Evolution of neuro-controllers for flapping-wing animats

Jean-Baptiste Mouret Stéphane Doncieux Laurent Muratet
Thierry Druot Jean-Arcady Meyer

LIP6 - AnimatLab
8, rue du Capitaine Scott 75015 Paris
http://animatlab.lip6.fr
{Stephane.Doncieux,Jean-Arcady.Meyer}@lip6.fr
{mouret,muratet}@poleia.lip6.fr

Abstract

This article reports preliminary results obtained with an evolutionary approach to the design of neural controllers for flapping-wing animats. This approach involves a multi-objective evolutionary algorithm and continuous-time neural networks. It has been used to automatically generate controllers securing an energetically thrifty horizontal flight at constant speed in a simulated artificial bird.

1 Introduction

Born as a reaction to the limitations of the *good old fashioned artificial intelligence (GOFAI)* [14], the animat approach [21] aims at designing autonomous and adaptive artefacts – mostly simulated or real robots – whose inner workings are inspired by living creatures. However, if most researches in this field have been targeted at designing wheeled robots – for instance Psikharpax, an artificial rat [10] –, walking robots [17, 13, 18] or swimming ones [15], up to now an important part of the animal kingdom has been neglected, at least from the point of view of its locomotive capacities: that of flying animals.

The flapping-wing locomotion mode has, indeed, clear potential for small-scale robots as demonstrated by the efficiency and manoeuvrability of real birds, who still outperform any artificial aircraft of the same size.

As standard airplanes only need to maintain their forward speed to fly, they do not expend a lot of energy for sustainment. However, the range of their flying abilities is limited, as they can't hover, for instance. On the other hand are helicopters, which are highly manoeuvrable, but consume a high amount of energy. Between these two extremes, flapping-wing aircraft are an interesting trade-off because they are able to both glide and hover. Such a skill has not been exhibited yet by any artificial platform, although several birds, like hummingbirds or hawks, are capable of hovering or near hovering flight. Thus, flapping-wing aircraft seem to have a promising future among flying robots, but their design and use require a thorough understanding of both the potentialities and constraints of their flying mode. The Robur project [1] aims at studying these potentialities and constraints from the point of view of the animat approach.

As a contribution to this project, the work described herein explores a methodology dedicated to the design of adaptive controllers for flapping-wing animats. This methodology involves continuous-time neural networks that are automatically generated by a multi-objective evolutionary algorithm using a dedicated encoding scheme, without any previous knowledge about the wing movements likely to secure an efficient flight.

A few research efforts are currently devoted to the design of flapping-wing UAVs [23, 25, 22, 2]. However, these efforts focus on aeronautical and mechanical aspects; none of them is targeted at the optimal control of such platforms, nor on the design of biomimetic controllers.

This paper is organized into two parts. The first describes the four major components of the proposed methodology, which is used, in the second part, to design controllers for flapping-wing aircraft flying horizontally and at constant speed.

2 The proposed methodology

2.1 Principles of flapping-wing flight

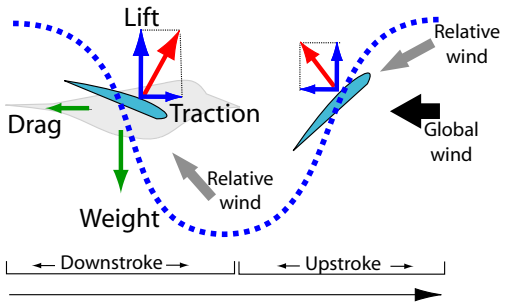

Figure 1: The force produced by the pressure difference between the intrados and the extrados is oriented forward and upward during a down-stroke. It can therefore be decomposed into a lift force and a traction force. Lift is produced as well during an up-stroke, but it is accompanied by a significant drag.

Birds use the pressure difference between wing intrados [1] and extrados [2] to counter their weight, and to generate a traction that allows them to maintain their relative speed or to accelerate. These two forces are both produced during the down-stroke (figure 1). The wing is powered downward with its leading-edge tilted down and, as a consequence, the relative wind – i.e., the sum of the bird's center of gravity speed in the air mass and of the speed of the wing relative to the bird's body – is directed upward. The corresponding force is therefore oriented both upward and forward and can be divided into two contributions, the lift and the traction. Lift is produced during the up-stroke as well, since birds change the angle of attack of the wing to adapt to the corresponding new relative wind. However, this lift is accompanied by a significant drag and, to reduce it, birds partially fold their wings during the up-stroke. Additional information on flight kinematics can be found in [24].

2.2 Oscillating patterns and continuous-time neural networks

While the understanding of the neural mechanisms involved in bird flight is not complete, comparative studies [5] suggest that flying animals could resort to similar solutions to those securing locomotion in walking and swimming creatures. These solutions would call upon oscillating neural circuits called *Central Pattern Generators* (CPG) that generate rhythmic movements even in the absence of any sensory input, and that have been successfully implemented on walking and swimming robots [16, 9, 18, 17, 13] for instance.

It turns out that these rhythmic behaviors include a temporal component which is difficult to generate with a standard neural network [19] because the state of such a device changes at

[1] lower part of the wing.
[2] upper part of the wing.

discrete time-steps only. This is why continuous-time neural networks are more appropriate for the design of CPGs. Among the corresponding models, those calling upon leaky-integrator neurons are the most widely used [16, 9, 18, 17, 13]. The behavior of these neurons is defined by the following differential equation:

$$\begin{aligned} \dot{y_i} &= \frac{1}{\tau_i}\left(-y_i + \sum_{j=1}^{N} w_{ij}\sigma(y_i + \theta_j) + I_i\right) \\ i &= 1, 2, .., N \end{aligned}$$

where τ_j is the time constant of neuron j, w_{ij} the weight of the connection from neuron j to neuron i, θ_j is a bias, I_i a constant input and $\sigma(x) = \frac{1}{1+exp(-x)}$.

Implementing CPGs as networks of leaky integrator neurons has four advantages [3]:

- the leaky integrator is the simplest, non-linear, model of a continuous dynamic neuron;
- networks of such neurons are universal dynamic approximators [12] that can approximate the trajectory of any smooth system arbitrarily well;
- they stem from a plausible neurobiological background;
- by varying the neurons' time constants, the frequency of the rhythmic behaviors generated can be modulated without changing the topology of the corresponding network.

This is why this variety of neural network was used here.

2.3 Evolutionary algorithms and multi-objective optimization

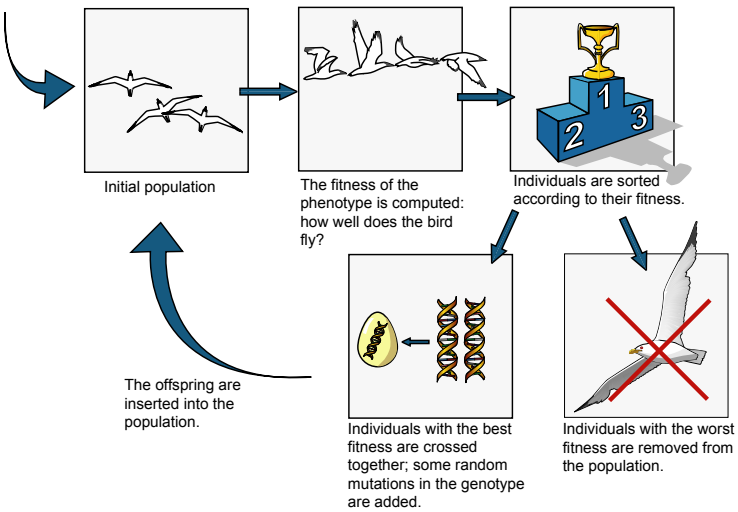

Figure 2: The general organization of an evolutionary algorithm.

The most convenient method for designing such controllers calls upon evolutionary algorithms, especially when the configurations of the networks likely to produce the desired behaviors are unknown. These optimization algorithms are loosely inspired by Darwin's theory of evolution, as illustrated in figure 2, and they unfold in the four following steps:

1. step 1. A random population is created;

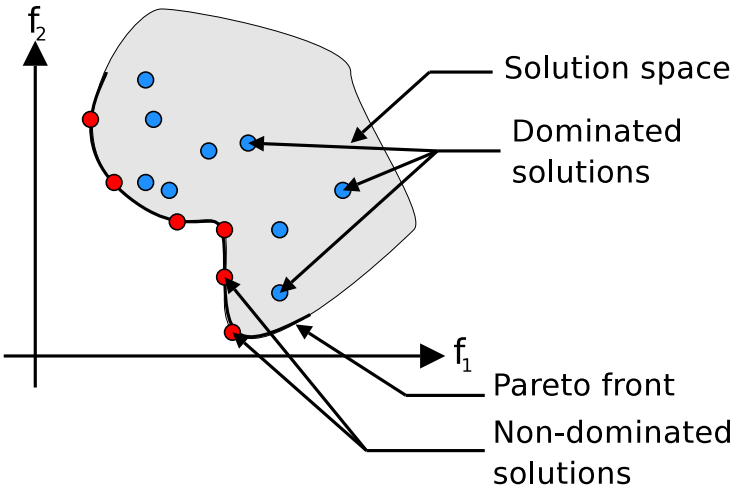

Figure 3: Among possible solutions to a given optimization problem, in which the two objectives f_1 and f_2 must be minimized, non-dominated solutions are represented using a red disk and dominated ones using a blue disk. No solution is better than a red one on both f_1 and f_2.

2. step 2. The fitness of each individual is computed and a subset of the population is selected according to the fitness;
3. step 3. Genetic operators like mutation and cross-over are used to form a new population;
4. step 4. if the desired number of generations or if performance criteria are not reached, return to step 2.

To implement these algorithms one must choose a way to encode each *phenotype* – i.e., each possible solution to the problem considered – into a *genotype* that will be manipulated by appropriate *genetic operators* that also must be hand-designed. The way this has been done for the current application will be described below.

One of the most interesting properties of such evolutionary optimization methods is their ability to deal with multi-objective problems. Indeed, while most optimization problems involve numerous objectives in practice, the standard approach to such problems is to transform them into single-objective ones, for example by using a weighted sum of the relevant objectives. Such practice raises the issue of choosing by trial and error the right set of weights, because no alternative method exists. Multi-objective optimization procedures find the set of all the compromises at once, among which a higher-level algorithm, or the user, may select the preferred one without the need to first choose relative weights.

Numerous algorithms have been proposed [4] to find the set of compromises. Most of them rely on the concept of *domination* and generate the so-called *Pareto Front* (figure 3).

Definition 1 *A solution $\mathbf{x}^{(1)}$ is said to dominate another solution $\mathbf{x}^{(2)}$, if both conditions 1 and 2 are true:*

1. *the solution $\mathbf{x}^{(1)}$ is not worse than $\mathbf{x}^{(2)}$ for all objectives;*
2. *the solution $\mathbf{x}^{(1)}$ is strictly better than $\mathbf{x}^{(2)}$ for at least one objective.*

This leads to the definition of the *globally Pareto optimal set*:

Definition 2 *The non-dominated set of the entire feasible search space is the globally Pareto-optimal set.*

In this work, to generate globally Pareto-optimal sets, we used MOGA (Multi-Objective Genetic Algorithm), an algorithm introduced in [11] that offers an interesting trade-off between ease-of-use and performance.

2.4 ModNet

The attempt to optimize both the structure and the parameters of a neural network raises important issues concerning the way the network should be coded and, then, handled by evolution. Classical approaches to the evolution of neural controllers [20] call upon individual neurons and connections that are assembled to generate whole networks. The problem with such approaches [8] is that evolution either starts from scratch – i.e., with randomly generated initial networks – or it is bootstrapped by a human designer who decomposes the original control problem into separate sub-problems to which partial solutions are sought. In the first case, the evolutionary process may need numerous iterations to converge. In the second one, it may be extremely difficult to decompose the initial problem and/or to recombine the separate networks into a single and coherent one. Moreover, it may happen that such an incremental procedure misses interesting solutions involving subtle couplings not foreseen by the experimenter [6].

Instead of manipulating individual neurons and connections, the ModNet encoding scheme [7] that is used in this work operates on *modules*, i.e., on small networks that are used to build the whole controller. They may be spontaneously discovered by the evolutionary process, or they may also stem from an initial pool of templates provided by the experimenter, either because, having been generated during previous evolutionary runs, they may be useful to solve the current problem, or because they encode some expert knowledge.

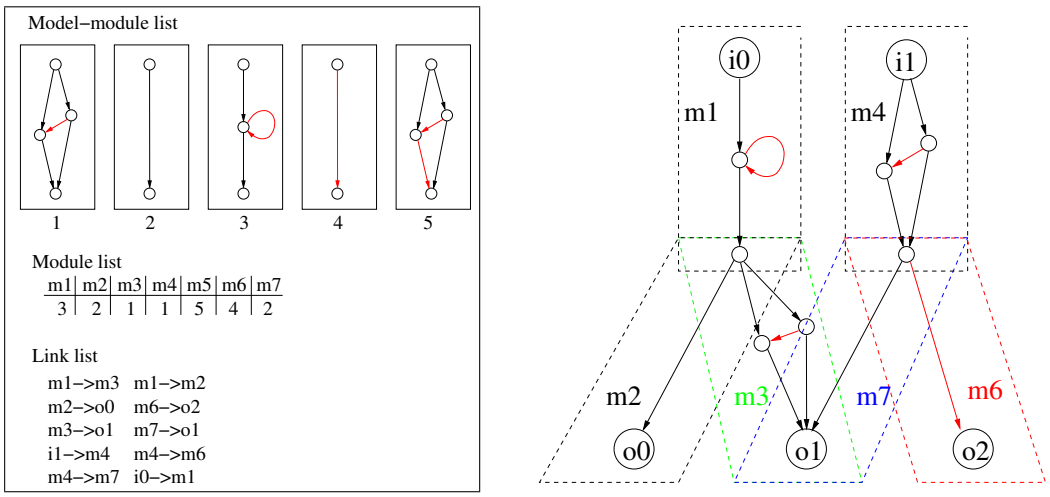

Figure 4: Left: Example of chromosome manipulated by ModNet. The chromosome is made of three components: a list of model-modules, a list of modules and a list of links. ml are modules, in are sensory (or input) neurons, and om are motor (or output) neurons. Right: the corresponding decoded neural network. Modules m3 and m4 are copies of the same model-module 1. Likewise, modules m2 and m7 are copies of model-module 2.

ModNet solutions are encoded in *chromosomes* made up of three components: a list of model-modules, a list of modules and a list of links (Figure 4). At the initialization of the algorithm, the first list contains a copy of randomly chosen modules of the initial pool. During the evolutionary process, some parameters of these modules – for instance connection weights – may evolve. The second list connects the number of a given module that will appear in the neural network with the corresponding model-module. This mechanism makes it possible for a given model-module to be included several times in the final controller, for instance to build a symmetric network or just to compress the representation of the neural network.

There are two categories of mutation operators in ModNet: those of the first category may modify parameters in the neural network., while those of the second category may change the network's structure by occasionally mutating each of a chromosome's three components. As for cross-over operators, they serve to exchange model-modules between genotypes. It turns out

that the set of these different operators fosters synergies that allow evolution to quickly propagate useful structures into the population [7].

2.5 Aerodynamic model

We have developed a computationally-efficient, but realistic, aerodynamic model (Figure 5) of any flapping-wing engine that makes it possible to compare various morphologies and flight controllers. This model is purely phenomenological and does not aim at taking all the complexities of aerodynamic behavior into account. In particular, it neglects the local direction of free flow velocity. Basically, it calls upon the classical splitting into lift and drag components of aerodynamic forces acting upon thin, small, quasi-plane, and rigid quadrilateral panels. These forces are computed as the sum of four independent contributions: friction drag, parachute, leading edge lift and leading edge vortex lift.

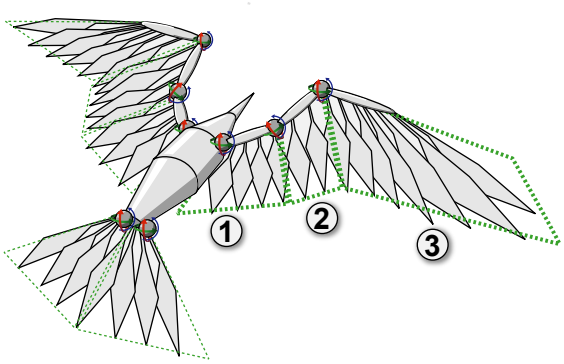

Figure 5: A modelled flapping-wing engine. Each wing is made of three panels. The body is made of cylinders and cones.

In the present work, each simulated engine's wing is modelled with three panels roughly representing the morphology of a bird's wing (figure 5) and each panel has three degrees of freedom : sweep, dihedral and twist. As for the engine's body, it is modelled with cones and cylinders.

3 Experiments

In principle, the components just introduced – a continuous time neural network, a multi-objective evolutionary algorithm, an appropriate encoding scheme, and an adequate aerodynamic model – allow the automatic generation of neural controllers able to take into account data acquired by miscellaneous sensors in order to modulate wing beats. In this preliminary work, we used these components to design simpler open-loop controllers according to which an artificial bird has to fly horizontally, at constant speed, and without sensory feedback. The objectives were to check the effectiveness of our approach, to assess the difficulties of synchronizing the degrees of freedom of a pair of wings, and to highlight the characteristics of any adapted rhythm that this approach would thus discover.

To this end, the evolutionary process was set so that it could generate only neural controllers exhibiting no sensory inputs, on the one hand, and three motor outputs corresponding to a wing's three degrees of freedom, on the other hand. Thus, symmetry constraints were imposed on the flying engines, which were forced to beat their two wings in synchrony. No other constraints

were imposed on the topology of the neural networks produced, which could thus include any number of hidden neurons and connections.

3.1 Fitness evaluation

The performance of each controller was assessed in a simulated wind-tunnel, in which the artificial bird faced a constant speed air flow and could only move its wings. The corresponding lift, traction and moments were evaluated at each time step and averaged after a 1024-step evaluation period in order to quantify the fitness of the current solution.

The fitness function we used had four different objectives to optimize. They describe the different aspects of the desired behavior of our bird:

1. The bird should fly at a constant speed. The difference between the traction force and the drag must then be minimized : $-\frac{1}{N}\left|\sum_N \overline{F_{traction}(n)} - \overline{F_{trainee}(n)}\right|$;
2. the bird should move horizontally. The difference between the lift and the weight must then also be minimized : $-\frac{1}{N}\left|\sum_N \overline{F_{portance}(n)} - \overline{F_{poids}}\right|$;
3. the bird must be stable. Its mean aerodynamical moments have to be as small as possible : $-\frac{1}{N}||\sum_N \overrightarrow{M_{aero}(n)}||$;
4. energy consumption should be minimized: $-\frac{1}{N}\sum_N E(t)$.

where $F_{traction}(n)$ is the traction force at time n, $F_{drag}(n)$ the drag force, $F_{lift}(n)$ the lift, F_{weight} the weight, $M_{aero}(n)$ the aerodynamical moment, and N the number of evaluation time-steps.

Finally, to avoid the drawbacks of evaluating fitnesses during short periods, three additional constraints were taken into account according to the *penalty method* described in [4]. This procedure forced evolution to discover solutions that forced the wings to move [3], and that were periodic, with a constant amplitude [4].

3.2 Parameters

Population size was 300. Only one template seeded the initial pool. Its internal structure was randomly generated when copied from the pool to a chromosome. This structure was then retained with a low mutation rate (10^{-4})). A maximum number of 10 modules per chromosome was allowed, with less than 5 hidden neurons and less than 10 connections each. Input and output neurons were all characterized by a time constant of $0.025s$, a bias of 0 and a slope of 1. All other parameters were submitted to evolution. The time constant of each hidden neuron could vary between 1.0 and 0.001, their bias between 0 and 2, and their slope between 0 and 5. Each connection weight could vary between 0 and 32.

3.3 Results

Figure 6 represents the Pareto front obtained after 420 generations. It thus appears that most solutions are both stable (first objective) and entail a low energy consumption (fourth objective), thus suggesting that stability and energy-efficiency are easy to optimize and to combine with the other objectives. On the contrary, the traction and lift objectives seem antagonistic since most solutions which generate a good traction fail on the lift objective – and reciprocally.

[3] This constraint penalized gliding behaviors that could otherwise get high fitness values, thus competing with non-optimal flapping wing behaviors especially on energy and stability objectives.

[4] This constraint ensured a steady-state behavior that couldn't be properly evaluated otherwise as the evaluation time is very short.

Besides the automatic and quantitative evaluation of its fitness through simulated wind-tunnel experiments, the quality of each solution could also be evaluated by letting the corresponding bird fly freely in a simulated landscape. Whereas, a majority of individuals in the first generations were unable to stay horizontal after a few wing beats and quickly crashed, more and more individuals able to generate convincing bird-like flights, with regular and energetically thrifty wing beats, were generated. Two such solutions are singularized on figure 6, with a dotted and a dashed line respectively.

Solution (a) (figure 7) is relatively smooth and energy efficient. The dihedral and twist oscillate at a frequency of approximately $7Hz$. Their phase difference is approximately of $\frac{\pi}{4}$. Successive snapshots of this behavior are shown on figure 8.

Solution (b) (figure 9) presents sharper twist variations and is more energy consuming. The dihedral and twist oscillate faster, at $12.5Hz$.

Both of these solutions share a common feature: the twist is positive during a down-stroke and negative during the up-stroke. It turns out that this characteristic is mandatory to generate an efficient flight and that evolution has spontaneously discovered it. It should also be noted that the corresponding behaviors are both periodic, but are not generated by mere sinusoid activations of the controllers' output neurons. This gives some clues about how diverse efficient flying behaviors may be.

The neural network that generated the behavior plotted on figures 7 and 8 is displayed on figure 10. This network contains five modules coming from three different model-modules. Module 3 forms a recurrent loop that allows the network to generate the cyclic behavior. The controllers of most of the individuals in the final generation included between 11 and 15 neurons. These network sizes are then small enough to be implemented on a dedicated on-board computing system.

Additional work will be devoted to the study of the inner workings of these neural networks with the hope that this will serve to extract relevant modules that could bootstrap further evolutionary runs, and prove to be useful for the resolution of more elaborate problems.

4 Conclusion

The results just described demonstrate the effectiveness of the proposed methodology. In particular, it seems capable of generating a variety of flapping-wing rhythms that may be adapted to the engine's morphology or to specific environmental constraints. Moreover, because evolving open-loop controllers thus appears to be relatively easy, one may hope that the same approach will also be efficient for relaxing the synchrony constraint and for designing close-loop controllers able to take sensory inputs into account and able to generate more complex and adaptive flight behaviors. This is the major short-term objective of the Robur project [1].

References

[1] Stéphane Doncieux Jean-Baptiste Mouret Laurent Muratet Jean-Arcady Meyer. Projet robur: vers un animat à ailes battantes autonome. In *Journées MidroDrones*, Toulouse, 2004. (sous presse).

[2] P. Augustsson, K. Wolff, and P. Nordin. Creation of a learning, flying robot by means of evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO02)*, 2002.

[3] R. Beer. On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469–510, 1995.

[4] K. Deb. *Multi-objectics optimization using evolutionnary algorithms*, volume -. Wiley, 2001.

[5] Michael H. Dickinson, Claire T. Farley, Robert J. Full, M. A. R. Koehl, Rodger Kram, and Steven Lehman. How animals move: An integrative view. *Science*, 288:100–106, April 2000.

[6] S. Doncieux and J.-A. Meyer. Evolving neural networks for the control of a lenticular blimp. In G. R. Raidl et al., editor, *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*. Springer Verlag, 2003.

[7] S. Doncieux and J.-A. Meyer. Evolving modular neural networks to solve challenging control problems. In *Proceedings of the Fourth International ICSC Symposium on ENGINEERING OF INTELLIGENT SYSTEMS (EIS 2004)*, 2004.

[8] S. Doncieux and J.A. Meyer. Evolution of neurocontrollers for complex systems: alternatives to the incremental approach. In *Proceedings of The International Conference on Artificial Intelligence and Applications (AIA 2004)*, 2004. (to appear).

[9] O. Ekeberg, A. Lansnser, and S. Grillner. The neural control of fish swimming studied through numerical simulations. *Adaptive Behavior*, 3:363–384, 1995.

[10] D. Filliat, B. Girard, A. Guillot, M. Khamassi, L. Lachèze, and J.A. Meyer. State of the artificial rat psikharpax. In *Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior*, Los Angeles, 2004.

[11] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization : formulation, discussion and generalization. In *Proceedings of the Fourth International Conference on Evolutionary Programming*, pages 416–423, 1993.

[12] DK. Funagashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6:801–806, 1993.

[13] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2):151–183, 1995.

[14] J. Haugeland. *Artificial Intelligence: The Very Idea*. Bradford/MIT Press, Cambridge, Massachusetts, 1985.

[15] A. J. Ijspeert, J. Hallam, and D. Willshaw. Evolving swimming controllers for a simulated lamprey with inspiration from neurobiology. *Adaptive Behavior*, pages 151–172, 1999.

[16] A. J. Ijspeert and J. Kodjabachian. Evolution and development of a central pattern generator for the swimming of a lamprey. *Artificial Life*, -, 1999. In Press.

[17] J. Kodjabachian and J.-A. Meyer. Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE Transactions on Neural Networks*, 9:796–812, 1997.

[18] M.A. Lewis, A.H. Fagg, and G.A. Bekey. Genetic algorithms for gait synthesis in a hexapod robot. In Y.F. Zheng, editor, *Recent trends in mobile robots*. World Scientific, 1993.

[19] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, (5):115–133, 1943.

[20] J.-A. Meyer. Evolutionary approaches to neural control in mobile robots. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, 1998.

[21] J.A. Meyer and A. Guillot. Simulation of adaptive behavior in animats: Review and prospect. In Meyer and Wilson, editors, *Proceedings of The First International Conference on Simulation of Adaptive Behavior*. The MIT Press, 1991.

[22] R. C. Michelson. *Neurotechnology for Biomimetic Robots*, chapter The Entomopter, pages 481–509. The MIT Press, 2002.

[23] R. C. Michelson and S. Reece. Update on flapping wing micro air vehicle research: Ongoing work. In *13th Bristol International RPV Conference*, 1998.

[24] U. M. Norberg. *Vertebrate Flight*. Springer-Verlag, 1990.

[25] J. Yan, R. Wood, S. Avandhanula, M. Sitti, and R. S. Fearing. Towards flapping wing control for a micromechanical flying insect. In *IEEE Int. Conf. Robotics and Automation*, 2001.

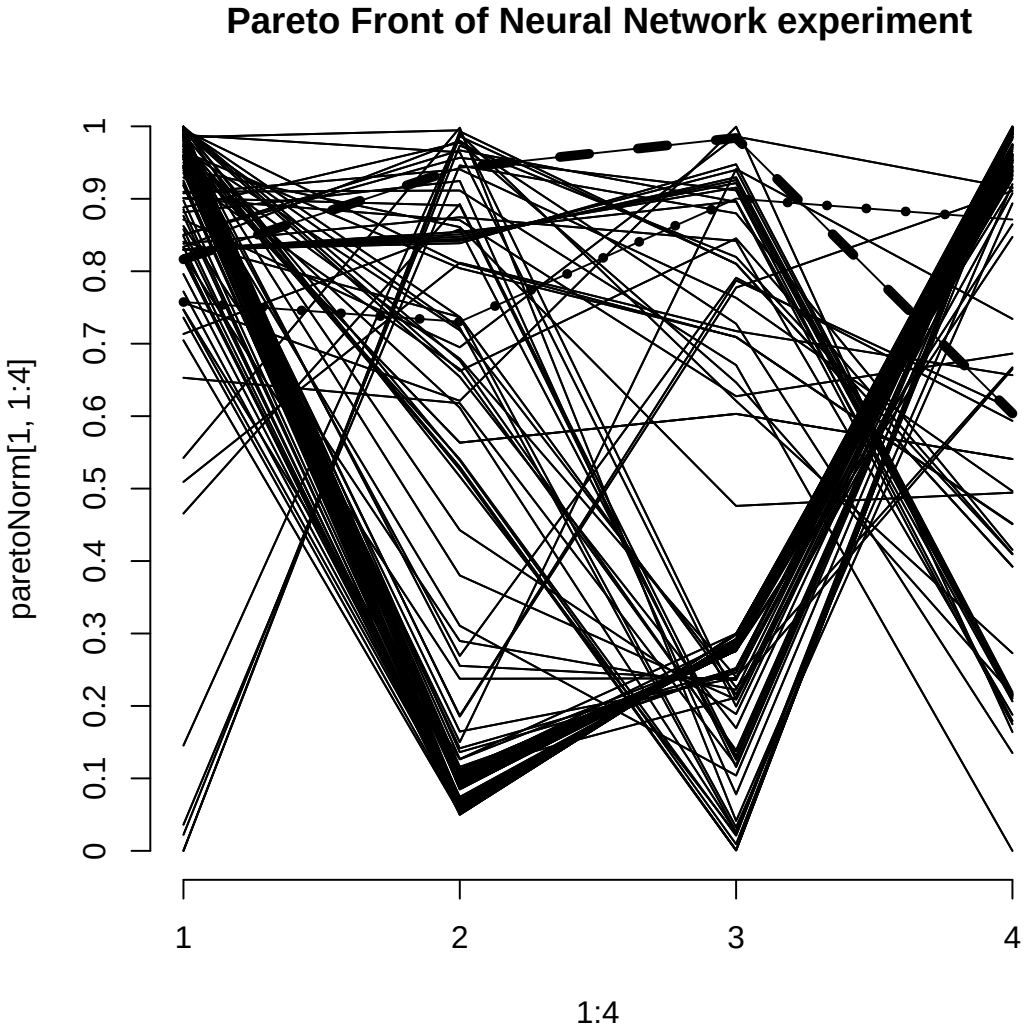

Figure 6: Pareto front obtained after 420 generations. Each line represents an individual of the Pareto front. The X axis represents the different criteria, and the Y axis the value of these criteria normalized according to the minimum and maximum values observed on each criterion (on all experiments). The first criterion is the mean aerodynamic moment, the second one corresponds to the lift, the third to the traction, and the fourth to the energy. The cases of two individuals, respectively represented by a dotted and a dashed lines, are singularized on the figure and discussed in the text.

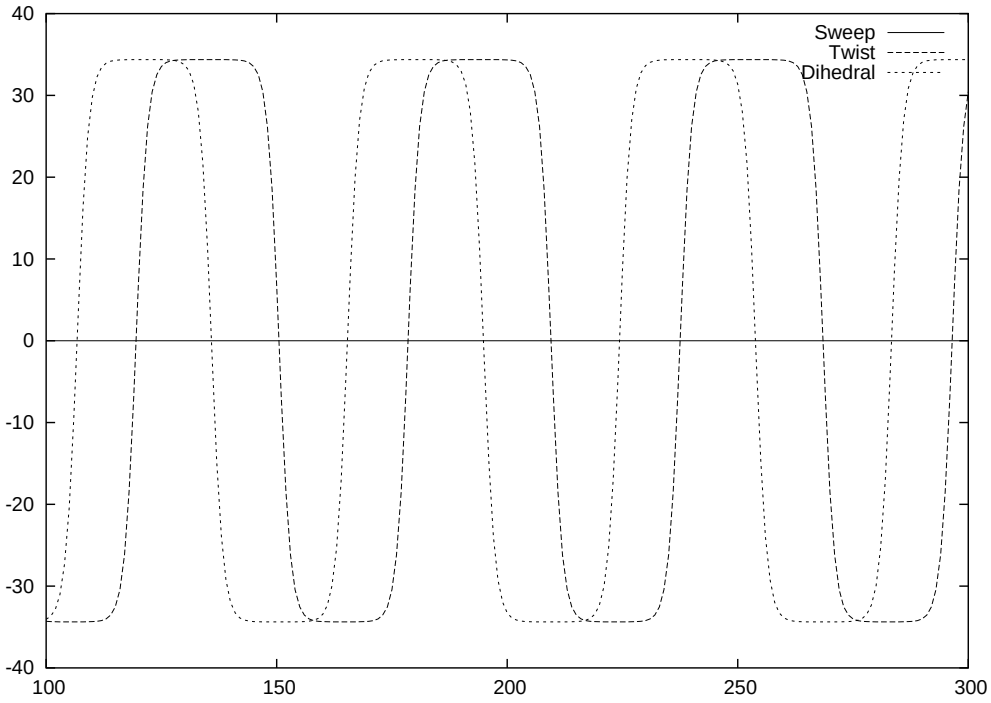

Figure 7:
Wing movements corresponding to solution (a) that are represented with a dotted line on figure 6. The X axis represents time steps (a time step equals 0.0025s), the Y axis represents the amplitude of each effector (in degrees, the maximum amplitude is 35 deg)

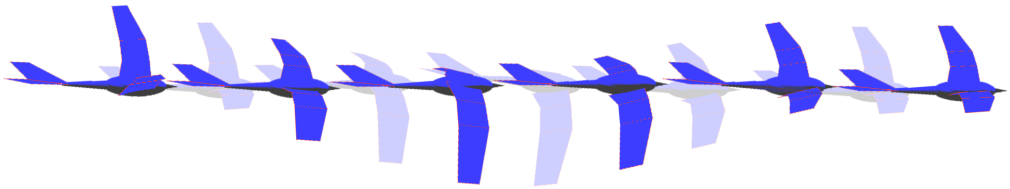

Figure 8: Example of a bird-like behavior generated by the evolutionary process.

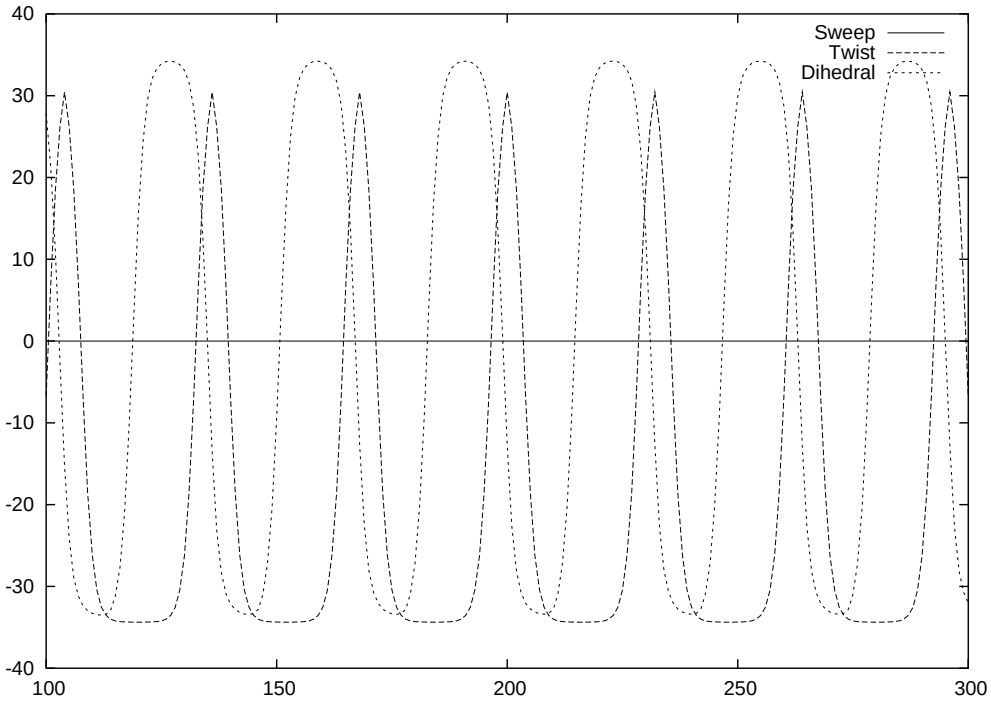

Figure 9: Wing movements corresponding to solution (b) that is represented with a dashed line on figure 6. The X axis represents time steps (a time step equals 0.0025s), the Y axis represents the amplitude of each effector (in degrees, the maximum amplitude is 35 deg)

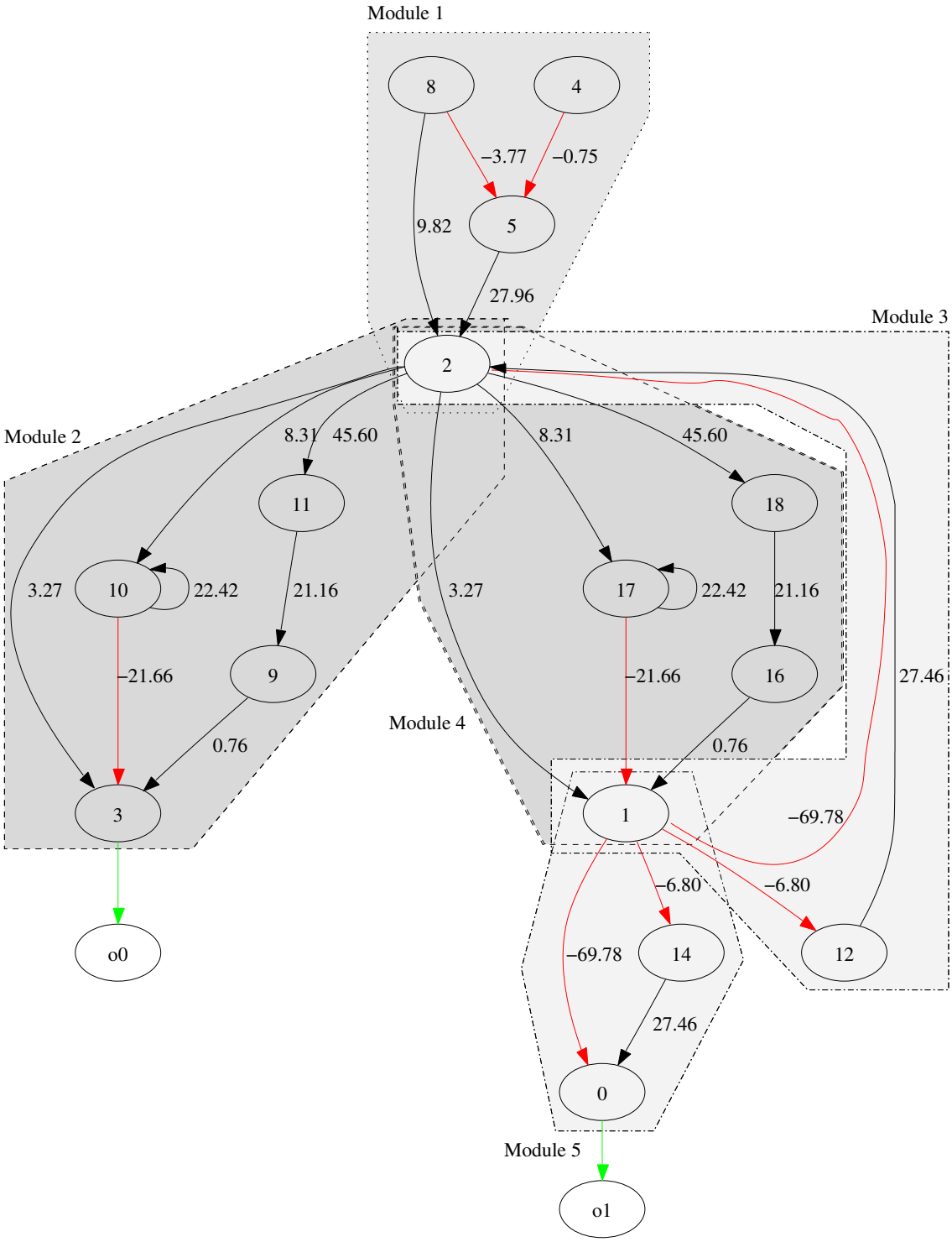

Figure 10: The neural network that generates the behavior plotted on figures 7 and 8. Modules are displayed using grey blocks. The neuron labeled $o0$ is linked to the dihedral and the one labeled $o1$ to the twist. This network did not use the sweep.