Production Rules As Chromosomes of GA for Robotic Swarm Applications

Kai Wing TANG and **Ray A. JARVIS**
ECSE, Monash University
Kai.Tang@eng.monash.edu.au and ray.jarvis@eng.monash.edu.au

Abstract

Fairly complicated data structures, other than a string of binary digits, as chromosome representations of genetic algorithms (GAs) have been tried, and published, by various researchers. How to choose an appropriate representation for a GA problem is not only necessary, but also a critical step. Once the genome [1] is not a string of binary digits, the recombination operators have to be re-defined. However, the merits gained through the suitability of genome representation, like faster convergence, easier effect interpretation etc. outweigh the extra effort required.
This paper presents results about an experiment on utilization of production rules as chromosomes of a GA problem on a common multiple robots cooperative task: environment exploration. It provides details of implementation and gives a narration about the new devised recombination operators like crossover and mutation.

1 Introduction

The success of application of GAs relies on two factors: an appropriate chromosome representation and a correct fitness evaluation function. Between these two factors, a good former one is especially evasive. This is due to the fact that researchers always use GAs on problems that cannot be completely modeled or problems that are not fully understood. If the problem is fairly easily comprehended, other types of approaches like dynamic programming, heuristic reasoning and hill-climbing etc. will outperform GAs. Hence a paradox exists: If the problem is not familiar enough, a pertinent chromosome representation will become very hard to realize.

For the experiment being reported here, GAs are not just used as a function optimizer to get some above average results. They are also used to create something, hopefully, that is compatible with human design. Based on this pre-requisite, the outcomes from this experiment, if they are really good enough, have to be interpreted or explained. Accordingly, a representation with a cause-effect relation will be desirable.

A production rule in the format of:

$$if[condition_1]and...and[condition_n]$$
$$then$$
$$execute[action_i]$$

will be the right choice of representation to fulfill the requirement of explainability.

Once the data structure of genome is confirmed as set(s) of production rules, some other related hurdles have to be surmounted, namely:

- What is the form of genome crossover operation?
- How is mutation carried out? Should we mutate the action of a rule, or mutate actions of a set of rules?
- How should we allocate the fitness value, or reward, to an individual rule?

In the following sections of this paper, our solutions to the above questions will be described in detail.

Including this introduction, this paper consists of seven sections.

The second section outlines the scope of the multiple robots environment exploration problem. The third section narrates the conditions / actions of the rule sets, parameters setting of GA, and recombination operators. The fourth and fifth sections are the simulation results and analysis of these results. The sixth section is a discussion about differences between the approach of this paper and learning classifer systems. The seventh section is the conclusion.

[1] chromosome / genome are interchangable in this context

2 Scope of Environment Exploration Problem

Environment exploration is one of the canonical applications used for testing the effectiveness of multiple robots' performance. It is such that:

A number of robots are placed in an unknown environment. Each robot has a range finder of limited effective range so that each can detect the existence of obstacles or other robots within a close proximity. Once a robot realizes it is too close to other robot(s), it will stop. As a result, all robots in the congested zone stop at once. Then randomly one of them will re-plan a new path and leave the zone. A short while later, another robot follows the same process. Hence, They disperse one by one away from the congested zone and a collision is avoided.

The task is for all robots working collectively to plot all the unknown environment into a known map by moving around and to complete this task within the shortest period of time.

Since the topic of this experiment is to study the possibility of using GAs to design individual behaviours that can accomplish certain collective goals and GAs normally require hundreds of generations to evolve out certain behaviours, computer simulations instead of physical robots are used in the experiment. The plan is such that, by exercising simulations in the beginning, if the simulated results are satisfactory, simple physical robots will later be built to verify the validity of these simulations.

All the simulations are run under a simplified environmental representation. This environment represents the real world by a grid of rectangular cells (50 by 50 cells).

Each rectangular cell represents one of the following classes:

- Space
- Obstacle
- Robot

In each robot's environmental map, the state of each cell is either:

- known, or
- unknown

The size of an obstacle is grown before mapping into the grid world, as shown in Figure 1 and hence a robot can pass through diagonally placed adjacent obstacle cells.

In this experiment, we take a de-centralized approach: there is no leader - follower relation among the robots in the group. Accordingly, every robot plans its own action independently. The objective of this experiment is to investigate emergence of cooperative behaviour. If

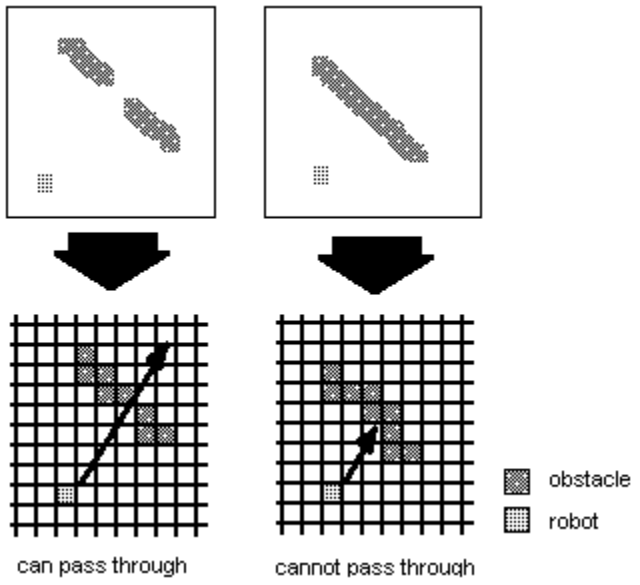

Figure 1: A robot can pass through diagonally placed adjacent obstacles.

a robot does not know the locations of the areas been explored by other robot(s), no cooperation can materialize. Consequently, a limited sharing of environmental information is necessary.

Our setting is such that a robot always broadcasts its coordinates to the others. Furthermore, once two robots are close, and visible, to each other (no obstacle in between), they will exchange their explored environmental map. The visibility requirement is included because the exchange of environmental maps may be performed via infra-red signals when physical robots are built.

Besides sharing of each other's location, a robot also transmits the count of unknown cells in its own environmental map to other robots. The purpose of sharing of this piece of information will be explained in the next section.

We can see that by sharing a limited amount of information, the problem becomes rather interesting. During the path planning process, every member of the robotic team has to make a choice between moving to another robot to obtain a more accurate map, or, executing the exploration task. Furthermore, since a robot only knows where the other members are, but not what they are doing; sometimes, even though a simple 'move to nearest team member' operation may not be easily accomplished. (The nearest team member may be moving away from the original robot. As both robots move with same speed, the original one cannot catch up the one which is moving away).

With such a problem, we, the ones who setup this

experiment, do not know what good results will look like in advance. Consequently, we pass the duty of individual action design to GAs, and try to evaluate how effective GAs will be.

2.1 Actions / Conditions of Rule Sets

The behaviours of each member of the robotic team are governed by two sets of rules named as:

- Revision-Rule
- Next-Action-Rule

The Revision-Rule Set decides when to change the action under execution whereas the Next-Action-Rule Set resolves what form of the upcoming new action will be.

The moment to revise the current action or re-plan a new path is a very critical factor that influences the final result of the collective operation. It is because the data available to each robot at any moment are (1) locations of other robots and (2) its own version of the environmental map. As a result, a new path depends on the environment data. If the elapsed time between two path planning operations is short, the robot reads environment data, and reacts, more frequently. Thus, the robot has more interactions with the environment. Different degrees of interactivity with the environment will cause various emergent results [Holland , 1998] and good emergent results rely on appropriate amount of of interactivity. Various moments to change action bring about different amount of interactivity, so that right moment to change action becomes a necessary criterion for good collective performance.

These two sets of rules, aided by a handful of built-in, exceptional conditions handling rules, form the central nervous system of the robotic team member. The following two subsections describe the available values of actions, whereas another next two about conditions, of these two sets of rules.

Actions of Revision-Rule Set

The action portion of the Revision-Rule Set is represented by a 'true/false' switch to control whether keep on running the current action or stop and switch to a new one.

Actions of Next-Action-Rule Set

The action portion of the Next-Action-Rule Set is a 5-valued indicator that specifies five different behaviours. They are:

1. Move to Nearest Unknown Cell
   Based on its own environmental map, use distance transform [Jarvis , 1984] to identify the nearest unknown cell and hence plan a path towards this cell.
2. Move to Other Robot
   Pick up one robot in the team and then move to it. The method to decide which one to pick is as follows:
   Based on the environmental map, count the number of unknown cells within a certain boundary around the robot. Figure 2 shows an example : It is the environmental map of the robot without a label, and the boundary around other robots for consideration is 7 by 7 cells. It illustrates that unknown count around robot C is the highest. The magnitude of this count shows how deep the robot is in unknown district. This count gives a rough estimation about the volume of information, which is unknown to itself, but already explored by that team member.

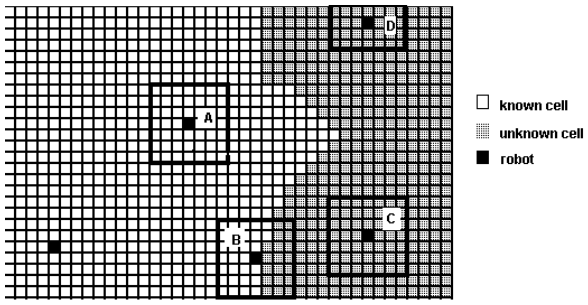

Figure 2: Count of Unknowns for robot A = 0, B = 19, C = 48, D = 27; so that C is the preferred one.

   Presumably, a higher count means the higher the amount of information to be gained by an approach.
3. To Spread
   Use distance transform to get the whereabouts of the unknown cell that is farthest from *ALL* other robots. Use this cell as destination to plan a path.
4. Move to Centroid
   Move to the centroid of the robotic team (include itself). This is an interesting action: If all robots take this action at the same time, it is a collective gathering operation. If only some of the robots take this action, they will merge and then follow an invisible, moving point. If only one robot takes this action, its effort may be rendered totally non-productive.
5. To Stop
   Remains stationary for four units of time. (A time unit is defined as the average time for a robot to move to its neighboring cell).

Conditions of Revision-Rule Set

The condition portion of the Revision-Rule Set consists of four different conditions:

1. Count of Own Unknown Cells
   This is the count of unknown cells in a robot's own

environmental map. This count is quantized to a grade between 0 to 7, i.e. a total of eight values. For the simulation, the whole environment is represented by 50 * 50 = 2500 cells. Seven numbers, n_1 to n_7, each is greater than 0 and less than 2500, with n_k less than n_{k+1} are assigned. Also, an eighth number, $n_0 = 0$ is used as an initial value. Grade of this unknown cell count will be equal to k if the unknown cells count is in the range:

$$n_{k+1} \geq Unknown.Cell.Count > n_k$$

Suitable sizes of ranges in between n_k and n_{k+1} (k = {0..6}) are parameters for GA to search.

2. Count of Group Unknown Cells
   This is the summation of counts of unknown cells of *ALL* robots' environmental maps. This is *not* the count of cells unknown to either, or all of the team members. The ratio of this count to Own Unknown Count is a rough indicator of how ignorant a robot is compared with the whole team. This count is also quantized to a grade between 0 and 7. (Proportionally, the values n_k are in the range between 0 and 2500 * number of robots).
3. Number of Steps Taken So Far
   This is the count of steps taken since last path planning. The value is in a range of {0..3}. Each value is coded to represent four steps. Therefore, if step count is less than or equal to four, this value is 0. If step count is greater than twelve, its value will be 3.
4. Current Action
   This is the type of action being taken. The values are as described in subsection Actions of Next-Action-Rule Set.

Thus, the rule [1][3][2][3] :: [0] means :

If
  unknown count of own map is grade 1;
  unknown count of total maps is grade 3;
  the current path has been take 5 to 8 steps;
  the current action is "to spread";
then
  continue current action.

By utilizing this set of rules, a robot can decide whether to re-initial a new action or not, given the information of :

1. its current action
2. its ignorance compared to that of the whole team and
3. how long since the current action been taken

Conditions of Next-Action-Rule Set
The condition portion of the Next-Action-Rule Set also consists of four different conditions:

1. Count of Own Unknown Cells
   Same as that of the Revision-Rule Set.
2. Count of Group Unknown Cells
   Same as that of the Revision-Rule Set.
3. If It is End of Path
   It is a true / false switch indicates when this rule is triggered, whether it has reached end of current path.
4. Current Action
   Same as that of Revision-Rule Set.

Thus, the rule [7][1][1][2] :: [4] means :

If
  unknown count of own map is grade 7;
  unknown count of total maps is grade 1;
  it has reached end of the path;
  the current action is "to other robot";
then
  move to the centroid.

By utilizing this set of rules, a robot can decide what the next action will be, given the information of :

1. its current action
2. its ignorance compared with that of the whole team and
3. whether the current action been completed by reaching end of the path

2.2 Exception Handling

This subsection explains the operation of the exception handling rules.

Normally, the Next-Action-Rule Set is triggered after firing of a Revision-Rule whose action is 'true'. There are situations when the Next-Action-Rule should be fired without prior firing of a Revision-Rule. The following paragraph describes these situations.

Sometimes, the current action will be stopped immediately and a new action, 'to nearest unknown cell', be taken. The second paragraph describes its reason.

Firing of Next-Action-Rule Alone Any one of the situations will trigger the firing of Next-Action-Rule immediately.

- The path is blocked by an obstacle cell
- The path is blocked by another robot
- The robot is at the end of current path

A Default New Action A robot always keeps a record of its previous sixteen steps (excluding stops). If the repetitive rate of these sixteen steps are greater than four, this triggers a 'loop-trapped' alert, which in turn forces the robot to take the default action of 'to nearest unknown cell'.

3 Implementation of Simple Genetic Algorithm (SGA)

3.1 GA Related Parameters

- Generation population: 100.
- Format of chromosome:
  The rule sets as described in previous section, with a full combination of values of conditions are used as the chromosome. (Figure 3 is an illustration) Take the Revision-Rule Set as an example, there are four conditions with a maximum of 8, 8, 4 and 5 values, respectively. Then the total count of Revision-Rules is 8 * 8 * 4 * 5 = 1280.

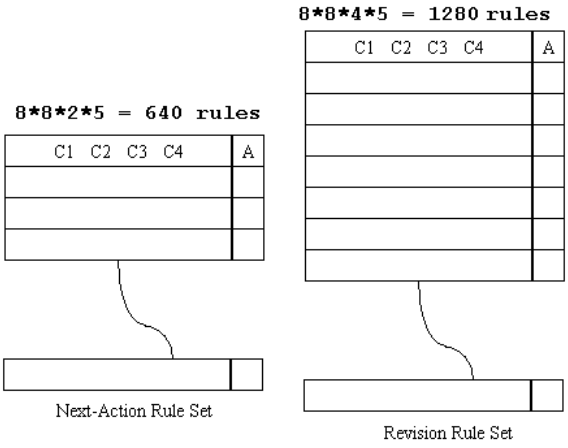

Figure 3: Two sets of rules form the chromosome.

- Instances of first generation :
  The action part of each rule is randomly assigned a valid value.
- Selection : Baker's Standard Universal Sampling algorithm [Mitchell , 1996].
- Elitism : During the selection, the best instance of previous generation is always copied to the new generation.
- Crossover : 60%.
- Mutation : 4%.
- Fitness Function : 1000 - time units spent for the whole exploration.

3.2 Recombination Operators

For the topics about cooperations of multiple robots, there is a special feature which makes short-term reward assignment very difficult to accomplish. With regard to cooperation, it is very hard to distinguish, or, to isolate sub-tasks from the whole task. Based on the total time consumed, the performance of the whole task can be assessed quantitatively. However, which portion of the task or which member of the team contributes most cannot be evaluated.

Using our experiment as an example, one of the members may explore just very few cells. However, its role as a map relaying agent may substantially improve the overall group performance.

Similarly, to discover which rule(s) among a whole set of rules could be the main contributor(s) to a good collective result, is very hard.

In the same way, a rule fired alone may not be beneficial at all. But, if it is fired within an episode of rules it may be functional as a bootstrapper and hence generates excellent results.

It is reasonable to think that certain combinations of rules fired within a whole exploration task will generate better results than the other combinations. (It is not necessary for them to be fired in sequence, or, be executed by the same robot).

To sum up, we can assign a score, or fitness value, to a bunch of rules (those fired during a run of exploration) but not to an individual rule. In such a manner, the credit assignment methods applied in learning classifer systems (LCS) are not feasible for robotic swarm experiment.

However, if we regard subset(s) of rules as building blocks [Holland , 1960] [Holland , 1995] in Holland's "Building Block Thesis", with some minor modifications, the SGA [Goldberg , 1989] can readily be applied.

Crossover Operator of Rule Sets

The sole function of a crossover operation is to pick out a good subset of genes from each parent and then try to combine them to generate an offspring better than each of the parents.

Now, because the chromosome and building block are a full set and subset of rules, respectively; the basic operation is to seek out a good subset of rules from the parents' full rule sets.

The steps are :

- Randomly pick out a rule which has at least been fired once during the previous exploration, from either of the parents going to be crossed over.
- Use conditions of this rule to generate a subset selection criterion.

1. Let the total count of conditions be n, then let the conditions of the chosen rule be C_1 , C_2 ... C_n
2. Generate a random number, m, in between the range $\{1..2^n - 1\}$. If treating m as a binary number, m will be n bits long, with p randomly placed bit-1's (p in the range of {1..n}).
3. Use m as a mask by regarding bit-1's as requisite, bit-0's as don't care; and matching the least significant bit with C_n, the most significant bit with C_1, etc. to generate a selection criterion.

This is illustrated by taking the Revision-Rule as an example, without loss of generality. Here, n = 4; consequently, conditions of picked rule be C_1 , C_2, C_3, C_4 and, m will be in the range of {1..15}.
If the randomly generated number, m, is 5, or, 0101 in binary, the criterion will be

$$Condition_2 = C_2$$

and

$$Condition_4 = C_4$$

If the randomly generated number, m, is 13, or, 1101 in binary, the criterion will be

$$Condition_1 = C_1$$

and

$$Condition_2 = C_2$$

and

$$Condition_4 = C_4$$

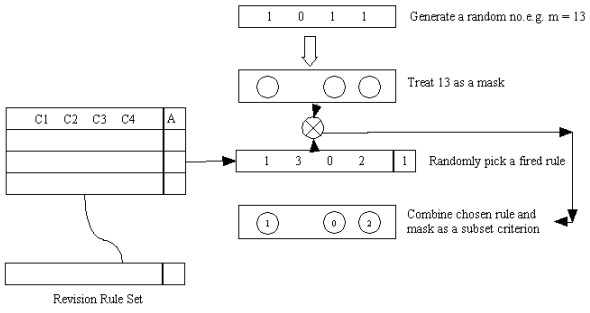

Figure 4: Formation of a subset criterion.

By using this method, a subset of rules, with the previous chosen rule among them, can be selected (figure 4). The size of this subset depends on number of bit-1's in m, or, p. If p = 4, the subset only consists of 1 rule. If p = 1, dependent on p's location, the size of subset varies from 160 to 320 (figure 5).

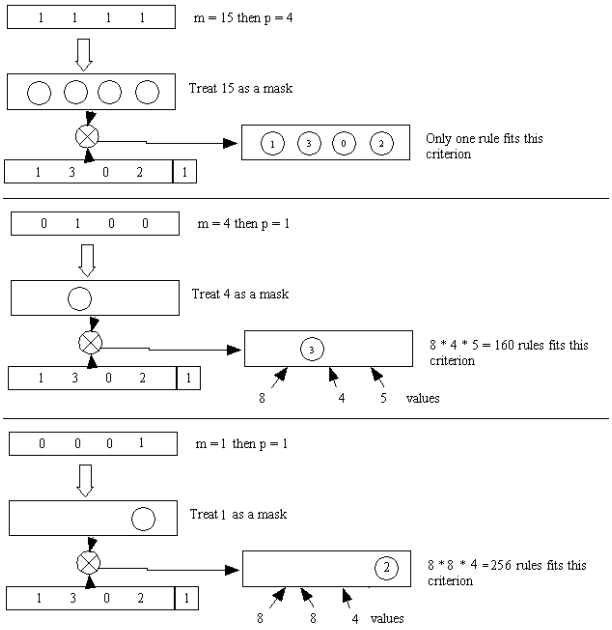

Figure 5: Size of subset depends value and position of p.

- From each parent, a subset of rules matching the criterion is picked out and then swapped to generate two new sets of rules (figure 6).

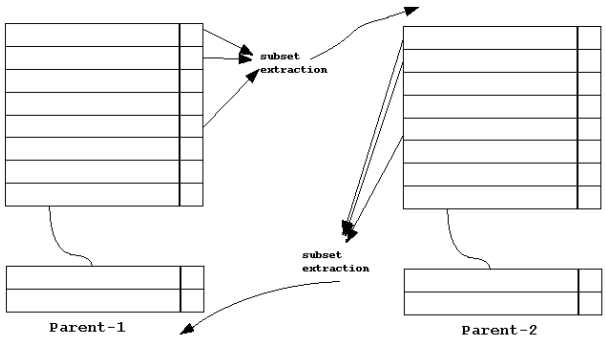

Figure 6: Crossover operation

Mutation Operator of Rule Sets
There are two approaches to defining the rule mutation operator. Namely,

- Based on the mutation rate, probabilistically choose a rule from a set and then alter its action.
- Similar to the crossover operation, set a subset criterion, and then set actions of *ALL* rules within the subset to a fixed value.

Both of these two operators have been tried. We found that the result of second approach is better than the first

one.

This is because the first approach introduces too little variance to the rule set. If the mutation rate is four to five percent, for a rule set with one thousand rules, an average of only forty rules will be affected by mutation. Those affected rules may scarcely or never be fired during the next exploration. Consequently, the desired effect of mutation does not occur and pre-mature convergence happens.

For the second approach, a bunch of rules are mutated at once. It introduces enough variations to the gene pool. Even though a mutation may inadvertently modify some good rules, since the crossover operations are preformed after mutation, the subset rules altered by mutation may be splitted and spread to other instances within the same generation. With the aid of the selection operation, good rules can be maintained and hence the unnecessary detrimental effects can be curbed (figure 7).

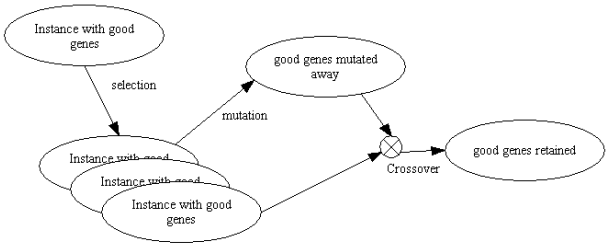

Figure 7: Good genes are retained after mutation.

4 Simulation Results

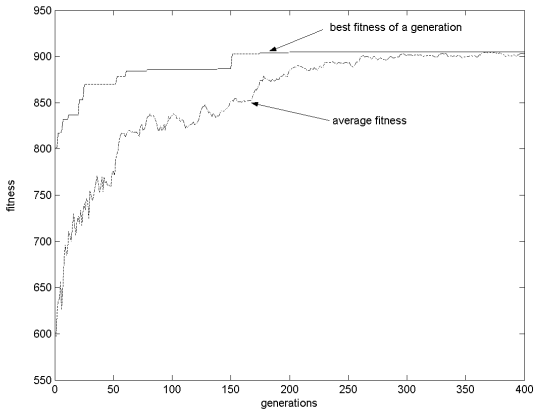

Figure 8: Convergence of fitness.

Figure 8 illustrates the rate of convergence of an empty environment explored by ten robots, with the position of the robotic team initially placed at the centre of the environment. It shows that the recombination operators defined in the above subsections could enhance the performance to 2/3 of the saturated result in 50 generations. It converged to the saturated level in the 270th generation.

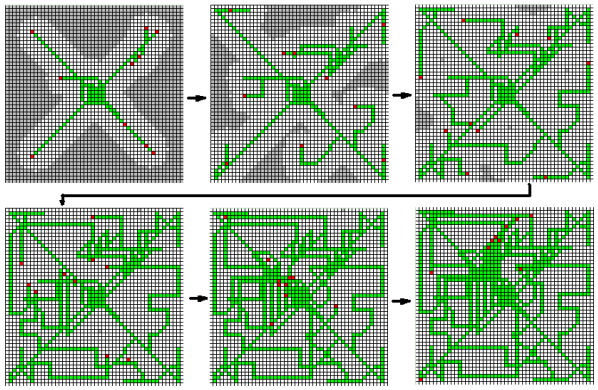

Figure 9: Exploration of an environment with no obstacle, the second best result.

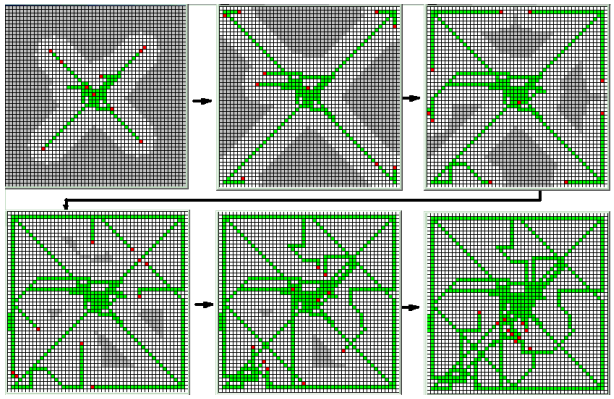

Figure 10: Exploration of an environment with no obstacle, the best result.

Figures 9 and 10 shows the conduct of cooperation in an empty environment. These are the snapshots in equal intervals. The grey region is unknown free, white region is explored free area, red squares are robots and finally, green squares are the tracks taken by some robots. Their behaviours are more or less the same : in the beginning, the robots spread out to the four corners. Once they take up a certain distance away from each other, they evenly cover the locally surrounding area. Finally, they move to the central area to exchange their maps. The main difference is such that, in figure 9, after gathered in the sixth snapshot, there are some unknown cells left. Whereas in fig. 10 the whole area are covered after merg-

ing in the sixth.

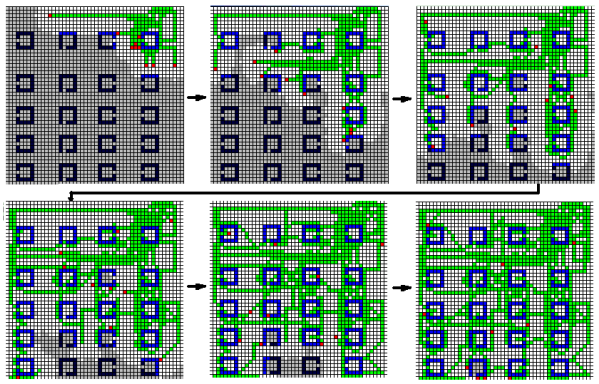

Figure 11: Exploration of an environment with chambers, the second best result.

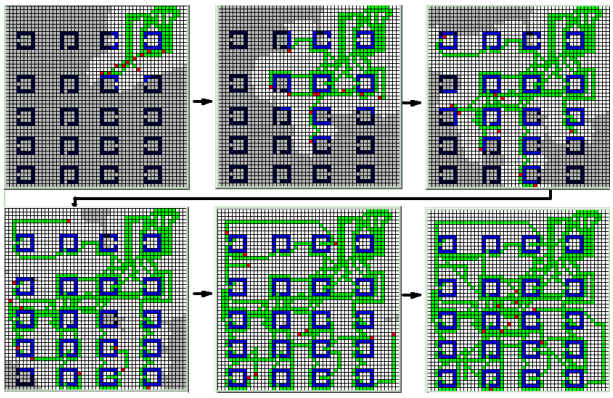

Figure 12: Exploration of an environment with chambers, the best result.

Figure 11 and 12 display the exploration of an area with rectangular chambers. The snapshots are in equal intervals also. The initial location of the team is at the upper-right corner. Black blocks are unknown obstacles, blue blocks are detected obstacles. This one is similar as the empty environment : they move to the central area and then spread to other regions, and finally merge.

These trials show that by using our rule-sets representation and recombination operations, GA successfully gives an answer to our question:

Question:

During the path planning process, every member of the robotic team has to make a choice between moving to another robot to obtain a more accurate map, or, executing the exploration task. When to do which ?

Answer:

In between the tasks of exploration and map exchange, always do the former in the beginning. Map exchange will be advantageous only after EACH *member is in possession of a certain amount of information for sharing.*

5 Result Analysis

Based on the emerged behaviour in the total empty space, we can analyse the exploration efficiency in the following way:

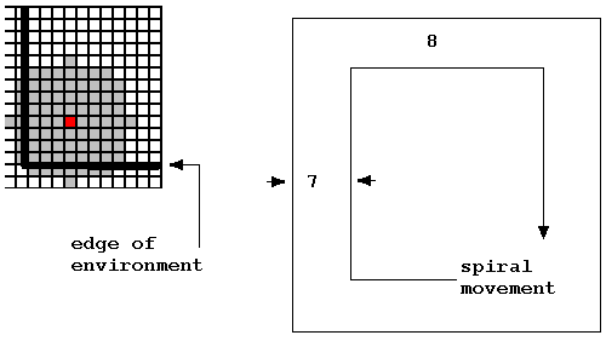

Figure 13: Left: Effective limit of range finder. Right: Spiral movement of a single robot in an empty area.

Figure 13 is the coverage area of range-finder of a robot, which is 5 cell in radius. If a single robot has to explore an environment of 50 by 50 cells by taking a spiral path from right bottom corner to the centre, it has to run three loops. Every loop reduces the dimension of unknown by 15 cells, and each loop takes steps:

$$(Length.of.Dimension - 7) * 4 - 8$$

Therefore, the total steps count is:

$$((50-7)*4-8)+((35-7)*4-8)+((20-7)*4-8)+6 = 317$$

In other words, the exploration cannot be completed in less than 317 time units by a single robot.
If there are 10 robots cooperating in a perfect sense, they will take 317 / 10 = 32 time units to cover the whole area. However, after exploring its own section of area, the robots have to gather at one location to exchange each other's map. Since the initial position of the team is at the centre, it is fair to claim that after 32 time units of synchronized exploration, they will take certain time to return from the boundary back to the centre. Among all the members, the one return from one of four corners will take the longest time. That is:

$$50/2 * \sqrt{2} = 35$$

Hence, the optimal performance will be $32 + 35 = 67$ time units.

For our evolved best instance, the optimal fitness is 909 which in turn equals to 91 time units and the efficiency is:

$$67/91 = 73.6\%$$

In short, it can attain 73 % of the best performance.

For the chambered environment, the analysis is not so straight forward. We use another metric to measure the performance, i.e. comparison with the *greedy approach*. The greedy approach is such that a robot always moves to its nearest unknown cells and it revises its action / path stepwisely. By taking the greedy approach for the chambered environment, we found out the required time is 223 time units. Since ours is 132, the improvement is:

$$223/132 - 1 = 68.9\%$$

Moreover, by taking the greedy approach for the empty environment, the required time is 263 time units. Hence, the improvement is:

$$263/91 - 1 = 189\%$$

6 Discussion

The use of genetic algorithm evolution to develop behaviours of robotic swarms is still in an exploratory stage. The decision about which control level to be evolved is a hard choice. One common approach is to evolve weights of a neural network which maps the sensory inputs to motor controls of a robot. The main advantage of this approach is its free of any designers' views of the problem, but its disadvantage is its high sensitivity to sensory information.

This experiment uses a set of rules to control the behaviours of a robotic swarm. By using this higher level representation, we take a bold tactic to investigate: if a deterministic approach is sufficient for this type of problems?

Initially, there is no idea about what these rules will look like. It is no guarantee that the conditions that form the antecede of a rule are the critical conditions which constitute some good behaviours. Although the rule-set is complete and sufficient to cover all possible situations, it may still be possible that no high-quality behaviours can be found.

The results of this experiment shows that by using GA evolution, good behaviours which are subject to the circumstantial constrains like the number of robots, initial position of the robot team and locations of obstacles, etc., can be found. A population of individuals with random sets of rules can evolve to a fixed rule-set which performs very well in the simulated environment. This evolved result may not be optimal, since optimality is hard to be proved, but it is certainly a good result. Therefore, we can claim that GA evolution with rule-based chromosome representation is a feasible approach to seek for some good collective behaviours.

The next paragraphs constitute a comparison between our approach and the Michigan-style learning classifier systems (LCS) [Lanzi and Riolo , 2000].
Learning classifier systems are machine learning systems which learn to interact with a partially unknown environment, using short- / medium-term rewards to guide a GA core to enhance its rule-based control system. The approach taken here has the following features in common with LCS:

- Rule-based representation of control system
- An partially unknown environment interaction
- GA-based new rules generation

However, there are three major differences:

- LCS requires short- / medium-term rewards whereas our approach only requires an end of task score
- LCS' rewards are dedicated to a single rule, our rewards to the whole set of rules
- LCS does not generate a full set of rules to represent the decision tree whereas ours has to create the full list for the running of the crossover operation

Originally, Holland et al's [Holland and Reitman , 1978] credit assignment scheme, bucket-brigade algorithm (BBA), is based on a market model: each rule acts as a business agent in chains leading from an external payoff back to a succession of suppliers / service providers. The scheme is simple and sound; however, it does not work as expected for robotic swarm problems because of two reasons. We can use the same business / market analogy to explain:

- Since it is difficult to decompose the task to smaller sub-tasks, the reward is very long-delayed.
  It is like a very long chain of service providers working in a row to offer services to an end-user. Except the one who directly serves the end-user, each service provider relies on its supplier's credit to maintain its service. Once the end-user clears its bill, each party in the entire chain can get its own share of reward. This method works fine for a chain with a few parties, but not for a very long chain. Those parties in the head of the chain will go bankrupt before the end-user pays.
- The sharing of reward is not evenly distributed as BBA expected.
  For BBA, the reward for nth party in the chain is:

$$reward * K^n$$

where K is a constant factor.
Nevertheless, the importance of various robots at different moments are far more complicated. In a genuine market, if any party believes its service is more valuable, it will adjust its price. Its supplier and customer will subsequently tune their prices until a balancing point is attained. Without this self-regulating mechanism, a correct allocation of reward can never be realized.

The major merit of our approach is that we successfully get around with credit assignment problem, which is an essential barrier for multiple robotic research.

On the contrary, the main drawback of our approach is the requirement to represent the whole combination of conditions of rules. If the counts and values of conditions are high, the total number of rules will grow geometrically to an unmanagable size.

7 Conclusion

The results presented here are promising. They demonstrate that, by using sets of rules as chromosome, associated with the right recombination operators, GAs can be used to generate useful behaviours in robotic swarms.

Even though the results shown in this paper (fig. 12 and fig. 10) are generated by two different set of chromosomes, we have successfully applied these results in the design of a team of robots for unknown environment exploration. A robust solution has been formulated, and it runs satisfactorily in simulation.

References

[Goldberg , 1989] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.

[Holland , 1998] J.H. Holland. *Emergence: From Chaos to Order.* Cambridge, Massachusetts, Perseus Books, 1998.

[Holland , 1995] J.H. Holland. *How Adaptation Builds Complexity.* Reading,Massachusetts, Addison-Wesley, 1995.

[Holland and Reitman , 1978] J.H. Holland, J.S. Reitman. *Cognitive systems based on adaptive algorithms*, Pattern-directed inference systems, New York: Academic Press, 1978.

[Holland , 1960] J.H. Holland. *On Iterative Circuit Computers Constructed of Microelectronic Components and Systems.* Proceedings of Western Joint Computer Conference (WJCC), 1960.

[Jarvis , 1984] R.A. Jarvis. *Collision-free trajectory planning using distance transforms*, Proceedings of National Conference and Exhibition on Robotics, Melbourne, 1984.

[Lanzi and Riolo , 2000] P.L. Lanzi and R.L. Riolo. *A Roadmap to the Last Decade of Learning Classifier System Research*, Learning Classifer systems - From Foundations to Applications, Springer, 2000, Page 35 - 61.

[Mitchell , 1996] M. Mitchell. *An Introduction to Genetic Algorithms.*, MIT Press, Cambridge, Massachusetts. 1996.