A Study on Scalable Representations for Evolutionary Optimization of Ground Structures

Alexandre Devert[1], **Thomas Weise**[2], **Ke Tang**[3]
Nature Inspired Computation and Applications Laboratory (NICAL)
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
[1]marmakoide@yahoo.fr [2]tweise@ustc.edu.cn [3]ketang@ustc.edu.cn

Abstract

This paper presents a comparative study of two indirect solution representations, a generative and an ontogenic one, on a set of well-known 2D truss design problems. The generative representation encodes the parameters of a trusses design as a mapping from a 2D space. The ontogenic representation encodes truss design parameters as a local truss transformation iterated several times, starting from a trivial initial truss. Both representations are tested with a naive Evolution Strategy based optimization scheme, as well as the state-of-the-art *HyperNEAT* approach. We focus both on the best objective value obtained and the computational cost to reach a given level of optimality. The study shows that the two solutions representations behave very differently. For experimental settings with equal complexity, with the same optimization scheme and settings, the generative representation provides results which are far from optimal, whereas the ontogenic representation delivers near-optimal solutions. The ontogenic representation is also much less computationally expensive than a direct representation until very close to the global optimum. The study questions the scalability of the generative representations, while the results for the ontogenic representation display a much better scalability.

Keywords

Genetic algorithms, evolution strategies, problems representation, artificial embryogeny, topological optimization, structural mechanics.

1 Introduction

Nowadays, optimization problems involving thousands and even millions of free variables are routinely solved. Such feats are possible because the optimization algorithms put to good use mathematical properties of the search space, like for linear programming, quadratic programming, gradient-based optimization etc. Another way to achieve high scalability is to exploit the domain knowledge specific to a problem. In shape optimization, for instance, rather than directly optimizing a shape, the practice is to optimize an iterative shape transformation that turns an initial shape into the optimal shape. Working on shape transformations allows integrating knowledge about

the physics involved in the problem, such as heat diffusion and elasticity. Unfortunately, the precise mathematical structure of problems is not always well understood and approaches such as shape optimization require rather sophisticated mathematical treatments, which can be intractable.

The original motivation of evolutionary algorithms is to deal with problems where the mathematical structure is mostly unknown and where we have little idea about how to exploit domain knowledge. Because evolutionary algorithms rely only on comparisons between the objective values of candidate solutions, they can find optima with very little a priori knowledge. This strength comes at a price. In comparison to more informed algorithms, evolutionary algorithms do not scale well if the number of free variables increases. For a large class of problems with $\mathbb{R}^n$ as search space, Fournier and Teytaud (2010) prove that evolutionary algorithms have a convergence speed linear in n at best, whereas more informed optimization algorithms (such as gradient-based search) are super-linear. In other words, the more decision variables need to be optimized, the slower will optimization processes get and in the case of EAs, this slowdown is at least linear in the number of variables.

One way to bypass that scalability issue is to use indirect representations of the problem solutions in order to reduce the number of dimensions as much as possible. Instead of directly optimizing candidate solutions of the problem, solutions are constructed by a solution generator and it is the solution generator which is optimized. An indirect approach allows to introduce a bias towards some families of solutions, and can greatly reduce the number of free variables, compared to the number of variables of the direct representation. If such a mapping is smooth around the global optimum, this can speed up the optimization process significantly. Note that a mapping can introduce a *modelling error*: it might not be able to perfectly represent the global optimum of the direct representation. In such case, one can only hope getting close the optimum.

The main contribution of this article is a comparison between two families of indirect representations on the basis of a benchmark from engineering mechanics, ground structures optimization. The first representation, the *generative* representation, proposes to optimize a function that transforms points from a low-dimensional space into a solution of the problem. The second representation, the *ontogenic* representation, proposes to optimize an iterated mapping. That mapping transforms an initial guess of the solution into an hopefuly better solution after several iterations. The iterated mapping uses local informations to compute the next iterate. We find that *1)* the ontogenic representation has both, better scalability and *2)* produces better results than the generative representation. *3)* Its results are nearly as good as results obtained with a direct encoding where the optimizer directly improves the solutions, *4)* but much less runtime is required to obtain close-to-optimal results. *5)* The better scalability of the ontogenic representation also holds for two different optimization schemes, one based on a state-of-the-art optimization scheme (*HyperNEAT*, *NEAT* to optimize *CPPN*s that in turn generates solutions), one based on a naive Evolution Strategy approach. *6)* An optimization scheme applied on an ontogenic representation can further be improved by introducing a staged development, i.e., by dividing the optimization process into stages where each stage optimizes iterated mappings which are applied to the best shape found in the previous stage.

In the following section, we will first give a discussion of related work. We study both indirect representations on an established benchmark from structural mechanics:

topology design of ground structures. This benchmark is introduced in Section 3 and the experimental protocol used for the comparison is described in Section 4. In a first set of experiments, naive generative and ontogenic representations are applied to this problem, with optimization carried out using the same evolutionary optimizer. Thus, a clear and accessible analysis of the results is possible. The analysis of the experimental results, given in Section 5, includes the statistics of both the best objective value reached and the computational cost of each representation. As a baseline, a direct representation for the benchmark problem is also included.

Then, in Sections 5.1 and 5.2, complementary experiments are described and analyzed, in order to detail and explain the behavior of each representation. Building on the analysis of the ontogenic representation behavior, Section 5.4 introduces a staged development approach to reduce the computational cost of an ontogenic representation.

So far, we chose a naive optimization scheme for the experiments with generative and with ontogenic representations. It could be questioned whether such an optimization scheme may be biased against the generative representation. In order to refute such considerations, we also apply a state-of-the-art optimization scheme for a generative representation, namely the *NEAT* optimizer to optimize *CPPNs*, to the same experimental protocol in Section 6. In these experiments, generative and ontogenic representations display a behavior equivalent to the one obtained with our naive optimization scheme. The article ends with the conclusions in Section 7.

2 Background and Related Work

When an evolutionary algorithm is applied, in most cases, the only information it gets from the problem to solve are the objective values of potential solutions. This is equivalent to seeing the objective function as a black-box. Yet in practice, the computation of the objective value itself is far from being a black-box and can provide a lot more information relevant to the optimization problem. For many shape design problems, for instance, the objective value is derived from a finite element method (FEM) computation. The FEM provides very relevant local information (like air pressure, temperature, electrical charge, mechanical stress) about the behavior of the shape. For optical systems design, the objective value can be derived by tracing rays of light, giving local information about the light propagation through a candidate optical system. For neural network design, the training of a neural network can provide local information such as local error or local correlation of the neurons activity. For constraint satisfaction problems, the violated constraint and the amount of violation is used to compute an objective value. Although much other information is available from the evaluation of a candidate solution, still only one synthetic element of information is used by an evolutionary algorithm, the objective value. In that sense, the usual assumption that the objective function is a black-box is often untrue.

2.1 Indirect Solution Representations

Information provided by a candidate solution evaluation can be advantageously exploited by the problem representation. Estévez and Lipson (2007), for instance, optimize shadow-making shapes with an evolutionary algorithm. The shapes are defined on a square grid where a square is either full or empty. Full squares stop the light, ac-

cumulate heat when hit by light, and diffuse heat efficiently. Empty squares are transparent to light and do not diffuse heat well. The goal is to find shapes that minimize the average temperature of the grid when light is cast. Estevez et al. define shapes as the result of a cellular automaton work. When evaluating a candidate cellular automaton, the cellular automaton changes the state of a square depending on the temperature and the state of the surrounding squares. In return, the changes made by the cellular automaton change the local temperature. Thus, a shape is the result of the interaction between the cellular automaton dynamics and the heat diffusion dynamics. The only information used is the information created by the simulation which is necessary for computing the objective anyway, but the representation of the solutions can exploit the local heat knowledge. Estevez et al. show that this local information from the light and heat simulation is effectively used by the best solutions found by their evolutionary algorithm. Also, encoding shapes as the result of a cellular automaton process makes the genotype size more independent from the complexity of the shape. The complexity of the resulting shapes emerges out of the interaction with the simulated environment. Because this kind of representation of solutions is based on iterated transformation, an ontogeny, we call then *ontogenic* representations. Each iteration of the transformation on a initial solution form a *development*.

Using iterative transformations for each candidate solution evaluation seems potentially costly. Moreover, Stanley (2006, 2007) hypothesizes that most of the features attributed to an ontogenic representation (capacity to express symmetries, the reuse of functional parts with alterations, etc.) can be subsumed by more efficient and conceptually simpler representations. Stanley proposes to represent solutions of a problem as a spatial transformation. It is the core idea of the *HyperNEAT* approach. For instance, in (D'Ambrosio and Stanley, 2007) and further works, the synaptic weights of a large neural-network are represented as a transformation of a four-dimensional space. A synaptic weights of a neural network are computed by a function F that takes as input the spatial coordinates of the synapse input and output neurons. The neurons are embedded in a two-dimensional space, therefore, F takes four arguments. Stanley et al. then optimize F, as a direct acyclic graph of elementary basis functions, named a *CPPN*. Each evaluation of a candidate *CPPN* has a modest computational cost, just one evaluation of the spatial transformation. Since that type of representation generates solutions in a single-shot fashion, we will call them *generative* representations.

2.2 Taxonomies: Generative and Ontogenic Representation

Both generative and ontogenic representations are indirect representations of candidate solutions. Being indirect, they can reduce the search space dimensionality, eventually making it independent of the scale of the problem. Moreover, they both introduce a bias which hopefully makes the optimization problem easier. Generative and ontogenic representations differ notably in their way of introducing this bias. A generative representation maps potential solutions to a manifold. Therefore, the bias is fully contained in the generative function. On the other hand, an ontogenic representation can exploit problem specific knowledge by making the iterated transformation dependent on some specific local information. The representation bias comes from the candidate solution behavior itself. The objective value evaluation is likely to be more expensive with an ontogenic approach, since it consumes several expensive operations (the computation in each iteration). Applying an ontogenic representation also leads to the question of when to stop to iterate transformations and to evaluate the final result. The influence

of the stopping criteria for ontogenic representations is studied in (Devert et al., 2011).

The distinction done here between generative and ontogenic representations loosely matches one existing representation taxonomy. That taxonomy is introduced in (Bentley and Kumar, 1999), and distinguishes definitions of three categories of representations: *external non-evolved embryogeny*, *explicit evolved embryogeny*, and *implicit evolved embryogeny*. With an implicit evolved embryogeny, the connection between genotype and phenotype is of emergent nature, the result of localized transformations. With an *explicit evolved embryogeny*, the phenotype transformations are explicitly defined by the genotype. Finally, an *external non-evolved embryogeny* defines the mapping from genotype to phenotype in a single shot way, without a development sequence. An ontogenic representation decodes the genotype by iterating a local, context-sensitive transformation on the phenotype, making it an implicit evolved embryogeny. A generative representation is an external non-evolved embryogeny, since no development takes place when decoding the genotype.

Another taxonomy of representations for evolutionary algorithms is introduced in (Stanley and Miikkulainen, 2003). It also is an informal taxonomy and proposes five traits to qualify and distinguish representations. However, our distinction between generative and ontogenic does not really fit into that taxonomy. One of the proposed traits is the *complexification*, whether the genotype size is completely fixed or changes during the optimization. In our work, we do not assume anything about the way in which either a generative or an ontogenic representation are optimized. The *targeting* and *cell fate* traits concern characteristics of the transitions rules for a cells-based representation. When mentioning an ontogenic representation, we do not assume any mechanism to carry out the iterated transformation of the phenotype. We simply assume iterated transformations, with a feedback on the result of the transformation. The iterated transformations can be carried out by cells, by a differential equation system, or anything else. The *canalization* trait classifies representations by their brittleness when exposed to random mutations of the genotype. No such distinction is done in our work. Finally, the *heterochrony* trait concerns the handling of time when decoding from genotype to phenotype. As we defined it here, a generative representation does not have a notion of time when decoding a genotype. This is a sharp contrast with the ontogenic representation, where time is an essential aspect of the decoding of the genotype. Thus, in the framework of the Stanley et al. taxonomy, the central distinction between generative and ontogenic representation is the *heterochrony* trait.

The work presented in this article attempts to demonstrate that generative and ontogenic representations exhibit very different behaviors, in terms of scaling, convergence, and solution quality. In that sense, it challenges Stanley's hypothesis that a generative representation is essentially a faster equivalent to an ontogenic representation. Devert (2009) empirically demonstrates the striking differences between the two representations. He used the maximization of the overhang of a stack of blocks for demonstrating this dissimilarity. His ontogenic representation encodes the block positions as the result of an iterated transformation. The transformation moves blocks according to the interaction forces between blocks due to the gravity. With the ontogenic representation, the applied optimization algorithm always converges to very close to the global optimum. The number of fitness evaluation required to reach the optimum is shown to be independent from the number of blocks in the stack. The generative representation encodes the block positions as a function of their position in the stack. The same optimizer applied with the generative representation fails to converge close

to the global optimum, and the results worsen with increasing number of blocks in the stack. However the problem used in that study might seem too artificial to make it relevant. Also, Devert (2009) did not study the computational cost associated with the use of generative and ontogenic representations. As a fitness evaluation for an ontogenic representation have to compute iterated transformations, such an evaluation is likely to be very expensive. Thus, it remains unclear if the higher computational cost of an evaluation an ontogenic represention is worthy.

3 Truss Design with Ground Structures

Trusses are rigid structures, usually composed out of metal beams, which are supposed to hold or support some weight. They are amongst the most basic and widely used elements in engineering and architecture. The goal of the truss design process is to find a suitable arrangement of beams which can at least withstand a specified force at minimal or fixed material requirements.

The ground structure approach is a common and widely used formulation of an optimal truss topology design problem (Achtziger, 2007). This approach was first introduced in (Dorn et al., 1964). A ground structure is defined by a set of *nodes* in the plane or in space. A set of n *potential bars* link together pairs of nodes. The structure with support conditions is a *ground structure*.

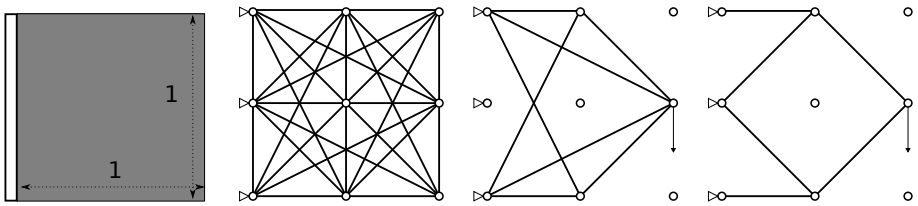


Figure 1: A ground structure is defined over a domain (1). Here the domain is a rectangle, on a Euclidean plane. A ground structure is made of joints connected by bars. Joints are at fixed positions and the connection matrix is also fixed. (2) The optimization variables are the cross-sections of the bars. A fixed quantity of bar material is available. Bars can have null cross-sections. (3) The same ground structure can express different trusses. (4) is another truss, made from the same ground structure as (3).

The bars behave like linear springs, with a Young modulus (linear elasticity of a given material) of 1. The nodes behave like joints between bars. When M forces ($f_1, \ldots, f_M$, the *loads*) are exerted on some nodes of the ground structures, the bars will react and the nodes will move. The displacements u of the nodes is the solution of the linear system

$$K(a)u_k = f_k \tag{1}$$

where $K(a)$ is the *stiffness* matrix of the structure and u_k, f_k respectively the displacements and the loads in global reduced coordinates. The construction of the stiffness matrix is done by a standard procedure called *assembling* which will not be detailed here, see Felippa (2010) for an in-depth description. For understanding this work, it is sufficient to know that $K(a)$ is a function of the nodes positions and the bars' cross section areas $a_1, \ldots, a_n$. Assembling $K(a)$ and computing u_k from f_k is an example of

Finite Element Method (FEM) computation, here applied to an elasticity problem. From u_k, it is relatively straightforward to retrieve information such the mechanical strain on each bar.

The nodes positions' are kept fixed: the only design variables are the bars' cross section areas. The reason for such a restriction is that simultaneous optimization of geometry and topology requires significantly more complex mathematical treatments when using classical optimizers. To compensate for the nodes positions' restriction, dense grids are used to obtain refined solutions. Grids with thousands of bars are not uncommon. Since the position of the nodes are not design variables, the bars length l_j remain constants too.

We use four benchmark problems, *square-beam*, *wheel*, *short-bridge*, and *beam*, as introduced by Achtziger and Stolpe (2007). Figure 3 provides detailed description of the four associated ground structures. For those problems, the optimal solutions are known, so meaningful comparisons of different problem representations are easier to do. The ground structure is the only difference between the four problems, and they share the following common optimization objective:

$$\min \frac{1}{2} f_k^T u_k \tag{2}$$

subject to the constraints

$$\sum_{j=1}^{n} a_j l_j \leq V, \tag{3}$$

$$a_j \in [0, 1] \tag{4}$$

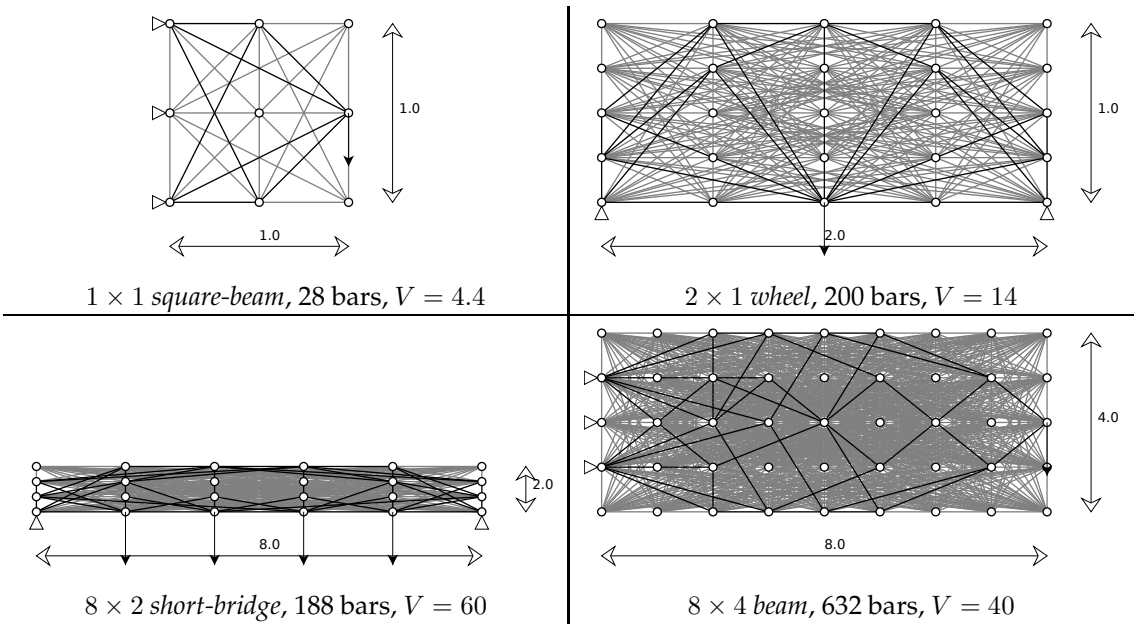
where V is a constant defined for each problem.

Such benchmark problems, despite their apparent simplicity and similarity, are relevant for and similar to many kinds of mechanical structure design tasks. Moreover, to study the scalability of a representation by introducing more design variables, we used the same problems with more refined ground structures. In these versions, the grid resolution is simply doubled, defining the *big-square-beam*, *big-wheel*, *big-short-bridge*, and *big-beam* problems. For these problems, the global optimum is unknown.

4 Experimental Protocol

We define three representations for the cross-section areas of the bars in a ground structure, *direct*, *generative*, and *ontogenic*, which we apply to all of the introduced benchmark problems.

- With the *direct* representation, the cross-section area of the bars are directly stored as a real-valued vector X. The cross section areas are first taken from X, according to the $a_i = X_i^2$ relationship, where a_i is the cross section area of the i-th bar. Then, the volume constraint of the ground structure is enforced by a linear scaling of the cross sections. The purpose of the square term is to enforce positive only cross-sections, without creating discontinuities in the fitness landscape.
- With the *generative* representation, the cross section area of each bar is a function F of the coordinates of its end-points. More precisely, the cross section a_i of the i-th



1×1 *square-beam*, 28 bars, $V = 4.4$ | 2×1 *wheel*, 200 bars, $V = 14$

8×2 *short-bridge*, 188 bars, $V = 60$ | 8×4 *beam*, 632 bars, $V = 40$

Figure 2: The ground structures for the *square-beam*, *wheel*, *short-bridge*, and *beam* problems. V is the volume of materials to use in the ground structure. The triangles mark the joints with freedom restrictions. The arrows show the 1-Newton loads. The domain size for each problem is given in front of the problem name. The bars appearing in black rather than gray are the bars with a non-null cross area from the optimal structure.

bar is computed as $a_i = F(U_i, V_i)$. (U_i, V_i) are the coordinates of the end-points of the i-th bar. Once the cross-sections areas are computed, the volume constraint of the ground structure is enforced by a linear scaling of the cross section areas. The function F is a perceptron, with 5 inputs (two 2D points plus one fixed bias input), one hidden layer of 8 $tanh$ neurons, and one single output. The weights vector of the perceptron is thus the genotype for this representation. The genotype size is $5 \times 8 + 8 \times 1 = 48$, independently from the number of bars.

- With the *ontogenic* representation, the cross section areas of the bars are the 32-th term of the sequence $a_i = G(s_i)$, where a_i is the cross section area of the i-th bar, s_i is the mechanical strain of the i-th bar, and G is a function. As sketched in Figure 4, after each computation of the cross section areas, the mechanical strain of each bar is updated, thus each term of the sequence costs one FEM computation. Also, the volume constraint of the ground structure is enforced by a linear scaling of the cross sections at each term of the sequence. The function G is a perceptron, with 2 inputs (bar strain plus one fixed bias input), one hidden layer of 8 $tanh$ neurons, and one single output. The weights vector of the perceptron is the genotype in this representation. The choice of the 32 steps is mostly arbitrary. The genotype size is $2 \times 8 + 8 \times 1 = 24$, independently from the number of bars.

For all three representations, the genotype are real-valued vectors. The *direct* representation genotype size equals the number of bars in the ground structure. In contrast, the two other representations feature a genotype size independent of the number of bars in

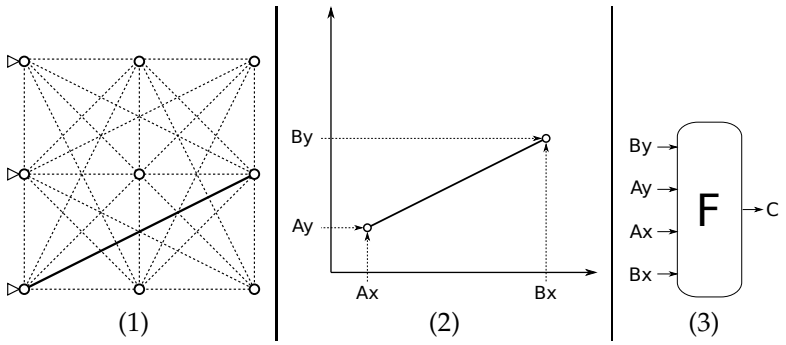


Figure 3: Generative representation for ground structures. (1) We consider each bar independently. (2) Each bar geometry is defined by its end-points coordinates. (3) A function F takes as input the end-points coordinates and outputs the bar cross-section. F is the same for all the bars.

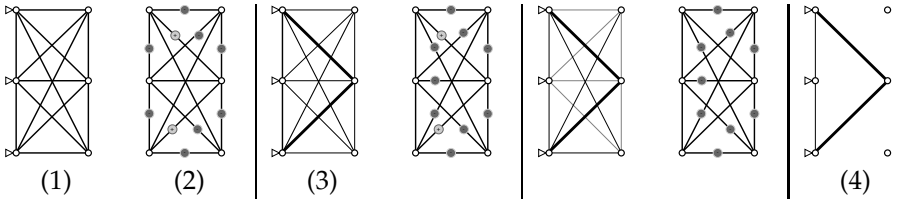


Figure 4: Ontogenic representation for ground structures. (1) The development starts with the bars of equal cross-section areas. (2) For each bar, a function G returns an increment for the bar cross-section area. (3) The cross-section area of the bar is modified according to these variations. (4) These steps are repeated several times. The resulting structure is evaluated to compute the objective value.

the ground structure. Table 1 displays the genotype size for the *generative* and *ontogenic* representations as functions of the number of neurons. 64 independent optimization runs are performed for the three encodings, applied for each benchmark problem. The optimizer used for every run of each experiment is the *SepCMA* evolution strategy with its default settings, as published by Ros and Hansen (2008). A run is stopped when the mutation operator parameters are below machine precision. As opposed to using a fixed time budget for each run, we this way can be sure that no further improvements would be obtained after a run terminates. The results thus would be the same as in scenarios with higher or unlimited time budgets.

The *SepCMA* evolution strategy is a variant of the *CMA* evolution strategy by Hansen (2006). The *CMA* evolution strategy is a (μ, λ) evolution strategy that deter-

no. neurons	4	8	12	16	32	64
generative	24	48	72	96	192	384
ontogenic	12	24	36	48	96	192

Table 1: The genotype size for the two proposed indirect representations, as a function of the number of neurons used in the representation.

ministically adapts all the parameters of its mutation operator, a Gaussian distribution. Although there are a wide choice of stochastic optimizers, *CMA* is known to be very competitive on recent, well-known benchmarks, as in Auger et al. (2009). Moreover, *CMA* have *proven* invariance properties (see Hansen (2000)) that makes it a robust choice. The *SepCMA* evolution strategy adapts a restricted mutation operator: an axis aligned Gaussian distribution. This makes *SepCMA* more scalable than *CMA* in high dimensional problems, such as those obtained when using the direct representation on the larger versions of the problems introduced in the previous section.

For all three representations, the initial *sigma* (initial standard deviation of the Gaussian noise) is set to 1.0. For the *direct* representation, and only for that one, a penalty $P(X) = \sum_i |X_i - 1|^2$ is added to candidates points which are outside the $[-1, 1]^n$ bound. Such a penalty scheme was necessary for the *direct* representation to converge closely to the global optimum.

The computation of the objective value requires solving relatively large linear systems. To take advantage of the high sparsity of the linear systems involved here, we employed the preconditioned conjugate gradient algorithm (as described by Shewchuk (1994)), an iterative method. The employed preconditioner is the Jacobi preconditioner. If the conjugate gradient residual (a measure of the result accuracy) is not below 10^{-15} after $20N$ iterations (where N is the dimension of the linear system), the genotype is considered invalid. In this case, the evaluation is stopped and the genotype is discarded. For an evolution strategy, this simply means that a new candidate individual is generated as replacement for the deleted one. In practice, such cases are extremely rare.

For comparing two statistical distributions, we rely on the two-tailed Mann-Whitney U test. When the p value of the test is above 0.02 – a very conservative limit – we consider that there is not significant differences between the two distributions considered.

5 Experimental Results

The statistical evaluation of the objective values obtained on each problem for the three representations is shown in Figure 5 and 7 as standard box plots. The *direct* representation reliably reaches the global optimum, showing very little dispersion across the optimization runs. The *ontogenic* representation also manages to produce solutions close to the global optimum, although not as close as the *direct* representation and also with a larger dispersion across the optimization runs. The objective value gap between the *direct* and *ontogenic* representation grows for the largest benchmark problems. The runs with the *generative* representation, while being able to reach the global optimum for the simplest problem (the *square-beam*), converge to sub-optimal minima for the other problems.

The median computational effort to reach given levels of objective values is illustrated in Figures 6 and 8. The computational effort is defined here as the number of calls to the FEM routine made during the optimization run, to reach a given level of fitness. In a typical design optimization scenario, the physical simulation of a design is by far the dominant cost of an optimization run. As explained in Section 4, the *direct* and *generative* representations consume one call to the FEM routine per objective value evaluation, whereas the *ontogenic* representation consumes 32 calls to the FEM routine

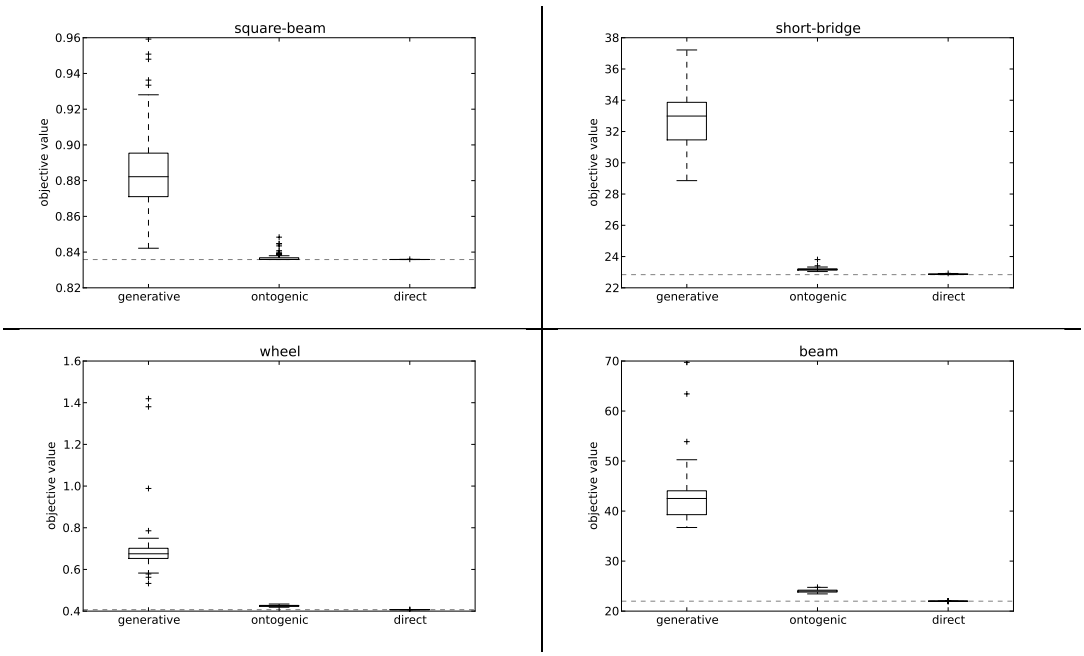


Figure 5: Lowest objective value reached statistics for the *square-beam*, *short-bridge*, *wheel*, and *beam* problems, across 64 independent optimization runs. The thin dotted horizontal lines show the lowest possible objective value for the problem considered.

per objective value evaluation. For evaluating one candidate solution, its costs are thus 32 times as high as for the other representations.

However, for all the problems but the simplest one (*square-beam*), the *ontogenic* representation still has a significantly smaller or similar computational cost to reach close-to-optimal fitness levels, compared to the *direct* representation. The reduced computational cost of the *ontogenic* representation is very visible for the largest instances of the benchmark problems. The computational costs of the *direct* and *ontogenic* representations increase at similar rates when very closely approaching the global optimum. In contrast, the *generative* representation's computational cost consistently increases faster than the other representation's computational costs. Although initially very efficient, the *generative* representation always leads to convergence far from the optimum. Also, the local optimum reached with the *generative* representation is always obtained at a higher computational cost compared to the ontogenic representation. The only exception for this trend is the *beam* problem, where the *generative* representation reaches its local optimum with the lowest cost.

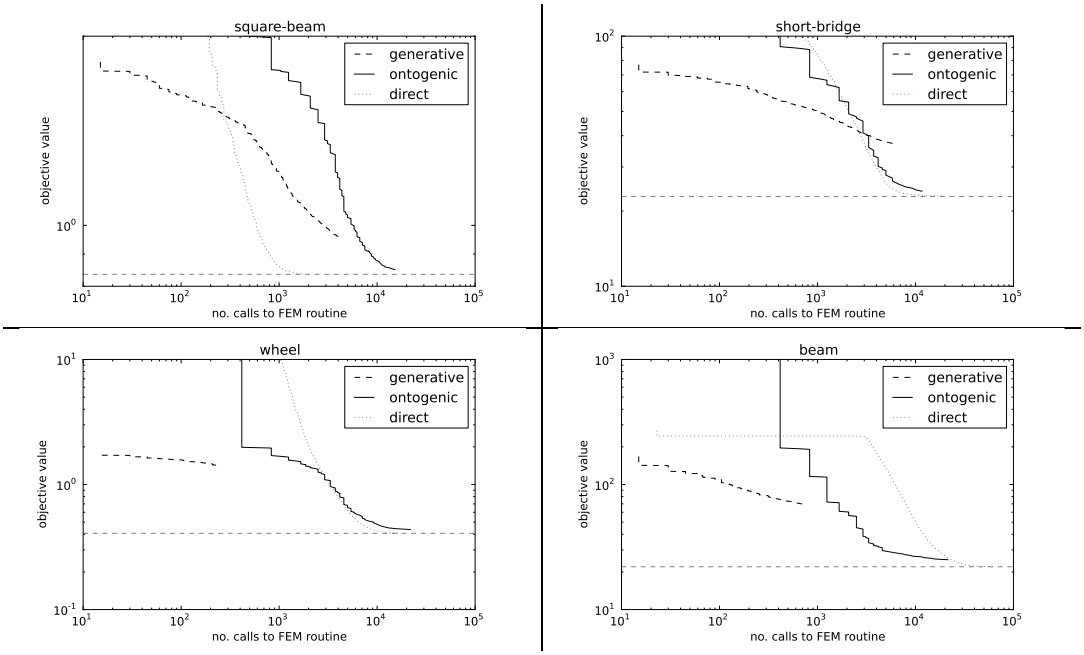


Figure 6: Median computational effort for the *direct*, *generative*, and *ontogenic* representations, on the *square-beam*, *short-bridge*, *wheel*, and *beam* problems, across 64 independent optimization runs.

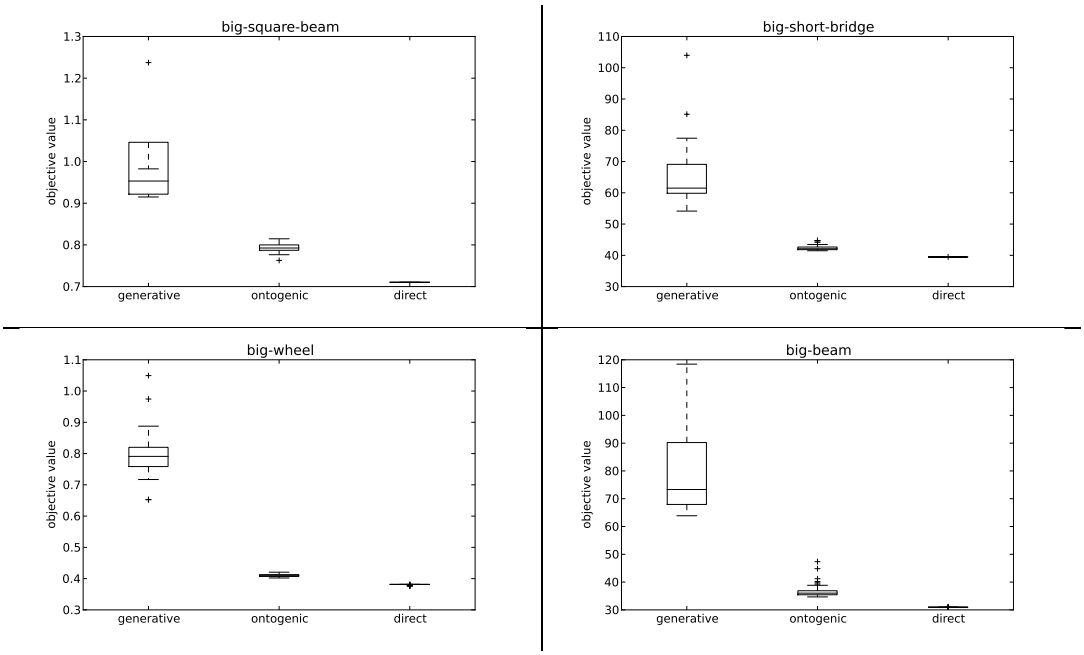


Figure 7: Lowest objective value reached statistics for the *direct* and *ontogenic* representation, on the *big-square-beam*, *big-short-bridge*, *big-wheel*, and *big-beam* problems, across 64 independent optimization runs.

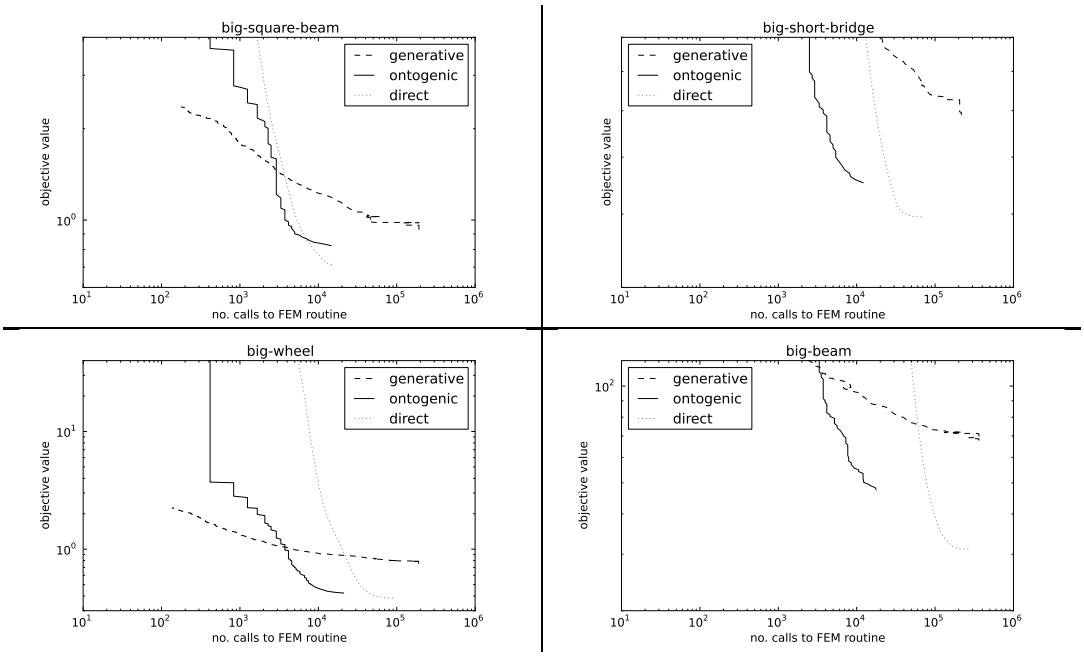


Figure 8: Median computational effort for the *direct*, *generative*, and *ontogenic* representation, on the *big-square-beam*, *big-short-bridge*, *big-wheel*, and *big-beam* problems, across 64 independent optimization runs.

5.1 Behavior of the Generative Representation

Both the *generative* and *ontogenic* representations converge farther from the global optimum than the *direct* representation. The *direct* representation can represent all possible solutions whereas the *generative* and *ontogenic* representations cannot reach all possible solutions, which is a consequence of the smaller search space. The resulting decrease in the number of discoverable solutions can be considered as a modeling flaw. Due to this modeling flaw, the best genotype possible for one indirect representation does not necessarily correspond to the best solution. Indeed, the best solution might even be unreachable for the representation. This can partly explain the differences with the direct representation, regarding the lowest objective value reached.

For the *generative* representation, the size of the genotype depends on the perceptron F. More neurons in the hidden layer allow a more accurate estimation of the optimal cross section areas of a ground structure. A given perceptron has a universal approximation capability, up to a given precision (see Hornik (1991) for a demonstration). Here, it seems that 8 neurons are not enough to match solutions close to the global optimum, as shown by the statistics of the lowest objective value reached. As explained in Section 4, the optimization is stopped when the optimizer cannot make any further improvements. While approaching the approximation limit of the perceptron, the optimizer is bound to do just vanishing small adjustments with little to no effects on the objective value and stops. In such a case, one might increase the number of neurons used in the perceptron, to allow more refined approximations. This addition of neurons means increasing the genotype size, which might increase the number of evaluations used by the optimizer to reach a given fitness level. Thus, adding more neurons for the *generative* representation would improve the best objective value reached, but likely at the price of an increased computational effort.

To validate this hypothesis, runs with 4, 8, 12, and 16 neurons for the perceptron F have been performed, with exactly the same protocol used for the initial experiments with 8 neurons (64 independent runs with SepCMA, default settings). Figure 9 shows the statistic of the lowest objective value reached. As predicted, using more neurons for the *generative* representation significantly decreases the lowest objective value. Figure 10 shows the median computational effort. The number of neurons does not significantly affect the increase rate of the computational effort when approaching the optimum. Even with 16 neurons, the objective values obtained with the generative representation are still far from those obtained with the two other representations. It seems that a large number of neurons are required for this problem, thus a large genotype is necessary. This somewhat defeats the purpose of using the generative representation, originally intended to reduce the genotype size.

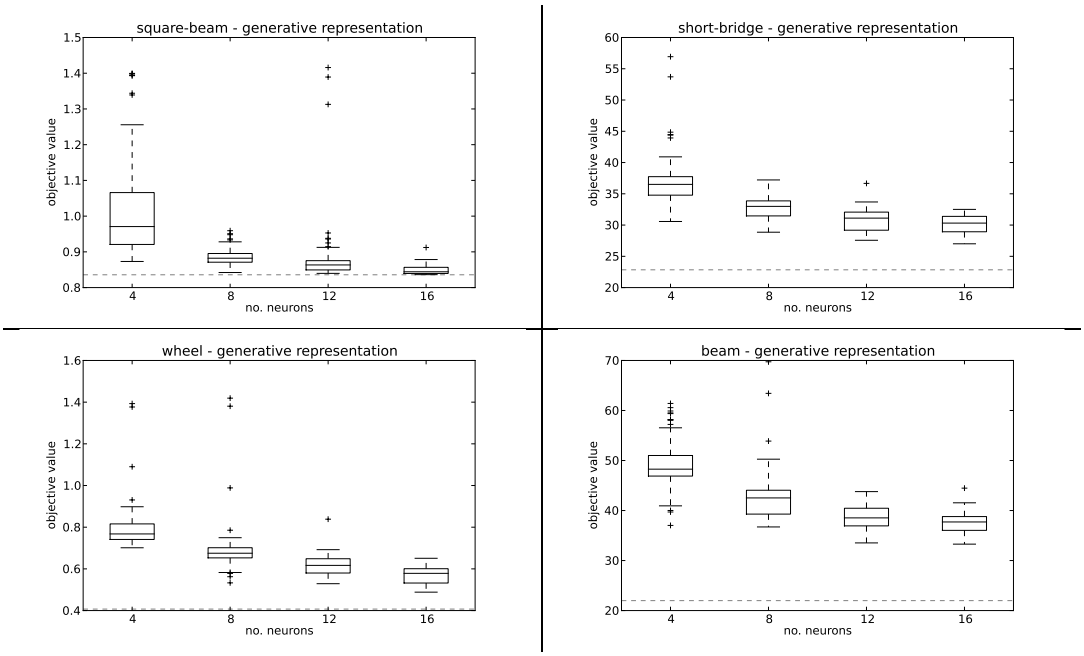


Figure 9: Lowest objective value reached statistics for the *generative* representation with 4, 8, 12, and 16 neurons, across 64 independent optimization runs. The thin dotted horizontal lines show the lowest possible objective value for the problem considered.

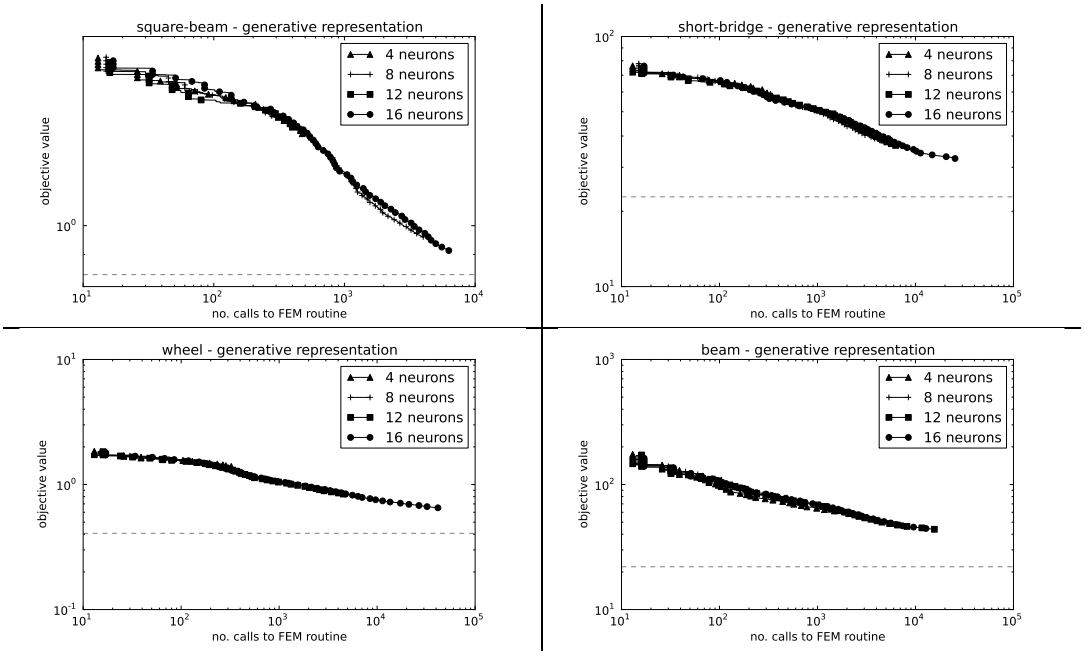


Figure 10: Median computational effort for the *generative* representation with 4, 8, 12, and 16 neurons for the perceptron F, across 64 independent optimization runs.

5.2 Behavior of the Ontogenic Representation

The *ontogenic* representation behaves much differently from the *generative* representation. First, the *ontogenic* representation reaches objective values much closer to the one reached by the *direct* representation, compared to the *generative* representation. Second, the computational cost to reach a given objective value level associated to this representation is close to the one of the *direct* representation. The computational effort increase rate is lower that for the *generative* representation, and close to the computational effort increase rate of the *direct* representation. Until the later stages of the optimization process, the *ontogenic* representation consistently has a reduced computational effort compared to the *direct* representation. It should be noted that this lower computational effort is achieved despite having fitness evaluations with a cost of 32 FEM calls, in contrast to the cost of 1 FEM call per genotype decoding for the *direct* representation.

Since the *ontogenic* representation also relies on a perceptron, it can do approximations only up to a certain precision. Two parameters control the *ontogenic* representation: the number of neurons used for the iterated function G, and the number of development steps. The number of neurons for G might be a less critical parameter to enhance the approximation power, compared to the number of development steps, for the following reason. In the case of the *ontogenic* representation, the optimal genotype codes for a function G which, after 32 iterations, will generate the optimal ground structure. But there might exist functions G that generate the optimal ground structure in 33, 42, or more iterations. Indeed, longer developments allow for greater variety of development trajectories that eventually stabilize around the optimal ground structure. Thus using more development steps would improve the lowest objective value statistic for the *ontogenic* representation. More neurons for the function G would allow to define more precise development trajectories. But what matters is only the end point of the development trajectory, the final ground structure. In the case of a development that tends to converge to a given ground structure, less precise development control would just converge more slowly to the attractor point. In the case of a development that does not converge, the development trajectory might be important and thus, more neurons would be very beneficial for the lowest objective value statistics.

To validate this hypothesis, runs with 4, 8, 12, and 16 neurons for the perceptron G at 32 development steps have been performed. Additionally, runs with 8, 16, 32, 64, and 128 development steps and 8 neurons for the perceptron G have been performed as well. In both cases, exactly the same protocol is used as in the initial experiments with 8 neurons (64 independent runs with SepCMA, default settings). The lowest objective value statistics are shown on Figures 11 and 13, whereas the median computational effort is shown on Figures 12 and 14. As predicted, longer developments introduce a significant improvement in the lowest objective value reached. Adding neurons to the perceptron G has no significant influence on the objective value.

The development sequences for the best performing genotypes have similar dynamics, seemingly independent from the problem. All the best development processes initially reduce the cross-area of the beams. Then, when the number of development steps is close to the number of steps used during the optimization, the development shows a pseudo-oscillatory behavior. Further development steps modify the truss, cycling through different but similar states. The frequency, the amplitude and the regularity of such cycles are highly variable, without significant correlation with the number of steps used during the optimization.

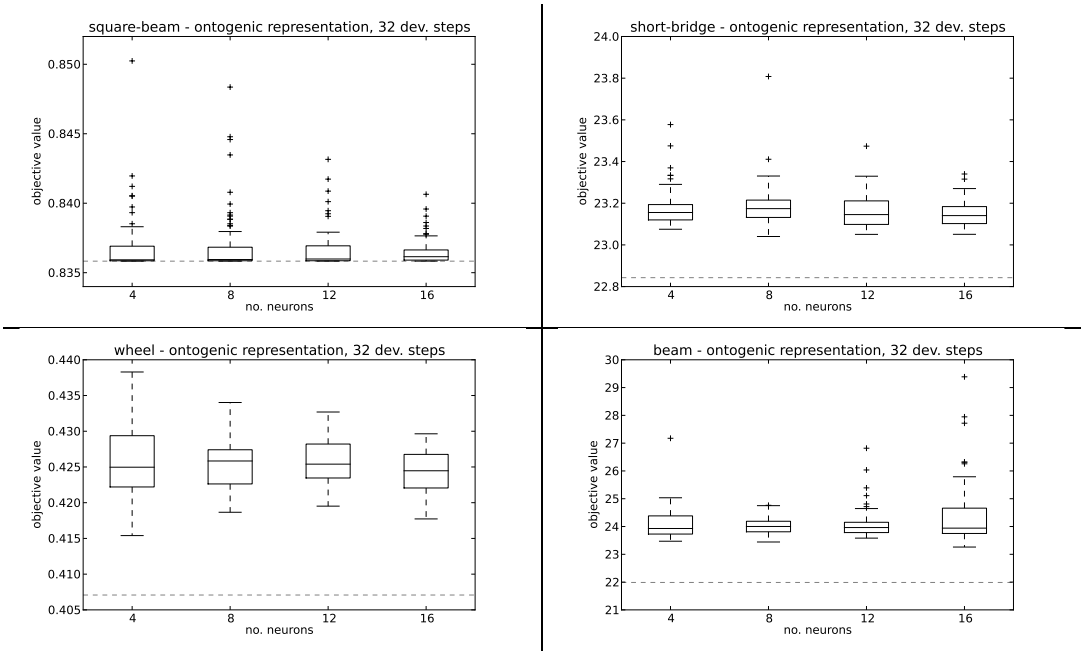


Figure 11: Lowest objective value reached statistics for the *ontogenic* representation with 32 development steps, 4, 8, 12 and 16 neurons, across 64 independent optimization runs. The thin dotted horizontal lines show the lowest possible objective value for the problem considered.

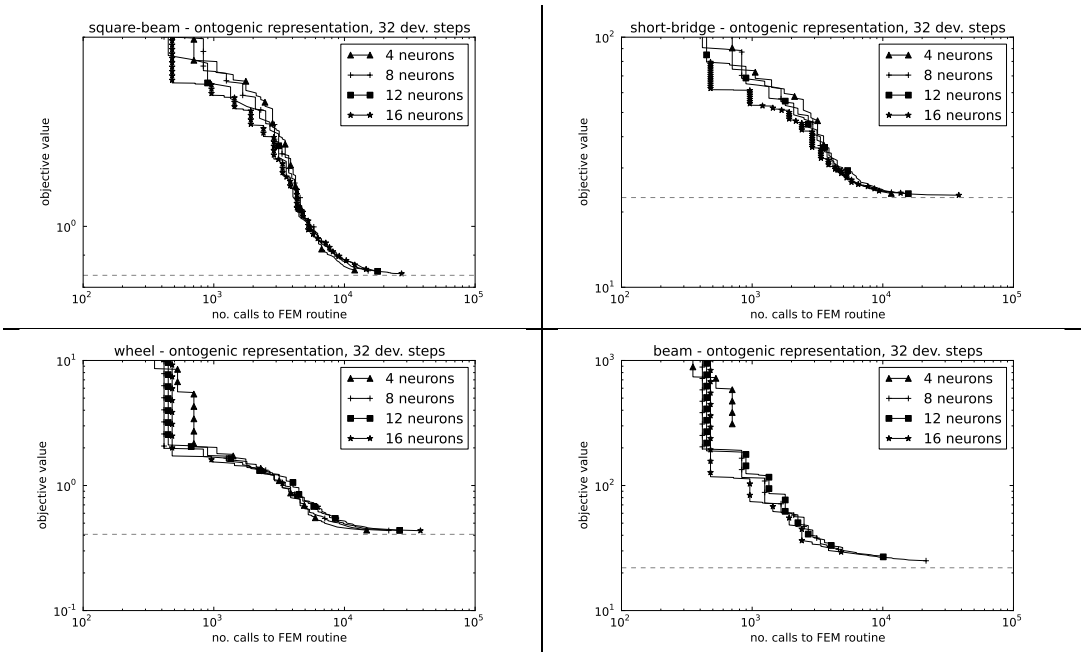


Figure 12: Median computational effort for the *ontogenic* representation with 32 development steps, 4, 8, 12 and 16 neurons, across 64 independent optimization runs.

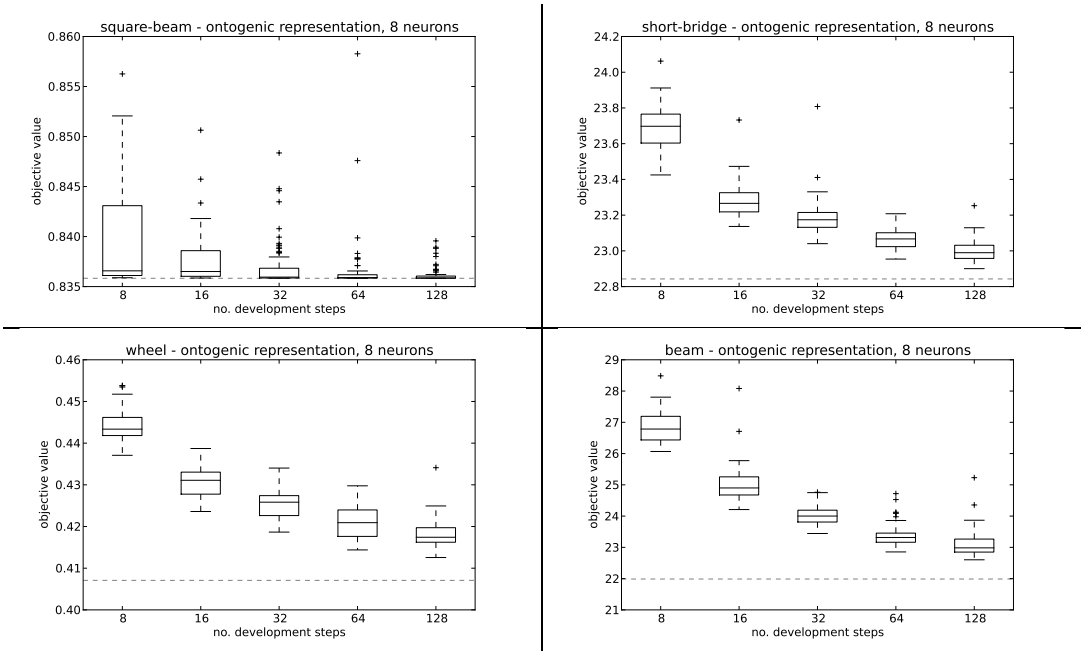


Figure 13: Lowest objective value reached statistics for the *ontogenic* representation with 8, 16, 32, 64 and 128 development steps, 8 neurons, across 64 independent optimization runs. The thin dotted horizontal lines show the lowest possible objective value for the problem considered.

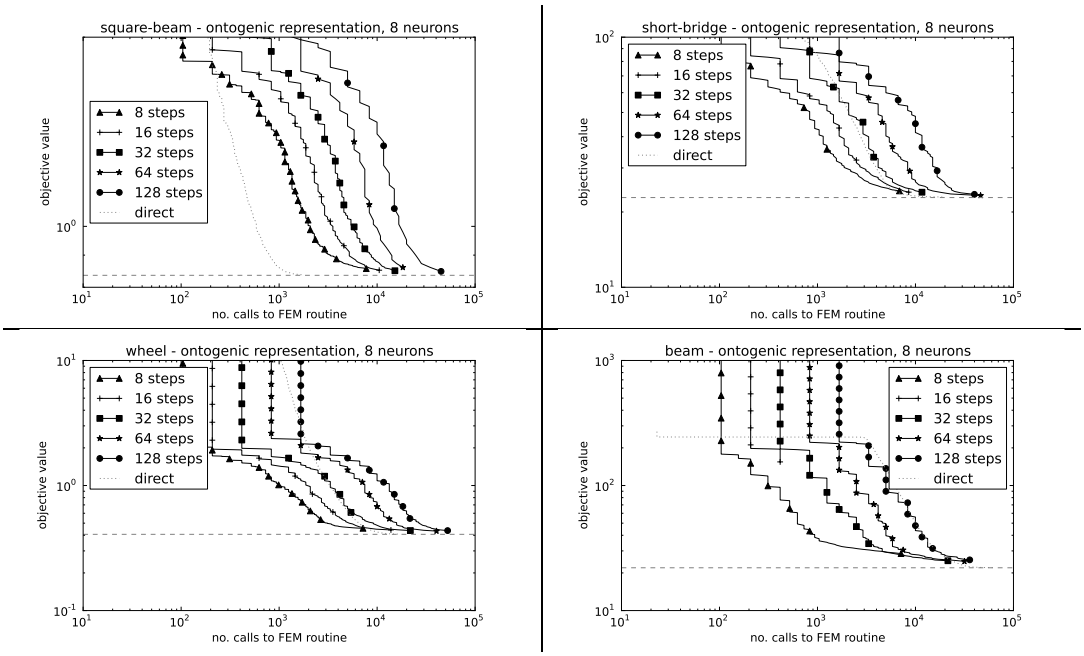


Figure 14: Median computational effort for the *ontogenic* representation with 8, 16, 32, 64 and 128 development steps, 8 neurons, across 64 independent optimization runs.

5.3 Fitness Landscape

From our results, it is now clear that the studied problems, the genotype size alone does not determine the behavior of a representation. By observing how the fitness statistic varies when increasing the distance from an optimum, we can further get a rough idea of the fitness landscape. The dispersion of the fitness gives a measure of the smoothness of the fitness landscape. The trend of the fitness over the distance to the optimum gives an idea of the topology of the fitness landscape. To compute the statistic of the fitness at a distance d from the optimum X, n normalized vectors Z are drawn from a uniform distribution. The fitness of the n points $X_i + dZ_i$ are then evaluated. We used $n = 4096$, with d varying from 10^{-3} to 10^{0}.

The Figure 15 shows the fitness statistic for the *generative* and *ontogenic* representations, at increasing distance from the best solutions ever found for the four problems we defined. For all problems, the fitness statistics of the two representations are significantly different. The *ontogenic* representation features an almost flat valley of low fitness dispersion, which ends with a sharp increase of the fitness and its dispersion far from the optimum. In the other end, the *generative* representation has a smaller low fitness dispersion area. The differences of the fitness statistic, when changing the number of neurons for a representation, or the number of development steps for the *ontogenic representation*, is not statistically significant.

The *CMA* evolution strategy and, by extension, most evolution strategies, can be seen as biased random walks in the fitness landscape. Evolution Strategies are biased to be move toward better solutions, so the random walk eventually ends up trapped by a local optimum. The optimum for the *ontogenic* representation seems a more effective trap than the *generative* representation for such a biased random walk. This observation is consistent with the statistic of the best fitness of a run. The best fitness statistic of the *ontogenic* representation has a reduced dispersion compared to the one of the *generative* representation. Since the optimum for the *ontogenic* representation behaves as a better trap, optimization runs are more likely to terminate in that optimum.

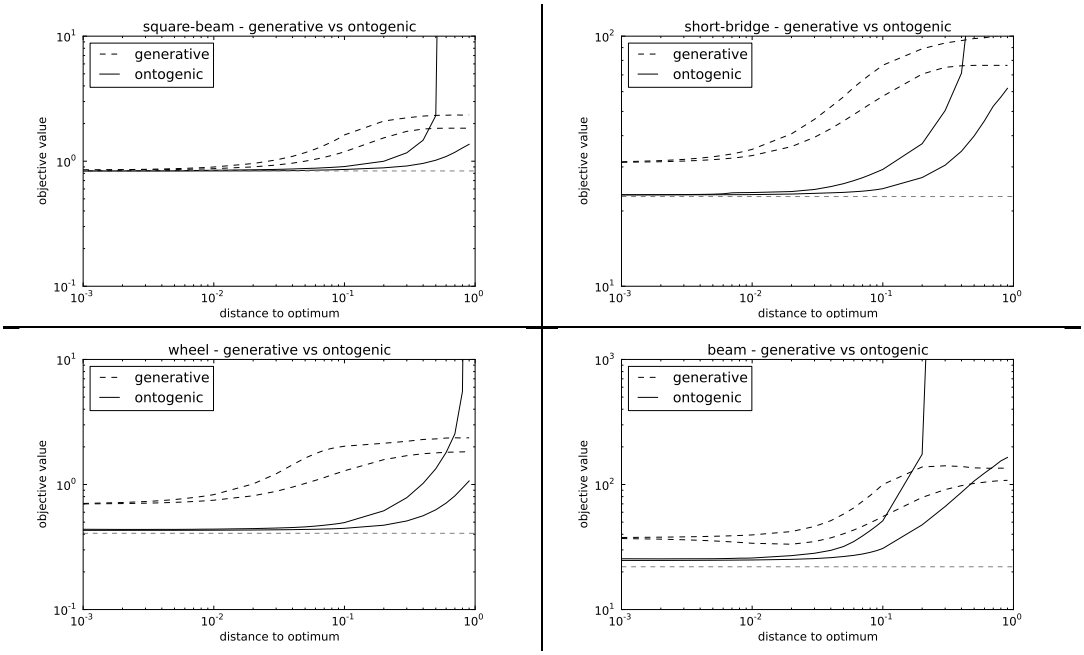


Figure 15: Higher and lower quartile of the fitness of candidate solutions at a given distance from the best solution ever found, for two representations. The *generative* representation uses 8 neurons. The *ontogenic* representation uses 32 development steps and 8 neurons. 4096 samples per points are used for computing this statistic. The thin dotted horizontal lines show the lowest possible objective value for the problem considered.

5.4 Staged Development

The genotype size for the ontogenic representation is independent from the number of bars in the ground structure. Although a modeling flaw is introduced, this flaw is much less pronounced than for the generative representation. By using more development iterations, the gap to the direct representation in terms of lowest objective value can be reduced. Although this allows keeping a reduced genotype size, doing so increases the computational effort. A way to reduce the computational effort when using an ontogenic representation, would be to increase the number of development iterations by stages. Staged development for ontogenic approaches have been introduced by Federici and Downing (2006). Each run starts with a first stage S_1: an optimization round where just a few development iterations per evaluation is carried out. Once the optimizer converged, a new stage S_2 is started. The initial ground structure used for the development process in S_2 is the best solution found by S_1. Instead of using a single run with a long development time, dividing a single run into stages with short development time can be a way to reduce the computational effort. Federici et al. reported a nine-fold computational effort reduction for their experiments.

Figures 16 and 17 display respectively the lowest objective value reached statistics and the median computational effort for the ontogenic representation with 5 stages, 8 neurons for the function G. The initial stage S_1 uses 8 development steps, and the number of development steps is doubled at each stages. The last stage, S_5, uses 128 development steps. The experimental setup remains the same as used in the previously introduced experiments. The lowest objective value is slightly yet significantly improved over a single staged run with the ontogenic representation, 128 development steps and 8 neurons. The computational effort is significantly reduced (up to a factor ranging from 2 to more than 10) until close to the global optimum, where it became equal to or largely higher than the computational effort of the direct representation. Nevertheless, multi-staged runs significantly improve the computational effort of the ontogenic representation compared to a single staged run. Indeed, it makes the ontogenic representation much more efficient than the direct representation, until very close to the global optimum.

The idea of staged runs can be transposed to a generative representation. A first optimization round, the stage S_1 would generate a ground structure. A second optimization run, the stage S_2 would generate a ground structure from the ground structure obtained at S_1. S_2 can even use the same local information used by the ontogenic representation. The stages $S_{i>1}$ would transform a ground structure rather than generate one from scratch. Such a staged run approach for generative representation is indeed equivalent to a staged run approach for ontogenic representations, with a single iteration for the development. Stages incorporate optimization results which, in turn, represent information gathered from the objective value evaluations. By doing so, we would introduce feedback into a generative approach. This extends beyond the pure generative idea and is out of the scope of this paper.

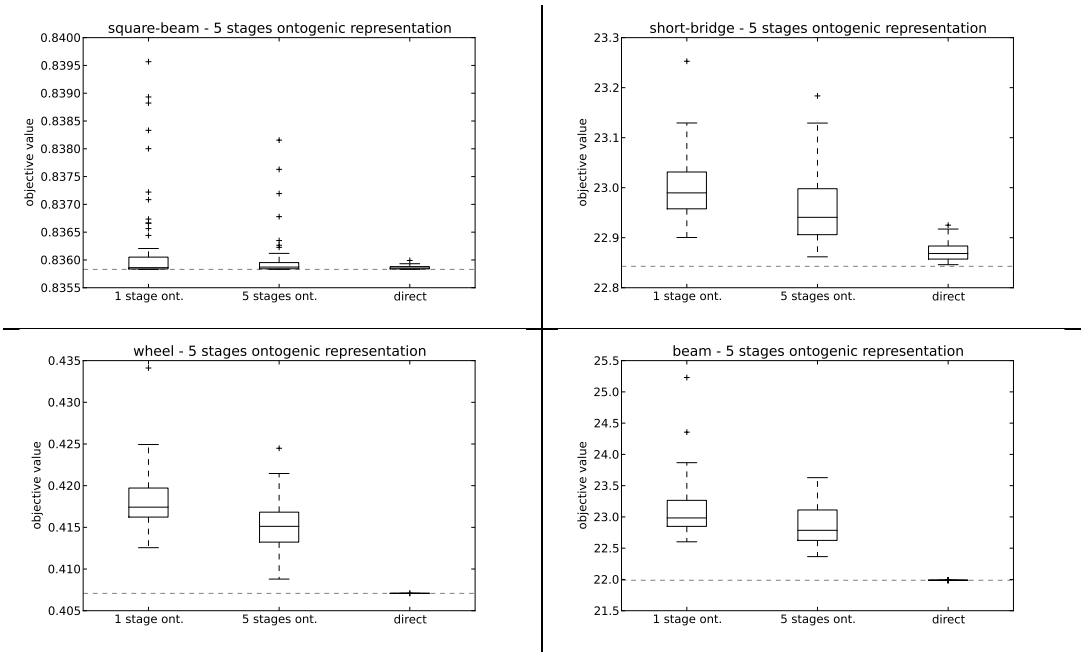


Figure 16: Lowest objective value reached statistics for the *ontogenic* representation with 5 stages 8, 16, 32, 64 and 128 development steps, 8 neurons, across 64 independent optimization runs. The thin dotted horizontal lines show the lowest possible objective value for the problem considered. The lowest objective value reached statistics for the single stage *ontogenic* representation is for 128 development steps and 8 neurons.

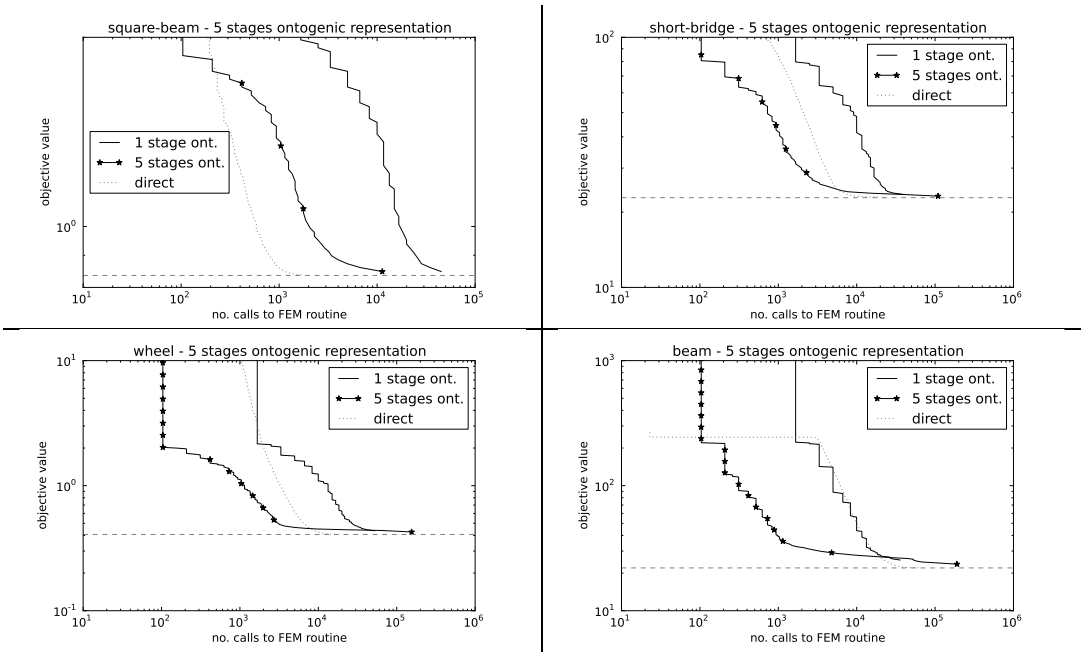


Figure 17: Median computational effort for the *ontogenic* representation with 5 stages of 8, 16, 32, 64, 128 developments steps, 8 neurons, across 64 independent optimization runs. The median evaluation statistics for the single stage *ontogenic* representation is for 128 development steps and 8 neurons.

6 NEAT

In all the experiments introduced in the previous sections, we relied on the same experimental setup. The function approximator is a neural network with an ad-hoc topology, a perceptron with a hidden neuron layer. The optimization algorithm is the *CMA* Evolution Strategy. The reason for choosing this straightforward setup is that it can easily be reproduced and, more importantly, does not introduce any effects or phenomena unrelated to the representation behavior that we wish to investigate. Using, e.g., a more sophisticated generative technique could lead to better results, but the question would arise whether these results are obtained because of the features of the optimizer, the peculiarities of the specific algorithm, or the internal structure of the representation instead of general generative idea.

Yet, choosing the trivial generative method may also be considered as making the results rather specific, even if the explanations for those results do not rely heavily on specificities of the setup. Doing the same experiments with a significantly different experimental setup would allow to see to which extends our observations can be generalized.

A natural choice for an alternative setup are *CPPN*s and *NEAT*. *CPPN*s are general feed-forward neural networks, where the neurons' transfer function is taken from a ad-hoc set of functions. *NEAT* (Stanley and Miikkulainen, 2002) is a state-of-the-art evolutionary optimizer for neural networks. *NEAT* is used to optimize both the parameters and the topology of *CPPN*. In the *NEAT* and *CPPN* method applied here, the *generative* encoding devised in Section 4 is very close to the encoding introduced as the *HyperNEAT* approach, adapted to the truss benchmark problem. In *HyperNEAT*, a *CPPN* generates the connection weight of a neuron network, where the neurons of the generated network have 2D Cartesian coordinate systems. Here, we use a *CPPN* to generate the thickness of the beams as a function of the end-point coordinates, also 2D. The *ontogenic* encoding, for this alternative setup, is in the other end unique, up to our knowledge. The *CPPN* and *NEAT* setup crucially differ from the *Perceptron* and *CMA-ES* setup.

- *NEAT* starts with an elementary neural network, which topology is then augmented by random mutations. Thus, the topology of the neural networks optimized with *NEAT* is plastic and unbounded. In contrast, the experiments in the previous sections always relied on a neural network of fixed complexity. An unbounded complexity for the neural network topology allows arbitrary precise matching of any function.
- In D'Ambrosio and Stanley (2007), the transfer function set for *CPPN*s is claimed to be well-suited for expressing *repetition with variations*. The transfer function used in the previous sections is the classical *tanh* function.
- *CMA-ES* is a self-adaptive evolution strategy. The *CMA-ES* mutation operator is an anisotropic additive Gaussian mutation, which parameters are adapted by an effective deterministic strategy. *NEAT* uses an additive uniform mutation without strategies to adapt that operator.
- *NEAT* uses a complex breeding operator, featuring a *niching* strategy. The niching strategy clusters the candidate solutions with a genotype similarity measure.

Crossover is performed only with candidate solutions from a same cluster. Moreover, premature convergence is countered by preserving recently introduced candidate solutions.

6.1 Experimental Protocol

The *NEAT* implementation used for all the experiments is the one provided by the *HyperNEAT 3.0* C++ package. The employed parameter setting specific to this implementation is fully described in Table 2. The population size is set to 100. As the number of parameters for *NEAT* is rather high, an extensive parameter tuning work was beyond our computational resources. Default values have been used, with some slight modifications. The simplest problem (*square-beam*), e.g., is solved without relying on a very small strength for the parameter mutation. Moreover, the parameters controlling the population clustering have been set such as the clusters size stays within a reasonable range during a run. The transfer function set for the neurons is the same as in D'Ambrosio and Stanley (2007). Recurrent connections in the *CPPN*s are not allowed, as we use the *CPPN*s as pure functions.

NEAT does not provide stopping criteria that terminate a run when that run is very unlikely to do any further progress. All runs are stopped after 2500 generations, which in practice is enough to reach a local optimum from which none of the runs could escape. Like the previous experiments, the *square-beam*, *beam*, *wheel*, and *square-bridge* problems are considered. As *NEAT* can arbitrarily expand the complexity of a neural network, the *generative* encoding does not require exploring various network topologies. For the *ontogenic* encoding, 8, 16, 32, and 64 development steps have been considered. For the four problems and encodings, 64 runs are performed.

NEAT is a maximization algorithm. Also, *NEAT* accepts only positive fitness values, which is a requirement of its breeding operator. Since the four benchmark problems are minimization problems, we have to apply a transformation to the compliance value of a truss. We used the following transformation.

$$F = \max(3 - \log_{10}(C), 0) \tag{5}$$

To ensure that this transformation does not modify significantly the difficulty of the four problems, we employed the same transformation on *CMA-ES* based runs. The best fitness of the runs with this fitness transformation is not statistically different from the runs using original fitness value directly (based on 64 runs for each four problems, with the *generative* and *ontogenic* encoding). The logarithmic scale employed by the proposed transformation proved to be important. Without it, *NEAT* is not able to converge near the global optimum even for the simplest problem (*square-beam*). As *NEAT* relies on a proportional selection operator, not a rank-based one, *NEAT* can be very sensible to the fitness scaling. For our four benchmark problems, the difficulty is roughly exponential, i.e., it is as hard to improve from 10^0 to 10^-1 as improving from 10^-1 to 10^-2. The logarithmic fitness transformation proved to be very effective, based on exploratory experiments.

6.2 Experimental Results

Figure 18 shows the best fitness statistic for all the experiments. The *ontogenic* encodings provide significantly better results, for all problems. The dispersion of the fitness

parameter name	**value**
AddBiasToHiddenNodes	0.0
AdultLinkAge	18.0
CompatibilityModifier	0.30
CompatibilityThreshold	100.0
DisjointCoefficient	2.0
DropoffAge	50.0
ExcessCoefficient	2.0
ForceCopyGenerationChampion	1.0
LinkGeneMinimumWeightForPhentoype	0.0
MutateAddLinkProbability	0.30
MutateAddNodeProbability	0.030
MutateDemolishLinkProbability	0.0
MutateLinkProbability	0.10
MutateLinkWeightsProbability	0.80
MutateOnlyProbability	0.250
MutateSpeciesChampionProbability	0.0
MutationPower	0.5
SmallestSpeciesSizeWithElitism	5.0
SpeciesSizeTarget	8.0
SurvivalThreshold	0.20
WeightDifferenceCoefficient	0.1
AgeSignificance	1.0

Table 2: The parameters setting used for all the NEAT-based experiments

is much smaller for the *ontogenic* encodings compared to the *generative* encoding. This is very similar with the *CMA-ES* based runs with ad-hoc neuron network topologies. An exception is the *beam* problem, where the dispersion of the best fitness is more important than for the *generative* representation. The *beam* problem is the hardest of the four problems, and here, *NEAT* did not fully converge. As for the *CMA-ES* based experiments, increasing the number of development steps for the *ontogenic* encoding triggers a better fitness statistic.

Figure 19 shows the effort statistic for all the experiments. For moderate levels of fitness, the *generative* encoding proves to be significantly less expensive. But as the runs are moving towards the global optimum, *ontogenic* encodings with 8 or 16 development steps provide better fitness at a lesser expense. Overall, the effort statistic with *NEAT* behaves similar to the one of the generative method with *CMA-ES*. The only difference is the scale of the computational effort: The best fitness is reached with a cost of roughly 10^6 to 10^7 invocations of the FEM with *NEAT*. With the *CMA-ES* based approach, that cost is within 10^4 to 10^5 for similar levels of fitness.

Overall, the behavior of the *generative* and *ontogenic* representations seems to be independent from the optimization method. For our four benchmark problems, it is clearly visible that the *generative* encoding cannot achieve results equivalent to those of the *ontogenic* encoding. The same phenomena are observed with two conceptually very different optimizers. Thus, we believe that the non-equivalence of *generative* and *ontogenic* representations, at least on the problem studied here, is mostly independent from the underlying optimization approach employed. Moreover, the better behavior of the *ontogenic* representation in regard of best fitness reached and scalability seems also fairly independent from the optimization approach.

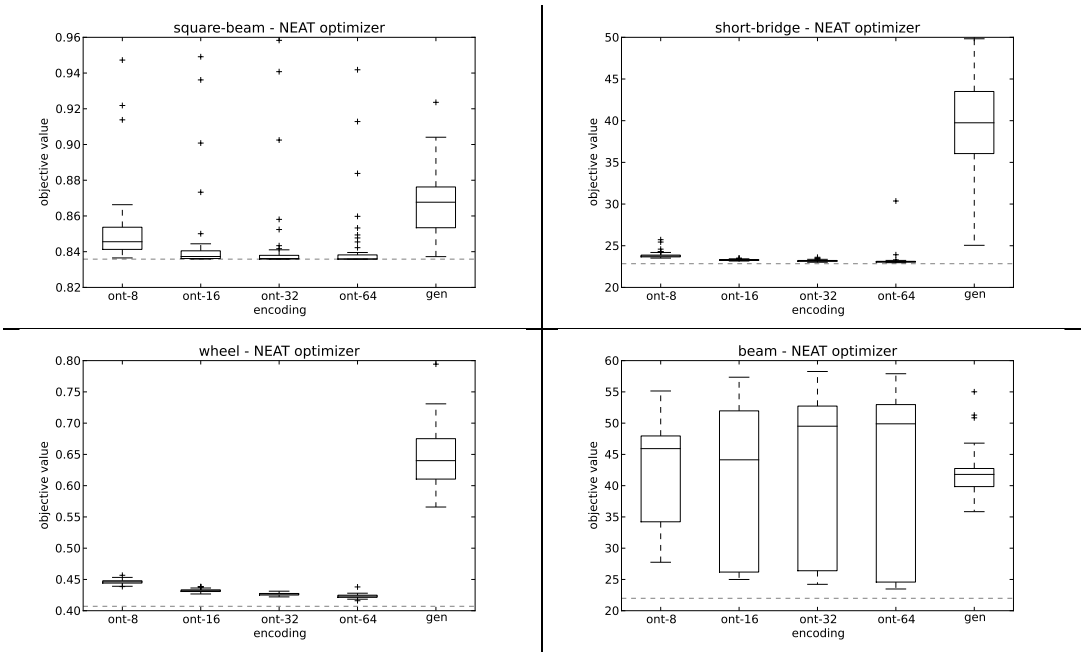


Figure 18: Lowest objective value reached statistics for the *generative* representation and the *ontogenic* representation with 8, 16, 32, and 64 development steps, across 64 independent *NEAT* optimization runs. The thin dotted horizontal lines show the lowest possible objective value for the problem considered.

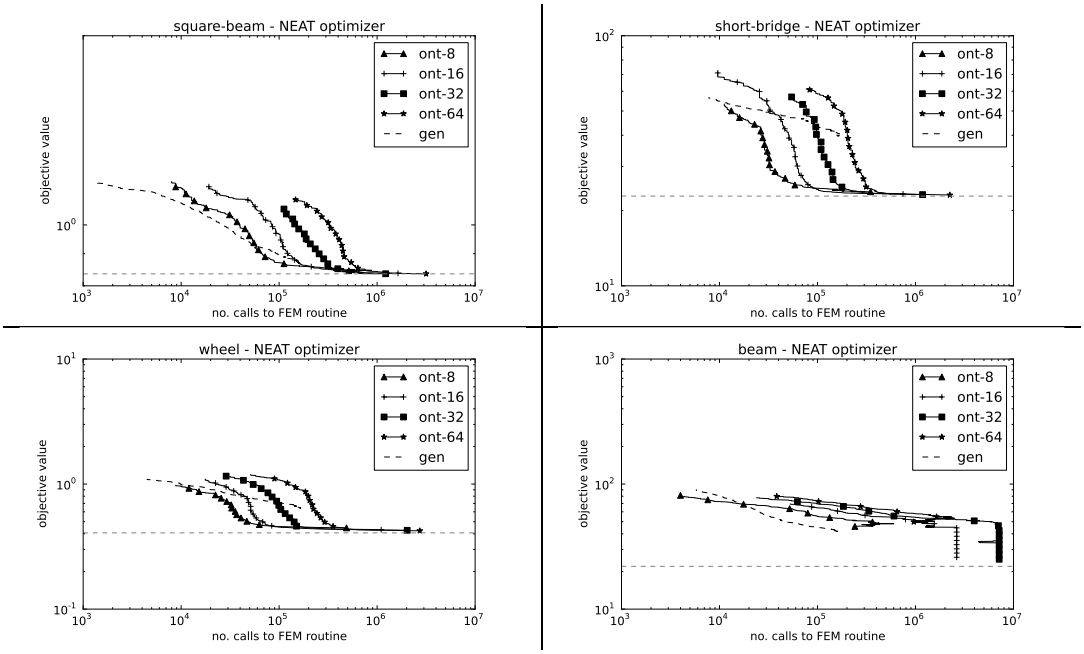


Figure 19: Median computational effort for the *generative* representation and the *ontogenic* representation with 8, 16, 32, and 64 development steps, across 64 independent *NEAT* optimization runs.

7 Conclusion

On a benchmark problem from structural mechanics, generative and ontogenic solution representations are compared and analyzed. The same evolution strategy (*SepCMA-ES*) is used for both representations. Both the generative and ontogenic method are indirect representations that attempt to bring a better scalability to evolutionary algorithms.

The generative representation encodes solutions of a problem as a single-pass spatial transformation. Here, the transformation is a function approximator: a multi-layer perceptron. The ontogenic representation encodes solutions as the result of an iterated transformation on an initial solution, using feedback from a simulation or objective value computation. Here, the iterated transformation is a function approximator (also a multi-layer perceptron) fed with local mechanical strain information. Both generative and ontogenic representations can only *approximate* the global optimum. Thus, comparisons with a direct encoding makes sense only *close* to the global optimum, not *at* the global optimum.

In terms of the lowest objective value reached, the ontogenic representation, by a large margin, provides better results, close to the global optimum. The ontogenic representation can further obtain better results by increasing the development time, which does not affect the genotype size. The ontogenic representation is a truly scalable representation for ground structures. In terms of computational cost, the generative representation is not better than the ontogenic representation. Far from the global optimum, the generative representation is much cheaper. But if it should approach the global

optimum, the generative representation becomes more expensive than the ontogenic one. Especially with multi-staged runs, the ontogenic representation remains much less costly than a direct representation, until being very close to the global optimum.

We showed that increasing the number of neurons in the generative representation does not improve the objective value which can be discovered. Moreover, adding neurons increases the genotype size, which goes against the very purpose of a scalable representation. Other changes could have been carried out to improve the approximation power of the perceptron. We can change the transfer function of the perceptron. A careful choice of transfer function would allow a perceptron with a few neurons to perfectly match the global optimum. But the ability to craft such a transfer function would imply a great knowledge about the solution of the problem. Indeed, in that case, using an evolutionary algorithm with a sophisticated representation would be irrelevant.

Another way to improve the approximation power of a neural network is to use more complex neural network topologies. This is what is done with *HyperNEAT* (D'Ambrosio and Stanley, 2007): the spatial transformation is a general feed-forward neural network. These neural networks are initially trivial, but an evolutionary algorithm will incrementally augment them. This is a big difference compared with first experiments described here, where the function approximators are never augmented during the optimization. In order to verify whether this can fundamentally change the trends for the generative and ontogenic representations, we conducted a second set of experiments. As the results of these experiments show, even with a much more complex approach such as *HyperNEAT*, the relation between the performances of two representations remains the same, as the ontogenic method still leads to better results and provides better scalability. Thus, the difference of behaviors between the generative and ontogenic representations is likely to be independent from the optimization scheme employed.

In summary, the study introduced here directly contradicts the claim that a generative representation is a much cheaper alternative to an ontogenic representation. On the problems introduced here, the generative representation suffers much more from approximation accuracy problems. The ontogenic representation, although relying on a function approximator of similar power, suffered much less from such issues. Thus, at least in the problems studied here, a generative representation is not qualitatively equivalent to an ontogenic representation. Moreover, the ontogenic representation is shown to be an overall less expensive way to find solutions close to the global optimum, despite having a higher cost per evaluation. A development process can be an effective way to build a scalable and computationally efficient representation for optimization problems.

Finally, it should be mentioned that in our experiments, the direct representation manages to reach the global optimum with a lower computational effort than the other representations. Thus one might question the relevance of the ontogenic representation. For the introduced benchmark, the direct representation is guaranteed to converge to the *known* global optimum. The two indirect representations can only *approximate* the global optimum. Thus, relevant comparisons can be done only at close-to-optimum levels. Then, the ontogenic representation demonstrates a very scalable behavior whereas the generative representations fall short of this. On real-world problems, the global optimum might be unknown. What an ontogenic representation can offer is to come reasonably close to the global optimum with a much improved scalabil-

ity, i.e., lower computational costs. In that sense, our results are a clear demonstration of that ability.

Acknowledgments

This work is partially supported by the Chinese Academy of Sciences Fellowships for Young International Scientist (Grant No.2009Y2BG15), China Postdoctoral Science Foundation Grant (No. 20100470843), and the Natural Science Foundation of China grant (No. U0835002).

References

Achtziger, W. (2007). Topology optimization of discrete structures: An introduction in view of computational and nonsmooth aspects. In Rozvany, G., editor, *Topology Optimization in Structural Mechanics*, pages 57–100. Springer.

Achtziger, W. and Stolpe, M. (2007). Truss topology optimization with discrete design variables-guaranteed global optimality and benchmark examples. *Structural and Multidisciplinary Optimization*, 34:1–20.

Auger, A., Hansen, N., Perez Zerpa, J., Ros, R., and Schoenauer, M. (2009). Empirical comparisons of several derivative free optimization algorithms. In *Acte du 9ieme colloque national en calcul des structures*, volume 1.

Bentley, P. and Kumar, S. (1999). Three ways to grow designs A comparison of evolved embryogenies for a design problem. In *Genetic and Evolutionary Computation Conference*, pages 35–43. Morgan Kaufmann.

D'Ambrosio, D. B. and Stanley, K. O. (2007). A novel generative encoding for exploiting neural network sensor and output geometry. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 974–981, New York, NY, USA. ACM.

Devert, A. (2009). When and why development is needed: generative and developmental systems. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1843–1844, New York, NY, USA. ACM.

Devert, A., Bredeche, N., and Schoenauer, M. (2011). Robustness and the halting problem for multi-cellular artificial ontogeny. *IEEE Transactions on Evolutionary Computation*, 15(3):387–404.

Dorn, W. S., Gomory, R. E., and Greenberg, H. J. (1964). Automatic design of optimal structures.

Estévez, N. S. and Lipson, H. (2007). Dynamical blueprints: exploiting levels of system-environment interaction. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 238–244, New York, NY, USA. ACM.

Federici, D. and Downing, K. (2006). Evolution and development of a multicellular organism: Scalability, resilience, and neutral complexification. *Artif. Life*, 12(3):381–409.

Felippa, C. (2010). Introduction to finite element methods. `http://www.colorado.edu/engineering/cas/courses.d/IFEM.d`.

Fournier, H. and Teytaud, O. (2010). Lower bounds for comparison based evolution strategies using vc-dimension and sign patterns. *Algorithmica*.

Hansen, N. (2000). Invariance, self-adaption and correlated mutations in evolution strategies. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., and Merelo, J., editors, *Proceedings of PPSN VI, Parallel Problem Solving from Nature*. Springer.

Hansen, N. (2006). The CMA evolution strategy: a comparing review. In Lozano, J., Larranaga, P., Inza, I., and Bengoetxea, E., editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

Ros, R. and Hansen, N. (2008). A simple modification in CMA-ES achieving linear time and space complexity. In Rudolph, G., Jansen, T., Lucas, S. M., Poloni, C., and Beume, N., editors, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 296–305. Springer.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. `http://www.cs.cmu.edu/~jrs/jrspapers.html`.

Stanley, K. O. (2006). Exploiting regularity without development. In *AAAI Fall Symposium on Developmental Systems, Proceedings*.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162.

Stanley, K. O. and Miikkulainen (2003). A taxonomy for artificial embryogeny. volume 9, pages 93–130.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.