# Sharing and Refinement for Reusable Subroutines of Genetic Programming

Naohiro Hondo
Complex Systems Engineering
Division of Systems and Information
Engineering, Hokkaido University.
N-13 W-8, Sapporo,
Hokkaido 060, Japan
hondo@complex.hokudai.ac.jp

Hitoshi Iba
Machine Inference Section,
Electrotechnical Lab.
1-1-4 Umezono, Tukuba Science
City, Ibaraki, 305, Japan
Iba@etl.go.jp

Yukinori Kakazu
Complex Systems Engineering
Division of Systems and Information
Engineering, Hokkaido University.
N-13 W-8, Sapporo,
Hokkaido 060, Japan
kakazu@complex.hokudai.ac.jp

*Abstract* **This paper presents a new approach to Genetic Programming (i.e., GP). The aim of this study is to indicate an approach to make GP fit for practical use. The objective of our study originates in the fact that the program by human tends to be divided into some subroutines and to reuse the subroutines frequently. In a traditional GP, the program is structured by one sequence. Moreover there is no room to reuse the subroutines in a traditional GP. For sake of the division the program to some subroutines, there have been a few techniques proposed, which attempt to discover certain subroutines. However, the reusability of GP have never been discussed so far. In this paper, we propose an approach for reusability. The proposed method has a library for keeping some subroutines in order to share and reuse them. We make use of the Wall Following Problem to indicate the efficiency experimentally.**

## I. Introduction

This paper introduces a new approach to automatic discovery of functions for Genetic Programming (i.e., GP). The objective of our study is derived from the fact that a program is often decomposable into certain subroutines. Moreover, the subroutines is often reused at any point of the program. For a practical use, a program by GP should be divided into some subroutines. So our goal of this study is to generate an effective subroutine and reuse them.

In order to realize the above, we propose a new system based on an ADF (i.e., Automatic Defining function) [1]. GP system with ADF defines functions (or subroutines) automatically. The objective of the ADF is derived from the fact that a program is often decomposable into certain subroutines. An effective subroutine is expected to work as a building block for searching a solution effectively. As a result, ADF makes it possible to improve learning efficiency and reduce computational effort[2].

However, so far the reusability and robustness of GP have never been discussed using ADF. We aim at the acquisition of a generalized knowledge or subroutine and realizing the above improvements. A generalized knowledge refers to a rule which can be applied in various situations, giving the system flexible behavior.

In order to realize this facility, we introduce a new method based ADF. Our proposal extends an ADF so as to establish a library for acquiring effective subroutines with each generation. As the generations proceed, the subroutines are shared among the population, whereby any individual can use this generalized knowledge by referring to the library. Moreover, to apply to *generalization* operator, the knowledge is refined upon. As a result, through these shared subroutines, our proposal gives robustness to the system.

The rest of this paper is structured as follows. Section II describes the related work as to the automatic discovery of functions in GP. Section III presents our proposal method. Section IV describes some experimental results, by which the effectiveness of our proposal is shown. Section V discusses these results and the feasibility of our approach. Section VI denotes the related work. Section VII concludes.

## II. Automatic Discovery of Functions in GP

Traditional techniques for discovering functions automatically are mostly aimed at improving the efficiency of the GP search. They select and/or make a certain subroutine automatically, which can then be used as a useful building-block for searching a solution.

ADF [1] developed by Koza works with conventional GP, and is called ADF-GP here. ADF-GP offers alternative populations which work as subroutines. That is, ADF-GP contains material which turns into subroutines in advance, as another population. An individual in the main population can refer to an individual in the alternative populations as a subroutine, and the reproduction of the population applies crossover and mutation operations such those found in the main populations. Effective subroutines in the population are

0-7803-2902-3/96/$4.00©1996 IEEE


Authorized licensed use limited to: Wikipedia. Downloaded on June 22,2024 at 15:19:50 UTC from IEEE Xplore. Restrictions apply.

made by interaction with an object. ADF-GP gives a remarkable performance in limited domains needing subroutines. However, some problems remain unsolved: The large amount of computational effort needed and how to set up the number of the population.

## III. Sharing subroutines for an acquisition of knowledge

### A. Our Objective

The goal of this paper is the scaling up of GP in terms of reusability of acquired knowledge.

For the sake of improving the reusability, the system should have the follows. In the first place, the system must find a knowledge with high generality as a subroutine. For the sake of this, it needs the additional operation to generalize to refine the knowledge. Second, to make sure the reusability, it is necessary to preserve the knowledge. So as to realize it, we take into consideration of introduction of a library. The knowledge in the library is shared among a population. In other word, each individual can refer the knowledge through the library. Therefore, the library helps to reuse the subroutines generated by ADF. In addition, the system is able to cope with the changeable problem to arrange the subroutines or reuse them.

In this paper, from above considerations, we propose an approach to the automatic discovery of functions based on ADF with a library. In the next section, we denote the algorithm.

### B. Sharing ADF

Our proposal method, namely a Sharing ADFGP is structured in two main parts: The former part generates the main solutions and subroutines based on the ADF-GP. The latter part stores effective subroutines acquired by the ADF. In this paper, this latter part is called the library. The subroutines acquired by ADF are drawn into the library according to a rule. The subroutine in the library (called *SR* here) may be referred to by any individual in the main GP.

In the first place, we describe the description GP. The terminal set and function set of the GP is defined as follows:

$$T_{GP} = \{t_1, t_2, \ldots, t_n\}, \quad (1)$$

$$F_{GP} = \{f_1^{\arg 1}, f_2^{\arg 2}, \ldots, f_m^{\arg m}\}. \quad (2)$$

where m and n are the number of used terminals or functions and depend on the object. argm is the number of arguments of each function. Secondly, we describe the description ADF-GP. The function set of the ADF-GP takes the additional form of that of GP. That is,

$$F_{ADFGP} = \{f_1^{\arg 1}, \ldots, f_m^{\arg m}, adf_1^{\arg 1}, \ldots, adf_p^{\arg p}\} \quad (3)$$

where $p$ is the number of adfs. The role of these additional functions is to refer to these adfs. The terminal set is the same that for the GP.

$$T_{ADFGP} = \{t_1, t_2, \ldots, t_n\}. \quad (4)$$

In the adf population, the terminal set and function set are as follows.

$$T_{adf} = \{t_1, \ldots, t_n, \arg_1, \ldots, \arg_q\}, \quad (5)$$

$$F_{adf} = \{f_1^{\arg 1}, f_2^{\arg 2}, \ldots, f_m^{\arg m}\}. \quad (6)$$

Then the Sharing-ADFGP with the library is formulated as follows.

$$Sharing\ ADFGP := \left\{ \begin{array}{c|ccc|c} body_1 & adf_{11}^{\arg 1} & \cdots & adf_{p1}^{\arg p} & SR_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ body_M & adf_{1M}^{\arg 1} & \cdots & adf_{pM}^{\arg p} & SR_w \end{array} \right\} \quad (7)$$

Moreover the function set of the Sharing-ADFGP has additional functions to refer $SR_i$ in the library.

$$F_{Sharing-ADFGP} = \left\{ \begin{array}{l} f_1^{\arg 1}, \ldots, f_m^{\arg m}, \\ adf_1^{\arg 1}, \ldots, adf_p^{\arg p}, \\ SR_1^{\arg 1}, \ldots, SR_w^{\arg w} \end{array} \right\} \quad (8)$$

The terminal set of the Sharing-ADFGP is as follows.

$$T_{Sharing-ADFGP} = \{t_1, \ldots, t_n\}. \quad (9)$$

The subroutine the drawing operation to the library would be executed under the following condition.

$$if\ f_{\max}^{t} > f_{\max}^{t-1} \quad then\ drawing\ \text{into}\ library \quad (10)$$

where $f_{max}^{t}$ is the maximum fitness value in generation $t$. Then, the $SR_i$ in the library is discarded based on each $SR_i$ fitness. The $SR_i$ fitness is defined as following equation.

$$fitness_{SR_i} = \frac{\sum fitness_k}{ref_{SR_i}}. \quad (11)$$

where $fitness_k$ is the fitness value of individual $k$ which refers to $SR_i$ and $ref_{SR_i}$ is the total number of referred $SR_i$. The $SR_i$ with the lowest fitness may be discarded and replaced with a new subroutine acquired by ADFGP.

In the library, toward to $SR_i$ which has the salient number of $ref_{SR_i}$, a *generalization* operation may be implemented. The aim of generalization here is to own wholly stable mean for each subroutine so as to exclude any argument. Therefore, for some subroutine, the argument, *argi* may be changed to other terminals stochastically. The condition for whether or/not the operation executes is as follows.

$$if\ ref_{SR_i} = 0\ then\ operate\ to\ Generalization \quad (12)$$

Figure 1 shows a diagram of Sharing-ADFGP. The next section describes experimental results for the wall



Authorized licensed use limited to: Wikipedia. Downloaded on June 22,2024 at 15:19:50 UTC from IEEE Xplore. Restrictions apply.

following problem.

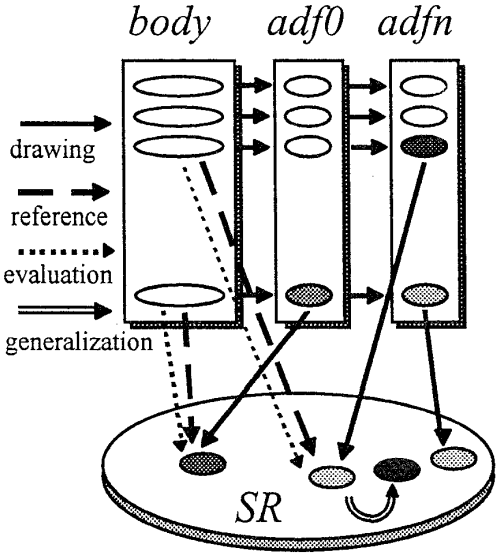


Figure 1: Diagram of Sharing-ADFGP.

## IV. Experiments

In this section, we experimentally examine our proposal method mentioned above. The main parameters are population size M = 700, number of generations G = 60, selection by tournament strategy. Other GP parameters have typical values as reported in [3]. We shows some experimental results of GP, ADF-GP and Sharing-ADFGP for fluctuating objects so as to verify the robustness and reusability.

The wall following problem is a typical robot navigation problem in GP [4]. The aim of this problem is to acquire a program which can navigate a robot to follow a closed wall within a map.

A problem space is represented by a field which is divided into grids and surrounded by wall. Some tiles are set along this wall. The robot can obtain a reward if it goes through a tile. However, there are some obstacles near the wall and the tile are set along those obstacles.

The robot can move within the space with no effect on the grid. The robot can select from the following actions: MoveForward, MoveBack, and rotating left or right at (LeftTurn, RightTurn). Moreover the robot has eight sensors(from 12 o'clock by counter clock, ) and can measure its distance from the wall. The robot consumes its energy (E) through its movement and sensing. The energy decreases E-1 units if it moves on a tile, E-2 units if it moves on the field and E-1 units if it senses. If E=0, the robot stops. The evaluation of the program which navigates the robot is given as the number of passed tiles (i.e. rewards).

The terminal set and function set of GP, ADF-GP, and the Sharing-ADFGP which are used in these experiments are as follows.

$$T_{GP} = \left\{ \begin{matrix} TurnRight, TurnLeft, MoveForward, MoveBack, \\ s0, s1, \ldots, s7 \end{matrix} \right\} \quad (13)$$

$$F_{GP} = \left\{ IFLTE_4, \Pr og2_2 \right\} \quad (14)$$

$$T_{ADF\text{-}GP} = \left\{ \begin{matrix} TurnRight, TurnLeft, \\ MoveForward, MoveBack, \\ s0, s1, \ldots, s7, \arg_0, \arg_1, \arg_2 \end{matrix} \right\} \quad (15)$$

$$F_{ADF\text{-}GP} = \left\{ IFLTE_4, \Pr og2_2, adf2_2, adf3_3 \right\} \quad (16)$$

$$T_{Sharing-ADFGP} = \left\{ \begin{matrix} TurnRight, TurnLeft, \\ MoveForward, MoveBack, \\ s0, s1, \ldots, s7, \arg_0, \arg_1, \arg_2 \end{matrix} \right\} \quad (17)$$

$$F_{Sharing-ADFGP} = \left\{ \begin{matrix} IFLTE_4, \Pr og2_2, adf2_2, adf3_3 \\ SR_1^{\arg 1}, \ldots, SR_w^{\arg w} \end{matrix} \right\} \quad (18)$$

The library size $W$, which is the peculiar parameter of the Sharing-ADFGP, is 7. The experiments make use of 2 different types of conditions:

**ex. 1**: The robot starts from a random point facing in a random initial direction as in Figure 2.

**ex. 2**: The robot learns 2 maps which change every 5 generations in Figure 3 and Figure 4.

In ex (1), as the action sequence must change at each generation, the robot needs a generalized strategy to follow. Moreover, in order to read the difference in the map, it needs to preserve some of the acquired knowledge. Figure 5 and Figure 6 present the improvements to the standardized fitness (best of generation) of ex. 1 and ex. 2 respectively. Figure 7 shows a typical trajectory which is derived from an acquired subroutine (ex. 1). Figure 8, Figure 9 and Figure 10 are the program generated by using ADFGP, GP and the Sharing-ADFGP respectively. Figure 11, 12 are a typical trajectory of the standard GP in ex. 2. Figure 13, 14 are a typical trajectory of Sharing-ADFGP in ex. 2. All the experimental results are calculated as over 10 trials.

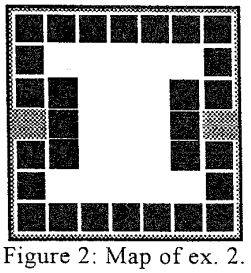

Figure 2: Map of ex. 2.



Authorized licensed use limited to: Wikipedia. Downloaded on June 22,2024 at 15:19:50 UTC from IEEE Xplore. Restrictions apply.

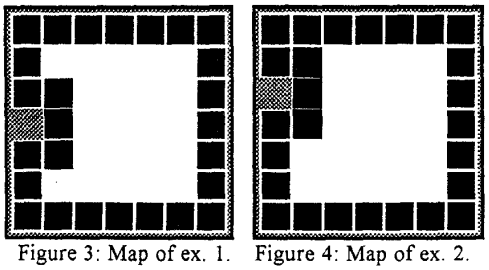

Figure 3: Map of ex. 1.



Figure 4: Map of ex. 2.

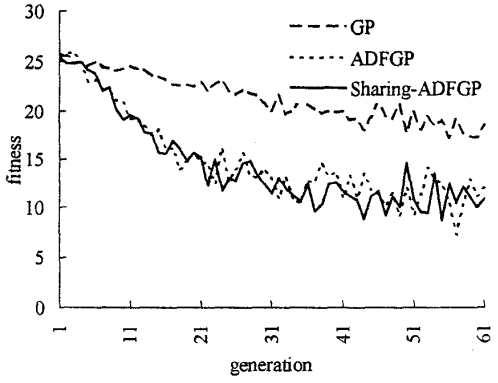


Figure 5: Improvement of fitness in ex. 1.

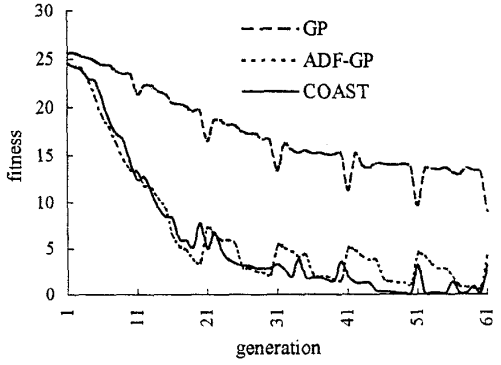


Figure 6: Improvement of fitness in ex. 2.

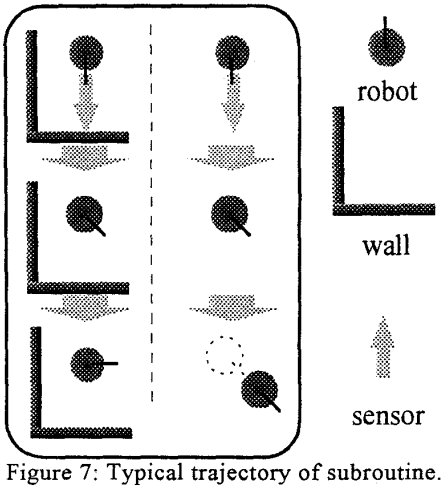


Figure 7: Typical trajectory of subroutine.

Body:
(IFLTE (IFLTE (ADF1 MoveBack S2 S6) (ADF1 TurnRight TurnRight MoveForward)(ADF1 S6 S0 (ADF1 MoveBack MoveForward S6)) (PROG2 (PROG2 S6 TurnLeft) (ADF0 MoveBackS 4))) (ADF1 (IFLTE (ADF1 MoveBack (ADF1 MoveBack MoveForward S6) S0) (ADF1 (ADF3 MoveBack MoveForward S6) S0 S2) (ADF1 MoveBack MoveForward S6) (PROG2 S4 TurnLeft)) TurnLeft MoveForward) (ADF1 S2 MoveForward S2) (ADF1 S4 S4 S7))
ADF0:
(IFLTE (IFLTE (IFLTE Arg0 (PROG2 S3 S3) S7 Arg1) Arg1 MoveForward S5) MoveForward (IFLTE TurnLeft S0 S5 S2) (IFLTE Arg1 MoveForward TurnRight S3))
ADF1:
(PROG2 (IFLTE (IFLTE TurnRight Arg0 MoveBack S5) (IFLTE MoveBack MoveBack TurnRight S2) (PROG2 MoveBack MoveBack) (IFLTE S2 (IFLTE S3 TurnRight MoveBack S2) (IFLTE S1 TurnRight TurnRight S2) TurnRight)) (IFLTE (IFLTE TurnRight MoveBack MoveBack S2) S2 (PROG2 MoveBack (IFLTE MoveBack Arg0 (IFLTE MoveBack Arg0 MoveBack Arg0) S3)) Arg2))

Figure 8: Typical program of ADF-GP in ex. 2.

(IFLTE (PROG2 (IFLTE (PROG2 MoveForward S6) (PROG2 MoveForward MoveForward) (IFLTE S6 TurnLeft S6 S5) (IFLTE S4 TurnLeft S6 S5) ) (IFLTE S4 TurnLeft (PROG2 MoveForward MoveForward) S3) ) (IFLTE (IFLTE MoveForward S7 MoveForward MoveForward) (IFLTE (PROG2 MoveForward TurnLeft) (IFLTE (PROG2 MoveForward S6) (PROG2 MoveForward MoveForward) (IFLTE S4 TurnLeft S6 S5) MoveForward) (IFLTE S3 MoveForward TurnRight TurnLeft) S7) TurnLeft MoveForward) (IFLTE S1 TurnLeft S6 MoveForward) (IFLTE (IFLTE (IFLTE (PROG2 MoveForward MoveForward) (IFLTE MoveForward S0 MoveForward MoveForward) (IFLTE MoveForward TurnLeft S5 MoveForward) (IFLTE TurnLeft MoveForward S6 S7) ) (IFLTE MoveForward S4 MoveForward MoveForward) (IFLTE TurnRight S5 S6 S5) (IFLTE S7(PROG2 MoveForward TurnLeft) (IFLTE MoveForward MoveForward MoveForward S4) (IFLTE S4 MoveForward MoveForward S4) ) ) (PROG2 (IFLTE (PROG2 MoveForward S6) (PROG2 MoveForward MoveForward) (PROG2 MoveForward TurnLeft) (PROG2 MoveForward TurnLeft) ) (IFLTE S6 TurnLeft S6 S3) ) (IFLTE (PROG2 MoveForward (PROG2 MoveForward TurnLeft) ) (PROG2 MoveForward MoveForward) (IFLTE S4 TurnLeft S6 (PROG2 MoveForward MoveForward) ) S6) (IFLTE S5(IFLTE (PROG2 MoveForward S6) (PROG2 MoveForward MoveForward) (IFLTE S4 TurnLeft S6 S5) (IFLTE S4 TurnLeft S6 S5) ) S5 (IFLTE S4 TurnLeft S6 S5))))

Figure 9: Typical program by GP in ex. 2.



Authorized licensed use limited to: Wikipedia. Downloaded on June 22,2024 at 15:19:50 UTC from IEEE Xplore. Restrictions apply.

Body:
(IFLTE (SR[5] S7 S0 S0) (ADF0 S2 S3 (SR[0] (SR[2] (SR[2] TurnLeft S4 S5) S4 MoveBack) (IFLTE S6 MoveBack S3 (IFLTE MoveBack MoveBack S3 S4)) (SR[5] MoveBack TurnLeft MoveForward))) S4 (SR[3] **((SR[4] S6 MoveForward MoveForward)** TurnLeft MoveBack))
ADF0:
(PROG2 (IFLTE (PROG2 (PROG2 (PROG2 Arg1 TurnLeft)(IFLTE Arg2 Arg2 Arg2 MoveForward)) (IFLTE Arg2 Arg2 Arg2 MoveForward)) (PROG2 (PROG2 Arg1 TurnLeft) (IFLTE Arg2 Arg2 Arg2 MoveForward)) Arg2 MoveForward) (IFLTE Arg2 Arg2 (PROG2 Arg2 (IFLTE Arg2 Arg2 Arg2 MoveForward)) MoveForward))
ADF1:
(IFLTE(IFLTE S2 MoveBack S0 TurnLeft) (IFLTE S3 (PROG2 S4 TurnRight) S1 TurnLeft) (IFLTE S0 S7 S1 MoveBack) (IFLTE 1 S0 S7 S1 MoveBack))
SR[0]:
(PROG2 (IFLTE (PROG2 (PROG2 (PROG2 Arg1 TurnLeft) Arg2) (IFLTE Arg2 Arg2 Arg2 MoveForward)) (PROG2 (PROG2 MoveForward TurnLeft) (IFLTE Arg2 Arg2 Arg2 S3)) Arg2 Arg2) (IFLTE Arg2 Arg2 Arg2 MoveForward))
SR[1]:
(IFLTE (IFLTE S0 S7 S1 TurnRight) (IFLTE S3 (PROG2 S4 TurnRight) S1 TurnLeft) (IFLTE S3 S7 S1 (IFLTE S1 S1 MoveBack S1)) (IFLTE S2 MoveBack (IFLTE S3 (IFLTE S3 S7 S2 S2) (PROG2 MoveBack TurnRight) (IFLTE MoveBack S2 S3 TurnLeft)) TurnLeft))
SR[2]:
(IFLTE S2 MoveBack (IFLTE (IFLTE S0 S7 MoveBack TurnLeft) (IFLTE S3 S0 S1 S3) (IFLTE TurnRight S7 S1 TurnRight) (IFLTE S3 MoveBack S1 TurnLeft)) TurnLeft)
SR[3]:
(PROG2 (PROG2 TurnLeft Arg2) (IFLTE S1 (PROG2 Arg1 Arg0) (PROG2 (PROG2 (PROG2 Arg0 Arg2) (PROG2 Arg1 Arg0)) (PROG2 Arg1 Arg0)) (PROG2 Arg2 (PROG2 (PROG2 Arg0 Arg1) S3))))
**SR[4]:**
**(IFLTE Arg0 TurnRight Arg1 TurnRight)**
SR[5]:
(IFLTE (IFLTE S0 S7 S1 TurnRight) (IFLTE S3 (PROG2 Arg1 TurnRight) S1 TurnLeft) TurnRight (IFLTE TurnLeft MoveBack S3 Arg1))
SR[6]:
(IFLTE (IFLTE S0 S7 S1 TurnRight) (IFLTE S3 (PROG2 S4 TurnRight) S1 TurnLeft)TurnRight (IFLTE TurnLeft MoveBack S3 TurnLeft))

Figure10: Typical program by Sharing-ADFGP in ex. 2.

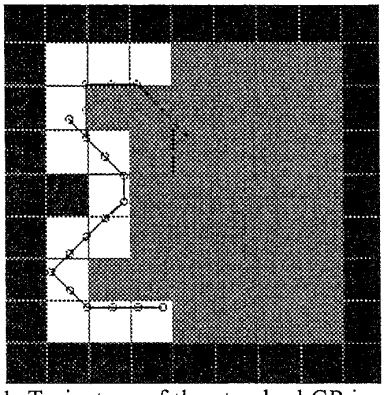

Figure11: Trajectory of the standard GP in ex.2 (1).

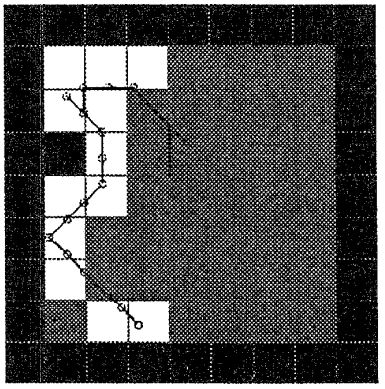

Figure 12: Trajectory of standard GP in ex.2 (2).

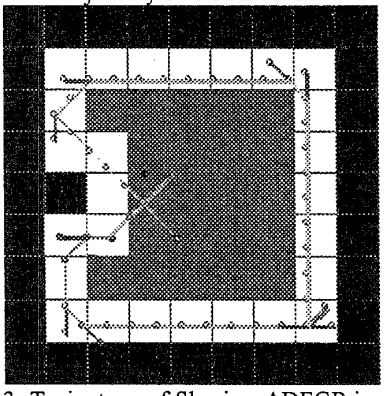

Figure 13: Trajectory of Sharing-ADFGP in ex.2 (1).

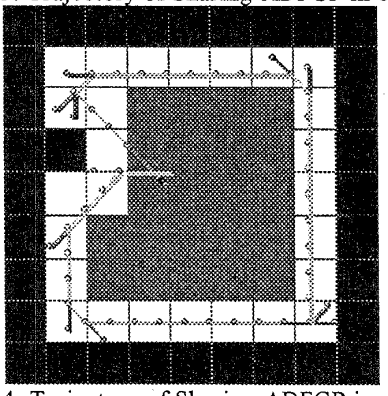

Figure 14: Trajectory of Sharing-ADFGP in ex.2 (2).

In Figure 10, the bold characters correspond to the trajectory in Figure 7.

## V. Discussion

The previous section was shown how the Sharing-ADFGP worked under various conditions.

In ex 1, the improvement of the Sharing-ADFGP is almost the same as that of ADF-GP. That is, because of the random stating point, the evaluation of each individual changes at each generation even if the same



Authorized licensed use limited to: Wikipedia. Downloaded on June 22,2024 at 15:19:50 UTC from IEEE Xplore. Restrictions apply.

program executed so the SR in the library is often replaced. Finally, the fitness of the individual referred to the SR becomes low, and as a result, the SR isn't referred and there isn't much difference between them.

In ex. 2, the Sharing-ADFGP shows remarkably excellent result. ADF-GP can't cope with multiple maps because it converges quickly. As a result, ADF-GP doesn't give robustness and adaptability. On the other hand, the Sharing-ADFGP indicates a stable behavior between two maps, not to mention that it reuses an acquired knowledge effectively.

The program generated by the Sharing-ADFGP employs some subroutines efficiently (see the body program in Figure 10). The body program refers to SR[2], SR[3], SR[4] and SR[5]. Some of these subroutines are generated by other individuals (or other ADFs) because the ADFs (ADF0 and ADF1) in the individual differ from these SR. Besides, SR[6] is generated by using generalization operator. So SR[6] doesn't have no argument.

In Figure 13, 14, the body tree in the program is very small because it refers four subroutines. The subroutines realize a characteristic trajectory shown in Figure 13, 14. That is, (1) the robot moves to the wall. (2) the robot perceives the distance to the wall with its sensors. (3) the robot retraces its steps and turns left. (4) the robot perceives the distance again. (5) the robot retraces its steps and turns left again. The above action sequence is repeated any number of times in Figure 13, 14., and can be utilized whenever the robot meets the wall. Thus, the robot can behave adaptable between two maps.

To conclude about the Sharing-ADFGP, our goal, namely, to pursuit the realization of the reusability in GP, was attained in our several experiments. The primary factor of this attainment exists in preserving of some effective subroutine which are acquired by ADF.

## VI. Related Work

### A. Module Acquisition

Module Acquisition (MA) [5] randomly selects a subtree from an individual and extracts a part of it as a *module*. Thus, the module is defined as a new function and is preserved in a library. This module can then be referred by other individuals, and works as a function with a fixed meaning. The main feature of MA is that it protects some effective subtrees against blind crossover operations. However, it is reported in [5] that MA could not improve learning efficiency or the acquisition of knowledge because of the random selection.

### B. Adaptive Representation GP

In Genetic Algorithms, schema theorem and building block hypothesis are regarded as the theoretical basis of learning. GP tends to seek the same basis. Rosca brought up a proposal where the schema in GA corresponded to a subtree in GP. Based on this assertion, he proposed an approach, named Adaptive Representation GP (AR-GP) [6]. AR-GP selects an effective subtree (i.e. a schema) based on an heuristic operation and adds that subtree as a new function to the function set. AR-GP can improve learning efficiency. However, the configuration of the evaluation function which works to select an effective subtree is not clear. And, moreover, it has been pointed out that this selection process isn't suited for general purposes.

## VII. Conclusion

This paper proposed the Sharing-ADFGP for scaling up of GP from the point of view of its reusability for generation of program for practical use. The effectiveness of our proposal was shown by the wall following problem experimentally. Especially, under the fluctuating environment, the Sharing-ADFGP showed stable performance because the library works efficiently to reuse an acquired knowledge.

## References


[1] Koza, J.: *"Genetic Programming II : Automatic Discovery of Reusable Subprograms"*, MIT Press, 1994.

[2] Koza, J.: *"Scaleable Learning in Genetic Programming using Automatic Function Definition: in Genetic Programming"*, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press.

[3] Koza, J.: *"Genetic Programming, On the Programming of Computers by Means of Natural Selection"*, MIT Press, 1992.

[4] Koza, J.: *"Evolution of subsumption using genetic programming"*, Proc. of the First European Conference on Artificial Life (ECAL 91), MIT Press, 1991.

[5] Kinnear, K.: Alternatives in Automatic Function Definition: *"A Comparison of Performance in Advances in Genetic Programming"*, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press.

[6] Justinian P. Rosca. and Dana H. Ballard.: *"Hierarchical Self-Organization in Genetic Programming"*, Machine Learning, Proc. 11th International Conference, p251-258, 1994.




Authorized licensed use limited to: Wikipedia. Downloaded on June 22,2024 at 15:19:50 UTC from IEEE Xplore. Restrictions apply.