Optimizing Morphology and Locomotion On a Corpus of Parametric Legged Robots

Grégoire Passault, Quentin Rouxel, Remi Fabre, Steve N'Guyen, Olivier Ly

LaBRI, University of Bordeaux, France.
{gregoire.passault, quentin.rouxel, remi.fabre, steve.nguyen, olivier.ly}@labri.fr

Abstract. In this paper, we describe an optimization approach to the legged locomotion problem. We designed a software environment to manipulate parametrized robot models. This environment is a platform developed for future experiments and for educational robotics purpose. It allows to generate dynamic models and simulate them using a physics engine. Experiments can then be made with both morphological and controller optimization. Here we describe the environment, propose a simple open loop generic controller for legged robots and discuss experiments that were made on a robot corpus using a black-box optimization.

1 Introduction

Eadweard Muybridge did an important early work in studying animals locomotion, analyzing photographs of animals during different phases of strides [1]. What is noticeable is that there is a lot of similarities in how different animals move at corresponding paces.

These similarities are explained by studies on Central Pattern Generators (CPGs), that pointed out that the nervous system is able to produce rhythmic motor patterns even in vitro [2]. This is called fictive motor patterns, and this points out that animals locomotion may be based on hardware encoded *a prioris*.

Simple physical models such as the SLIP (spring-loaded inverted pendulum), can also describe running animals. It is interesting to mention that animals with two to eight legs can fit this model when running, where groups of legs acts in concert so that the runner is an effective biped [3].

One of our goals is to explore how some classical gaits like the trot, and in lesser extent the walk, could appear naturally by optimizing trajectory-based locomotion controllers. We explore how the locomotion solutions of animals, provided by hundreds of millions years of evolution, corresponds to solutions issued from statistical optimization on pure physics based criteria, and in this way appear to be canonical.

What we propose here is to optimize the locomotion trajectories on a parametric robot corpus as well as morphology/anatomy. In this context, we propose

a controller with *a priori* knowledge (such as symmetry) allowing to simplify the search space of the optimization process.

This work is based on a technology developed by authors allowing to simulate a parametric family of legged robots. It is part of the Metabot [4] project, which consist in developing an open-source low cost robot for education and research.

We will first describe the architecture of the system that was developed, and then the generic controller that can make all the robots of the corpus walk. Then, we will discuss the optimization experiments that were made and the obtained results.

2 Related Works

The GOLEM project [5] proposed to evolve both the robot's morphology and its controller, using neural networks and quasi-static simulation. They generated manufacturable walking robots without any *a priori*.

More recently, Disney Research [6] proposed a system allowing casual users to design and create 3D-printable creatures. They proposed a method to generate motion using statically stable moves on a robot model given by the user. They also proposed a way to generate printable geometry from the custom parts.

The presented work is not restricted to statically stable motions. Instead, dynamic locomotion patterns are explored - which mean that it is not assumed that the robot have to be statically stable at each time step. The ability to print the robots is also proposed.

[7] co-evolved pure-virtual robots in dynamics simulator and proposed a script language for robot structures.

[8] produced real-world robots whose morphology and controller were optimized by simulation. The study underlines an important gap between simulation the physical world. Our simulation tries to reduce this gap by including an estimation of the shocks and the backlash.

[9] studied the legged locomotion space, trying to adapt the locomotion when there is damaged parts using pre-computed behaviour-performance maps.

There are also several works on fixed architecture robot's locomotion tuning, like the teams that worked on the Aibo robots [10] or Bostom Dynamic's Little Dog [11], or the locomotion learning from scratch [12].

In our system, not only the robot locomotion but also its morphology is optimized.

3 System Architecture

3.1 Robots Modelling

To model the robots, we developed a custom tool that uses the OpenSCAD [13] language as backend. This language can be used to describe 2D or 3D models using code, which makes extensive parametrization possible. This code is actually based on constructive solid geometry (CSG), where boolean operations are

applied to basic objects to combine them. OpenSCAD can produce meshes and CSG trees, which contains the basic objects, boolean operations and transform matrices.

We customized OpenSCAD to add metadata in the code. We extended the parts description language to add markers in the CSG tree so that we can automatically retrieve information such as reference frames such as anchor or tips. Thus, we are able to specify in the generated part some reference frames that we can retrieve once the part is compiled and to tag specific parts of the CSG tree.

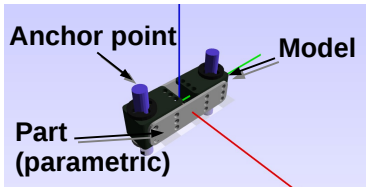

Fig. 1: A component contains models, parts and anchor points.

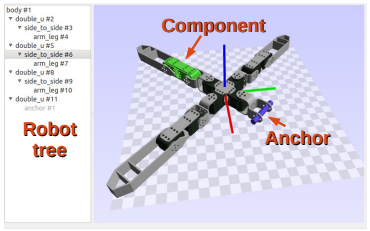

Fig. 2: An overview of the robot editor, using parametric components library.

Moreover, we also added the notion of *component*, which contains *models* of "real-life" existing objects (such as motors), *parts* that should be manufactured to make the robot and *anchors* that are points where other compatible anchors can be attached (see figure 1). Components can have parameters, that can modify some features (for instance mechanical lengths) of the parts that it contains.

We then created a components library and an editor that can be used to instantiate and attach components together. The edited robot is actually represented as a tree with components as nodes and anchors as child relation (see figure 2[1]). We also added global robot parameters that can impact multiple components. This results in parametric robots that can be re-generated with new parameters to change some morphological features.

All the parts of a robot can be exported, generating meshes that can be used for example for 3D printing.

3.2 Kinematic and Dynamics

Since we know the transformation matrices for each part of the robot tree, we can compute the kinematic model for a given robot topology and its parameters. Leg tips are also tagged in the components.

[1] Videos showing the editor is available at `https://www.youtube.com/watch?v=smHctwi05Ic`

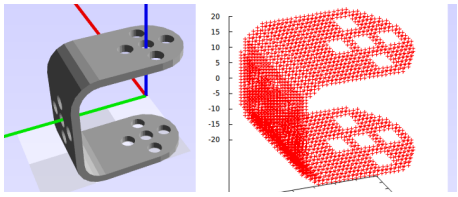

Fig. 3: Voxels are computed from the actual part meshes to compute the complete dynamic properties of the model.

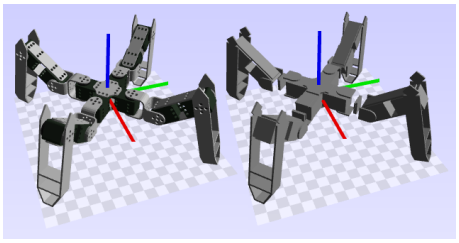

Fig. 4: Robot parts and models and its pure shape collisions approximation for simplified collision computations.

It is also possible to compute the dynamics model. To achieve this, parts and models are turned into small voxels (cubes) of typically $1mm^3$. This can be done using even-odd rule [14] on each point of the bounding box. With these cubes, we can deduce the volume of the part and the mass distribution (see figure 3). The center of mass can then be known using a weighted average and the inertia with an integration of the cubes.

Parts have a pre-determined homogeneous density, and models (for example a well-known motor) can have a fixed overall mass, in this case, the density is adjusted to fit the mass.

All the parts have a corresponding (parametric) pure shape approximation. This is made easy thanks to the constructive geometry, as it is already based on basic objects. Most of the part collision approximations simply consist in removing items from the tree such as screw holes. Then, we parse the CSG tree and directly retrieve the pure shapes (see figure 4). To avoid auto-collision glitches, these shapes can be automatically "retracted" (each pure shape is slightly downsized, so that near shapes such as a motor horn don't collide because of numerical approximations when rotating).

We can then inject all these information into a dynamics engine.

3.3 Corpus

We designed a corpus containing 22 robots, with 4 to 8 legs and 8 to 24 motors. The parts are designed to be manufacturable (3D printable). The modeled motor is similar to an on-shelf low-cost motor (Robotis XL-320). Robots vary in shape and thus have different morphological parameters (see figure 5).

All the robots are facing the X-axis, which is the arbitrary "front" of the robot. Some degrees of freedom range were limited artificially to disambiguate the kinematic (see below).

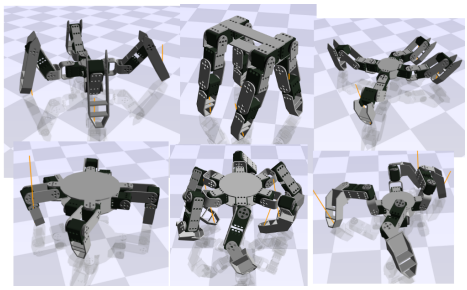

Fig. 5: An overview of six robots of the corpus.

4 Generic Controller

Here we introduce a generic controller for locomotion that takes a robot model and generates motor trajectories. This controller allows to steer the robot according to a given speed vector.

4.1 Principle

The locomotion task to be solved consists in following a given speed and rotational velocity described by the dynamic controls $(\dot{x}, \dot{y}, \dot{\theta})$. Since the robots have a significant number of degrees of freedoms, it is clear that there are many different ways to achieve this. For more convenience, the legs are ordered using their anti-clockwise position around the z-axis. Thus, the front left leg is always the first one and the front right the last.

In order to reduce the parameter's space, we consider that the robot's height during the walk remains constant and that the body yaw and roll are null.

If the robot is indeed following the dynamic controls, the leg trajectory on the ground during the support phase will then be determined. We then use iterative inverse kinematics using simple stochastic method (we apply random variations on the angles and check if one of these variation lead to a result closest to the target and iterate again until it doesn't). Since we are dealing with small number of degrees of freedom per leg, this is efficient enough (This consumes far less computational power than the dynamics engine itself). Thus, we are able to control the tip position in the Cartesian (x, y, z) coordinates.

The leg trajectory is described using splines. In order to ensure smooth motions (and finite speeds) we have chosen cubic splines position for trajectories representation. We define three couples point/velocity (or locus) for this spline. The first two are defining the segment followed by the leg on the floor and the last one being the point reached when the leg rises (C).

Here, the two loci on the floor have both a fixed position and speed: the position is determined by the steps length and the speed is null during support phase. The third locus height is a parameter that defines how high the robot

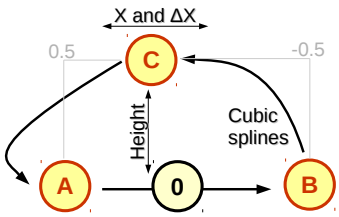

Fig. 6: Model of the leg trajectory that uses cubic splines and three locus, the "0" point is the position of the tip when the dynamic controls are null.

will rise its legs. Its position and speed belong to the optimized parameters of the controller that change the leg trajectory when raised (see figure 6).

We assume that the support duration is the same for each leg and is a parameter of the controller. Leg phases (i.e synchronization) are also parameters. The first leg (front left) is the reference phase that can be changed along with all the other leg phases accordingly. The frequency of the whole stride is another parameter.

Finally, the robot posture (i.e. the position of the legs when dynamic controls are null and the robot is static) can be parametrized with x, y and z, where z is the height of the robot and x, y are multiple of the leg position on the original robot model (if x=1 and y=1 the position of the legs are exactly the same as in the model, if x=0.5 and y=2, the legs are near the center of the robot along the sagittal plane and farther along the frontal plane, the symmetry is preserved).

4.2 Parameters summary

- **locus x, speed and height**: are locus parameters (see above and figure 6)
- **support**: the duration of the support of each leg
- p_2, p_3, **...** p_n (where n is the number of legs): the leg phases (p_1 is the "reference" leg, so there is only a set of parameters for a given gait)
- **frequency**: the number of stride per second
- **x, y and z**: are posture parameters that define the robot posture when the dynamic controls are null

Thus, the total number of optimized parameters for the walk controller is $(n-1)+8$ where n is the number of legs.

5 Experiments

5.1 Dynamics Engine

We use the Bullet physics library, an efficient open-source dynamics engine [15]. In this case, it can simulate rigid bodies, torque controllable hinges and collisions.

To simulate the motors, we applied torques directly on the components. Joints control is made by applying a target velocity from the position error (proportional), and then applying a target torque from the velocity error (proportional). To do so we use gains manually tuned. The maximum motor speed and torque curve is considered, the maximum torque that can be applied depends linearly on the current speed, reducing to zero when the max speed is reached.

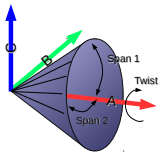

Fig. 7: The cone/twist constraint used to simulate backlash is a free degree of freedom, allowing the attached part to move in the span of the cone and twist.

In order to get more realistic results, we added a backlash simulation on the degrees of freedom, using a cone/twist constraint (see figure 7).

The collisions information can be retrieved from the dynamic simulation. They were used to detect both auto-collisions between the robot's parts and collisions between other parts than legs with the floor to add a penalty on the score (to avoid crawling behaviors for instance).

Simulations were made using a time accuracy (Δt) of $1ms$, and all the parts had a friction coefficient of 0.5.

In order to make experiments possible, morphological parameters were rounded to millimeters and a cache system was added to avoid compiling and voxelizing parts that were already met before (see figure 8).

5.2 Optimization

We use the black-box evolutionary algorithm CMA-ES [16] [17] as optimizing algorithm, with the following meta-parameters (all the problem parameters were normalized between 0 and 1):

Meta-parameter	**Value**
algorithm	BIPOP CMAES
restarts	3
elitism	2
f tolerance	10^{-6}
x tolerance	10^{-3}

Notice that the *f tolerance* is maybe the more critical meta-parameter, because it determines how accurately the score should be tuned. The optimization

indeed stops when the score is not optimized more than this tolerance during a certain number of iterations. In our case, since each fitness evaluation typically takes a few seconds, a bad value for this can lead to hours of useless computation trying to tune meaningless small values.

The dimension of experiments vary with morphological parameters and number of legs (because of the phases) from 15 to 20.

5.3 Score

The goal of the first experiment consists in walking forward, i.e. going as far as possible on the X axis. Two scores were tested, the first is simply the inverse of the distance walked (minimization), and the second is the inverse of the distance walked multiplied by the energy cost, which is the sum of all the impulses ($N.m.s$) sent to the motors.

A second experiment consists in reaching checkpoints. Four near points have to be attained successively by walking forward and turning to get the body aligned with the target. This introduces the trajectory control which adds a few more parameters to be optimized. Here, giant steps were used on the score with each checkpoint passed. In order to help the convergence of the optimization, if the last checkpoint is not reached, the score is related to the distance to the next missed checkpoint. If the last checkpoint is reached, two scores were tested, the first is the duration of the experience (trying to minimize it), and the second is the duration multiplied by the energy cost.

Finally, during this two experiments, we optimize the walk using a small fixed frequency to compare the resulting gait.

5.4 Workflow

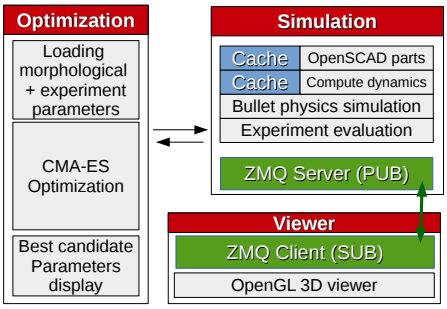

Fig. 8: Software architecture.

The overall architecture is shown on figure 8. CMA-ES runs parallel simulations with parameters set, each simulation first compile the dynamics of the

robot, trying to hit the (filesystem) cache, speeding up the process. Simulations are headless, but can also stream their state and are viewed using a custom 3D client GUI.

6 Results

While there is no guarantee that CMA-ES finds the global optimum, the optimization still provides very satisfactory sets of parameters for each robot.

6.1 Leg Phases

A first result can be viewed on the leg phases of the optimized robots. The (p_2, p_3, p_4) plot of all optimized quadruped robots (all experiments mixed) is shown on figure 9, where all the points appear to be along a long quasi ellipsoid.

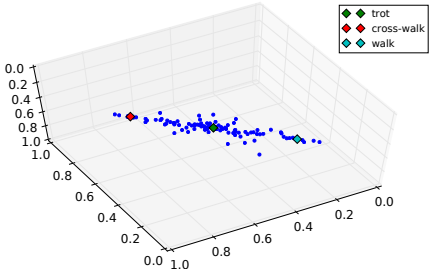

Fig. 9: p_2, p_3 and p_4 for all optimized quadruped are along an ellipsoid (p_3 was shifted by 0.5 to center it)

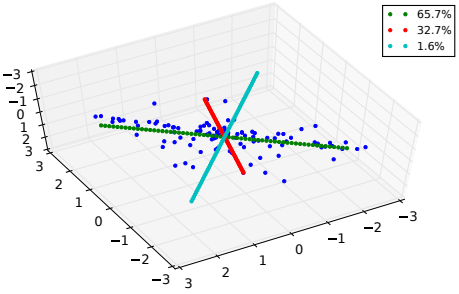

Fig. 10: PCA analysis of the points from figure 9, data are standardized and the legend corresponds to the proportion of variance explained by each of the principal components.

The trot gait is on the middle of the ellipsoid, the walk on one side and the "cross" walk on the other side (see figure 11). Doing a Principal Component Analysis (PCA) reveals that the first axis explains 65% of the variance and the second one 32% (see figure 10).

6.2 Locus

We plotted the histogram of locus for all experiments, resulting in figure 12.

We can notice that the locus X values are mostly in the $[0, 0.5]$ interval, which means that the leg trajectory will go slightly forward when rising.

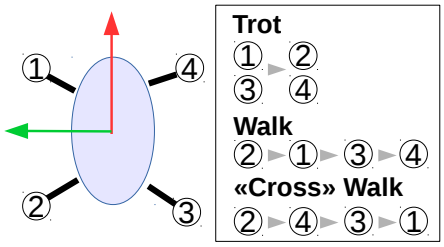

Fig. 11: Different quadruped gaits mentioned in this paper. The diagram on the right is the order of the rising legs (which correspond to leg phases).

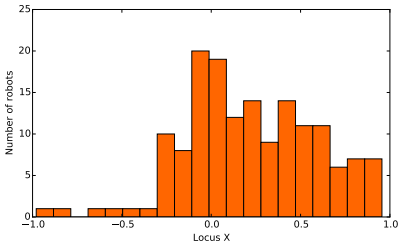

Fig. 12: Histogram of locus X for all experiments (see figure 6).

6.3 Walking with Low Frequency

We also tried optimizations with fixed frequencies that were not tuned.

We observe that this results in higher robot postures (see figure 13). Which indeed reduce energy cost but lose stability. More robots were able to exhibit a walk (or cross walk) during this experiment.

6.4 Checkpoints

The "walk forward" experiment often leads to over-tuned running robots, that sometime seem very unrealistic and likely not controllable. Moreover, walking forward doesn't prove that the robot can be reliably steered. This is why introduced the checkpoints experiment (see above). All the experiments converged to robots that successfully reach all the checkpoints.

The resulting robots walk slower (see Fig. 14) that those from the walk experiment.

6.5 Fast Walk

The experiment that consists in a fast walk disregarding the energy cost unsurprisingly converges to bigger robots in every cases (see Fig. 15). This is likely because the reachable space of legs is simply bigger, allowing bigger steps.

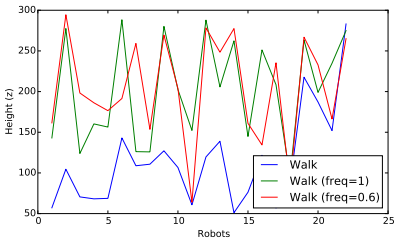

Fig. 13: The posture height (z) of the optimized robots from the corpus for optimized frequency and constrained frequency.

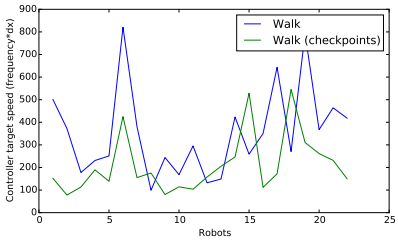

Fig. 14: The speed of the robots that the controller tries to reach ($frequency * dx$) for the walking and checkpoints experiments (with energy cost in the score).

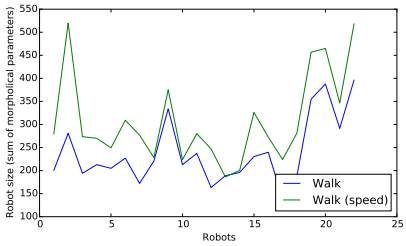

Fig. 15: The size of robots (which is the sum of morphological parameters) are compared in the walking experiment based on energy cost score and walking experiment based on pure distance score.

7 Conclusion and Future Work

We presented a framework allowing to generate and simulate parametric multi-legged robots. We also proposed a simple generic controller and used it through simulations experiments on a corpus of parametric robots. This controller was able to produce locomotion for all the robots of the corpus, but also to control them to reach arbitrary checkpoints[2].

Analysis of the simulations revealed that the search space could be reduced, like the leg phases for quadruped robots and the locus position that is almost always on the front. We also noticed that reducing the robot's energy cost imply a smaller morphology in every simulations, while making it steerable (with checkpoints experiments) reduces the speed. Moreover, the obtained leg phases for optimized quadruped robots seems very similar to animals trot and walk.

[2] A video of the obtained behaviors is available: `https://www.youtube.com/watch?v=GF1KM7JrmC0`

In a future work, the phases ellipsoid result will be checked on real robots with an appropriate experimental setup.

Some other optimizable parameters could also be added to the controller, like the orientation of the body.

The controller could also be improved in order to be able to handle robots with spine and legs kinematic chains that share degrees of freedom, thus greatly extending the capabilities of the robots.

It would also be interesting to compare CMA-ES with other optimizing algorithm for this specific experiment. For example a multi-objective optimization method could allow to find not only "optimal" candidates but also optimal populations of candidates.

Finally we could also produce automatically robot morphologies from scratch without a pre-defined corpus. This raises some problems like kinematic singularities, upstream filtering of bad candidates (for example all the motors on the same plane) and particularly adapting the optimization algorithm.

References

1. E. Muybridge, *Animals in motion*. Courier Corporation, 2012.
2. E. Marder and D. Bucher, "Central pattern generators and the control of rhythmic movements," *Current biology*, vol. 11, no. 23, pp. R986–R996, 2001.
3. P. Holmes, R. J. Full, D. Koditschek, and J. Guckenheimer, "The dynamics of legged locomotion: Models, analyses, and challenges," *Siam Review*, vol. 48, no. 2, pp. 207–304, 2006.
4. F. P. Grégoire Passault, Quentin Rouxel and O. Ly, "Metabot: a low-cost legged robotics platform for education," submitted.
5. J. B. Pollack and H. Lipson, "The golem project: Evolving hardware bodies and brains," in *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*. IEEE, 2000, pp. 37–42.
6. V. Megaro, B. Thomaszewski, M. Nitti, O. Hilliges, M. Gross, and S. Coros, "Interactive design of 3d-printable robotic creatures," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 216, 2015.
7. D. Marbach and A. J. Ijspeert, "Co-evolution of configuration and control for homogenous modular robots," in *Proceedings of the eighth conference on intelligent autonomous systems (IAS8)*, no. BIOROB-CONF-2004-004. IOS Press, 2004, pp. 712–719.
8. E. Samuelsen and K. Glette, "Real-world reproduction of evolved robot morphologies: Automated categorization and evaluation," in *Applications of Evolutionary Computation*. Springer, 2015, pp. 771–782.
9. A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.
10. B. Hengst, D. Ibbotson, S. B. Pham, and C. Sammut, "Omnidirectional locomotion for quadruped robots," in *RoboCup 2001: Robot Soccer World Cup V*. Springer, 2001, pp. 368–373.
11. P. D. Neuhaus, J. E. Pratt, and M. J. Johnson, "Comprehensive summary of the institute for human and machine cognition's experience with littledog," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 216–235, 2011.

12. P. Maes and R. A. Brooks, "Learning to coordinate behaviors." in *AAAI*, 1990, pp. 796–802.
13. "The programmers solid 3d cad modeller," http://www.openscad.org/.
14. K. Hormann and A. Agathos, "The point in polygon problem for arbitrary polygons," *Computational Geometry*, vol. 20, no. 3, pp. 131–144, 2001.
15. A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. ACM, 2007, pp. 281–288.
16. N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.
17. "Multithreaded c++11 implementation of cma-es family for optimization of nonlinear non-convex blackbox functions." https://github.com/beniz/libcmaes/.