HAL
open science

Evolutionary Design of Buildable Objects with BlindBuilder : an Empirical Study

Alexandre Devert, Nicolas Bredeche, Marc Schoenauer

To cite this version:

Alexandre Devert, Nicolas Bredeche, Marc Schoenauer. Evolutionary Design of Buildable Objects with BlindBuilder : an Empirical Study. Asia-Pacific Workshop on Genetic Programming, The Long Pham and Hai Khoi Le and Xuan Hoai Nguyen, Oct 2006, Hanoi, Vietnam, pp.98–109. inria-00118652

HAL Id: inria-00118652

https://inria.hal.science/inria-00118652v1

Submitted on 6 Dec 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Evolutionary Design of Buildable Objects with BlindBuilder : an Empirical Study

Alexandre Devert, Nicolas Bredeche, and Marc Schoenauer

TAO team - INRIA Futurs - LRI, Bat 490 - Université Paris-Sud - France

Abstract. In a previous paper, we presented *BlindBuilder*, a new representation formalism for Evolutionary Design based on construction plans. As for other indirect encoding approaches in the literature, *BlindBuilder* makes it possible to easily represent possible solutions but makes it difficult to perform structural optimization. While satisfying results are provided, it becomes more and more difficult to build larger structures during the course of evolution. This is due to the high disruptive rate of variation operators as construction plans grow. In this paper, we provide an analysis of such a problem and propose new construction operators to avoid this. Then, we perform extensive experiments so as to identify the key parameters and discuss the advantages, limitations and possible perspectives of the indirect enconding approach.

1 Introduction

Evolutionary Design addresses the issue of automatic design of buildable structures and objects with the use of Evolutionary Computation. This field has been growing along with the availability of more and more powerful computers. Major achievements in this field range from evolved creatures[14, 12, 2] to the design of satellite antennas that were actually launched in space [5] and various designs of everyday objects (e.g. chairs) and/or structures[1, 9, 7, 13] (e.g. brigdes, walls, etc.).

In a recent paper [3], we proposed *BlindBuilder*, a new representation language for Evolutionary Design. a *BlindBuilder* construction plan is defined as a directed acyclic graph (DAG) that specifies how to build the target object rather than describing what it looks like (indirect versus direct encoding). We already showed that this representation language is particularly powerful and enables hierarchy, modularity and generality, which are key features in Evolutionary Design. *BlindBuilder* can easily be used to represent 3-dimensional structures and is not limited to one set of domain-specific operators. As a consequence, *BlindBuilder* makes it possible to address a wide range of Evolutionary Design problems, from building 3D objects out of small elements such as Lego or Kapla to 3D photolithography and so on.

In the literature, the indirect encoding approach is largely favored[8, 11] and have lead to building objects as bridges, tables, etc [2, 9, 7]. In [3], we have achieved comparable results with *BlindBuilder* by building (among other) structures as bridges with cantilevers and archs. Despite these promising results, all

the works in the literature (including ours) are limited to the use of a small amount of elements (less than one hundred atomic elements), require days of computation and are either not buildable or buildable thanks to a very constrained representation (e.g. 2D structures such as in [9]).

As a consequence, two questions must be asked: (1) why does it work? (2) what are the limitations? We conducted further experiments with our work in order to get a deeper understanding of the evolutionary process. Our primary concern regarded the success of variations during evolution (i.e. mutations in the graph that lead to a better individual). Indeed, most of the variations are disruptive and produce children that are worse than their parents. During the course of evolution, such disruptive variations tends to occur more and more frequently, making it very difficult to improve the score of the best individual.

As a conclusion, relevant solutions are found because the search space is well-constrained by the very representation formalism rather than by the optimization process.

In the scope of this paper, we intend to provide an in-depth analysis in two steps. Firstly, we shall define more constrained operators and experimental setup so as to be able to better understand the optimization dynamics. Secondly, we intend to perform an extensive analysis of the impact of all parameters (population size, selection operators, variations operators, etc.) on evolution.

In the next section, we describe *BlindBuilder*, the representation formalism for construction plans, as well as the evolutionary and construction operators used in the scope of this paper. Then, we show extensive experiments on the bridge and wall problems and provide an analysis of the results, including a comparaison with standard approach (using Koza's ADF[6]). We conclude with a discussion of such results, the impact of the various parameters studied and the consequence in terms of benefits and limitations of the indirect encoding approach as well as future directions regarding representation issues.

2 The BlindBuilder representation

Basically, a *BlindBuilder* individual is a directed Acyclic Graph (DAG) where nodes can be either *atomic elements* (the physical atomic elements of the construction) or *construction operators* (that defines the way to fit the atomic elements together). A much more precise description of the *BlindBuilder* could be found in our previous paper [3]. No a priori assumption is given for the definition of operators and elements. It is hence possible to define a wide range of construction operators, as will be described later.

Thanks to this DAG-based representation, *BlindBuilder* is endowed with the following properties, which makes it a unique powerful representation formalism compared to that of the literature [1]: (1) *modularity*: the ability to reuse a part of the construction plan. Modularity may or may not be recursive; (2) *hierarchy*: the ability to consider as one single element what has already been built as opposed

[1] Note that in [11], some of these terms are used in the context of programs rather than graphs, with different meaning.

to having to target specific sub-elements for any new operations; (3) *generality*: the property according to which the representation can be easily extended to accept new kind of elements; (4) *3D representation*: some representations only consider 2-D structures, or don't scale-up well to 3-D structures.

2.1 Variation operators

Variation operators are used during evolution in order to generate offspring from parents at a given generation. *BlindBuilder* only relies on mutation as variation operators - this is due to the great diversity between construction plans that makes it very difficult to perform non-disruptive cross-over. Two kinds of mutations are considered. The first kind alterates only parameters of a node in construction plan using a gaussian probability centered on the previous value. The second kind deals with structural mutations, i.e. modification of the topology of the DAG, by adding or removing a node[2]. In practical, there are four structural mutation operators:

1. *append*: A new non-terminal gets as input the top level node and, if its arity is greater than one, it gets other randomly chosen node as parameter. The internal numerical parameters of the new node are randomly generated using uniform distribution.
2. *fusion*: A node and all its input nodes are shrinked into one into a new operator with randomly generated parameters.
3. *reconnect*: The input edges of a node are randomly reconnected to other nodes then the operator parameters are renewed at random.
4. *permute*: The input edges of a node are randomly permuted and the parameters are randomly generated.

Any of the structural mutations are performed in a way so that the constraints on a construction plan (only one top level node) are never violated. If a mutation is impossible to apply, the plan remains untouched. An individual is initialised using all the possible terminal nodes, and a few *append* mutations so that after initialisation only one DAG remains. It is important to note that these variation operators are *problem-independent* and can be used with any construction plan.

2.2 Construction operators

During evaluation of a given individual, the resulting structure is build out of the construction operators and atomic elements that are referenced in the construction plan. While variation operators are not problem specific, construction operators should be carefully designed depending on the problem at hand. In the following experiments, the construction operators work on 2D structures, made

[2] However concerned with variation of individuals between generations, the nature of our operators are inspired in part by the embryogenic approaches of [10]

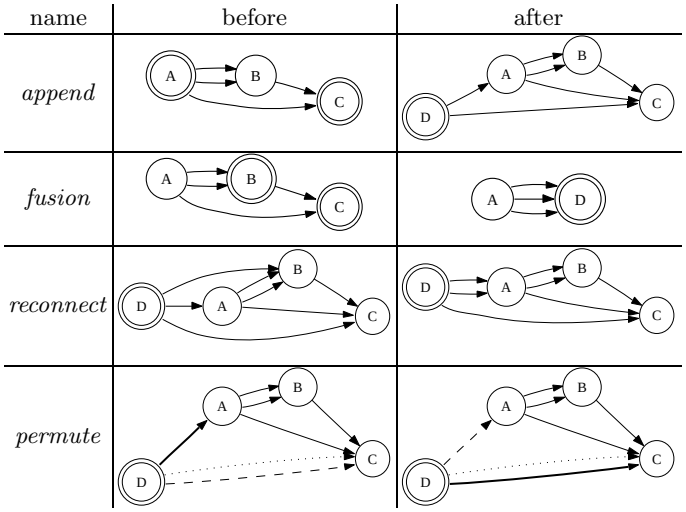

Fig. 1. The mutations operators (the double circled nodes are the affected nodes)

of layers of small wooden shelves[3]. Formally, a shelve is a box with an associated mass. The atomic operator is the operator that build a unique shelf. The other operators works on both a unique shelf or a group of shelves. *drop* puts a group of shelves on another group, with a variable shift. This shift is specified as a fraction of the length of a third building. *flip* flips upside-down a whole group of shelves, *mirror* also flip a group but horizontaly. *remove_top* removes the top layer of a group of shelves. Figures 1 illustrate the function of these operators.

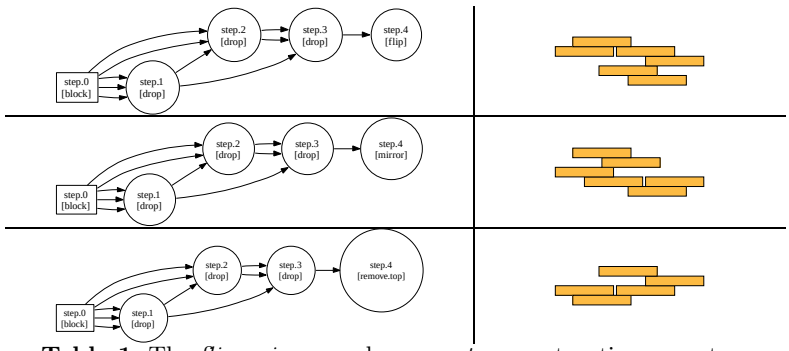

Table 1. The *flip*, *mirror*, and *remove-top* construction operators

[3] Compared to our previous work in [3], this setup is made simpler and makes and allows an in-depth analysis of underlying mechanisms that takes place during evolution.

3 Experimental settings

In this section, we present two evolutionary design problems, the tower problem and the bridge problem, along with their corresponding objective functions. Then, the evaluation process that makes it possible to compute the fitness of an individual, i.e. the adequacy of a given candidate solution with regard to the objective function, is described.

3.1 Objective Functions

The tower construction problem The tower problem is simply defined as a maximization problem where the objective is maximize the fitness of an individual. The fitness of an individual is defined as:

$$\frac{min(H_t, h)}{H_t}. \tag{1}$$

With the height of the current building h and the desired height of the tower H_t (i.e. the desired height defined by the user prior to evolution). H_t was set to the height of 300 times the thickness of a shelf.

The bridge construction problem The bridge problem consists in building a bridge defined as the longest construction including a "roadway" above the water with as few shelves as possible in contact with the ground. For this problem, two parameters must be taken into account: the height of the water plane H_w and the target length L_b of the bridge. $\Phi(height)$ is the total number of shelves that intersect the horizontal plane at a given $height$. This is a three objective maximization problem :

1. Minimal height ($fitness_1$):
$$\frac{min(H_w, h)}{H_w} \tag{2}$$
2. Roadway coverage ratio ($fitness_2$):
$$\frac{1}{L_b}\left(\sum_{x \in \Phi(H_w+1)} length(x)\right) \tag{3}$$
3. Shore coverage ratio ($fitness_3$):
$$1 - \frac{1}{L_b}\left(\sum_{x \in \Phi(0)} length(x)\right) \tag{4}$$

With h the height of the construction and $length(x)$ the (horizontal) length of shelf x. During selection and reproduction of the most fitted individuals, individuals are ranked according to the following algorithm inspired for a lexicographic

fitness approach : Given two individuals ind_i and ind_j, ind_i is better than ind_j only if $fitness_1(ind_i)$ is better than $fitness_1(ind_j) + threshold$. if the two individuals cannot be ranked (because the two fitness values do not differ enough with regard to the *threshold* value), comparaison is performed once again using $fitness_2$ in the same way. Then, if the two individuals still cannot be ranked, $fitness_3$ is computed and comparaison is performed *without* considering the *threshold*. In the following, the *threshold* is set to 0.01. H_w was set to the height of 4 times the thickness of a shelf and L_b to 14 times the length of a shelf.

3.2 Evaluation Process

As shown in figure 2, candidate solutions are evaluated in a two-steps process: "building" and "evaluation". Firstly, construction plan are interpreted so as to build the corresponding structure - this step is very fast to compute. Secondly, this structure is evaluated within a solid body physics simulator[4] during 250 simulation steps[5]. As opposed to step 1, this step is computationaly expensive. During step 2, blocks may fall down, leading to the worst fitness possible for the individual at hand - on the contrary, if the structure is robust with regard to the simulation, fitnesses described in the previous section are computed. The combination of "smart" construction operators and a costly but accurate test ensure buildability and relevant results with a reduced cost (because a subspace of the quite plausible constructions is explored).

Table 2. Building and evaluation processes: before evaluation (left) ; after evaluation (right).

4 Experimental Results

4.1 Methodology

Optimization Parameters Studied In this section, we provide results for the two objective functions with a great variety of parameter settings. Table 3 show all of the settings experimented. Note that only probability for mutation on parameters m_p and probability for reproduction r in the generational case are shown. The probabilities of the N_p structural mutation operators can be deduced from the following: $m_s = \frac{1-(m_p+r)}{N_p}$ where m_s is the probability of occurence for one structural mutation operator.

[4] We rely on Newton Game Dynamics - see `http://www.newtondynamics.com/`.
[5] time integration step is 100msecs

Evolution Engine	$(\mu + \lambda)$-ES	(μ, λ)-ES
population size	1, 10, 50, 100, 250, 500	
μ/λ ratio	1, 2, 3, 5	
parameters mutation prob	0.15, 0.35, 0.55, 0.75, 0.95	

Evolution Engine	generational with tournament
population size	10, 50, 200, 500, 1000
tournament size	2, 5, 8, 10
reproduction prob	0.3, 0.5
parameters mutation prob	0.07, 0.21, 0.35, 0.49, 0.63

Table 3. The 440 settings experimented

Comparing results from different parameter sets: Two parameter settings are compared as follow: firstly, approx. 20 runs are performed on each settings. Secondly, the distribution of the fitness maximum at each evaluation[6] is compared using a Student T test. So for each evaluation, it is possible to identify which parameter settings is better, along with a confidence rate. If a parameter setting is better than another for 90% of the evaluation steps with a confidence over 95%, we conclude that this setting is better than the other - if not, we conclude to a draw. In order to compare all parameter settings, each setting is compared to every other setting. Every time it is concluded to be better/worse than another setting, is gets/looses 1 point (and vice versa); if this is a draw, both setting get 0 point. Once all possible comparaison combination are exhausted, we end up with a ranked list of parameter settings.

4.2 Results

Figures and show the results for both the tower and bridge problems. Concerning mutations, only the mutation on parameter rate is shown and the structural mutation rate should be deduced as stated before.

The Tower construction problem: The best settings are those with a tiny population and with the lowest probability of mutation on parameter (i.e. high structural mutation rate). In its vast majority, these settings rely on the tournament selection, except for the noteworthy exception of number 9 (mu,$lamba$). Similarly, the worst settings use large populations, a high probability of parameter mutation and are either relying on (mu,$lambda$) and (mu+$lambda$) evolution engine. All of the experiments converge towards the highest possible tower, composed of piled up shelves (see figure) : only convergence speed differs. For this problem, the *BlindBuilder* representation is very efficient and the result of setting number 9 highlights the fact that success is more related to representation issues (including construction operators) rather than optimization parameters. Indeed, setting number 9 corresponds to a stochastic hill-climbing.

The Bridge construction problem: Compared to the previous problem, bridges structure evolved looks very differents eventhough they share similar properties: underwater pillars and over-water pillars that guaranty stability of the roadway

[6] using the number of evaluations rather than the number of generations makes it possible to compare settings with different population sizes.

Rank	Evolution engine	Pop size	Mutations settings	Score
1	(μ, λ) $\mu/\lambda = 2$	1	mut param = 0.15	439
2	tournament size = 2	10	mut param = 0.07 repro = 0.3	434
3	tournament size = 5	10	mut param = 0.07 repro = 0.5	431
4	tournament size = 8	10	mut param = 0.07 repro = 0.3	431
5	tournament size = 10	10	mut param = 0.07 repro = 0.5	430
6	tournament size = 8	50	mut param = 0.07 repro = 0.5	427
7	tournament size = 5	10	mut param = 0.07 repro = 0.3	421
8	tournament size = 10	50	mut param = 0.07 repro = 0.5	420
9	(μ, λ) $\mu/\lambda = 1$	1	mut param = 0.15	418
10	tournament size = 10	50	mut param = 0.07 repro = 0.3	417
11	tournament size = 5	50	mut param = 0.21 repro = 0.5	417
12	tournament size = 8	10	mut param = 0.07 repro = 0.5	415
13	tournament size = 2	10	mut param = 0.07 repro = 0.5	410
14	tournament size = 10	50	mut param = 0.21 repro = 0.3	409
15	tournament size = 10	10	mut param = 0.07 repro = 0.3	406
16	tournament size = 10	10	mut param = 0.21 repro = 0.5	406
...				
431	$(\mu + \lambda)$ $\mu/\lambda = 5$	100	mut param = 0.95	-397
432	$(\mu + \lambda)$ $\mu/\lambda = 3$	250	mut param = 0.95	-397
433	$(\mu + \lambda)$ $\mu/\lambda = 5$	500	mut param = 0.95	-398
434	(μ, λ) $\mu/\lambda = 5$	250	mut param = 0.75	-399
435	(μ, λ) $\mu/\lambda = 1$	500	mut param = 0.95	-410
436	(μ, λ) $\mu/\lambda = 1$	50	mut param = 0.95	-412
437	(μ, λ) $\mu/\lambda = 1$	250	mut param = 0.95	-415
438	(μ, λ) $\mu/\lambda = 1$	100	mut param = 0.95	-419
439	$(\mu + \lambda)$ $\mu/\lambda = 3$	500	mut param = 0.95	-423
440	$(\mu + \lambda)$ $\mu/\lambda = 5$	250	mut param = 0.95	-423

Table 4. the best and the worst settings for the Tower fitness case (440 settings)

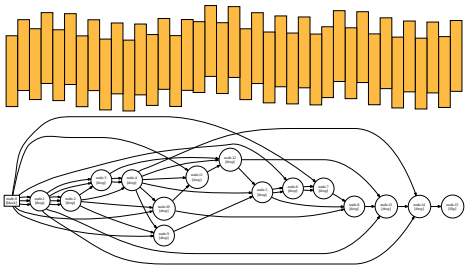

Fig. 2. A tower structure (rotated) and its corresponding plan

Rank	Evolution engine	Pop size	Mutations settings	Score
1	tournament size = 8	10	mut param = 0.07 repro = 0.5	159
2	(μ, λ) $\mu/\lambda = 3$	1	mut param = 0.15	151
3	(μ, λ) $\mu/\lambda = 2$	1	mut param = 0.15	151
4	tournament size = 10	10	mut param = 0.07 repro = 0.5	151
5	tournament size = 8	50	mut param = 0.07 repro = 0.5	151
6	tournament size = 2	10	mut param = 0.07 repro = 0.3	150
7	tournament size = 10	50	mut param = 0.07 repro = 0.5	146
8	tournament size = 5	10	mut param = 0.07 repro = 0.3	145
9	(μ, λ) $\mu/\lambda = 5$	1	mut param = 0.15	139
10	(μ, λ) $\mu/\lambda = 1$	1	mut param = 0.15	139
11	tournament size = 5	10	mut param = 0.07 repro = 0.5	139
12	(μ, λ) $\mu/\lambda = 1$	1	mut param = 0.35	135
13	tournament size = 5	50	mut param = 0.07 repro = 0.3	131
14	tournament size = 10	10	mut param = 0.07 repro = 0.3	128
15	tournament size = 10	50	mut param = 0.07 repro = 0.3	128
16	tournament size = 8	10	mut param = 0.07 repro = 0.3	127
...				
151	(μ, λ) $\mu/\lambda = 3$	50	mut param = 0.95	-137
152	(μ, λ) $\mu/\lambda = 1$	50	mut param = 0.95	-139
153	(μ, λ) $\mu/\lambda = 5$	1	mut param = 0.95	-143
154	(μ, λ) $\mu/\lambda = 2$	100	mut param = 0.95	-145
155	(μ, λ) $\mu/\lambda = 2$	50	mut param = 0.95	-147
156	(μ, λ) $\mu/\lambda = 2$	10	mut param = 0.95	-148
157	(μ, λ) $\mu/\lambda = 5$	100	mut param = 0.95	-149
158	(μ, λ) $\mu/\lambda = 1$	100	mut param = 0.95	-150
159	(μ, λ) $\mu/\lambda = 3$	10	mut param = 0.95	-153
160	(μ, λ) $\mu/\lambda = 5$	10	mut param = 0.95	-158

Table 5. the best and the worst settings for the Bridge fitness case (160 settings)

are frequently encountered. Here again, high structural mutation rate along with tournament selection is favored, but population are larger - in this case, simple stochastic hill-climbing is worse and the population-based approach is much more efficient. Moreover, high selection pressure for tournament selection are favored and reproduction rate has no significant impact. As for computational ressources, the best runs took around 1 to 3 hours on a single 2 Ghz CPU.

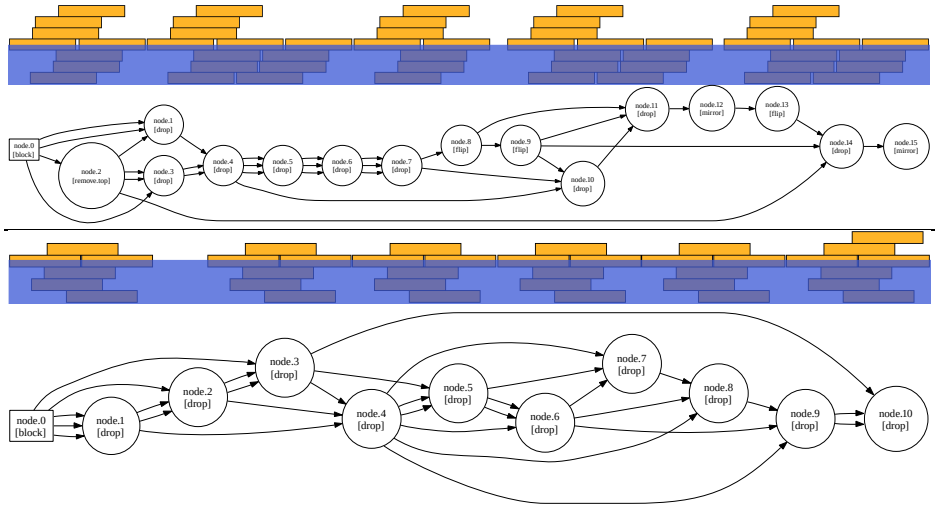

Fig. 3. Some of the bridges structures obtained with their corresponding plans

For validation purpose, we also performed experiment using classic canonical GP with ADFs - replacing DAGs with trees. Tree terminal set corresponds to *BlindBuilder* terminal nodes and operator parameters (depending on the context) and function set corresponds to *BlindBuilder* operators. It should be noted that there is a mapping between *BlindBuilder* construction plans and tree-based GP from the final construction viewpoint. Results (not shown here) are **very** deceptive : we tried common GP settings: tournament selection with 8 opponents, population size of 100 and 500 individuals, 100% mutation (no crossover) and 90% crossover (no mutation), and several ADFs number and depth. Runs were much slower, for the same number of evaluations (3 to 10 hours versus 1 to 3 hours with *BlindBuilder*) and resulted with individuals genotype mostly composed of bloat (unused code) and poor perfomance wrt. objective functions. We also conducted runs so as to guess other good parameter settings but results and CPU time required were discouraging enough, even for the tower problem, that we did not dig any further. From all our experiments, tree-based GP with ADF is not adapted for the Evolutionary Design problems considered here.

To conclude with our approach, we observe that as problems become more difficult (bridge vs. tower), evolution parameters increase in relevance (vs. stochas-

tic hill climbing): larger population and stochastic selection/replacement (tournament selection vs. other evolution engine). To sum it up, exploration (via structural mutation and tournament selection) is favored over exploitation (parameter mutation, more conservative approach during selection/replacement with ES-like evolution engine). With large constructions, we begin to pay the price of such an exploration strategy as fine (but difficult) tuning of parameters become necessary. For example, tower constructions evolved are more and more unstable during the course of evolution because blocks were not exactly aligned at first. Note that this could be addressed with a baldwinian approach to fine tune the parameter at evaluation time (e.g. on-line optimisation or wrapper-approach), but will not suppress the problem of poor performance for mutation on parameters.

5 Discussion

Previously shown experiments produced results comparable or even bigger in size than that of the literature, with (sometimes dramatically) less evaluations. However, it quickly becomes very difficult to expand existing structures eventhough both construction and variation operators could make it possible. This is caused by the fact that even a small alteration on a construction operator is likely to be more and more disruptive as the graph grows (i.e. altering deeper construction operators can be very disruptive). Indeed, the less disruptive way to expand an existing individual is to add new construction operators to the construction plan rather than to modify existing ones. As a consequence, trajectories in the search space differ greatly between evolution, even with the same parameter settings[7]. To address this issue, we could experiment with Islands Model[4] so as to better exploit hardware resources by eliminating non-promising runs.

However, the problem of scalability is only addressed in part using Island Models: the major results of our work is that current representation for indirect encoding are not truly scalable. This work has lead us to consider two possible extensions of graph-based representation for evolutionary design. The first is based on adding a new recursivity level, and the second is to consider context-aware construction plans. We shall describe these hereafter.

In its current form, our representation is already able to reuse substructures in the construction process. However, this could be much easier by adding construction operators defined as a parameterized production rule. For example, such a production rule could recursively build a subgraph with only the number of recursive calls as parameter (e.g. $produce(A, 5)$ would create the subgraph $A \leftarrow A \leftarrow A \leftarrow A \leftarrow A \leftarrow ?$, with A a unary construction operator). Once defined, such production rules are very interesting because of the very few parameters to deal with (i.e. the number of recursive calls) - moreover, resulting construction plan should be shorter, helping in reducing the dirsuptive mutation problem seen before.

[7] In this case, of course, only the seed for random number generation differs.

Another promising approach is to take advantage of the environment, i.e. to consider context-aware construction plans. In the standard setup, a construction plan is interpreted in an environment so as to produce the candidate structure. In some case however, it is possible to rely on information *from* the environment during the construction process: for example in the bridge problem, the water level is considered to evaluate the resulting construction, but could also be used to trigger construction phases. This would have many advantages: optimizing construction plans that could be executed in different environments (i.e. different water levels, different length needed) as well as keeping construction plans short (i.e. simple recursive process to build the underwater part of the bridge independently of the water level). Compared to our previous approach, the emphasis is put on the on-line construction process rather than off-line optimization.

Of course, both approaches can easily be combined, since they tend to address the same problem (scalability issue) along with being compatible (addressing smaller construction operator and developmental process triggering). To some extent, *BlindBuilder* can be extended in this direction and will be in the short term.

6 Conclusion

In this paper we have addressed the problem of Evolutionary Design with *BlindBuilder*, a new representation formalism that has been proved to be very efficient with regard to other works in the literature. We have focused our attention on specific construction and variation operator for optimizing classic evolutionary design problems. We conducted an extensive set of experiments so as to provide an in-depth analysis of comparable representation in the field of Evolutionary Design.

The results shown in this paper are comparable in size and time, and even better, to that of the literature. However the results were satisfying, our conclusions have lead us to reconsider the current approach and to highlight the drawbacks of commonly known indirect encoding approaches. In particular, the optimization process tends to be more and more inefficient as the size of graphs grow. While *BlindBuilder*'s specific properties make it possible to reduce the impact of such considerations, it cannot really scale up. Nevertheless, our in-depth analysis of the construction and optimization process have lead us to identify the current conceptual locks and to draw some insightful conclusion and perspectives on representation formalism for Evolutionary Design.

In the discussion section, we have highlighted several promising perspectives for our work to address before-mentionned problems. The first approach, yet more classical, is to expand our representation formalism with new construction operator embedding production rules. The second approach, more original, intends to take into account the very environment so as to perform a context-aware construction process, enabling both greater adaptivity to environmental

variations and smaller construction plans. These perspectives are currently under investigation and will be integrated in *BlindBuilder* in the next few months.

References

1. Peter J. Bentley. *Evolutionary Design by Computers*. 1999.
2. J. Bongard and R. Pfeifer. Evolving complete agents using artificial ontogeny, 2003.
3. Alexandre Devert et al. Blindbuilder: A new encoding to evolve lego-like structures. In *EuroGP*, pages 61–72, 2006.
4. Forrest H Bennett III et al. Building a parallel computer system for $18,000 that performs a half peta-flop per day. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1484–1490, 1999.
5. Jason Lohn et al. Evolutionary antenna design for a NASA spacecraft. In *Genetic Programming Theory and Practice II*, chapter 18, pages 301–315. 2004.
6. John R. Koza et al. *Genetic Programming 3: Darwinian Invention and Problem Solving*. 1999.
7. Maxim Peysakhov et al. Using assembly representations to enable evolutionary design of lego structures. *Artif. Intell. Eng. Des. Anal. Manuf.*, 17(2):155–168, 2003.
8. Peter Bentley et al. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 35–43, 1999.
9. Pablo J. Funes and Jordan B. Pollack. Computer evolution of buildable objects for evolutionary design by computers, 1998.
10. F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Ecole Normale Suprieure de Lyon, France, 1994.
11. Gregory S. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In *GECCO 2005*, volume 2, pages 1729–1736, 2005.
12. Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.
13. John Rieffel. *Evolutionary Fabrication: The Co-Evolution of Form and Formation* .. PhD thesis, Brandeis University, USA, 2006.
14. Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial Life*, 1(4), 1994.