Implements an Evolutionary Self-Organizing Map Based on Graph Evolution

Maiga Chang, Jia-Sheng Heh
張明治，賀嘉生
Department of Electronic Engineering of Chung Yuan Christian University
中原大學電子工程系

Abstract

Optimization problems in the real world are very difficult to be solved by conventional optimization techniques. Evolutionary algorithms can be used to reach the global minimal in the problem domain by the characteristics of adapting to environment and evolving from generation to generation. The widely known evolutionary algorithm, genetic algorithm provides the most considerable solving mechanism of evolutionary algorithms as applied in the area of industrial engineering. Unfortunately, not all problems in real world can be reformulated into genetic algorithm easily based on the forms of genes.

A prototype of evolutionary algorithm for graph-formed problem is proposed in this paper, which is so-called graph evolution. The graph evolution theory not only can cover all features of the traditional genetic algorithm, but also can extend the solvable problem domain to graphs. Up to now, this paper tries to apply the prototype of graph evolution to some graph-related systems. Evaluation is made for proving the advantage of using graph evolution instead of traditional problem-solving mechanisms.

Keywords: evolutionary algorithm, mutation, crossover, cut-vertex, fitness, homeomorphism

1. Introduction

Many optimization problems from the real world are very difficult to be solved by conventional optimization techniques.[Sin96][Hit96] The characteristics of adapting to environment and evolving from generation to generation, evolutionary algorithms can be used to reach the global minimal in the problem domain.[Bäc96] Recently, genetic algorithms provide the most considerable solving mechanism of evolutionary algorithms as applied in the area of industrial engineering.[GeC97] Unfortunately, not all of problems in the real world can be formulated and translated into genetic algorithm easily. This kind of problem translation is made with the fundamental feature of genetic algorithm.

This paper presents a prototype of evolutionary algorithm for graph-formed problem, which is so-called graph evolution. The graph evolution not only can cover all features of the conventional genetic algorithm, but also can extend the solvable problem domain to graphs. Consistent formulation between graph evolution and genetic algorithm is analyzed and designed in this paper for keeping the graph evolution can fit all solvable problems of genetic algorithm. Since the objective of the proposed evolutionary algorithm is graph-formed problems, the characteristics of graphs and the corresponding genetic operations in evolutionary algorithms are also necessarily defined and investigated in advance.

Section 2 describes fundamental pre-knowledge about the concept of graph theory, which is useful for transforming the operations in graph to genetic operations. Besides graph theory, the widely-known type of evolutionary algorithms, genetic algorithm, is also be illustrated in Section 2 for the consistency analyzed and designed in Section 3 and Section 4. Detailed analysis of genetic phase in genetic algorithm is demonstrated in Section 3. And Section 4 designs these preliminary corresponding graph genetic operations, including mutation and recombine, are analyzed for graphs in the graph evolution of evolutionary algorithms. Experiment system uses the graph evolution to improve performance is implemented to verify the purpose of this paper in Section 5. And Section 5 also shows the results of evaluation are quite acceptable not only at the reduced error rate, but also for the reduced processing time. Section 6 makes a simple summary for this paper and discusses what can be done in the future.

2. Problem Formulation of Graph Evolution

Evolutionary algorithm is a stochastic optimization technique derived from simulating the natural evolutionary process of human beings. Such algorithms can often outperform conventional optimization methods when applied to difficult real-world problems. [BäS96][Sch94][Mic96] There are currently three main avenues of this research: genetic algorithms (GAs), evolutionary programming (EP) and evolution strategies (ESs). Among them, genetic algorithms are perhaps the most widely known types of evolutionary algorithms today. [GeC97]

Genetic algorithms, different from conventional search techniques, start with initial set of random solutions, called population. Each individual in the population is called a chromosome, representing a solution to the problem at hand. A chromosome is a string of symbols; it is usually, but not necessarily, binary bit string. The chromosomes evolve through successive iterations, called generations. During each generation, the chromosomes are evaluated, using some measurement of fitness. [FoG96]

After all the individuals (the parents) of the generation are evaluated by the fitness function, it is able to produce a new generation formed by the new individuals (the offspring). To create the next generation, new chromosomes, called offspring, are formed by either

(a)merging using a crossover operator

(b)modifying using a mutation operator.

Those individuals in the population of next generation are composed by

(a)selecting based on the fitness value

(b)rejecting to keep the population size constant.

Those fitter chromosomes with high fitness values have higher probabilities of being selected. After several operations these chromosomes can represent the optimum or suboptimal solution to the problem. In fact, there are only two kinds of operations in genetic algorithms:

1. genetic operations: crossover and mutation,
2. evolution operation: selection.

The genetic operations mimic the process of heredity of genes to create new offspring at each generation. Although the evolution operation mimics the process of Darwinian evolution to create populations from generation to generation, the description for selection of evolution operation given by Holland is to obtain parents for recombination. [Hol75]

Crossover is the main genetic operator. It operates on two chromosomes at a time and generates offspring by combining both chromosomes' features. A simple way to achieve crossover would be to choose a random cut-point and generate the offspring by combining the segment of one parent to the left and the cut-point with the segment of the other parent to the right of the cut-point.

It has been proven that genetic algorithms are able to solve some difficult real-world problems, especially optimization problems. Since many problems can be expressed as graph forms and will be solved more simply, the well-known type of evolutionary algorithm, genetic algorithm, possibly acts as the mechanism of problem solving. Unfortunately, the genetic operations described above in string-formed genetic algorithm are not suitable to the problems with graph form. Graph evolution in the evolutionary algorithms presented by this paper is a solution to these graph form problems. Hence, the genetic operations and evolution operation have to be analyzed and well designed in the rest of this paper.

The isomorphic graphs play an important role in developing the graph reformation of a problem. Two graphs are probably isomorphic indicates they both are solutions with different meanings of the same problem. Which one is better should then be taken into consideration. Besides the isomorphic issue, the mechanism that making two graph-formed chromosomes to do several genetic operations is also necessary to analyze and well design. In the rest of this paper, the genetic operations for graphs are first analyzed. Preliminary designing of graph evolution operators is then made based on the analysis of graph evolution operations.

3. Analysis of Evolution Operations

Definitions of *graph evolution* are analyzed in this section. First the structure of individual in graph evolution is given with the *graph* is defined as $G(\nu,\varepsilon)$, where $\nu \subseteq \overline{V}$ is the vertex space, $\varepsilon \subseteq \overline{V}\times\overline{V}$ is the edge space and $\varepsilon \subseteq \nu\times\nu$. Definitions of graph evolution are analyzed here also. The structure of *individual* $\vec{a}$ in graph evolution can be described as:

$$\vec{a}=\left(\vec{x},\vec{s}\right)=\left(\tilde{x}_1,\tilde{x}_2;\vec{s}\right), \tag{1}$$

where $\tilde{x}_1=\nu$ is the vertex set and $\tilde{x}_2=\varepsilon$ is the edge set.

Depending on the definition of individual in graph evolution, here is an example as Figure 1 to demonstrate how a *graph chromosome* be. A simple graph with six vertices and seven connections (or edges) is shown in Figure 1, and the corresponding graph chromosome with the same form of modified definitions of gene in the evolutionary algorithm of this paper is formulated as $G_1(\nu_1,\varepsilon_1)$, where $\nu_1=\{A,B,C,D,E,F\}$ and $\varepsilon_1=\{e1,e2,e3,e4,e5,e6,e7\}$, then the individual $\vec{a}$ can be expressed as:

$$\begin{aligned}\vec{a}&=\left(\nu,\varepsilon;\vec{s}\right)\\&=\left(\{A,B,C,D,E,F\},\{e1,e2,e3,e4,e5,e6,e7\};\vec{s}\right)\end{aligned}. \tag{2}$$

As the evolution strategy parameter $\vec{s}$ is ignored, we have

$$\vec{a}=(\nu,\varepsilon). \tag{3}$$

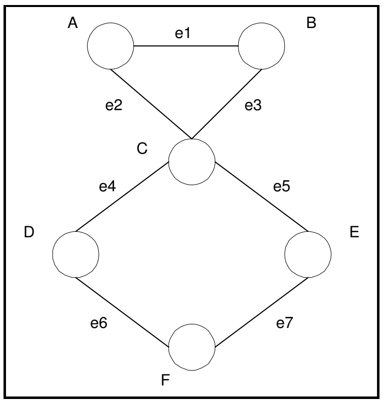

Figure 1. A graph chromosome

Similarly to the traditional genetic algorithm, there are also two kinds of operations: unary operation and binary operation, applied on the graph chromosome of graph evolution. We talk about the unary one at first. As the mutation operation, which is one of unary operation in the conventional genetic algorithm, two kinds of operations can be made: insertion and deletion of list chromosome.

There is a little difference between list chromosome and graph chromosome. Generally speaking, a graph can be differentiated or pruned by adding or deleting vertices or connections (edges) in graph theory. The meanings of mutating a graph are more complex than those applied in a list chromosome, because both variants of vertices and edges should be taken into consideration.

Supposed that a connected graph is kept during the process of mutation, Table 1 lists the possibilities between variants of vertices or edges and sorts of mutation operations. Let $G(\nu,\varepsilon)$ be the original graph as Figure 1 and $G'(\nu',\varepsilon')$ be the graph after the mutation process.

Table 1. Possible combinations of mutations in graph chromosome

Simple operations	$\varepsilon \subset \varepsilon'$	$\varepsilon = \varepsilon'$	$\varepsilon \supset \varepsilon'$
$\nu \subset \nu'$	Node differentiation	X	X
$\nu=\nu'$	Edge differentiation	Duplication	Edge pruning
$\nu \supset \nu'$	X	X	Node pruning

As we can see in the above Table 1, different operations gives different rules, which can be used to determine what kind of a new graph should be made through the mutation process.

Differentiation:

Node differentiation: If $\varepsilon \subset \varepsilon'$ and $\nu \subset \nu'$ (4)

Edge differentiation: If $\varepsilon \subset \varepsilon'$ and $v = v'$ (5)

Pruning: If $\varepsilon \supset \varepsilon'$ and $v \supseteq v'$ (6)

Reproduction: If $\varepsilon = \varepsilon'$ and $v = v'$ (7)

Impossible conditions:

If $\varepsilon \subseteq \varepsilon'$ and $v \supset v'$ (8)

If $\varepsilon \supseteq \varepsilon'$ and $v \subset v'$ (9)

The reasons of these rules as above Eq.(4) to Eq.(9) come from on the properties in the graph theory. Besides the unary operation, there is also binary operation in the graph evolution algorithm. Recombine is one binary operation in graph evolution, just like it in genetic algorithm. Two graph chromosomes will be taken as parameters for the sexual crossover purpose. But recombine operation in graph evolution is far difficult than mutation.

The operation of recombining two graphs in graph evolution has to be described in advance. The first step of recombining two graphs is *partition*, which means to divide a graph into two sub-graphs with some specific methodology. Each graph will possess two parts after partition. One of sub-graph in both original graphs is then picked up for the recombining process. There are several methodologies to divide one graph into more than one parts, such as cut-set and cut-vertex. And we also find out that two homeomorphic graphs can be recombined easily when their two sub-graphs is cut on the specific cut-vertex.

4. Preliminary Designing of Graph Evolution Operators

According to the analysis of genetic operations in graph evolution, two preliminary operations such as mutation and recombine are designed in this section. With those rules analyzed above, mutation operator, σ, in graph evolution can simply take a graph G as its parameter. Then a new graph G' is generated as:

$$\sigma(G(v,\varepsilon)) = G'(v',\varepsilon'). \tag{10}$$

Recombine operator not only takes one graph but two graphs as its parameters. Hence, the recombine operator is more complex than the mutation one. Firstly, two sub-graphs would be divided from each parent graphs by the recombine operator. And then two sub-graphs come from different sets is recombined to form the new generation of chromosomes, it is so-called *descendent chromosomes*. Figure 2 and Figure 3 indicate two parent chromosomes are cut as two sub-graphs by the gray vertex C. And then those descendents can be re-merged depend on the gray vertex C. However, not all graphs can be merged so easily.

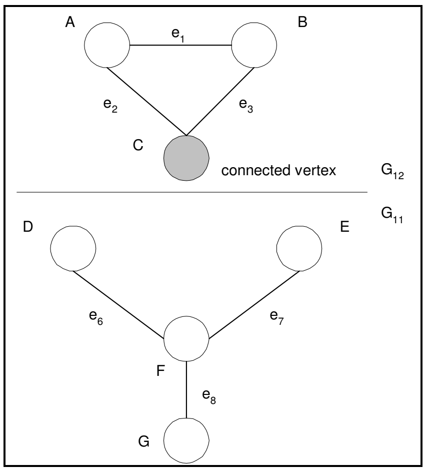

Figure 2. Decomposed sub-graphs of G_1

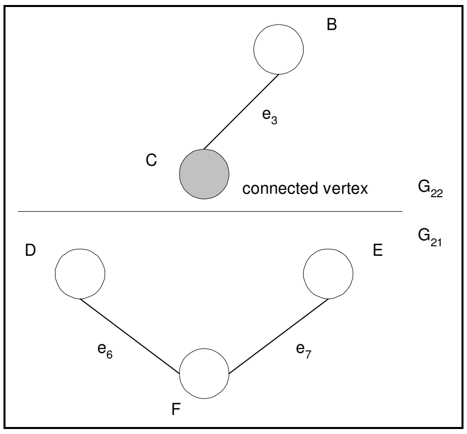

Figure 3. Decomposed sub-graphs of G_2

As description of the above successful example, the descendant graphs can be combined from its parents G_1 and G_2. Let $|v_{C_i}|$ be the number of vertices in the connected vertex set v_{C_i}, the premise that two decomposed graphs can be merged is that $|v_{C_1}| = |v_{C_2}|$, as the following states.

Lemma 1.

if $|v_{C_i}| \neq |v_{C_j}|$, then it is difficult or even impossible to generate a descendant graph from parent graph. □

Hence, the solving for recombining two graphs has to avoid the condition of $\left|V_{C_i}\right| \neq \left|V_{C_j}\right|$ obviously. When the methodology of partition graph is replaced by cut-vertex, it should be solved quite easily.

Even the advantage of cut-vertex in graph theory are discovered and help to solve the partition step, we are still not able to randomly select two graphs to proceed the recombine operation. This is because the condition that one of the selected graphs contains none of the cut-vertex could happen. Hence, some more sufficient condition such as homeomorphism for doing the recombine operation should be met.

Lemma 2.

Two graphs should be at least homeomorphic to process recombine operation. □

Finally, the recombine operator can be precisely designed after the analysis of these important properties for successful operation. *Recombine operator* π takes two *parent* graph chromosomes, says $G_1 \overset{\text{hom}\,eo.}{\sim} G_2$, as its parameters:

$$G_3(v_3,\varepsilon_3) = \pi\left(G_1(v_1,\varepsilon_1), G_2(v_2,\varepsilon_2)\right). \qquad (11)$$

5. Implementation and Evaluation

Our experiment system focuses on applying the theory of graph evolution to the Self-Organizing Map in neural network. MATLAB is chosen to be the development environment. Besides the toolboxes of mathematical manipulation, MATLAB also provides several toolboxes about some application domains, such like neural network, fuzzy theory, digital signal processing and so on. And the MATLAB toolbox of neural network is the fundamental of our experiment system, called Evolutionary SOM.

The whole processing module design of the ESOM is shown as Figure 4, where three phases can be recognized. The only difference between Kohonen's SOM and ESOM is the graph evolution phase. Two corresponding modules, including evolution module and genetic module, of the genetic algorithm are involved in the graph evolution phase. Evolution module does the selection operation in the graph evolution, whereas the genetic module processes those evolution operations on the graphs.

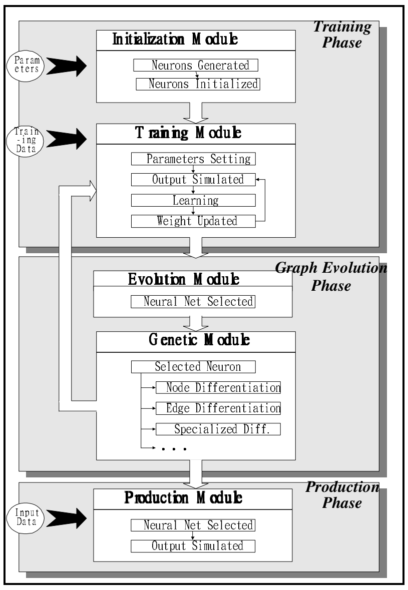

Figure 4. Evolutionary SOM

Experiment systems in this paper use the data on educational transitions for a sample of 500 Irish schoolchildren aged 11 in 1967. The data were collected by Greaney and Kelleghan (1984) and placed on the Web site http://lib.stat.cmu.edu/datasets/irish.ed. After some erroneous records are eliminated, there are 380 records leaving as training patterns and 93 as test patterns. Table 2 describes the meanings of fields in this datasets.

Table 2. Irish educational transition database

Name of Field	Content of Field
Sex	Male female
DVRT	Drumcondra Verbal Reasoning Test Score
Educational level attained	1. Primary terminal leaver 2. Junior cycle incomplete: vocational school 3. Junior cycle incomplete: secondary school 4. Junior cycle terminal leaver: vocational school 5. Junior cycle terminal leaver: secondary school 6. Senior cycle incomplete: vocational school 7. Senior cycle incomplete: secondary school 8. Senior cycle terminal leaver: vocational school 9. Senior cycle terminal leaver: secondary school 10. 3rd level incomplete3rd level complete
Leaving Certificate	1.if Leaving Certificate not taken 2.if taken
father's occupation	Prestige score for father's occupation calculated by Raftery and Hout, 1985, 0 if missing.
School Type	1. Secondary 2. Vocational 9. Primary terminal leaver

In this data sets, classification is made by using four fields, includes sex, DVRT, education level and father's occupation. Based on the classification problem, different kinds of contrast systems are also implemented for the evaluation of graph evolution.

Two evaluation issues are used to verify the advantage of using graph evolution theory in the contrast systems, include training time and classification error. From the view of classification error, Tables 3 shows out both classification error value of Kohonen's Linear SOM and Evolutionary SOM. The whole neural network begins from only one neuron growing up in ESOM, and is differentiated when the network is stable. Lower classification error will produce can be found in Table 3. Graphical representations are also made as Figure 5 shown.

Table 3. Linear SOM vs. ESOM

Name	Epoch	5	10	15	20	25	30
SOM	500	2.0740	1.5090	1.4232	1.3420	1.3292	1.3205
ESOM	**500**	**2.1144**	**1.4890**	**1.1903**	**1.1065**	**0.9646**	**0.8865**
SOM	1000	2.1345	1.4976	1.392	1.3060	1.3003	1.2787
ESOM	**1000**	**2.1044**	**1.4829**	**1.1892**	**1.1032**	**0.9605**	**0.8848**
SOM	2000	2.1190	1.4908	1.3896	1.2928	1.2973	1.2690
ESOM	**2000**	**2.1010**	**1.4770**	**1.1889**	**1.1020**	**0.9599**	**0.8835**
SOM	5000	2.1259	1.4937	1.3866	1.2868	1.2934	1.2659
ESOM	**5000**	**2.1000**	**1.4734**	**1.1881**	**1.1001**	**0.9593**	**0.8823**

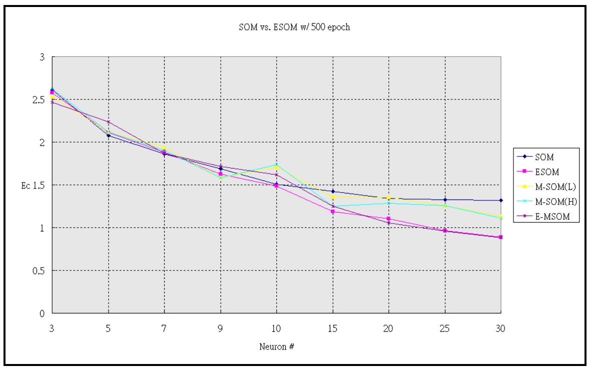

Figure 5. Comparison of learning curve with training epoch = 500

6. Conclusion

This paper proposes a new evolutionary algorithm for graph evolution, and makes a basic research on the graph evolution by using the biological evolution as a reference for the concept of evolution. Fundamental analysis of genetic phases of evolution is made. And graph evolution operations such as mutation, recombine and selection are illustrated. Besides the analysis and design of the graph evolution algorithm, Evolutionary Self-Organizing Map is also implemented for evaluating the performance of using graph evolution instead of the original systems. Not only the curves of learning rates and output errors are better, but also the timing of solving problem is acceptable.

Although the analysis of evolution phases have been made, detailed analysis of genetic operations, especially pruning operators of mutation, is still necessary. Besides the genetic phrase of graph evolution is analyzed and well designed, the evolution phrase is also needed to complete the whole graph evolution algorithm. More actual data should be taken and some useful systems should be implemented for the platform of real evaluating and testing the performance of graph evolution. Variant systems can be also established to verify the adaptation of graph evolution algorithm. Such systems should involve string-form problems in traditional genetic algorithm.

Reference

[Bäc96] T. Bäck, Evolutionary Algorithm in Theory and Practice, Oxford University Press, New York, 1996

[BäS96] T. Bäck and H. P. Schwefel, "Evolutionary computation: an overview," Proceedings of the Third IEEE Conference on Evolutionary Computation, Nagoya, Japan, 1996

[FoG96] D. Fogel and A. Ghozeil, "Using fitness distributions to design more efficient evolutionary computations," Proceedings of the Third IEEE Conference on Evolutionary Computation, pp. 11-19, Nagoya, Japan, 1996

[GeC97] M. Gen and R. Cheng, Genetic Algorithms and Engineering Design, WILEY, 1997

[Hit96] K. Hitomi, Manufacturing Systems Engineering, 2nd ed., Taylor & Francis, London, 1996

[Hol75] J. Holland, Adaptation in Neural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975

[Mic96] Z. Michalewicz, "Evolutionary computation: practical issues," Proceedings of the Third IEEE Conference on Evolutionary Computation, pp. 30-39, Nagoya, Japan, 1996

[Sch94] H. P. Schwefel, Evolution and Optimum Seeking, John Wiley & Sons, New York, 1994

[Sin96] N. Singh, System Approach to Computer-Integrated Design and Manufacturing, John Wiley & Sons, New York, 1996