Evolution of Rewriting Rule Sets Using String-Based Tierra

Komei Sugiura[1], Hideaki Suzuki[2], Takayuki Shiose[1],
Hiroshi Kawakami[1], and Osamu Katai[1]

[1] Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
{sugiura, shiose, kawakami, katai}@sys.i.kyoto-u.ac.jp
[2] ATR Human Information Science Laboratories
2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
hsuzuki@atr.co.jp

Abstract. We have studied a string rewriting system to improve the basic design of Tierra. Our system has three features. First, the Tierra instruction set is converted into a set of string rewriting rules using regular expressions. Second, every agent is composed of a string as the genome and a set of string rewriting rules as its own instruction set. Third, a genetic operation and selection of rewriting rules are introduced to allow the agents to weed out the worst rules. We carried out experiments on how agents evolve through self-replication. The results have shown that our system can evolve not only the agents' genomes but also the rewriting rule sets.

1 Introduction

Artificial Life (ALife) research attempts to not only investigate complex phenomena of life but also to engineer artificial systems with life-like properties such as autonomy and evolvability. Accordingly, it is necessary to design the process for constructing the system rather than to design the system itself. Designers have to prepare two main components: a set of objects (symbols [1], character sequences [2–4], binary strings [5], λ-terms [6]) standing for agents and environments and a set of reaction rules (rewriting rules [1–5], λ-calculus [6]) defining the interaction among objects.

The evolvability of an ALife system depends heavily on this basic design. Hence, these objects and reaction rules must be robust to changes and capable of open-ended evolution. Nevertheless, previous studies have focused mainly on the evolution of agents using fixed reaction rules. Man-made rules, however, do not always have the capability of open-ended evolution.

Suzuki has reported the optimization of string rewriting grammar [3], and Matsuzaki proposed a way to translate the Tierra [7] instruction set [8]. However, no previous research has realized a Tierra-like ALife system as a string rewriting system nor evolved a set of string rewriting rules.

In this paper, we propose an ALife system in which both the agents and the set of reaction rules can evolve together by providing the system with the ability to improve the rule set by itself. In order to realize this, we take three steps to develop a string rewriting system based on Tierra. First, we convert 32 Tierran instructions into a set of 140 independent rewriting rules (initial rule set). Each rule is represented as a sequence of the matching or substitution of regular expressions, which makes the rule robust against changes to it. Second, we equip every agent with both a string as the genome and a set of string rewriting rules and then make the agent self-replicate using its own rule set. Lastly, we introduce a genetic operation and selection of rewriting rules so that the agents can weed out the worst rules.

2 String-Based Tierra

2.1 Comparison with Tierra system

In this system, every Tierran instruction is denoted by a character, and the execution of an instruction is represented as the application of a string rewriting rule. Each agent is composed of a string as the genome and a rewriting rule set. An agent replicates its genome by using its own rewriting rules. In order to improve the rewriting rule set, we introduce a genetic operation and natural selection into the system. Table 1 compares our system with Tierra in terms of object, genome, and reaction rule.

Table 1. Comparison of proposed system and Tierra

	Proposed system	Tierra
Object	Character	Machine code
Genome	String	Sequence of machine code
Reaction rule	String rewriting rule	Execution of instruction

2.2 Set of Objects

In our system, the objects have three kinds of characters: instruction, register, and membrane.

Instruction Characters Each instruction character corresponds to one of thirty-two Tierran instructions. For example:

NOP0 = '`f`', NOP1 = '`t`', PUSHA = '`A`', ..., DIVIDE = '`z`'

Register Characters Register characters represent an instruction pointer, registers (as, bx, cx, dx), and a ten-word stack contained by a Tierra CPU. Each agent has these characters in its genome as well as instruction characters. Instead of storing an address in RAM, a register character implies that it stores the number of instruction characters between the beginning of the genome and itself and also indicates the next instruction character. Containing these characters in a genome has two advantages:

- It makes it possible to represent the movement of a pointer, etc., as rewriting a string, and
- it enables agents to have multiple pointers, etc., which means the execution of the instruction can be made in parallel.

Membrane Characters These characters are introduced to show the beginning and the end of a genome. The register characters of an agent are contained only inside its membrane.

Table 2 shows the characters used in our system.

Table 2. Characters used in proposed system

	Proposed system	Tierra
Instruction characters	`t, f, A,..., Z, w, x, y, z`	Instruction words
Register characters	`p`	Instruction pointer (IP)
	`a, b, c, d`	Registers (ax, bx, cx, dx)
	`0, 1,..., 9`	Ten-word stack
Membrane characters	`[, ]`	

2.3 Rewriting Rules Using Regular Expressions

We represent the execution of a Tierran instruction as string rewriting. Each Tierran instruction is converted into one or more string rewriting rules. This approach has the following three advantages over Matsuzaki's proposal [8]:

1. *Using regular expressions*
   We write string rewriting rules with Perl regular expressions [9], which enables us to carry out complicated rewriting. On the other hand, the rewriting rules proposed by Matsuzaki use their own form.
2. *Sequence of matching/substitution*
   A string rewriting rule is represented as a sequence of the matching or substitution of a regular expression combined by *AND* operators. This method simplifies the rules and provides them with robustness against changes.

3. *Independency among rules*
   We represent 32 Tierran instructions as 140 string rewriting rules, which are independent from each other. This rule set has an advantage over Matsuzaki's, where a priority order is needed owing to the non-independency among rewriting rules. In our system, rules are applied from top down sequentially, but the order is changeable. This means that our system has no priority order among rewriting rules.

The following two examples show how our rewriting rules are applied for 'INCA' and 'IFZ'.

Example 1 INCA (increment ax) is represented as follows:

```
s/pN/Np/g && s/a($J)/$1a/g
```

where:

`N` : instruction character corresponding to INCA
`$N` = `[`*register characters*`]`
`$J` = `$N*[`*instruction characters*`]$N*`

`$N` is the character class representing register characters, and `$N*` represents zero or more `$N`. Therefore, `$J` matches one arbitrary instruction character that has zero or more `$N` before and after.

Fig. 1 shows the execution of INCA as string rewriting. First, the former substitution makes `p` move right for one instruction character. Next, the latter substitution makes `a` move right for one instruction character as well. These operations correspond to the fact that the IP and register ax are incremented in Tierra.

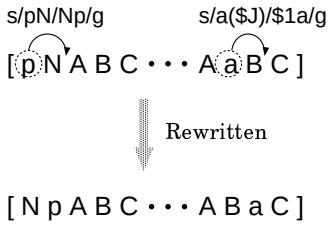

Fig. 1. Execution of INCA as string rewriting

Example 2 IFZ (if cx == 0 perform next instruction, otherwise skip it) needs multiple rules. IFZ is represented as follows:

```
  /\[$N*c/ && s/pU/Up/g
! /\[$N*c/ && s/pU($J)/U$1p/g
```

where 'U' is the instruction character corresponding to IFZ.

Fig. 2 shows the execution of IFZ as string rewriting. The operation of "/\[$N*c/" returns *false*, i.e., the regular expression does not match the genome. Hence, the substitution of the upper rule is not executed, as shown in the left-hand figure. On the other hand, the operation of "! /\[$N*c/" returns *true*, so that the substitution of the lower rule is executed, as shown in the right-hand figure.

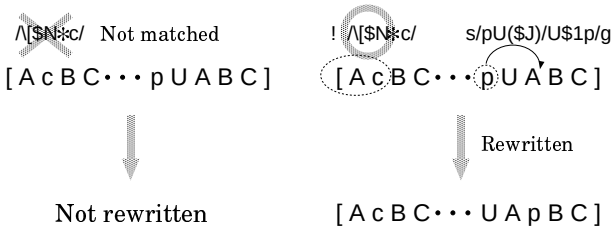

Fig. 2. Execution of IFZ as string rewriting

2.4 Selection of Rewriting Rules

In our system, each agent has its own rewriting rule set **R**, and it self-replicates using **R**. Here, every rewriting rule in **R** is executed from top down sequentially. The number of applied rules per iteration, therefore, depends on the order of the rules. Even if a couple of agents have the same genome, the more the order of its own rules is adapted to the genome, the fewer iterations the agents need to self-replicate.

We introduce the selection of rules in order to improve rewriting rule sets. Here, the fitness value of a rewriting rule is defined as the number of applied times. Fig. 3 shows schematically the selection of rewriting rules. Selection in **R** is executed as follows:

1. sort all of the rules according to the number of applied times,
2. delete l rules from the bottom,
3. generate new l rules by giving l elite rules genetic operation,
4. insert new rules in the middle of **R**.

This operation is executed with probability p per iteration. Through this operation process, rules with fewer applied times are removed and rules with many applied times are selected.

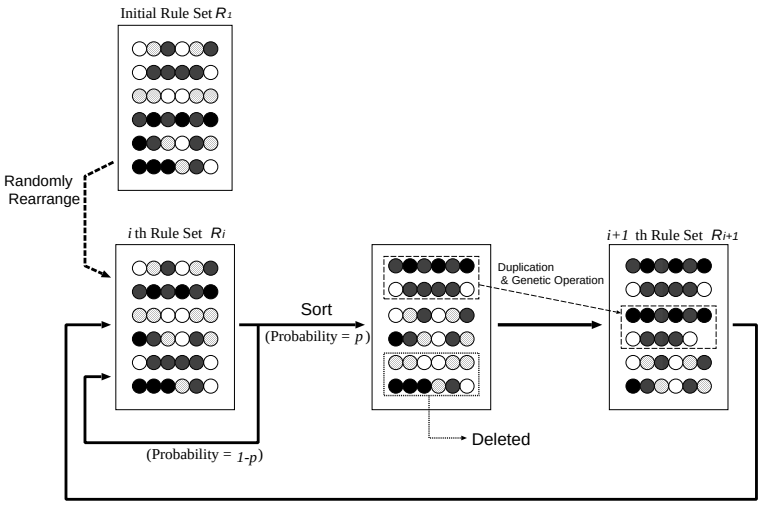

Fig. 3. Selection of rewriting rules

2.5 Selection of Individuals

In our system, a genome is represented as a string, so that its length is elastic. To avoid the infinite reproduction of agents, the maximum population N_{max} is introduced. If the population of agents exceeds N_{max}, randomly chosen agents are deleted.

Our system also deletes agents that are not able to self-replicate. Suppose that an agent cannot apply any rule due to a disadvantageous mutation in its genome. To remove such agents, the system checks the number of applied rules per iteration and weeds out every agent without an applied rules in its rule set.

2.6 Mutation

Our system has the following two mutations for both genomes and rewriting rules, while the Tierra system has only mutation for genomes.

Genome Mutation This string-based system has two other mutations, insertion and deletion, in addition to substitution, which is used in Tierra.

Rule Mutation The rule mutation contains three kinds of operations: duplication, deletion, and modification. As an example, the modification of a rule R is represented as follows:

$$R = r_1 \ \& \ r_2 \ \& \cdots \& \ r_i \ \& \cdots \& \ r_n \Longrightarrow r_1 \ \& \ r_2 \ \& \cdots \& \ r'_i \ \& \cdots \& \ r_n$$

In order to change r_i into r'_i, we introduce three kinds of operations as rule mutation: insertion, deletion, and substitution.

3 Results and Discussion

In our system, the ancestor self-replicates using the initial instruction set (140 rules), which is equivalent to the Tierran instruction set. The ancestor's genome in our system is a string converted from the genome of the Tierra ancestor. We started the experiment under the condition of that the number of ancestors is N_{max}. The parameters we use are as follows:

$$
\begin{aligned}
N_{max} &= \text{maximum number of agents} \\
m_g &= \text{genome mutation rate} \\
m_r &= \text{rule mutation rate} \\
p &= \text{probability of rule selection per iteration} \\
l &= \text{number of rules weeded out per rule selection}
\end{aligned}
$$

We obtained the most remarkable result under the condition of $N_{max} = 32$, $m_g = 0.01$, $m_r = 0.9$, $p = 0.02$, $l = 14$. It took about four hours for a server with a 2.8-GHz CPU to carry out 100,000 iterations.

In Fig. 4, the number of applied rules is plotted against the number of iterations. The number of applied rules is defined as the number of rewriting rules that are applied from the set of 140 rules within one individual. In the figure, the solid, dotted, and broken lines show the maximum number (best) of applied rules, the minimum number, and the average number, respectively. Fig. 4 shows that the maximum number of applied rules increases with the increase in iterations. Specifically, the maximum number is approximately 13 for iterations between 10,000 to 60,000 and increases to 25 for iterations of more than 80,000. This result means that the agents obtain important rules while weeding out unnecessary rules.

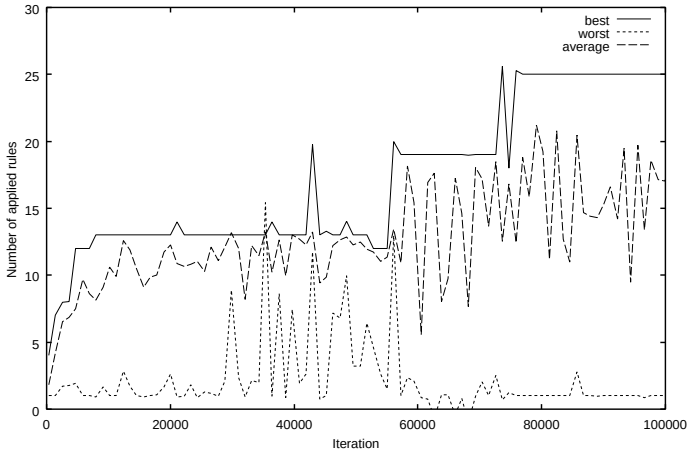

Fig. 4. Variation of applied rules

In Fig. 5, the interval of cell division is plotted against the number of iterations. The interval of cell division is defined as the number of iterations from the latest to the second latest cell division. Fig. 5 shows that the best interval, the average interval, and the worst interval, all drastically decrease with the increase in iterations. Specifically, the average interval is approximately 500 at 1000 iterations but decreases to 30 at 100,000 iterations. This result can be explained by the fact that the agents after 100,000 iterations have a better rule set than the ancestors.

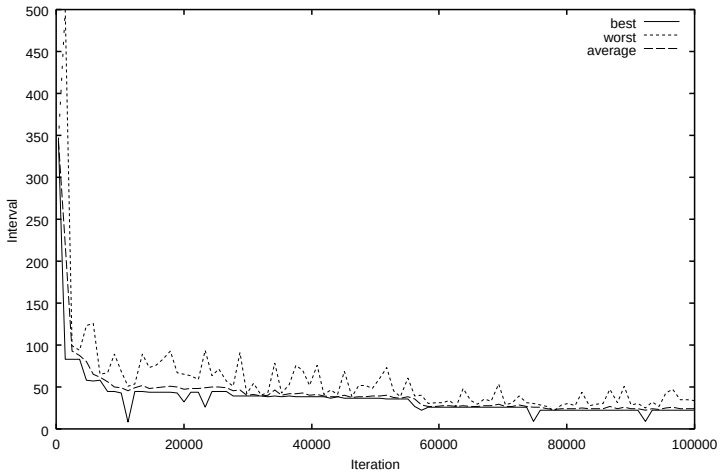

Fig. 5. Variation of interval of cell division

We also studied the variation of the genome length. The length of the ancestor was 87, while the length of the descendants after 100,000 iterations was reduced to 69. This indicates that the genome was improved through the iterations.

Thus we obtained three fundamental results: an increase in the number of applied rules per iteration, a decrease in the average interval, and a decrease in the genome length. These results clearly indicate that both agents' genomes and rewriting rules sets evolved in our system.

Finally, we briefly describe an example of newly generated rules. The ancestor replicates its genome by executing the reproduction loop dozens of times. The rewriting rules are performed 10 times in one loop. We have found that an agent after 40,000 iterations generates a new rule:

```
s/pfN/fNp/g && s/a($J)/$1a/g
```

This rule can execute the sequence of instructions '`fN`' (NOP0 and INCA) in one substitution. Namely, it reduces the number of rules necessary for the loop to 9. This result also supports the fact that both the agent's genomes and rewriting rule sets can evolve in our system.

4 Conclusions

In this paper, we studied the evolution of rewriting rule sets in order to improve the basic design of Tierra. In order to realize this, we took three steps to develop a string rewriting system based on Tierra. First, we converted 32 Tierran instructions into a set of 140 independent rewriting rules (initial rule set). Each rule is represented as a sequence of the matching or substitution of regular expressions combined by *AND* operators. Second, we equipped every agent with both a string as the genome and a set of string rewriting rules as its own instruction set, and we made the agents self-replicate by string rewriting. Lastly, we introduced a genetic operation and selection of rewriting rules so that the agents could weed out the worst rules. The ancestor of our system consists of both the genome equivalent to the Tierra ancestor and the initial rule set, and it self-replicates by string rewriting.

We carried out experiments and obtained three key results: an increase in the number of applied rules per iteration, a decrease in the average interval, and a decrease in the genome length. These results indicate that our system can evolve not only the agents' genomes but also the rewriting rule sets.

References

1. Suzuki, Y., Tanaka, H.: Chemical evolution among artificial proto-cells. In Bedau, M.A., McCaskill, J.S., Packard, N.H., Rasmussen, S., eds.: Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life, MIT Press (2000) 54–63
2. Suzuki, H.: Evolution of self-reproducing programs in a core propelled by parallel protein execution. Artificial Life **6** (2000) 103–108
3. Suzuki, H.: String rewriting grammar optimized using an evolvability measure. In Kelemen, J., Sosik, P., eds.: Advances in Artificial Life (6th European Conference on Artificial Life Proceedings), Springer-Verlag, Berlin (2001) 458–468
4. Suzuki, H., Ono, N.: Universal replication in a string rewriting system. In: Proceedings of the Fifth International Conference on Humans and Computers (HC-2002). (2002) 179–184
5. Dittrich, P., Banzhaf, W.: Self-evolution in a constructive binary string system. Artificial Life **4** (1998) 203–220
6. Fontana, W.: Algorithmic chemistry. In Langton, C.G., Taylor, C., Farmer, J.D., Rasmussen, S., eds.: Artificial Life II: Proceedings of the Second Artificial Life Workshop, Addison-Wesley (1992) 159–209
7. Ray, T.S.: An approach to the synthesis of life. In Langton, C.G., Taylor, C., Farmer, J.D., Rasmussen, S., eds.: Artificial Life II: Proceedings of the Second Artificial Life Workshop, Addison-Wesley (1992) 371–408
8. Matsuzaki, S., Suzuki, H., Osano, M.: Tierra instructions implemented using string rewriting rules. In: Proceedings of the Fifth International Conference on Humans and Computers (HC-2002). (2002) 167–172
9. Friedl, J.E.F.: Mastering Regular Expressions. O'Reilly (1997)