Reactive Planning with Evolutionary Computation

Chaiwat Jassadapakorn and Prabhas Chongstitvatana
Intelligent System Laboratory, Department of Computer Engineering
Chulalongkorn University, Bangkok 10330, Thailand
E-mail: chaiwat.ja@student.chula.ac.th, prabhas@chula.ac.th

Abstract

This work proposes a method to generate robot plans by evolutionary computation. The main focus of the work is the representation of the plan. The reactive plan can be represented with a fixed length string that is suitable to be evolved by Genetic Algorithms. Two experiments are performed comparing the reactive plan with the ordinary plan: controlling a manipulator and the artificial ant problems. The results show that evolving reactive plans requires much less computational effort than ordinary plans.

Key-Words: Reactive planning, robot planning, evolutionary computation.

1. Introduction

Reactive planning [1,2,3,4] is a form of robot planning where the plan focuses on response to the environment. The main aim of this type of planning is to create systems that sense the environment and act in real-time. To achieve this goal reactive systems have strong coupling between sensing and action. The cycle sense-think-act is very short in a reactive system. Instead of a monolithic centralised control, a reactive system uses distributed and concurrent control. Many small sense-act control loops co-operate to achieve a global behavior. Brooks advocates this type of architecture for robot control, called "subsumption architecture" [5]. The layer of control composed of many goal-achieving units that work concurrently where the output from a unit can subsume the output of other units or altering the activities of other units. The composition of these units must be carefully designed to achieve the required behavior. Reactive systems have been used successfully in many robotic systems [6,7,8,9,10] including multi-agent robots or the robots that work in team such as [11].

Koza used Genetic Programming [12] to evolve programs to control robots to perform the desired task. Robot programs can be regarded as a "plan". The structure of plan is in the form of a tree where the internal nodes are the connectives such as `{if-and, if-or, if-not}` and the leaf nodes are the primitive commands, including sensing and action such as moving a joint or sensing the environment. This representation has variable length. Genetic programming has been used successfully to evolve robot plans for many tasks. One of the main concerns of using Genetic programming is that for a variable length representation the size of solutions (i.e. robot plans in our domain) grows quadratically [13]. Therefore, it requires considerable computational effort to evolve robot plans. Applying the concept of reactive planning we design the plan to be a set of if-then rules where the if-clause is the sensing command and the then-clause is the action command. The reactive plan then can be represented by a fixed length string. Genetic algorithm [14] is applicable to evolve such a plan. We compare two tasks of evolving robot programs using a tree representation of the plan and the fixed length string representing the reactive plan. In terms of computational effort, our reactive plan is much more efficient. The sections that follow describe the representation of the reactive plan and the details of the experiments and discuss the results.

2. Reactive plan

An ordinary robot plan generated by Genetic programming has all the actions in the leaf nodes. Sensing commands are used to affect the flow of the plan via the connectives, such as "if the arm *hits* obstacles then move the shoulder joint else perform the branch X". The sensing command "hits" causes the flow of the plan to execute the move shoulder joint or to execute the branch X. The state (when some memory is required in the task) is representing by the path of the plan. This type of representation causes a plan to be variable length.

A reactive plan is a set of if-then rules. The if-clause contains all sensing commands therefore it captures all possible sensing situations that can occur. The then-clause contains the action commands. If a state is required, it can be represented by an integer and is included in the then-clause. This number also appears in the if-clause and the if-clause becomes the combination of

the state number and possible sensing situations. The interpretation of such state number is as follows "perform the action in the action command and go to the state specified by the state number". This interpretation regards a reactive plan as a finite state machine when the state is required and as a combinational circuit when no state is required. The if-clause does not have to be explicitly represented, it becomes the index into the table of then-clause.

An example of a reactive plan is shown in Figure 1. There are 3 states in this example. The input is one bit. The action set is `{ move, right, left }`. The if-clause is consisted of state and input. The then-clause is consisted of action and next state. We encode the reactive plan as individual with two chromosomes as shown.

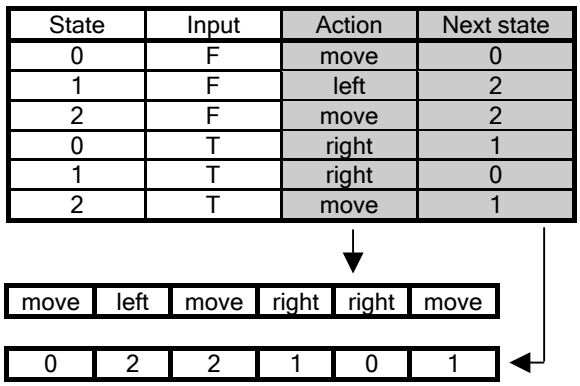

Figure 1. Individual encoding of an example of reactive plan

To compare the computational effort to generate a reactive plan with an ordinary plan generate by Genetic programming, two problems are used. The first problem is controlling a manipulator [15] and the second problem is an artificial ant [16]. The metric we used for comparison is the computational effort as defined in Koza [12].

3. Controlling a manipulator

The problem of controlling a manipulator to reach a target is taken from [15,17,18]. GP is used to generate robot programs to control a robot manipulator to reach a target while avoiding obstacles. The robot system composed of a 3 DOF manipulator and a vision system mounted above and overlooking the whole workspace. The vision system monitors the positions of various objects: the manipulator, the obstacles and the target. The manipulator is a 3 links planar arm moving on a plane. Movement of each joint is limited. The environment for the experiment is shown in the figure below: (solid blocks are obstacles and a cross is the target)

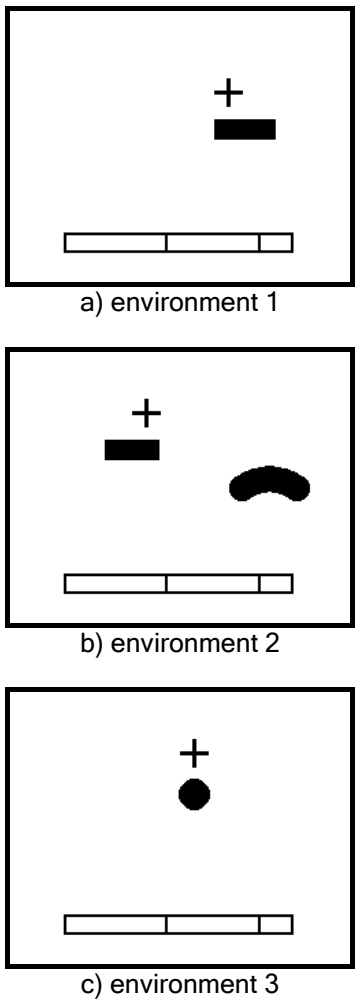

Figure 2. Three environments used in the problem of controlling a manipulator

Each robot plan composed of symbols from the function set and the terminal set. The function set is `{ IF-AND, IF-OR, IF-NOT }` and the terminal set is `{ s+, s-, e+, e-, w+, w-, HIT?, SEE?, INC?, DEC?, OUT? }` The terminal `s+` (shoulder) moves the shoulder motor clockwise 1 step (5 degrees) and `s-` moves the shoulder motor anticlockwise 1 step. The similar meaning applies for `e+`, `e-` (elbow) and `w+`, `w-` (wrist). All of these functions always return true. The terminal `HIT?` checks whether each link of the robot arm hits the obstacle. The terminal `SEE?` checks whether the path from the fingertip to the goal has any obstacle. The terminal `INC?` checks whether the distance between the fingertip and the goal is increasing. The terminal `DEC?` checks the opposite. The terminal `OUT?` checks if each joint of the robot arm moves out of bound. The bound is defined to prevent the arm from going out of the view of

Table 1. The reactive plan for controlling a manipulator

Hit?	See?	Inc?	Dec?	Out?	Shoulder	Elbow	Wrist
T	T	T	T	T	+/0/-	+/0/-	+/0/-
T	T	T	T	F	+/0/-	+/0/-	+/0/-
T	T	T	F	T	+/0/-	+/0/-	+/0/-
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
F	F	F	F	F	+/0/-	+/0/-	+/0/-

the camera. The function `IF-AND` is a four-argument comparative branching operator that executes its third argument if its first argument and its second argument are true, or otherwise, executes the fourth argument. The function `IF-OR` is a four-argument operator that executes its third argument if its first argument or its second argument is true, or otherwise, executes the fourth argument. The function `IF-NOT` is a three-argument operator that executes its second argument if the negation of its first argument is true, or otherwise, executes the third argument

Since the move command contained in any robot program is altered when the situation change. The environment is monitored by five binary sensing flags `{ HIT?, SEE?, INC?, DEC?, OUT? }`. Without using any state the reactive plan has $2^5 = 32$ entries and the then-clause consisted of commands to move shoulder, elbow and wrist joints. This combination circuit is shown as Table 1.

Table 2. Parameters of the evolutionary process

Parameter	Ordinary plan	Reactive plan
population	400 plans	400 plans
size of an individual	80 symbols (initial)	96 symbols (3 chromosomes with 32 symbols)
maximum generation	10	10
number of repeated run	1000 runs	1000 runs
elitism	40 plans	40 plans
crossover	160 plans	360 plans
mutation	200 plans (addition 100 plans and extension 100 plans)	mutation rate = 0.0333 per gene

Each joint command is either: move the joint motor clockwise 1 step (+), move the joint motor anticlockwise 1 step (-) or no action (0). The genetic algorithms (GA) is used to find the solution of this problem. The action command in the reactive plan is encoded as three chromosomes. Table 2 shows the parameters used for evolving the ordinary plan by GP and the reactive plan by GA. Table 3 shows the comparison of computational effort for evolving both types of plan. The performance of the reactive planning evolved by GA is much better for every environment.

Table 3. The computational effort in evolving the ordinary plan and the reactive plan in each environment

Environment	Ordinary plan (from [18])	Reactive plan
1	14,400	800
2	18,000	2,000
3	220,800	18,400

4. Artificial ant

The problem of artificial ant used in our experiment is "Santa Fe trail" [19]. This problem is frequently referred to and used as a standard problem for comparing various learning techniques. The goal of Santa Fe trail problem is to train the ant to eat all food contained in the square 32 x 32 grid with a limitation amount of time (400 actions in our experiment). An ant has a sensor that can senses the food in next grid ahead. The ant has three actions: turn right, turn left, and move. The environment of the Santa Fe trail is shown in Figure 3.

From the work [12], GP is used to evolve the solution for the Santa Fe trail problem. The function set is `{ IF-FOOD-AHEAD, PROGN2, PROGN3 }` and the terminal set is `{ move, right, left }`. The terminal `move` is a command to move an ant to the next grid ahead. The terminal `right` turns the ant 90 degrees clockwise and the terminal `left` turns 90 degrees anticlockwise. The function `IF-FOOD-AHEAD` is a function with two arguments. It will process the first argument if the next grid ahead has food, and will process the second argument if not. The function `PROGN2` is a function with two arguments. It will process the first argument and the second argument in order. The function `PROG3N` is the same as the function `PROGN2` except it has three arguments.

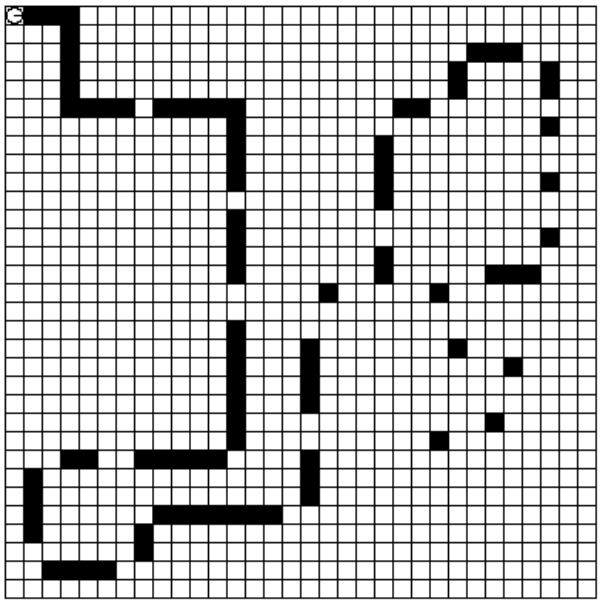

Figure 3. The Santa Fe trail problem

The reactive plan for this problem required state variables. It is known that five states are necessary to perform this task. The reactive plan is shown in Table 4. The number of states used in the plan must be greater than the minimum, we set it to eight. Therefore, the size of table is 16 entries. GA can be used to evolve this plan by encoding the action and next state into two chromosomes. Two experiments are performed, one with bounded perimeter where the ant stops at the perimeter and the other one with wraparound perimeter where the ant can "wrap around" from one side of the board to the other side. Table 5 shows the parameters used for the experiment and Table 6 shows the comparison of the computational effort. Again, the performance of the reactive planning evolved by GA is much better in both experiments.

Table 4. The reactive plan for the Santa Fe trail problem

State	Food ahead?	Action	Next state
0	F	move/left/right	0-7
1	F	move/left/right	0-7
. . .	. . .	. . .	. . .
7	F	move/left/right	0-7
0	T	move/left/right	0-7
. . .	. . .	. . .	. . .
6	T	move/left/right	0-7
7	T	move/left/right	0-7

Table 5. Parameters of the experiments

Parameter	Ordinary plan	Reactive plan
population	500 plans	500 plans
size of an individual	20(initial) - 80(max) symbols	32 symbols (2 chromosomes with 16 symbols)
maximum generation	51	51
number of repeated run	1000 runs	1000 runs
elitism	50 plans	50 plans
crossover	450 plans	450 plans

Table 6. The computational effort of both methods

Type of area	Ordinary plan	Reactive plan
wraparound	1,045,500	130,000
bound	867,000	270,000

5. Discussion and Conclusion

In both problems, controlling a manipulator and the artificial ant, we found that evolving the reactive plan is much easier than evolving the ordinary plan. This fact leads to the observation that the representation of reactive plan may be the factor in reduction of computation effort. Why the representation of an ordinary plan by a tree structure required more computational effort than the representation of a reactive plan by a fixed length string? We hypothesise that:

Firstly, for the tree representation, the effort in evolution must be spent on evolving not only the suitable terminal set, but also the suitable function set. Unlike the fixed length string that only terminal set is evolved. The searching for solutions in the ordinary plan has more work than the reactive plan. Moreover, for the problem of controlling a manipulator, the fixed length string representation has an advantage over the tree representation that the number of terminal is smaller because all sensing terminals are not included. The if-clause is represented implicitly.

Secondly, the solution encoded with the tree representation suffers from the inaccessible path. This path originates by some conflict of condition in the tree. Whereas for the reactive plan all clauses are in normal form.

Lastly, the crossover operator of the tree representation is not as efficient as the crossover operator of the fixed length string. When the crossover takes place at inaccessible or unused paths, the behavior of solution does not changed. In contrary, the crossover operator of the fixed length string does alter the solution almost every time.

It does not mean that the fixed length string is always more efficient than the tree representation. Since in the

experiment, the reactive plan is designed carefully. The size of table is chosen to be near optimal so that the search process works well. Table 7 shows the result of the Santa Fe trail problem with different number of states, which affect to the size of table. It can be seen that choosing fitter or looser table will degrade the performance.

In general, the representation of solution is dependent on the problem. Choosing a suitable representation leads to good performance. This work shows some success of the conversion of representation of an ordinary plan to the reactive plan which improve the performance. We believe that many problems in robot planning will have similar behavior.

Table 7. The computational effort with different value of parameter n for the Santa Fe trail problem

Methods	Wraparound area	Bound area
Ordinary plan	1,045,500	867,000
Reactive plan 5 states	172,500	340,000
Reactive plan 8 states	130,000	270,000
Reactive plan 16 states	700,000	1,872,000
Reactive plan 32 states	7,544,000	50,633,000

Acknowledgement

The idea of converting an ordinary plan to a reactive plan is originally suggested by Yodthong Rodkaew. The first author would like to acknowledge the support of the Royal Golden Jubilee Ph.D. Graduates program by Thailand Research Fund organization.

References

[1] Chapman, D. 1987. Planning for conjunctive goals. Artificial Intelligence 32:333-377.

[2] Agre, P. and Chapman, D. 1990, What are plan for? Robotics and Autonomous Systems 6:17-34.

[3] Maes, P. 1990. Situated agents can have goals. Robotics and Autonomous Systems 6:49-70.

[4] Nilsson, N. 1994. Teleo-reactive programs for agent control. JAIR 1:139-158.

[5] Brooks, R. 1987. A robust layered control system for a mobile robot. IEEE J. of Robotics and Automation 2:14-27.

[6] Schoppers, M. 1987. Universal Plans for Reactive Robots in Unpredictable Environments. Proc. of 10th IJCAI.

[7] Dorigo, M. and Colombetti, M. 1994. Robot shaping: developing autonomous agents through learning. Artificial Intelligence 71(2):321-370.

[8] Schaal, S. and Atkeson, C.G. Robot Juggling: Implementation of Memory-Based Learning, IEEE Control Systems, vol. 14, no. 1 (Feb. 1994) 57-71.

[9] Arkin, R. 1995. Reactive robotic systems, in Arbib, M. ed. The handbook of brain theory and neural networks, pp.793-796, MIT Press.

[10] Mataric, M. Williamson, M. Demiris, J. and Mohan, A. 1998. Behavior-Based Primitives for Articulated Control. Proc of 5th Int. Conf. Soc. for Adaptive Behavior, MIT Press, pp.165-170.

[11] Balch, T. Boone, G. Collins, T. Forbes, H. MacKenzie, D. and Satamaria, C. 1995. Io, Ganymede and Callisto -- a multiagent robot trash-collecting team, AI Magazine 16(2):39-51.

[12] Koza, J. 1994. Genetic Programming, MIT Press.

[13] Langdon, W. 1998. The Evolution of Size in Variable Length Representation, IEEE International Conference on Evolutionary Computation, pp. 633-638.

[14] Holland, J. 1975. Adaptation in Natural and Artificial System, Ann Arbor, Michigan : University of Michigan Press.

[15] Chongstitvatana, P. and Polvichai, J. 1996. Learning a Visual Task by Genetic Programming, Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and System, Osaka, Japan, pp. 534-540.

[16] Jefferson et al. 1991. Evolution as a theme in artificial life, in Langton, C. et al ed. Artificial Life II, Addison Wesley.

[17] Polvichai, J. 1996. Robot Learning by Genetic Programming, Master Thesis, Department of Computer Engineering, Chulalongkorn University (In Thai)

[18] Jassadapakorn, C. 1997. Reduction of Computational Effort in Genetic Programming Learning Method, Master Thesis, Department of Computer Engineering, Chulalongkorn University. (In Thai)

[19] Langton, C. et al ed. 1991. Artificial Life II, Addison Wesley.