Evolving Creativity

Penousal Machado[1,2], Jorge Tavares[1], Amílcar Cardoso[1], Francisco B. Pereira[1,2], and Ernesto Costa[1]

[1] Instituto Superior de Engenharia de Coimbra, Quinta da Nora, 3030 Coimbra, Portugal
[2] Centre for Informatics and Systems of the University of Coimbra, Pólo II - Pinhal de Marrocos, 3030 Coimbra, Portugal
{machado, jast, amilcar, xico, ernesto}@dei.uc.pt

Abstract. Transformational creativity requires a change of the search space. As such, Evolutionary Computation (EC) approaches are incapable of transformational creativity. In this paper, we discuss how canonical EC techniques can be extended in order to yield the potential of transformational creativity. We present a formalized description of how this can be attained, and the experimental results achieved with a meta-evolutionary scheme.

1 Introduction

In her works on creativity, Margaret Boden (e.g. [1]) identifies two types of creativity: exploratory and transformational (for short: *e-creativity* and *t-creativity*), and considers the latest as more important. A typical EC approach is thought to yield the potential for *e-creativity.* However, it is deemed as being incapable of *t-creativity.* In this paper we propose several extensions of EC, which we consider to confer to EC the potential for *t-creativity.*

In section 2 we make a synthesis of key concepts related with creativity, and of a formalization of some of Boden's ideas. In the next section we present a generic EC algorithm, and propose the evolution of some of its components. Then, in section 4, we make a short overview of EC systems in which some of these items are evolved. In section 5 we establish a relation between the evolution of EC components and *t-creativity.* The experimental results attained with a meta-evolutionary approach are presented and discussed in section 6. Finally, we draw some conclusions and present future research directions.

2 Exploratory and Transformational Creativity

In this section we introduce some key concepts related with creativity that are relevant for the remainder of the paper. We do not address most of the questions in detail, since that is clearly beyond the scope of the paper.

Boden [1] establishes two axis for the characterization of creativity. The first relates with proprieties of the product, and establishes a distinction between Historical and Psychological creativity: creating something that was never created before versus creating something that is novel only to the eyes of the agent. This distinction, although important from a social perspective, is of little consequence for our study.

The second axis characterizes the type of creativity – exploratory versus transformational – and has wider implications. Boden views the creative process as a search of new objects in a conceptual space. This space can be extremely convoluted, and therefore some of its points hard to reach. The discovery one point of this type is deemed as *e-creativity*. While *e-creativity* is an exploration of a conceptual space, *t-creativity* refers to the change of the conceptual space itself. As such, great breakthroughs that provoke paradigm shifts fit into this class of creativity [1].

Wiggins [2] formalized some of the ideas presented by Boden. This formalization – which sheds a new light into the somewhat vague description provided by Boden, making it more clear and precise – will be described in the next section. For a more thorough description we refer the reader to the original paper.

2.1 Formalization

The formalization proposed by Wiggins [2] includes the following elements:

- U – The space of all possible concepts. Or, for parsimony, the set of all possible concepts relevant to a given domain.
- L – A language that enables the definition of constraints and construction rules.
- $[[.]]$ – An interpreter for selecting concepts from U according to a set of constraints specified in L.
- $\langle\langle . \rangle\rangle$ – A search engine for traversing U, or one of its subsets, according to rules specified in L.
- R – A set of rules, in L, that defines a subset of U.
- T – A set of rules, in L, defining the search strategy.
- E – A set of rules, in L, which allow the evaluation of concepts.

These elements allow the modelling of Boden's ideas with more precision. Lets represent by C the conceptual space that experts in a given area usually consider in the search for new concepts. This space is a subset of U, defined by the rules in R.

$$C = [[R]](U) \tag{1}$$

Additionally, T is a set of rules encoding a search engine that allows the traversal of C, thus defining the connectivity between points of the space:

$$c_{i+1} = \langle\langle R \cup T \rangle\rangle(c_i) \tag{2}$$

E-creativity is modelled as an exploration of the search space C, using the search methodology specified by T, that leads to new points of C which are highly valued by E. One can add the additional constraint of these points being hard to reach, but this is not strictly necessary. Moreover, E can take into account aspects other than adequacy (e.g. novelty). According to the formalization, *t-creativity* can be achieved in several ways, namely:

– By changing the set of rules, R, that defines the elements of C, thus creating a new conceptual space, C'.
– By changing T, the rules governing the traversal of C. In this case the elements of C do not change, what is changed is the connectivity between the points of C.

This distinction only became clear due to the formalization [2]. Wiggins also notices that: "...a change in E opens up a whole new area of conceptual space, and possibly of universe, for consideration." [2]. In a human society a change in E is probably not easy to achieve. Nevertheless, we put forward the possibility of this being yet another way of achieving *t-creativity.*

Wiggins' formalization allows the definition of *t-creativity* in precise terms. Simply put, *t-creativity* is *e-creativity* at the meta-level. An exploratory creative system can be described by a sextuple:

$$\langle U, L, [[.]], \langle\langle . \rangle\rangle, R, T, E \rangle \quad (3)$$

T-creativity is the search for a new R or T or both. Since they are both expressed in L, L becomes the space of all possible concepts. A new language L_L is needed, to allow the definition of constraints and rules for traversing this space. We also need R_L, a set of constraints in language L_L that specifies the search space of *R's* and *T's* that will be considered, and T_L a set of rules in L_L for traversing that space. Additionally, we have the corresponding interpreters: $\widehat{[[.]]}$ and $\widehat{\langle\langle . \rangle\rangle}$. Finally we need an evaluation function E_L, also in L_L, that assesses the quality of the *R's* and *T's.* Thus, *t-creativity* can be described by the following sextuple:

$$\langle L, L_L, \widehat{[[.]]}, \widehat{\langle\langle . \rangle\rangle}, R_L, T_L, E_L \rangle \quad (4)$$

Hence the argument of *t-creativity* being *e-creativity* at the meta-level [2]. This also gives rise to the possibility of considering further meta-levels of creativity. There are, of course, practical implications, which include how to build: R_L, T_L, and E_L. We will return to these issues later.

3 Evolutionary Computation

Historically EC is divided into four families namely: Evolution Strategies (ES); Evolutionary Programming (EP); Genetic Algorithms (GA); and Genetic Programming (GP). In spite of their differences they can all be seen as particular instances of the Generic Evolutionary Algorithm (GEA) presented in figure 1.

The first step is the creation of a random set of genotypes, $G(0)$. These genotypes are converted to phenotypes through the application of a mapping function (map). In most cases there is not a clear distinction between genotype and phenotype, so this step is typically omitted. The next step consists on the evaluation of the individuals. This is performed at the phenotype level using a fitness function, $eval$. The main evolutionary cycle follows. A set of parents is selected, using sel, followed by the application of genetic operators, op, which

```
$t \leftarrow 0$
$G(0) \leftarrow generate\_random(t)$
$P(0) \leftarrow map(G(0))$
$F(0) \leftarrow eval(P(0))$
while $stop\ criterion\ not\ met$ do
    $G'(t) \leftarrow sel(G(t), P(t), F(t))$
    $G''(t) \leftarrow op(G'(t))$
    $G(t+1) \leftarrow gen(G(t), G''(t))$
    $P(t+1) \leftarrow map(G(t+1))$
    $F(t+1) \leftarrow eval(P(t+1))$
    $t \leftarrow t+1$
end while
return $result$
```

Fig. 1. Generic Evolutionary Algorithm.

yields a new set of genotypes, $G''(t)$. The next steps consist in the generation of the phenotypes and their evaluation. The evolutionary cycle continues until some termination criterion is met.

We are now in position to return to the original motivation for this paper: "How can transformational creativity be achieved by means of Evolutionary Computation?". Assuming that EC has the potential to achieve *e-creativity*, which seems to be the case; and that *t-creativity* is *e-creativity* at the meta-level, which follows from the work of Wiggins [2]; then meta-evolution should yield the potential to perform *t-creativity.* More precisely, what we propose evolving the following aspects of the evolutionary algorithm: mapping function, genetic operators, selection procedure, replacement strategy, and evaluation function.

4 Related Work

The area of Adaptive Evolutionary Computation (AEC) focuses on the evolution of specific parameters of EC algorithms. Angeline [3] makes a formal definition and classification of AEC, proposing three levels of adaptation: population-level, individual-level and component-level.

There are several AEC approaches that allow the dynamic resizing of the genotype, allowing its expansion and contraction according to environmental requirements. Angeline and Pollack [4] propose the co-evolution of: a high-level representational language suited to the environment; and of a dynamic GA where the genotype size varies. Also related to genotype-phenotype mapping, is the work of Altenberg [5] about the notion of "evolvability" - the ability of a population to produce variants fitter than previous existing ones. In [6, 7] Altenberg explores the concept of Genome Growth, a constructional selection method in which the degree of freedom of the representation is increased incrementally. This work is directly connected to the concepts presented by Dawkins in [8], where he clearly differentiates genetics, the study of the relationships between genotypes in successive generations, from embryology, the study of the relationships between genotype and phenotype in any one generation. This leads us to the concept of embryogeny, the process of growth that defines how a genotype is mapped onto a phenotype, and to the work of Bentley. In [9], the use of such growth processes within evolutionary systems is studied. Three main types of EC embryogenies are identified and explained: external, explicit and implicit. A comparative study between these different types, using an evolutionary design problem, is also presented.

The evolution of the genetic operators is the obvious next step. In [10], Teller describes how genetic operators can be evolved using PADO, a graph based GP system. In [11] GP is extended to a co-evolutionary model, allowing the co-evolution of candidate solutions and genetic operators. Another approach to the evolution of genetic operators is described in [12]. In this study, an additional level of recombination operators is introduced, which performs the recombination of a pool of operators. In [13] Spector and Robinson discuss how an autoconstructive evolutionary system can be attained using a language called Push.

5 Evolutionary Transformational Creativity

In this section we analyze the consequences of evolving several components of the traditional EC, and propose propose several alternatives for their evolution.

5.1 Transforming the Search Space

We start by analyzing what kind of transformation of the search space can be achieved by changing the *map* function. This function is responsible for the mapping between genotype and phenotype. A phenotype is a fully grown individual, or, from a more computational point of view, a candidate solution to a given problem. A genotype, is a piece of genetic code that once expressed via the *map* function, results in a phenotype. The genotype has no independent meaning, it only gains meaning after the application of a mapping function.

We can establish a connection between the *map* function and Wiggins' formalization. Considering that *map* is expressed in some language, that [[.]] is an interpreter for that language, and that u is the set of all genotypes – we have:

$$c = [[map]](u), \tag{5}$$

where c is the space of all possible phenotypes under *map*. Thus, *map* is *roughly* equivalent to R.

To be totally equivalent we would have to consider the space of genotypes, u, to be equivalent to U. According to Wiggins' formalization C and U are both concept spaces, and C should be a subset of U. In general, these propositions do not hold for c and u. In the typical scenario u and c are sets of different types, genotypes and phenotypes. Thus, we can establish a parallelism between C and c, however, in general, the same parallelism cannot be established between U and u. Nevertheless, for particular cases, e.g. when there is an identity relation between genotypes and phenotypes, u can be considered equivalent to U. Moreover, establishing an equivalence between u and U is not strictly necessary. Since *t-creativity* is about the transformation of C (or T), the equivalence between C and c is, from our view point, sufficient to be in accordance with Wiggins' formalization.

Additionally, and due to the genetic operators being applied at the genotype level, a transformation of *map* may also lead to a change of the connectivity of the space, and thus of T.

The way the space is traversed depends, mostly on the operators employed. They are the main ingredient in the definition of the connectivity of the space. In a less direct way, the connectivity also depends on the selection procedure, and on the replacement strategy (*gen*). As such, a change of *op*, *sel*, or even *gen*, can be seen as a transformation of T – the rules governing the traversal of C.

Like we stated before, the genetic operators are performed at the genotype level. This causes no conflict with the formalization. Equation 2.1 clearly states that the interpreter $\langle\langle . \rangle\rangle$ is applied to $R \cup T$.

Changing *sel* or *gen* does not appear to be as interesting as changing the genetic operators. Nevertheless, a change of these functions has the potential to change the search method, and as such yield *t-creativity*.

The relation between the fitness function, *eval*, and E is obvious. In section 2.1 we suggested that a change in E could also be a type of *t-creativity*. In the case of a EC approach this is certainly the case. The *sel* of the progenitors takes into account their fitness value. Therefore, a change in E can change the way the space is traversed.

5.2 Evolving EC Components

The most obvious approach to the evolution of EC components is the use of Meta-Evolution. Lets assume we are interested in evolving one of the components of EC, for instance the mapping function, and that we resort to GP to evolve populations of these functions.

Each genotype, G_i^{map}, is an encoding of a candidate mapping function; once expressed, via map^{map}, it results in a phenotype, P_i^{map}. Thus, the phenotypes are mapping functions expressed in some language L^{map}. For simplicity sake, we can assume that this language to be Turing Complete, which implies that any computable mapping function can be evolved.

We also need to define: sel^{map}, op^{map} and gen^{map}. Like map^{map} and $eval^{map}$ these functions are static. Therefore, we can use standard GP selection, operators and replacement strategy.

Moreover, we need to develop a way to assign fitness to different mapping functions, $eval^{map}$. One possibility is to use a set of rules that specify the characteristics that are considered desirable in a mapping function, and assign fitness based on the accordance with those rules. Although technically possible, this can be both difficult and pointless. To attain *t-creativity* we must be *e-creative* at this level. The chances of finding an interesting, innovative, and adequate mapping function, with a search procedure guided by a pre-established set of rules, or heuristics, seems slim. Moreover, in most scenarios, we might not even have a good idea about the kind of mapping function we are interested in. Additionally, EC is usually good at finding holes in the fitness function.

There is, however, something that we usually can take for granted: we are mainly interested in mapping functions that promote the discovery of good solutions for the original problem. We can, for each individual being evaluated, run an EC algorithm in which P_i^{map} is used as mapping function. By changing the GEA presented in figure 1 so that we can pass as arguments one, or several, of the following functions: *map*, *sel*, *op*, *gen* and *eval*, we can use an $eval^{map}$ such as the one presented in figure 2. The fitness of each phenotype is the result of a lower level EC. This result can indicate, for instance: the fitness of the best individual (at the lower level); the time necessary to reach the optimum; the average fitness of the last population; etc.

```
$eval^{map}(P^{map})$
    for $i = 1$ to $\#(P^{map})$ do
        $F_i^{map} \leftarrow AEG(P_i^{map})$
    endfor
    return $F^{map}$
```

Fig. 2. Meta-level fitness function.

We have stated that the evolved mapping functions could be expressed in some Turing Complete language. If this is the case, since the lower level EC runs P_i^{map}, we must deal with the Halting Problem. The typical solution is to impose a time constraint. Thus, if P_i^{map} does not halt after a pre-specified amount of time it is assumed that it will never stop and, accordingly, its fitness value will be low.

It should be more or less obvious that we can employ the exact same strategy to evolve: *sel*, *op*, *gen* or *eval*. If we are evolving evaluation functions, the lower level EC is guided by P_i^{eval}. However, we are interested in a P_i^{eval} which allows the discovery of individuals which are fit accordingly to some original fitness function. As such, the return value of GEA should reflect its performance according to this original function.

There is, of course, the possibility of adding more levels enabling the evolution of several components. Alternatively, we can also evolve several components simultaneously. In this case, each genotype would be an encoding of several functions, and the phenotype a set of functions, which are passed to the lower level EC. For instance, considering that all components are being evolved, we would have $P_i^{\{map,sel,op,gen,eval\}}$.

The main problem of the architecture presented in this section is its high computational cost. There are several other alternatives to the evolution of EC components, among which: Dual-Evolution and Co-Evolution. Due to space restrictions these approaches will not be analyzed in this paper, for a brief overview please consult [15].

6 Experimental Results

To test our ideas we decided to apply meta-evolution to evolve mapping functions. We use a two level meta-evolution scheme composed by a GP algorithm and a GA. At the higher level we have the GP algorithm, which is used to evolve the mapping functions. At the lower level we have the GA, whose task is finding the maximum value of a mathematical function. The goal is to evolve mappings that help the GA to accomplish its task.

The function being optimized by the GA, $f(x)$, is defined over the interval $[0, 1]$, and is the sum of $f_{peak}(x)$ and $f_{wave}(x)$, which are specified as follows:

$$f_{peak}(x) = max(0,\ |1 - 2|x - peak| - (1 - \frac{1}{r})| \times 0.1 \times r) \tag{6}$$

$$f_{wave}(x) = cos(2\pi \times r \times (x - peak)) \tag{7}$$

f_{wave} creates a sine wave with r repetitions in the $[0, 1]$ interval, returning values between -1 and 1. By adding f_{peak} we change the values of one of the repetitions, making it reach a maximum value of 1.1. In figure 3 we present a graph of $f(x)$. To increase the complexity of the problem, variable r is set to 100, which means that the wave function repeats itself 100 times in the $[0, 1]$ interval.

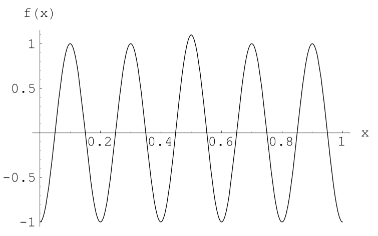

Fig. 3. Test function, $r = 5$, $peak = 0.5$

By changing the value of the variable *peak* we can change the coordinate of the maximum of $f(x)$. This ability to change the maximum is fundamental. If this value does not change the GP algorithm could find programs that output a constant x value corresponding to the maximum of $f(x)$. This is not, obviously, the desired behavior. What we aim to achieve is a GP program that transforms the search space in a way that helps the GA to find the maximum value. Thus, for each value of x the GP programs should compute a new value, x'. The re-organization of the search space induced by the x to x' transformation should make the task of finding the optimum easier.

To ensure that it is possible to find a good mapping function we decided to design one by hand. We were able to develop the following function:

$$g_{optimal}(x) = \frac{x + floor(frac(x \times 10000) \times r)}{r} \tag{8}$$

Where $frac$ returns the fractional part. This function has the effect of folding the space r times, and then expanding it back to $[0, 1]$. By using this mapping function the topology of the search space is changed, resulting in a less convoluted fitness landscape. In figure 4 we present a chart of this mapping function, $g_{optimal}(x)$, and of the search space resulting from its application, $f(g_{optimal}(x))$. Since $g_{optimal}$ maps $[0, 1]$ to $[0, 1]$, R is not changed. Instead, what is changed is the connectivity of the points of the space and, as such, T. This becomes possible due to the genetic operations being performed at the genotype level.

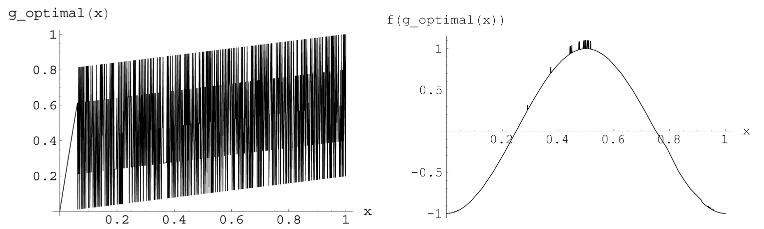

Fig. 4. On the left $g_{optimal}(x)$; on the right $f(g_{optimal}(x))$, $r = 5$.

To assign fitness, we run, for each GP individual, $mapping_j$ a GA. Each GA genotype, G_i^{sol}, is a binary string of size 50, encoding a value, $G_i'^{sol}$, in the $[0, 1]$ interval. The GP individual is used as mapping function for the GA. The value $G_i'^{sol}$ will be mapped to x_i' (thus, $x_i' = mapping_j(G_i'^{sol})$). x_i' is then used as the x coordinate for the function being optimized by the GA, f.

In order to get a good estimate of the quality of the mapping $functions$ we perform 30 runs of the GA for each GP individual. To prevent the specialization of the GP individuals, the value of $peak$ is randomly chosen at the beginning of each GA run. The fitness the GP individual is equal to the average fitness of the best individual of the last population of the lower level GA.

We use the following function and terminal sets: $\{+, -, \%, \times, floor, frac\}$, where % is the protected division; $= \{G_i'^{sol}, 1, 10, 100\}$, where $G_i'^{sol}$ is a variable holding the value of the GA genotype that is currently being mapped.

The settings for the GP algorithm were the following: Population size = 100; Number of generations 500; Swap-Tree crossover; Generate Random Tree mutation; Crossover probability = 70%; Mutation probability=20%; Maximum tree depth = 10. The settings for the GA where the following: Population size = 100; Number of generations = 30,100; Two point crossover; Swap Mutation; Crossover probability = 70%; Mutation probability=$\{1\%, 2.5\%, 5\%\}$.

6.1 Analysis of the Results

The main objective of our approach is to find a mapping function that consistently improves the performance of the GA algorithm. To evaluate the experimental results we need some reference points. Therefore, we conducted a series of experiments in which the mapping function was not subjected to evolution. In these tests we used the following static mapping functions: $g_{optimal}$, already described; and $g_{identity}$ with $g_{identity}(x) = G'^{sol}$. These results will be compared with the ones obtained using as mapping functions the best individuals of each GP run, $g_{evolved}$.

Table 1 shows the average fitness of the best individual of the last population of a traditional GA (Number of generations = 100) using as mapping functions $g_{identity}$, $g_{optimal}$ and $g_{evolved}$. The results are averages of 100 runs for the static mappings and of 3000 runs for the evolved mappings (100 runs per each evolved mapping).

Table 1. Average fitness of the best individual of the last population. The entries in bold indicate a statistically significant difference between these values and the corresponding $g_{identity}$ values ($\alpha = 0.01$).

Mutation	$g_{identity}$	$g_{optimal}$	$g_{evolved30}$	$g_{evolved100}$
1%	1.0134617	**1.0806543**	**1.0647961**	**1.0632421**
2.5%	1.0242724	**1.0942163**	**1.0843632**	**1.0821193**
5%	1.0323492	**1.0982930**	**1.0993311**	**1.0946481**

As expected using $g_{optimal}$ considerably improves the performance of the algorithm, yielding averages close to the maximum attainable value, 1.1. Table 1 shows that the use of the evolved mapping functions significantly improves the performance of the GA. However, the results attained are usually inferior to the ones achieved using $g_{optimal}$. This difference diminishes as the mutation rate increases. Additionally, using a smaller number of generations in the lower level GA leads to better overall results. The higher evolutionary pressure increases the need for the transformation of the search space.

To get a better grasp of how the use of the evolved mapping functions alter the GA, we present, in figure 5, the evolution of the fitness of the best individual during the GA run. For static mappings the fitness increases abruptly in the first generations, stagnating for the remainder of the run; with the evolved mappings the fitness increases steadily during the entire run. An analysis of the evolution of the average fitness of the GA populations gives insight to how the evolved mappings are improving the GA performance. The use of evolved mappings decreases significantly the average fitness of the populations. These results indicate that the evolved mappings improve the performance of the GA by promoting phenotypic diversity, preventing the early stagnation of the GA runs.

The evolved mappings are not similar to $g_{optimal}$, which cannot be considered surprising. As is often the case when analyzing the results of a GP program, it

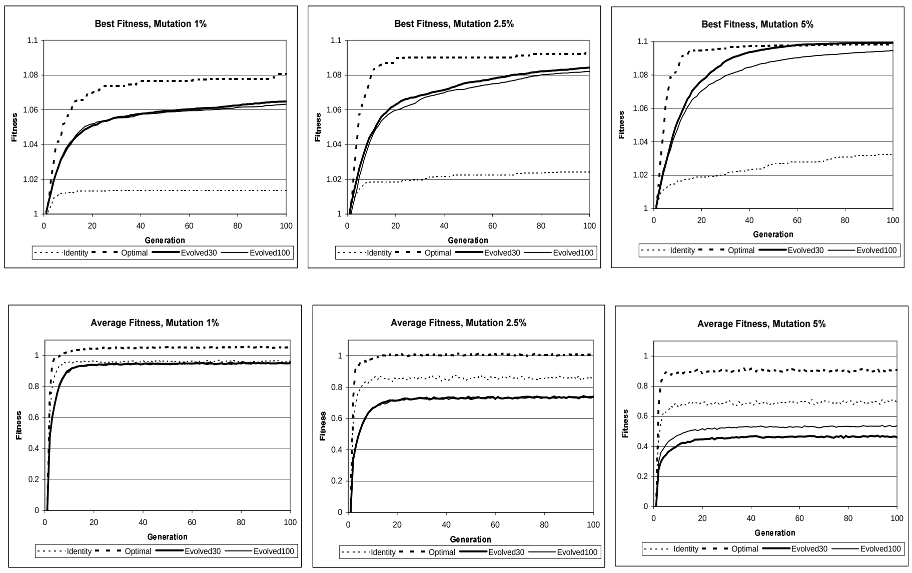

Fig. 5. Evolution of the fitness of the best individual and of the average fitness of the GA populations.

is not clear how the evolved mappings solve the problem of improving the GA performance. The charts suggest that reducing the number of GA generations used in the GP fitness assignment procedure, thus increasing the difficulty of the evolved mappings task, leads to better results. Taking into account the average of the GA populations when assigning a fitness value for the GP individuals, may also prove useful to achieve mappings closer to $g_{optimal}$.

7 Conclusions and Further Work

In this paper we discussed how the canonical EC algorithm can be extended in order to yield the potential for *t-creativity*. The proposed changes involve the evolution of several components of EC which are typically static. We establish a relation between the evolution of these components and the change of R and T, introduced in Wiggins' formalization of *t-creativity* [2]. Additionally, the evolution of these components may prove useful in improving the EC performance, lessen the burden of researchers, and provide indications about the characteristics of the problems being solve.

The attained results are promising, and provide pointers for the improvement of the approach. Future research will include: making a wider set of experiments; applying the proposed approach to a different set of domains; and using dual-evolution and co-evolution to evolve EC components.

References

1. Boden, M.A.: The Creative Mind: Myths and Mechanisms. Basic Books (1990)
2. Wiggins, G.: Towards a more precise characterisation of creativity in ai. In: Proc. of the ICCBR 2001 Workshop on Creative Systems, Vancouver, Canada (2001)
3. Angeline, P.J.: Adaptive and self-adaptive evolutionary computations. In: Computational Intelligence: A Dynamic Systems Perspective. IEEE Press (1995)
4. Angeline, P.J., Pollack, J.B.: Coevolving high-level representations. In: Artificial Life III. Volume XVII of SFI Studies in the Sciences of Complexity. (1994) 55–71
5. Altenberg, L.: The evolution of evolvability in genetic programming. In Kinnear, K.E., ed.: Advances in Genetic Programming. MIT Press (1994) 47–74
6. Altenberg, L.: Evolving better representations through selective genome growth. In: Proceedings of the 1st IEEE Conference on Evolutionary Computation. Part 1 (of 2), Piscataway N.J., IEEE (1994) 182–187
7. Altenberg, L.: Genome growth and the evolution of the genotype-phenotype map. In Banzhaf, W., Eeckman, F.H., eds.: Evolution as a Computational Process. Springer-Verlag, Berlin (1995) 205–259
8. Dawkins, R.: The evolution of evolvability. In: Artificial Life, SFI Studies in the Sciences of Complexity. Volume VI., Addison-Wesley (1989) 201–220
9. Bentley, P., Kumar, S.: 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. (1999) 35–43
10. Teller, A.: Evolving programmers: The co-evolution of intelligent recombination operators. In Angeline, P.J., Kinnear, Jr., K.E., eds.: Advances in Genetic Programming 2. MIT Press, Cambridge, MA, USA (1996) 45–68
11. Edmonds, B.: Meta-genetic programming: Co-evolving the operators of variation. CPM Report 98-32, Centre for Policy Modelling, Manchester Metropolitan University, UK, Aytoun St., Manchester, M1 3GH. UK (1998)
12. Kantschik, W., Dittrich, P., Brameier, M., Banzhaf, W.: Meta-evolution in graph GP. In: Genetic Programming: Second European Workshop, Springer (1999) 15–28
13. Spector, L., Robinson, A.: Genetic programming and autoconstructive evolution with the push programming language. Genetic Programming and Evolvable Machines **3** (2002) 7–40
14. Koza, J.R.: Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Stanford University (1990)
15. Tavares, J., Machado, P., Cardoso, A., Pereira, F.B., Costa, E.: On the evolution of evolutionary algorithms. In Keijzer, M., O'Reilly, U.M., Lucas, S.M., Costa, E., Soule, T., eds.: Genetic Programming 7th European Conference, EuroGP 2004, Proceedings. Volume 3003 of LNCS., Coimbra, Portugal, Springer-Verlag (2004) 389–398