Program Evolvability Under Environmental Variations and Neutrality

Tina Yu

Department of Computer Science
Memorial University of Newfoundland
St. Johns, NL A1B 3X5 Canada
tinayu@cs.mun.ca
http://www.cs.mun.ca/~tinayu

Abstract. Biological organisms employ various mechanisms to cope with the dynamic environments they live in. One recent research reported that depending on the rates of environmental variation, populations evolve toward genotypes in different regions of the neutral networks to adapt to the changes. Inspired by that work, we used a genetic programming system to study the evolution of computer programs under environmental variation. Similar to biological evolution, the genetic programming populations exploit neutrality to cope with environmental fluctuations and evolve evolvability. We hope this work sheds new light on the design of open-ended evolutionary systems which are able to provide consistent evolvability under variable conditions.

1 Introduction

Biological organisms live in an ever-changing world. However, early population genetics theory assumed the environment to be constant while the mathematical ecology assumed the genetic makeup of the species involved to be constant. About 40 years ago, Richard Levins published a seminal work which modeled the situation where evolution is taking place while the environment changes[7]. This work has influenced the population genetic theory to consider environmental fluctuations in studying the theory of evolution by natural selection.

While population genetic theory focuses on population-level adaptation, other empirical studies investigate molecular basis of phenotypic changes to cope with environmental variation. For example, it is shown that bacteria turn on the machinery for taking up iron from the environment, that is they synthesize siderophores, only when a lack of iron triggers the expression of over ten genes involved in the regulation of this system [11][2]. Currently, there is a rich volume of information revealing diverse strategies that organisms have evolved to cope with environmental fluctuations. One recent trend in ecology and evolution research is to integrate the diverse forms of 'adaptive variation' into a single conceptual framework [10].

In the evolutionary computation community, there are also interests in investigating the impact of environmental variation on the evolution of artificial

F. Almeida e Costa et al. (Eds.): ECAL 2007, LNAI 4648, pp. 835–844, 2007.

life. The most noticeable work is by Lipson and colleagues [8], who argued that modularity is an emerged property resulting from environmental variation. In that work, they used computer simulation to study modular structures induced under different environmental changing rates. They reported that the amount of modular separation is logarithmically proportional to the changing rate.

Kashtan and Alon [4] went farther by devising the environmental changes in a modular manner: the two objective functions used to evolve two different Boolean functions contained two identical modules. The only difference between these two objective functions was that they used two different logic operators to combine the two modules: one used AND and the other used OR. By switching the two objective functions periodically throughout the evolution process, they reported that the evolved Boolean functions not only contained both modules but also had a structure that could be easily switched into the other Boolean function with a small number of mutations.

This research investigates program evolvability [1] under environmental variations and implicit neutrality (semantic redundancy in genetic programming) [13]. We employed two objective functions which have two opposite goals. The first one is even-4-parity, whose genotypes contain an *even* number of all 4 input variables. The second one is always-on, whose genotypes contain an *odd* number of all 4 input variables [6]. We periodically switched these two objective functions using various rates. The interplay between program evolvability and neutrality under these changing rates are then analyzed.

This work is motivated by our interest in building computer systems for open-ended evolution, i.e. a system that provides consistent evolvability under environmental variation. According to the experimental results, under environmental fluctuations, populations exploited implicit neutrality to evolve evolvability. We hope this work sheds new light on the design of open-ended evolutionary systems for Artificial Life research.

The rest of the paper is organized as follows. Section 2 highlights the mechanisms biological organisms have developed to cope with variable conditions. In Section 3, we explain implicit neutrality in the genetic programming framework. The two objective functions of even-4-parity and always-on are then presented in Section 4. Section 5 details the computer experimental setup while Section 6 reports the experimental results and provides analysis. We discuss the implications of these results in Section 7. Finally, Section 8 concludes the paper and outlines our future research.

2 Background

Variation is the fuel of evolution. Under different environmental variation rates, different individual-level adaptation strategies may evolve [10]. When the environment changes rapidly, mechanisms such as *physiological plasticity* and *learning* may occur for individual organisms to respond to these changes. As environmental changes slow down, *phenotypic variation* may rise resulting from stochastic or directed heterogeneity in the developmental pathway. For even slower rates

of changes, mutations may produce *novel phenotype*. If environmental fluctuations are rare, populations may have a period of directional selection and thus have sufficient time to achieve *genetic robustness*. These phenotypic and genotypical changes in responding to environmental variation have been observed in viruses/bacteria and in experimental studies [9].

Genotypes and phenotype (or genes and traits) have a many-to-one relationship [3]. For any particular trait value, there exists a large number of genotypes that give rise to that value. Metaphorically, one can imagine the genotypes that map to the same phenotype as a network connected by mutations. Within the network, a mutation from one genotype to another is neutral, having no impact on the physiology, behavior or fitness of the organism. However, depending on the location in the network where the mutation takes place, there will be different outcomes. Near the periphery of the network, mutations are likely to produce different phenotype, whereas near the center of the neutral networks, mutations have little impact on the phenotype.

Under environmental variations, populations evolve toward genotypes in different regions of the neutral networks. In [9], Meyers and colleagues used a simple mathematical model and a single amino acid site to study evolution under two alternate environments. They reported that when the variation is rare, the populations swing back and forth between genetic robustness, which is located at the center of the neutral networks, of the two phenotypes. At intermediate rates of fluctuation, populations favor the edge of the neutral networks. Thus, mutation between the two phenotype occurs frequently. Finally, for highly variable environments, populations settled in a phenotype that has an intermediate fitness in both environments. This phenotype corresponds to organismal flexibility - individuals tolerate both conditions, but neither one exceptionally well.

Inspired by these biological phenomena and mathematical modeling results, we used computer simulation to study programs evolution under environmental variation. Similar to the biological evolution, there is a many-to-one mapping between genotypes and fitness. The computer evolutionary system used is genetic programming, which is described in the following section.

3 Genetic Programming and Implicit Neutrality

Genetic programming (GP) [5] models the process of natural evolution for problem solving by generating a population of possible solutions, which are selected and mutated to reach a near-optimum. In a GP system, a genotype represents a point in the search space and is operated on by genetic operators. In contrast, a phenotype represents a point in the solution space and is evaluated by the fitness function. In other words, selection is based on phenotypes, while reproduction operates on the underlying genotypes [12].

A GP system may or may not distinguish genotypes from phenotypes, depending on the implementation. In the first case, there can be a many-to-one mapping between genotypes and phenotypes while in the later case, there can be a many-to-one mapping between genotypes and fitness. Under either imple-

mentation, a mutation may transform genotypes from one to another without affecting the fitness; they are neutral mutations.

In that standard GP, the evolved genotypes are computer programs, whose behaviors are interpreted to give the phenotype and the corresponding fitness. In other words, fitness is evaluated directly on the genotypes. Since computer programs are semantically rich, many syntactically different programs may be interpreted to the same behavior, hence have the same fitness. Consequently, there is a many-to-one mapping between genotypes and fitness. We call this implicit neutrality [13] as it is embedded in the genotype representation without explicit encoding. In the tree-based GP representation, there are two forms of implicit neutrality: functional redundancy and introns.

Functional redundancy refers to the case when many different programs (genotypes) represent exactly the same function (phenotype). For instance, the following three programs give the same function XOR:

G1: NOR (AND x_1 x_2) (NOR x_1 x_2)
G2: NOR (NOR x_1 x_2) (AND x_1 x_2)
G3: NOR (AND x_1 x_2) (NOR (NOR x_1 x_2) (NAND x_1 x_2))

Genetic transformation from one genotype to another (e.g., G1 to G2) has a neutral effect on the program's behavior.

Introns are code that is part of a program but is semantically redundant. For example, the OR operator with input FALSE is an intron in the following genotype because (OR FALSE ANY-BOOLEAN-VALUE)=ANY-BOOLEAN-VALUE.

G4: OR FALSE (NOR (AND x_1 x_2) (NOR x_1 x_2))

Genetic transformations that remove introns from a genotype (e.g., G4 to G1) have a neutral effect on the program's behavior.

Functional redundancy and introns can emerge within an evolving genetic program and render implicit neutrality. This work will study genetic programs evolution under environmental variation. The two objective functions used to define the two alternating environments are even-4-parity and always-on, which are explained in the following section. We will use the term 'program' to refer to genotype and 'function' for phenotype.

4 Even-Parity and Always-On

Boolean parity is well studied in the GP community. These programs can take any n number of Boolean inputs. An even-n-parity returns TRUE if an *even* number of the inputs are TRUE while an odd-n-parity returns TRUE if an *odd* number of the inputs are TRUE.

An even-n-parity program can be constructed using only EQ logical operator if n is even [6]. Moreover, the EQ-only even-n-parity programs have an unique feature: they contain an *odd* number of all n input variables. In other words, if

any of the n input variables is missing or if there is an *even* number of any of the n input variables, the program is *not* an even-n-parity. This is because EQ is symmetrical, which allows the input variables to be in any order in the program without changing the semantics. Moreover, x_1 EQ x_1 is TRUE, hence (x_2 EQ x_1 EQ x_1) = x_2. With these two premises, any pair of repeated input variables can be removed from a program or added to a program without changing the program behavior. An even-n-parity program therefore always has an *odd* number of all n input variables.

Among those EQ-only programs that are *not* even-n-parity, the subset that contains an *even* number of all n input variables are always-on. These programs effectively ignore all input variables (since they all come in pair) and always return TRUE. The remaining EQ-only programs contain an *even* number of some input variables and an *odd* number of other input variables. We call these programs *in-between* as they are in between the two extremes of even-n-parity and always-on. With implicit neutrality, more than one program can be interpreted as even-n-parity or as always-on or as in-between. In [6], Langdon and Poli showed that for any program length l, only a small fraction (less then 1%) of the programs are interpreted as even-n-parity or as always-on. The majority of the EQ-only programs are in-between.

A program tree can have any length. However, because EQ is a binary operator, an EQ-only program tree always has an odd length: $l = 2t - 1$, where t is the number of leaf nodes in the program tree. These leaf nodes may contain any of the n input variables. As mentioned, even-n-parity has an *odd* number of all n input variables, $t = n + 2i, i = 0, 1, 2, \ldots$. The length of an even-$n$-parity program is therefore $l = 2t - 1 = 2n + 4i - 1$. In contrast, always-on has an *even* number of all n input variables, $t = 2p + 2i, p = 1, 2, 3, \ldots, n, i = 0, 1, 2, \ldots$. The length of an always-on program is therefore $l = 2t - 1 = 4(p + i) - 1$.

A program tree with length l can have many different shapes: $s = \dfrac{(l-1)!}{((l+1)/2)!((l-1)/2)!}$. The following table lists some of the program lengths and structures that are common to even-4-parity and always-on. The length ranges from the smallest of 7 up to 35. Figure 1 shows three example program trees with length 7 that are even-4-parity, always-on and in-between functions.

even-4-parity			always-on				
i	t	l	p	$p+i$	t	l	s
0	4	7	2	2	4	7	5
1	6	11	2	3	6	11	42
2	8	15	2	4	8	15	429
3	10	19	2	5	10	19	4862
4	12	23	2	6	12	23	58786
5	14	27	2	7	14	27	742900
6	16	31	2	8	16	31	9694845
7	18	35	2	9	18	35	129644790

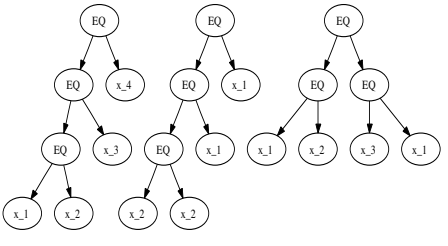

Fig. 1. (a) even-4-parity (b) always-on (c) in-between.

5 Experimental Setup

We allowed the program trees to have a length between 7 and 35. Since the internal nodes of a program tree are constant (EQ), only the leaf nodes were considered for evolution. Additionally, program trees with the same length might have different shapes. We therefore used a vector of length $t + 1$ to encode all possible variations: the first t cells contain the 4 possible input variables (x_1, x_2, x_3, x_4) while the last cell gives the shape of the program tree, with a value between 1 and s. During the initialization of the first population, uniform distributions were used to allocate genotypes to each of the 8 possible vector lengths.

The evolutionary system is steady-state with population size 10. At each generation, 5 offspring were generated by mutating 5 selected parent individuals. Among the 15 parent and offspring individuals, the top 10 were kept as the new generation. The selection method is fitness-proportional without scaling. Each selected individual was mutated on each cell of the vector with 10% probability. We let a run last for 1,000 generations, which is equivalent to 5,000 mutations and fitness evaluations. Three sets of experimental runs were made, one for each of the 3 environmental epoch lengths (λ=10, 20 and 50 generations). Each set consists of 100 runs.

Initially, the objective function was always-on. After λ generations, the objective function was switched to even-4-parity. The switching between these two objective functions took place every λ generations until the end of the run.

Fitness of an evolved program was calculated based on the number of test cases it solved. With 4 Boolean inputs, each can be TRUE or FALSE, the number of test cases is $2^4 = 16$. When the objective function is even-4-parity, the EQ-only programs have a needle-in-haystack fitness landscape: a program either solves 16 or 8 test cases. This property also applies to the always-on objective function program fitness landscape [6]. An evolved program, therefore, has fitness value either 16 or 8 under both objective functions.

6 Results and Analysis

Under different environmental variation rates, populations exhibit different program distributions. We show the results averaging over 100 runs in Figures 2, 4 and 6. Note that we only plot the first 100/200 generations as the rest of the generations have a similar pattern.

For all variation rates, populations were able to adapt to the new environment and evolved phenotype to meet the new objective function. However, the populations never completely converged to the target phenotype. Under rapid environmental fluctuation ($\lambda = 10$), populations did not have enough time to reach a stable condition. Nevertheless, the in-between programs are consistently present in the populations with a noticeable proportion (30-40%). These genotypes have half of the features required for both objective functions. They correspond to organismal flexibility - individuals tolerate both conditions, but neither one exceptionally well.

At an intermediate rate of fluctuation ($\lambda = 20$), populations were able to settle in a stable condition for a short period of time, during which the target

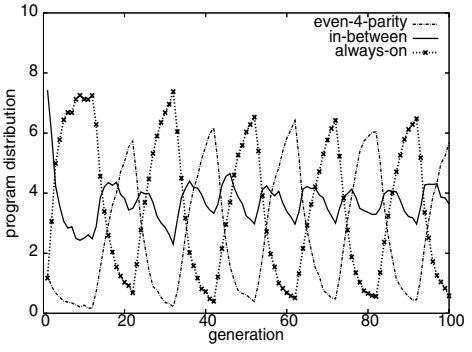

Fig. 2. program distributions under λ=10.

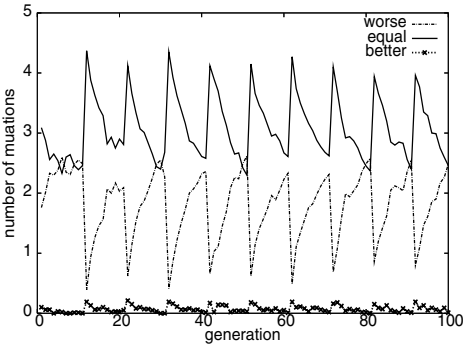

Fig. 3. program evolvability under λ=10.

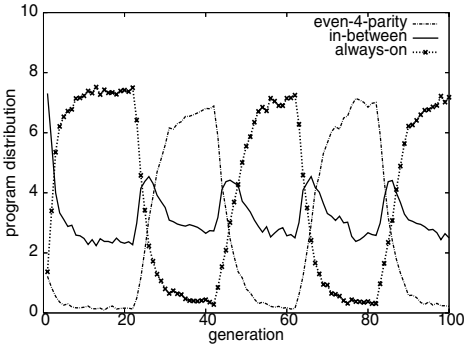

Fig. 4. program distributions under λ=20.

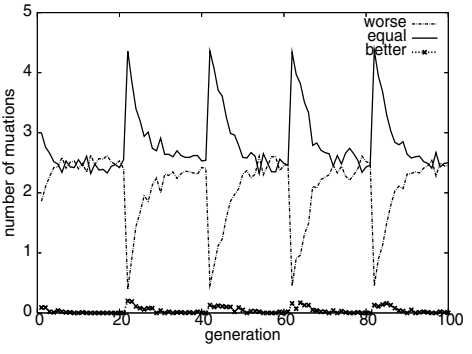

Fig. 5. program evolvability under λ=20.

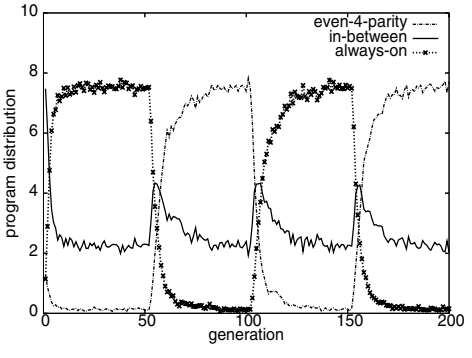

Fig. 6. program distributions under λ=50.

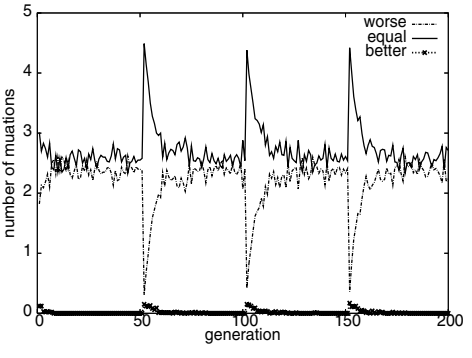

Fig. 7. program evolvability under λ=50.

genotypes occupied about 70% of the populations. Meanwhile, the number of in-between programs in the populations (around 25%) was lower than that under the fast variation rate. With an even slower environmental variation rate ($\lambda = 50$), the populations had a longer period of stable condition, where the number of in-between programs decreased to 20% while the number of the target genotypes increased to 75%. This indicates that populations start to evolve programs which are more resilient to changes, hence a less number of new phenotypes (in-between) were produced.

Although both intermediate and slow variation rates produced populations that evolved a similar combined number of in-between and the target genotypes, these genotypes have different characteristics. We collected individuals from the populations prior to switching to the new objective function from these two sets of runs. These genotypes were then evaluated with their terminal size (t). Figure 8 gives the results when the objective function was even-4-parity and Figure 9 gives the results when the objective function was always-on.

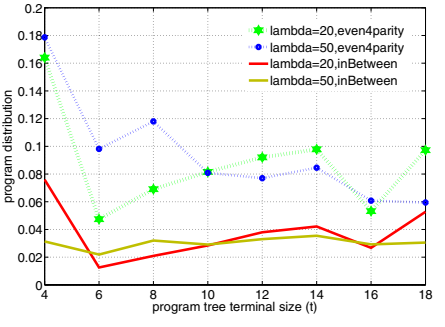

Fig. 8. program terminal size (t) distribution under even-4-parity environment.

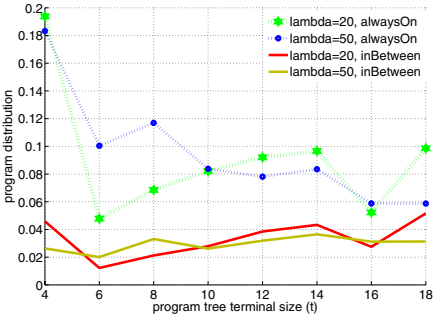

Fig. 9. program terminal size (t) distributions under always-on environment.

In general, populations evolved programs with a larger terminal size (t) under $\lambda = 20$ (one exception is $t = 16$, which might be caused by data distribution). These genotypes are more sensitive to mutations under the experimental setups we used. As mentioned in Section 5, each cell of a selected vector was mutated with 10% probability. Hence, the individual-level mutation rate is $\sum_{i=0}^{t+1} 0.1$. In other words, the larger the terminal size, the higher the individual mutation rate. These genotypes are located near the edge of the neutral networks, where mutations are more likely to produce different phenotypes to adapt to new environments when the change takes place.

Did the populations that evolved genotypes with a heightened mutation sensitivity produce more beneficial offspring? Figure 10 gives the information of all mutations that produced better offspring.

As shown, the populations which evolved genotypes with larger terminal sizes (under $\lambda = 20$) produced more beneficial offspring than those evolved genotypes of shorter terminal sizes (under $\lambda = 50$). Moreover, the majority of the geno-

	λ=10		λ=20		λ=50	
parent	qty	%	qty	%	qty	%
opposite	2129	28.29	1200	26.99	457	24.75
in-between	5396	71.71	3245	73.01	1389	75.25
Total	7525	100%	4445	100%	1846	100%

Fig. 10. Information of mutations that produced better offspring.

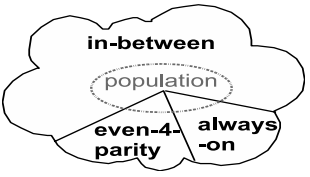

Fig. 11. Populations evolve toward boundaries of the 3 neutral networks.

types that produced better offspring are in-between. We also examined all these parent genotypes and found that they *all* have terminal size 18, hence are all located at the very end of the neutral networks. This indicates that populations evolved genotypes around the boundaries of the three phenotype neutral networks (see Figure 11). As shown in Figures 3, 5 and 7, all beneficial mutations took place right after the environmental switches. These provide strong evidences that under environmental variations, populations exploit neutrality to increase the like-hood of beneficial mutations.

7 Discussions

Using a simple GP system that switches between two opposite environments periodically, we have observed the populations favor genotypes with heightened mutation sensitivity, hence increase the rate of phenotypically consequential mutations, even though the genotypic mutation rate is the same. This evolution of evolvability [1] enhances the populations' ability to respond to selection and mutations according to the changes of their environments.

The emergence of self-adaption of mutation rates to improve evolvability has also been observed in a different GP system with a many-to-one genotype-phenotype mapping [13]. In that study, the objective function was constant throughout the evolution process. However, the populations exhibited an incremental progress that evolved from genotypes with lower fitness to higher ones step by step until the target phenotype was reached. We can therefore view it as an environment with continuous changes of target phenotypes from lower-fitness to higher-fitness. This pattern is similar to that of this study in that the populations focus on one phenotype at a time. We speculate that the emergent of self-adaption of mutation rates is likely to hold for a larger class of systems if the neutral networks contain genotypes with variable mutation sensitivities.

In this work, the prior and future environments are identical. If, instead, the environment continually shifts to completely novel states, the evolutionary history of a population might not prepare it for future adaptation. We speculate that populations will still evolve a certain number of genotypes with heightened mutation sensitivity, as long as they exist in the phenotype neutral networks. When considering open-ended evolution, populations which are capable of adapting mutation rates according to the environment they live in are more likely to continuously evolve evolvability.

8 Conclusions

Simulating biological evolution process for problem solving has been the backbone of evolutionary computing. While many aspects of the process have been integrated in computer systems to study the evolution of artificial life forms, not much attention is given to environmental variations. With the recent research development in evolutionary biology that clarifies the relationships between molecular, functional and ecological variations, we become better equipped to explore the implications of artificial life evolution under heterogeneous environments.

This paper is our initial effort in researching the interplay between program evolvability and neutral networks under environmental variations. Using a simple GP system under 2 alternating objective functions, we have observed populations exploiting neutrality to cope with environmental fluctuations and evolve evolvability. We will continue this work by investigating more complex environment variation patterns and other forms of neutrality.

Acknowledgment. This research is supported by NSERC of Canada.

References

1. Altenberg, L.: The evolution of evolvability in genetic programming. In: Kinnear Jr., K.E. (ed.) Advances in Genetic Programming, pp. 47–74 (1994)
2. Crosa, J.H.: Signal transduction and transcriptional and post transcriptional control of iron-regulated genes in bacteria. Microbiology and Molecular Biology Reviews 61(3), 319–336 (1997)
3. Huynen, M.A., Stadler, P.F., Fontana, W.: Smoothness within ruggedness: The role of neutrality in adaptation. Proceedings of the National Academy of Sciences of the United States of America (PNAS) 93(1), 397–401 (1996)
4. Kashtan, N., Alon, U.: Spontaneous evolution of modularity and network motifs. Proceedings of the National Academy of Sciences of the United States of America (PNAS) 102(39), 13773–13778 (2005)
5. Koza, J.R.: Genetic Programming. MIT Press, Cambridge (1992)
6. Langdon, W.B., Poli, R.: Why "building blocks" don't work on parity problems. Technical Report CSRP-98-17, University of Birmingham (1998)
7. Levins, R.J.: Evolution in Changing Environments. Princeton Univ. Press, Princeton (1968)
8. Lipson, H., Pollack, J.B., Suh, N.P.: On the origin of modular variation. Evolution 56(8), 1549–1556 (2002)
9. Meyers, L.A., Ancel, F.D., Lachmann, M.: Evolution of genetic potential. Computational Biology 1(3), 236–243 (2005)
10. Meyers, L.A., Bull, J.J.: Fighting change with change: adaptive variation in an uncertain world. Trends in Ecology & Evolution 17(12), 551–557 (2002)
11. Neilands, J.B.: Iron absorption and transport in microorganisms. Annual Review Nutrition 1, 27–46 (1981)
12. Yu, T., Bentley, P.: Methods to evolve legal phenotypes. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) Parallel Problem Solving from Nature - PPSN V. LNCS, vol. 1498, pp. 280–291. Springer, Heidelberg (1998)
13. Yu, T., Miller, J.: Through the interaction of neutral and adaptive mutations, evolutionary search finds a way. Artificial Life 12(4), 525–551 (2006)