EVOLUTION OF INTELLIGENT AGENTS: A NEW APPROACH TO AUTOMATIC PLAN DESIGN

Dániel Kovács László

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, H-1117, Magyar tudósok körútja 2., Hungary
dkovacs@mit.bme.hu

Abstract: Evolution of intelligent agents, a new approach to automatic planning, is presented here for the first time as a new technique for evolving systems capable of generating "optimal" plans without any prior knowledge of the environment, or any method (i.e. schemata) concerning plan design. The classical model of system-environment interaction is extended by making it more "natural". By the use of the recent gene expression programming (GEP) technique a fully functional, multi-layered system architecture capable of solving complex tasks is proposed. The power of the new approach is demonstrated by testing it on several micro-world problems. *Copyright © 2003 IFAC*

Keywords: Agents, Genetic algorithms, Learning systems, Planning, Problem solvers, Self-optimizing systems

1. INTRODUCTION

The mathematical model of natural evolution, namely genetic algorithms (GAs) are globally convergent, stochastic search methods (Holland, 1975), discovered in the early '70's. The word *"agent"* cannot be defined with the same precision. Typically a system called *"agent"* is an autonomous, adaptive entity placed in an environment, where it tries to satisfy a given task. So far not much success was shown in connecting the two fields in a natural way, despite that it would be analogous to the empirical sense of human evolution and thus potentially fruitful.

This gap is targeted by the proposed concept of agent evolution, which is a theoretical approach of how to find the best possible problem solver for a given task, i.e. an adaptive system that can automatically discover the rules of plan design required for solving complex problems.

The principle is based on the classical system-environment interaction model, expanding it in a way by reinterpreting the hitherto used decision-making mechanisms, the mental representations of the environment, and other agent components.

The topic becomes more and more vital due to the growing need for such "intelligent" applications both in scientific and daily activities. The increased computer power, the global use of the Internet, and other issues concerning our everyday life make it worth, if not indispensable, to design "intelligent tools", such as robots, software agents, etc. (IBM, 1999). The need for automatic decision-making is crucial, when thinking of exploration of such areas that are inaccessible, or unmanageable for humans.

Our goal was to design a system that – given a class of problems – can produce an optimal solver. The present work introduces the agent evolution by presenting the revised concepts of the agent-environment relation in detail. The paper then proceeds with the description of a possible (currently Prolog-based) implementation, illustrating its functionality on a set of test cases. Finally it evaluates the result, and draws its conclusion concerning the effectiveness of the concept.

2. AN OVERVIEW OF AGENT EVOLUTION

The idea to bring together evolution with agents within the classical framework of system-environment interaction model is as follows:

Given an environment, i.e. a problem-space, where a population of agents is being developed via an unsupervised evolution, every agent evolves its best action via an inner, supervised evolution.

Lets us define the meaning of the most important terms:

- An **environment** is a problem-space representing a task, where the solution of the problem means the solution of the task.
- **Evolution** is a process, which tends to evolve elements of the same type within the environment using natural means of selection.
- An **agent** is an adaptive, problem solving system, which – being a part of an evolution – tends to solve a task.

The conceptual framework of agent evolution is shown in Figure 1.

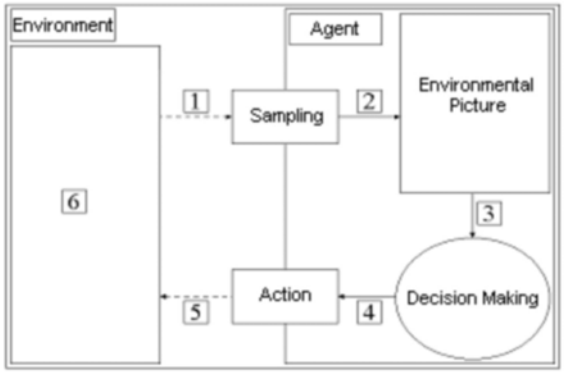

Fig. 1. General framework of agent evolution

An agent first samples the environment (1), creating its picture (2), which then is used by a decision-making mechanism (3) to evolve the best possible action (4) toward the environment (5) modifying its present state (6).

The agent evolves the best possible action via an "inner" evolution. This means, that it must possess a fitness function to classify the goodness of a given action, and this implies, that it must construct the fitness function all by itself. The agent constructs a fitness function during the sampling stage, which then is used to evaluate the goodness of an action by evaluating the final state of the inner representation, generated by that action. Secondly, the agent must generate populations of such actions, to be able to run an evolution. For this purpose an extension of genetic programming (GP) (Koza, 1992), namely gene expression programming (GEP) (Ferreira, 2002) is used, to make the evolution of actions possible. Actions are coded as multi-genic GEP chromosomes, and can be considered as conditional action-chains, i.e. programs. The developed framework, which extends the standard approach, can be seen in Figure 2.

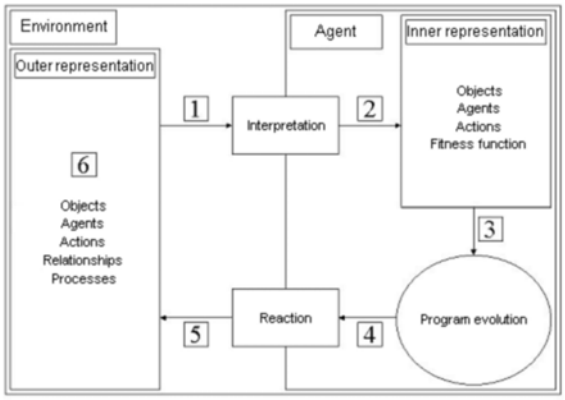

Fig. 2. Detailed framework of agent evolution

The outer representation of the environment consists of objects, other agents, possible elementary actions, relationships between objects and agents, as well as dynamic processes. The agent interprets the outer representation (1) and constructs its inner equivalent (2). The inner representation consists of pictures of objects, agents, possible elementary actions and a fitness function to evaluate them. The agent then generates an initial population of conditional action-chains, consisting of interpreted elementary actions (3), and evolves them using a fitness function, to get the best possible variant (4). This is then used as an action toward the outer representation (5), modifying its present state (6). The agent continues this loop, until it stops by itself, or is externally interrupted.

Many question arise by examining the above concept:

1. *What is the difference between the outer, and the inner representation?*
2. *On what basis does the agent construct the inner representation?*
3. *In what way are actions tested and evolved?*
4. *How does the agent know, how to calculate the fitness of a given action, how to interpret the environment, how to run, and parameterize the inner evolution?*

Let us answer the questions one-by-one: Interpretation can be any linear, or non-linear transformation, so the inner representation can differ from its outer equivalent depending upon the specific realization of its functionality. The inner and the outer representations are rarely equal, usually only a "surface" of the outer is "projected" into the inner.

Agents use only a part of the outer representation to construct the inner, i.e. that part, which is sensed by them at the moment. Consequently agents can only interpret their local environment. This leads us to an extension of our model, and thus some more definitions:

- The complete environment, i.e. the outer representation is the **global reality** of an agent.

- The part of the global reality sensed by an agent is the agent's **local reality**.

Agents use their local reality to construct the inner representation of the global reality, i.e. the outer representation.

It is evident, that actions can only be tested on inner representations of the environment, before they are used. At first an agent senses the global reality. The resulting local reality is then interpreted by the agent. There may be several global realities producing the same sensual impression, i.e. the same local reality, so the agent constructs alternative global realities, called **fantasies**. Fantasies are global realities "thought" to be possible by an agent (similarly to the possible worlds in models of modal logic). It then continues by generating an initial population of conditional action-chains, which are executed – one after the other – within every fantasy, so that the goodness of an action is calculated as its average goodness on all the fantasies. An action being part of a fantasy is called an **agent's double**. A double can only sense a local fantasy, analogously to the local reality in the real environment. The extended concept of the agent-environment relation is shown in Figure 3.

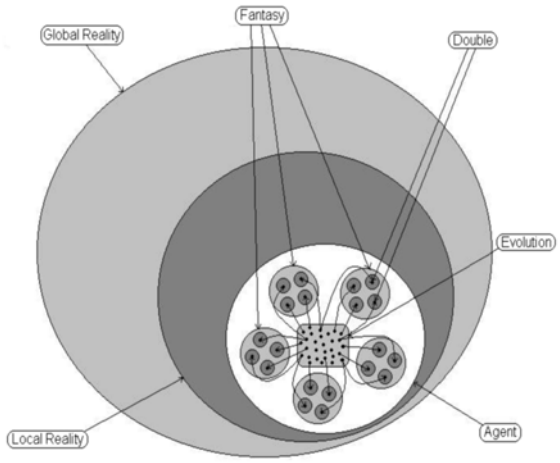

Fig. 3. Extended agent-environment relation

The last question is the "how" of the above concept. The answer is quite simple: An agent's double is a GEP chromosome, i.e. a conditional action-chain. At a meta-level every agent can be represented with a set of such GEP chromosomes. One of them is responsible for representing production-rules, which can produce a fitness-function in interaction with the environment; the other can be responsible for representing the interpretation mechanism, etc. Every agent holds its chromosome-package, waiting for an other agent to make an exchange, crossover, or any other genetic manipulation.

This way the agent evolution works from top to bottom. The only uncovered theoretical aspect is the make-up of the inner and outer evolutions. The paper continues with a brief overview of the GEP theory, that makes the program evolution possible.

3. USE OF THE GEP THEORY

As mentioned before, gene expression programming (GEP) (Ferreira, 2002) is an extension to genetic programming. It was used because it is "*more natural*" and in some cases considerably faster than other genetic programming methods (Koza *et al.*, 1999).

In genetic programming the genotype, on which the genetic operators work, is the same as the phenotype, which is evaluated by the fitness function (like in classical genetic algorithms), while in gene expression programming these two concepts are different. That's why it is called "*more natural*" in the first place.

A GEP chromosome (genome) consists of genes, which can be "expressed" to be equivalent with a GP program-tree. A gene is a string, consisting of terminals and functions. Functions have a number of arguments, where an argument can be either a function, or a terminal. Terminals are the leaves of the program-tree, while functions are its inner vertices. For example $F=\{+, -, *\}$ can be a function-set, with decimal numbers as terminals. Figure 4 shows a program-tree build from the above example.

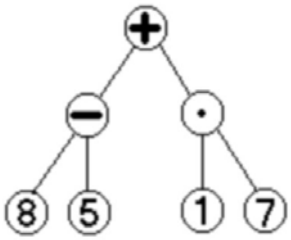

Fig. 4. Example of a program-tree

There are many ways to describe such a tree. The above program tree is *+(-(8,5),*(1,7))* in prefix notation, and *((8-5)+(1*7))* in infix notation. By evaluating this tree, a decimal value of 10 is calculated. In gene expression programming the gene for the above tree is shown in Figure 5.

Fig. 5. GEP gene example

When coding a tree, its layers are simply put beside each other. By knowing the functions' arity a gene can be easily decoded into a program tree.

In general a gene has two parts. The first part consists of functions and terminals and is called the "head" of the gene. The other part consists only of terminals,

and is called the "tail". Given the length of the head *(h)*, and the maximal number of arguments (*n)* for a function, the maximal length of the tail is $h(n-1)+1$, and so the length of the gene is $hn+1$. Given a set of functions, terminals, and the length of the head, the length of the gene follows.

Usually when decoding a GEP gene, not all of the genetic material is transferred into the program tree. The end of the tail may be left untouched, because the head does not contain enough functions. For example, by changing the second position of the gene shown in Figure 5 to a constant, a very different program tree will arise, which does not possess all the genetic material the genome had. The modified genome and its program tree are shown in Figure 6.

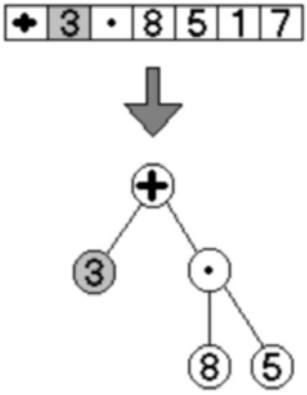

Fig. 6. Modified GEP example

By evolving, selecting, crossing, mutating GEP genes it is possible sometimes to achieve a "neutral change", i.e. when a genes' non-coding region is changed. This is the second reason, why gene expression programming is called "*more natural*" than any of its predecessors.

In the above examples functions with only numeric arguments were shown, and so program trees had to be evaluated BOTTOM-UP. In robotics and other fields of application the function set usually consists of conditional functions, while the terminal set consists of functions representing actions, movements, etc. In that case a program tree is more like a decision tree – which has conditions as its inner vertices and actions as its leaves – so it is evaluated TOP-DOWN.

4. IMPLEMENTATION ISSUES

Sicstus Prolog 3.9.1 (Szeredi and Benkő, 2000) was used for implementing the theory in particular, and so the TOP-DOWN method was preferred for evaluating program trees. The function set consisted of *cognitive* and *perceptive* actions, while the terminal set consisted of *modifying* actions.

- **Perceptive** actions are those, which evoke other actions on a given condition of the local reality.
- **Cognitive** actions evoke other actions on a given condition concerning the state of the global reality.
- **Modifying** actions modify the global reality.

Perceptive actions are like IF-THEN statements, which call other actions depending on the perception being present in the sensed environment (i.e. the local reality). Cognitive actions are much the same, but they have conditions concerning the state of the global reality, i.e. the outer representation. For instance, if a given action is a logical deduction, which – by knowing some things of the environment – can deduce some things till then unknown, then it is called a cognitive action. Modifying actions are those, which can modify the state of the surroundings, the state, or the position of its user, or any other aspect of the global reality. A program tree, consisting of the above-mentioned action types is shown in Figure 7.

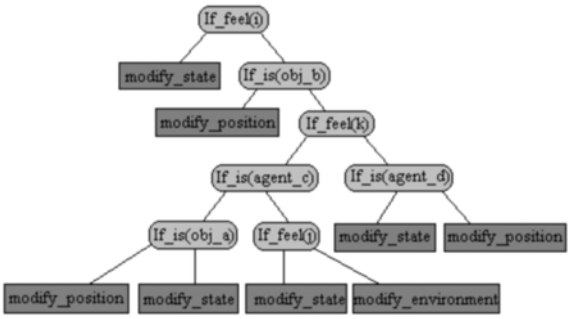

Fig. 7. A TOP-DOWN program tree

The above tree may be interpreted as follows: IF the *i*-th perception is present at the moment, then modify the state of the agent, ELSE IF an object of type "B" is close by, modify the position of the agent by stepping toward it, ELSE IF the *k*-th perception is present, etc. It represents only one action, which is chosen depending on the current state of the local reality so it may be called a conditional action. With a given number of genes representing such program trees (activated sequentially one after the other), a conditional action-chain – mentioned in the previous sections – can be obtained. An action-chain can be considered a **plan**, where each gene is a conditional step of the plan. Thus – by evolving conditional action-chains – plans are actually evolved. It is important to see that the methods of plan design are not preset; agents evolve them by themselves. Thus by evolving agents, and because of a globally convergent computational model of evolution (Goldberg, 1989), systems capable of designing "optimal" plans for dynamic environments are evolved.

5. TEST ISSUES

The approach was tested on several different problems, so called micro-worlds. Because of the numerous free parameters, testing was limited by time and machine capacity. For example, working with 6-7 parameters taking three values each, four agents, 100 agent steps, three problems and ten runs (to make an average), would require $3^6 \cdot 4 \cdot 100 \cdot 3 \cdot 10 = 8748000$ evolution-runs. On a single computer (i.e. Pentium 4 1.5Ghz) it would take months to do the necessary calculations for a relevant testing. Luckily tests can be easily parallelized.

Two test cases are shown: Wumpus-world and Table-world. Wumpus-world (Russell and Norvig, 1995) is a famous test bed for different agent-architectures. It is a $k \times n$ size grid-world, with two types of objects: Pit, and Gold. A pit is a place, where the agent can fall and die, while gold can be picked up. Except the other agents, there is only a monster called Wumpus. Meeting Wumpus is deadly.

An agent's task is to find all the gold scattered randomly over the grid, to kill the Wumpus, then to return to the starting coordinate, and climb out. An agent's local reality consists of five possible perceptions: it can feel *smell*, if being in the explicit neighbourhood of a Wumpus; *wind*, if in explicit neighbourhood of a pit; *shine*, if being over a gold piece; *push*, if being crashed into a wall; *scream*, if a Wumpus died recently. Every agent has one shot to kill a Wumpus. Possible actions for an agent are: Move-forward, Rotate-left, Rotate-right, Shoot, Pick-gold and Climb-out. Perceptive actions, such as If-Smell, If-Wind, If-Shine, If-Push and If-Scream and cognitive actions such as If-Wumpus-in-Front and If-Pit-in-Front can also be implemented. An example of Wumpus world is shown in Figure 8.

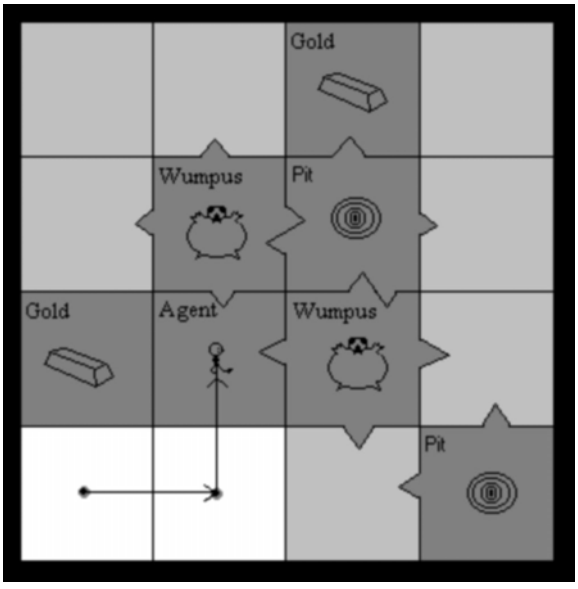

Fig. 8. Wumpus World

Wumpus-world is used to test an agent's "human-like" intelligence. To survive in this world, an agent must be able to deduce possible threats, to memorize places, etc. For being successful it must express a complex, "human like" behavior.

Table world (Zhang and Cho, 1999) is also a grid world, but much more simpler, than Wumpus world. There are also two types of objects: a table and obstacles. There are four agents, which have the same task of moving the table to a given destination. The table is too heavy for less, than four agents, so they must cooperate in solving the task. An agent has the following actions: Move-forward, Rotate-Clockwise-around-the-Table, Move-in-a-Random-Direction, Turn-toward-the-Table, Turn-toward-the-Target and Stay. If-Crashed-into-Obstacle, If-Crashed-into-other-Agent, If-Table-is-Around, and If-Target-is-Around can also be implemented as perceptive actions. An example Table world is shown in Figure 9.

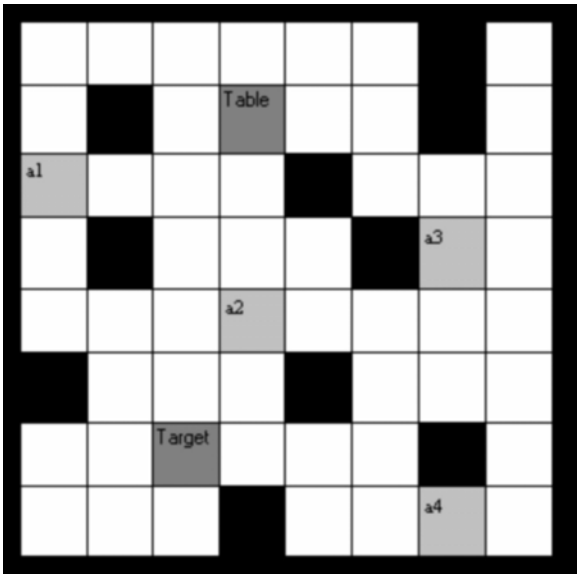

Fig. 9. Table World

Table world is used to test the ability of agents to cooperate. Collective, emergent behavior is needed to solve the task.

In Wumpus-world the following fitness function was used:

$$F = L \cdot \left(0.3 \cdot \frac{G}{G_{\max}} + 0.3 \cdot \frac{W}{W_{\max}} + 0.4 \cdot C \right) \qquad (1)$$

L is 1, if the agent is alive, 0, if not. G is the number of gold pieces collected by the agent, G_{max} is their maximal number. W is the number of Wumpuses killed by the agent, W_{max} is their maximal number. C is 1, if the agent climbed out successfully, 0, if it is still on the grid. Agents evaluated 32 genic chromosomes as their doubles during the inner evolution.

With the above definition tests have shown, that our approach is close to a human solution on the described Wumpus world test case. Approximately 90% of solutions generated by the evolution of agents produced the same result as human opponents on the average. The remaining 10% differed in both directions.

It is important to add, that Prolog's Constraint Logic Programming on Finite Domains, i.e. CLP(FD) library was used for creating cognitive actions, furthermore every agent had a memory map of the explored area.

In Table world the following fitness function was used:

$$F = \frac{1}{f_1 + f_2} \tag{2}$$

$$f_1 = \sum_{a=1}^{4} \{c_1 \cdot \max(X_a, Y_a) + c_2 \cdot S_a + c_3 \cdot C_a - c_4 \cdot M_a + K\}$$

$$f_2 = \sum_{a=1}^{4} \{c_1 \cdot \max(X_a, Y_a) + c_2 \cdot S_a + c_3 \cdot C_a - c_4 \cdot M_a + c_5 \cdot A_a + K\}$$

X_a and Y_a are the X and Y distances of the ath agent-double from its current target. S_a are the steps it made. C_a is the number of collisions it suffered. M_a is the distance between its start and end-coordinates. A_a is the penalty for moving away from other agents. c_i is the weight factor for the ith fitness component. K serves as a normalizing factor.

The four agents used the above fitness function in the following way: a 2-genic chromosome, i.e. action-chain was generated, whose F fitness value was calculated after every agent double in a given agent used the same action chain for 20 times. This meant, that the four doubles (generated in an agent) did the same conditional action. This lead to their cooperation within every of the agents. Because of the fact that they were using the same conditional action-chain, it is called a *homogeneous* cooperation.

The astonishing fact is that agents – evolving their respective actions thus separately – cooperated in a *heterogeneous* way. This means, that they used different conditional actions to achieve cooperation, similarly to human cooperation.

It is important to see, that f_1 is responsible for "homing", while f_2 is responsible for "herding". Homing means that agents get round the table, while herding is when they bring the table down to its destination. Consequently: cooperation is controlled by the choice of the fitness function.

6. CONCLUSIONS

The paper introduced a new theoretic approach using evolutionary computation and its application to an automatic plan design. It can be concluded, that a general, scalable and fast algorithm was found for solving the difficult task of plan design without the need for any particular preset method, or schemata. Test results have shown, that a closely "human level" of competition can be achieved. Moreover, it was possible to generate heterogeneous cooperation as an emergent behavior. The above benefits make the new technique a promising alternative for solving complex tasks.

REFERENCES

Ferreira C. (2001). Gene Expression Programming: A New Adaptive Algorithm for Solving Problems, *Complex Systems*, Vol.13, issue 2: pp.87-129.

Goldberg D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Holland J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press.

IBM (1999). *Journal on Pervasive Computing* Vol. 38, No. 4., www.research.ibm.com/journal/sj38-4.html

Koza J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza J. R., F. H. Bennett III, Andre D. and M. A. Keane (1999). *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann.

Russel S. J., Norvig P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Szeredi P., Benkő T. (2000). *Introduction into logical programming.* Technical University of Budapest.

Zhang Byouk-Tak, Cho Dong-Yeon, (1999). Co-evolutionary Fitness Switching: Learning Complex Collective Behaviours Using Genetic Programming, *Advances in Genetic Programming*, Vol. 3, 18, pp.425-445.