can be seen on Figure 1.

The associated matrix is given by (the numbers expressed in this matrix are in "d" unit):

$$D = \begin{bmatrix} 0 & 1 & \sqrt{17} & 4 & \sqrt{17} & \sqrt{10} & 3 & \sqrt{10} & \sqrt{5} & 2 & \sqrt{5} \\ 1 & 0.5 & \sqrt{10} & 3 & \sqrt{10} & \sqrt{5} & 2 & \sqrt{5} & \sqrt{2} & 1 & \sqrt{2} \\ \sqrt{17} & \sqrt{10} & 0.5 & 1 & 2 & 1 & \sqrt{2} & \sqrt{5} & 2 & \sqrt{5} & \sqrt{8} \\ 4 & 3 & 1 & 0.5 & 1 & \sqrt{2} & 1 & \sqrt{2} & \sqrt{5} & 2 & \sqrt{5} \\ \sqrt{17} & \sqrt{10} & 2 & 1 & 0.5 & \sqrt{5} & \sqrt{2} & 1 & \sqrt{8} & \sqrt{5} & 2 \\ \sqrt{10} & \sqrt{5} & 1 & \sqrt{2} & \sqrt{5} & 0.5 & 1 & 2 & 1 & \sqrt{2} & \sqrt{5} \\ 3 & 2 & \sqrt{2} & 1 & \sqrt{2} & 1 & 0.5 & 1 & \sqrt{2} & 1 & \sqrt{2} \\ \sqrt{10} & \sqrt{5} & \sqrt{5} & \sqrt{2} & 1 & 2 & 1 & 0.5 & \sqrt{5} & \sqrt{2} & 1 \\ \sqrt{5} & \sqrt{2} & 2 & \sqrt{5} & \sqrt{8} & 1 & \sqrt{2} & \sqrt{5} & 0.5 & 1 & 2 \\ 2 & 1 & \sqrt{5} & 2 & \sqrt{5} & \sqrt{2} & 1 & \sqrt{2} & 1 & 0.5 & 1 \\ \sqrt{5} & \sqrt{2} & \sqrt{8} & \sqrt{5} & 2 & \sqrt{5} & \sqrt{2} & 1 & 2 & 1 & 0.5 \end{bmatrix}.$$

It can be noticed that when a key is pressed two times, the associated distance is not zero but half *d*. So, for a given numerical code, the distance of the overall path is given by the summation of the individual distances between keys.

$$D(\tilde{N}'_i) = \sum_{j=1}^{|\tilde{N}'_i|-1} \left[d_{\tilde{N}'_{i\,j},\tilde{N}'_{i\,j+1}} \right]$$

where $|\tilde{N}'_i|$ represents the length of the considered compressed code ($\tilde{N}$ represents the induce code by a permutation and $\tilde{N}'$ after compression). For instance

the numeric code 753 has the distance $\sqrt{5}+\sqrt{2}+\sqrt{2}$. This distance being computed on the compressed numerical codes (code without superfluous last digits), its minimization meets two objectives (minimization of the number of pressed keys and the remaining distance). In order to take into account the statistics associated to each code, the distance is weighted by the number of flight having this code. So, a permutation P is evaluated the following way :

$$O(P) = \sum_{i=1}^{i=|N'(P)|} D(\tilde{N}'_i).N_B(\tilde{N}'_i)$$

where $N'(P)$ is the set of numerical codes associated to P after compression; and $N_B(\tilde{N}'_i)$ the number of flights with the code $\tilde{N}'_i$.

Having now defined the state space and the objective function, one has to identify the associated constraints of this problem. The main constraint can be summarized the following way: *two different alphabetic sequences have to produce two different numerical sequences*. For a given permutation P, the associated constraint violation criterium is computed by counting the identical compressed numerical code induced by a permutation P. The number of constraint violations is then given by :

$$N_v(P) = \sum_{i=1}^{i=|N'(P)|} \sum_{\substack{j=1 \\ j=i}}^{j=|N'(P)|} \delta[\tilde{N}'_i(P), \tilde{N}'_j(P)]$$

where

$$\delta[\tilde{N}'_i(P), \tilde{N}'_j(P)] = \begin{cases} 1 & \text{si} \quad \tilde{N}'_i(P) = \tilde{N}'_j(P) \\ 0 & \text{sinon.} \end{cases}$$

This number is taken into account in the optimization process by relaxation. Then, the constraint factor will be stronger at the end of the optimization process than at the beginning.

Having now defined a mathematical model for our problem one must identify the associated complexity. As it has been previously shown, one has to find an optimal assignment of 26 letters to 30 boxes. The number of possibilities is given by[3]: $\frac{30!}{4!}$.

The problem we have to solved is an assignment problem which is known to be NP_Hard and artificial evolution is quite adapted to address such a problem.

4 Artificial Evolution

Artificial evolution is an optimization algorithm belonging to the class of stochastic optimization techniques. Simulated annealing [8], taboo search [4], branch and probability bound [17] etc..., belong also to such class. Such techniques usually

[3] The first letter has 30 choices, the second 29 and so on: $30.29 \ldots 5 = \frac{30!}{4!}$ (the number 4 comes from the 4 null letters which have been included).

addressed problem with strong complexity for which the objective function has no specific feature such as convexity, continuity etc. The main feature of those stochastic optimization techniques is to randomly move in the state domain in order to improve the objective criterium. Evolutionary Algorithms (EA) [5,7, 3,11,9,2,14] are problem solving systems based on principles of evolution and heredity. An EA maintains a population of individuals,$P(t) = x_1, x_2, \ldots, x_n$ at iteration t. Each individual represents a potential solution to the problem at hand and is implemented as some (possibly complex) data structure S. Each solution x_i is evaluated to give some measure of fitness. Then a new population at iteration $t + 1$ is formed by selecting the fitter individuals. Some members of the new population undergo transformation by means of genetic operators to form new solutions. There are unary transformations m_i (of mutation type), which create new individuals by a small change of a single individual and higher order transformations c_i(crossover type), which create new individuals by combining parts from several (two or more) individuals. For example, if parents are represented by a five-dimensional vector $(a_1, a_2, a_3, a_4, a_5)$ and $(b_1, b_2, b_3, b_4, b_5)$, then a slicing crossover of chromosomes after the second gene produces offspring $(a_1, a_2, b_3, b_4, b_5)$ and $(b_1, b_2, a_3, a_4, a_5)$. The control parameters for genetic operators (probability of crossover and mutation) need to be carefully selected to provide better performance. The intuition behind the crossover operation is information exchange between different potential solutions. After some number of generations the program converges - the best individual hopefully represents the optimum solution.

The structure of the EA which has been used for our experiments is given Figure 2. To move from generation k to generation $k + 1$, the population is first

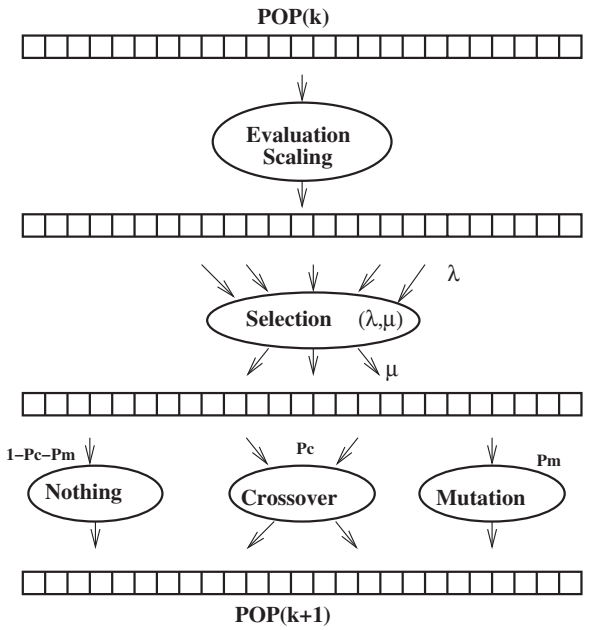


Fig. 2. Structure of the Evolutionary Algorithm

evaluated and the associated fitness is also scaled using a classical power law scaling principle [5]. Afterward, the best individuals are selected with a tournament selection $(\lambda; \mu)$ and then undergo the recombination process. Then, individuals can be put straightly in the next population ("nothing" operator), crossed over (crossover operator) or mutated (mutation operator). Those operators are randomly selected based on their associated probabilities ($(1 - P_m - P_c) \rightarrow$ nothing; $P_c \rightarrow$ crossover; $P_m \rightarrow$ mutation). The probability of mutation is self adapted to the problem using the Reichenberg rule with a constraint interval limiting this probability between $[0.5, 0.7]$. This scheme has been successfully used for our keyboard optimization experiments.

5 Application to Keyboard Optimization

5.1 Coding

In order to make run such Evolutionary algorithm, a coding of the state space has been developed. Two potential codings have been proposed. The first one is based on the pairwise decomposition property of any permutation of size N. As a matter of fact, any permutation of size N can be decomposed into a list of exchanges between pair of positions, subject that the minimum number of swaps is at least $N \log(N)$. From the mathematical point of view, this property can be formulated the following way. Suppose that a permutation P of size N and a pairwise swapping operator T are given. When the permutation P is applied to an initial list L_1, it produces a new list noted L_2. This means that $L_2 = P(L_1)$. This list L_2 can also be reach by the successive application of the operator T on some pairs of position: $L_2 = T_1 \circ T_2 \circ \ldots \circ T_K(L_1)$ subject that K is greater than $N \log N$ (where $\circ$ is the composition operator). The structure of such a coding is given Figure 3. This coding is convenient because crossover like slicing

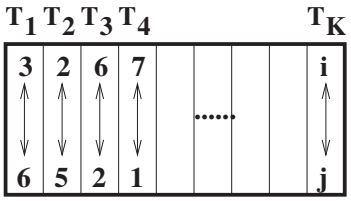


Fig. 3. Coding based on pairwise permutation operator T

or uniform can be straightly applied to any chromosome without checking the integrity of the children (as a matter of fact, the application of such a crossover insures that the produced permutation will be valid (one letter is assigned to only one numerical key)). In the some way, it is very easy to develop a mutation operator by randomly choosing a locus in the chromosome and put a new random pairwise permutation.

Unfortunately, this coding produced poor results in our experiments and a more straight one has been used without crossover operator. This new coding is just an array of integers describing the overall permutation as it can be shown Figure 4.

i	1	2	3	4	5		28	29	30
P(i)	17	24	9	27	12		1	3	15

Fig. 4. Straight coding of the permutation P

In this example the 17th letter is assigned to the first position, the 24th to the second one and so on.

The mutation operator consists in exchanging two random positions (i, j) in the chromosome like it can be seen Figure 5 (in this example the size of the chromosome is 12 instead of 30 in our problem).

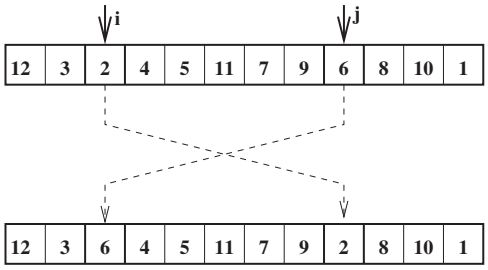


Fig. 5. Mutation operator

The associated fitness evolution takes into account the constraint criterium (N_v) and the overall distance induced by the keyboard. As it has been previously noticed, this distance minimization takes into account the code compressions (when a flight can be selected only by the first digits in the numerical code, the final ones has not to be pressed on the numerical keyboard). The fitness associated to a chromosome C is then given by :

$$Fitness(C) = e^{-\lambda . N_v(C)} . \frac{1}{\left(\frac{D(C)}{N_F}\right)}$$

where $\lambda = \frac{gen}{nb\_gens}$ (gen current generation number; nb_gens: total number of generations), $D(C)$ the total distance associated to the chromosome C and N_F the total number of flights. To compute $D(C)$ and $N_v(C)$, the keyboard is build from the permutation associated to C. Then all alphabetic codes are then converted into numerical codes and compressed when it is possible. Based on this set

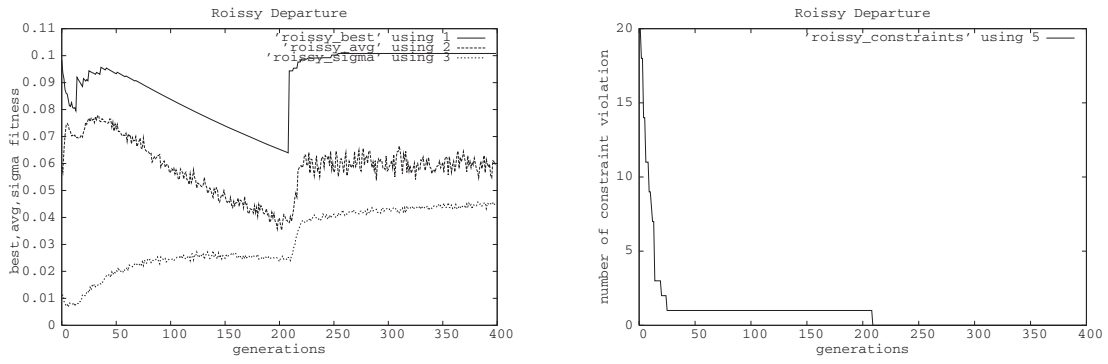
of numerical codes, the number of collisions is then computed (N_v: the number of numerical codes which are similar) which is the number of constraint violations. The overall distance on the proposed keyboard is then computed using the distance matrix. This distance takes into account the number of flights associated to a given numerical code.

6 Results

This method has been applied to the departure sector of Roissy Airport. During one year this sector is crossed by about 300000 aircraft. The set of alphabetic codes is given on the following table (only the beginning of the table is represented).

Airline ID	AFR	AF	DLH	BAW	FPO	AZA	SAS	BCY	BMA	...
Nb Flights	97071	23689	14324	9165	7359	7332	4862	4328	3944	...

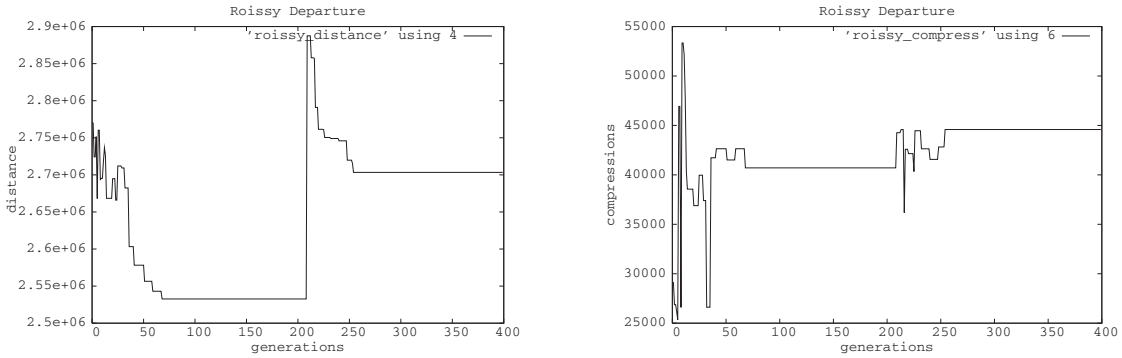
The parameters of the evolution were the followings: population size 300; number of generations 400; $\lambda = 7$, $\mu = 2$ and initial probability of mutation 0.6. The evolution of the best fitness, the average and the associated dispersion are given Figure 6(a). This fitness has to be maximized and the observed decreasing is due to the constraint relaxation for which the constraint violation criterium becomes stronger and stronger. The discontinuities of the fitness appear when the number of constraint violations decreases (see 6(b)). The associated keyboard distance reduction and the number of compressions are represented on figure 7.



(a) Best, Avg, Sigma (b) Number of constraint violations

Fig. 6. Fitness and constraint violations evolution

As it can be seen on the figure, the distance has a smaller value at the beginning of the evolution than at the end but it is also the period where the constraint are not respected. The final distance is about $2.7.10^6$ which is really smaller to the one induced by the actual keyboard: $3.9.10^6$. The associated reduction of pressed keys is about 45.10^3. Finally the synthesized keyboard is represented Figure 8. The optimization process has reduced significantly the distance of the most frequent airline codes like AFR, DLH, BAW (closest to the finger position).



(a) Distance minimization (b) Compression maximization

Fig. 7. Distance and compression evolution

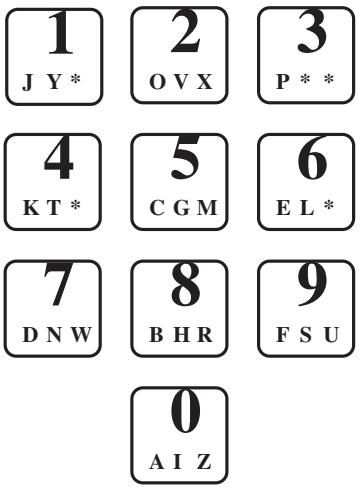


Fig. 8. Synthesized keyboard

7 Conclusion

This paper has presented a real word application of the Artificial Evolution technique. Results proposed by the algorithm really improve the selection performance of the flight ID on the control position. The paper has first described the operational context of the flight ID selection and the underlying keyboard optimization problem. The objective function, the state space and the associated constraints have been presented with the associated mathematical formulation. Due to the induced complexity of such a problem, stochastic optimization has been supposed to be a good candidate to solve this problem. Artificial evolution has been presented and adapted to the keyboard optimization problem. Finally, the algorithm has been tried on a real instance of the problem related to departure sector of Roissy Airport. The given results really improved the flight ID selection with a large reduction of pressed key. In a near future, this tool will be adapted to several control positions with different flight IDs in order to check if the produced results are as good as the ones given for the departure sector of Roissy Airport.

Acknowledgements. We would like to thank Herve Damiano for having proposed such a beautiful combinatoric optimization problem.

References

1. D Colin De Verdière. Le système français de contrôle du trafic aérien : Le CAUTRA. Technical report, CENA, Toulouse France, Aout 1992.
2. D.B Fogel. *Evolutionary Computation. Toward a new Philosophy of Machine Intelligence*. IEEE press, 1994.
3. L.J Fogel, A.J Owens, and M.J Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley and sons. NY, 1966.
4. A. Glover. *Taboo Search*. Kluwer Academic Publishers, 1997.
5. D.E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading MA Addison Wesley, 1989.
6. D Hankerson, G.A Harris, and P Johnson. *Introduction to Information Theory and Data Compression*. CRC Press, 1997.
7. J.H Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan press, 1975.
8. S Kirkpatrick, C.D Gelatt, and M.P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
9. J.R Koza. *Genetic Programming*. MIT press, 1992.
10. G Maignan. *Le Contrôle de la Circulation Aérienne*. Presse Universitaire de France, 1991.
11. Z Michalewicz. *Genetic algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.
12. M Nelson and J.L Gailly. *The Data Compression Book*. Hungry Minds, Inc, 1995.
13. C.H Papdimitriou and K Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Prentice-Hall, New York, 1982.
14. H.P Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.
15. P.L Tuan, H.S Procter, and G.J Couluris. Advanced productivity analysis methods for air traffic control operation. Technical report, Stanford Research institute, Menlo Park CA 94025, December 1976.
16. S. Zhai, B.A Smith, and M Hunter. Performance optimization of virtual keyboards. Human-Computer Interaction, 2001.
17. A. A. Zhigljavsky. *Theory of Global Random Search*. Kluwer Academic Publishers, 1991.