Genetic-Based Search for Error-Correcting Graph Isomorphism

Yuan-Kai Wang, Kuo-Chin Fan, and Jorng-Tzong Horng

Institute of Computer Science and Information Engineering
National Central University,
Chung-Li, Taiwan, R.O.C.

‡ This work was supported in part by National Science Council of R.O.C.
under contract NSC 83-0408-E-008-001
Abstract - Error-correcting graph isomorphism has been found useful in numerous pattern recognition applications. This paper presents a genetic-based search approach that adopts genetic algorithms as the searching criteria to solve the problem of error-correcting graph isomorphism. By applying genetic algorithms, some local search strategies are amalgamated to improve convergence speed. Besides, a selection operator is proposed to prevent premature convergence. The proposed approach has been implemented to verify its validity. Experimental results reveal the superiority of this new technique than several other well-known algorithms.

1. INTRODUCTION

Graph representation is a structural description which represents an object in terms of its parts and their interrelationships. There are several important issues in building structural description for an object such as the construction of a description from the given data, the classification of the given descriptions, etc. One of the most difficult but interesting issue is the optimum isomorphism between structural descriptions. This problem has been found useful in numerous applications, especially in the area of pattern recognition[1]. For instance, shown in Fig. 1 is a structural description called attributed relational graph of a Chinese character which can be utilized to represent the structural description embedded in the character.

However, in many practical applications, a certain amount of uncertainty or deformations may exist in structural descriptions. Deformations due to noise or distortion of an object may not find a match for an input structural description from all existing structural descriptions because the assignment of it to any class will be rejected by conventional graph isomorphism due to its exact matching nature. Hence, inexact matching, or the so-called error-correcting graph isomorphism, is the only way to remedy the distortion problem in this case. Since the error-correcting graph
isomorphism problem is basically an NP-search problem of combinatorial optimization[2], the development of an appropriate search strategy is crucial in reducing time complexity and guaranteeing the correctness of match.

Genetic algorithms are adaptive search methods which are devised based on the natural genetic processes. They are general-purpose optimization methods which have been successfully applied to difficult search, optimization, and machine learning tasks[3]. It has been shown that genetic algorithms can be applied to general NP-complete problems[4]. Moreover, simultaneous allocation of search effort to many regions of the search space contributes the power of genetic algorithms. It is called implicit parallelism in [5] and is proved that it can set a lower bound of an $N^3$ speedup over systematic sequential search.

In this paper, a genetic-based search approach is proposed to solve the problem of error-correcting graph isomorphism. Since direct implementation of genetic algorithms on this problem exhibits lower performance than traditional approaches, two local search strategies are incorporated to form a hybrid search mechanism. The proposed hybrid scheme reveals substantial improvement in performance by employing the global specialty of genetic algorithms as well as the local betterment capabilities of local search strategies. A proposed selection operator called inhibitive selection is incorporated to prevent premature convergence, which is a well-known serious problem in genetic algorithms. Besides, two fitness functions are devised and their effectiveness will also be discussed in the paper.

This paper is organized as follows. Section 2 describes the error-correcting graph isomorphism problem and surveys existing solutions to this problem. Detail descriptions of genetic algorithms are provided in Section 3. Section 4 describes the proposed genetic-based search approach in solving the error-correcting graph isomorphism problem. In Section 5, experiments are conducted to verify the performance of this approach in comparison with tree-search and eigendecomposition approaches. Finally, conclusions are given in Section 6.
2. **ERROR-CORRECTING GRAPH ISOMORPHISM (ECGI)**

Graph isomorphism has been extensively used to express basic structural similarity between two structural graphs. Two graphs are isomorphic if there is a one-to-one vertex mapping between them which preserves edge incidence relationship. Due to deformations, some structural information of the graph may be missed or distorted. The matching problem in finding the vertex mapping for two nonisomorphic graphs becomes the error-correcting graph isomorphism (ECGI) problem. An example illustrating the ECGI problem is shown in Fig. 2. It is pointed out in [8] that ECGI is a much harder problem than graph isomorphism problem.

The graph considered in ECGI is usually attributed relational graph[1]. The difference between ordinary graph and attributed relational graph is that both of the nodes and edges of attributed relational graph possesses weighted values. Hence, graph isomorphism problem can be regarded as a special case of ECGI problem. The algorithm for solving ECGI problem can also be applied to solve graph isomorphism problem.

2.1. **Earlier Approaches**

During the past decade, many algorithms including approaches of relaxation methods[6][7], tree search[1][8][9], and algebraic analysis[10] have been developed to solve ECGI problem.

*A. Relaxation method*

This method employs a set of locally interacting parallel processes that in an iterative fashion update assignments in order to achieve a global optimization. It is tolerant to noise. However, it is difficult to determine a suitable local compatibility measure. Besides, it is a local optimization aproach in intrinsic and fails to converge sometimes.

*B. Tree Search*
Tree search method is a global search strategy which will guarantee the finding of optimal solution. Earlier tree search approaches include backtracking[8], branch-and-bound[1][9]. Unfortunately, since they are combinatorial search methods, the NP optimization problem of ECGI will need exponential time to find the optimal solution. This impractical aspect makes such algorithms of small value which can only be utilized to those graphs containing no more than 10 nodes.

C. Algebraic analysis

By representing graphs as adjacency matrices, the algebraic theory, such as eigendecomposition, can be utilized to analyze the ECGI problem. In [10], an approximate solution close to the optimal one can be found when the graphs are sufficiently close to each other. However, if graphs are not sufficiently close to each other, this method sometimes fails to give the optima.

In this paper, we will propose a novel approach that adopts genetic algorithms combined with local search techniques to solve the problem of error-correcting graph isomorphism.

2.2. Problem Statement

In order to state the problem, the terminology of attributed relational graph[1] has to be defined first. An attributed relational graph $w$ over $Nv \cup Ne$ is a 4-tuple $w(V, E, \alpha, \beta)$

where

- $V$ is a finite nonempty set of vertices;
- $E \subseteq V \times V$ is a set of distinct ordered pairs of $V$ called edges;
- $\alpha : V \rightarrow Rv$ is a function whose range $Rv$ is a set of weighted values of vertices;
- $\beta : E \rightarrow Re$ is a function whose range $Re$ is a set of weighted values of edges.

The adjacency matrix of an attributed relational graph $G$ with $n$ vertices is an $n$ by $n$ matrix $M = [m_{ij}]$ such that
$$m_{ij} = \begin{cases} 
\alpha(v_i), & \text{if } i = j \\
\beta(v_i, v_j), & \text{if } i \neq j 
\end{cases}$$ (1)

Permutations of rows and of the corresponding columns in adjacency matrix imply reordering of the vertices. Hence two graphs $G$ and $H$ are isomorphic if and only if their adjacency matrices $M(G)$ and $M(H)$ differ only by permutations of rows and columns, i.e., $M(G)$ and $M(H)$ are related with a permutation $\sigma$:

$$M(H) = P \cdot M(G) \cdot P^T$$ (2)

where $P$ is a permutation matrix of $\sigma$. If we define the difference between two attributed relational graphs as follows:

$$J(\sigma) = \|M(H) - P \cdot M(G) \cdot P^T\|$$ (3)

where $\|$ is the matrix norm which will be discussed in Section 4.4, the error-correcting graph isomorphism problem can then be formulated as the search optimization problem in searching a permutation $\sigma$ that minimizes $J(\sigma)$. Since the search space of permutation is $n!$ for an $n$-node graph, any existing combinatorial search method cannot find optimal isomorphism within polynomial execution time. However, the implicit parallelism of genetic algorithms provides a possible improvement of search performance for obtaining optima.

3. Genetic Algorithms

Genetic algorithms are a type of stochastic search method inspired by natural population genetics to evolve solutions to problems. They have been applied to a wide range of optimization problems, including well-known NP-complete and NP-hard problems, scheduling and routing, configuration, and layout design[3]. They were modeled and developed by Holland in 1975[5] to mimic the processes found in natural evolution.
In genetic algorithms, a population is a set of "chromosomes" representing possible solutions. Given a (random) initial population, genetic algorithms use genetic operators to alter chromosomes in population through which a new generation is created. This process is repeated until a predetermined stop condition is satisfied. The detail descriptions of genetic algorithms are discussed below.

Basically, a genetic algorithm can be represented as a set of parameters \((M, \Delta, \mathbb{R}, f, \mathcal{S}, N)\), where \(M\) is the encoding scheme which is a mapping from domain elements to coding values, \(\Delta\) is the genetic operators applied to evolutionary search, \(\mathbb{R}\) is the rates controlling the frequencies with which genetic operators are applied, \(f\) is the fitness function to evaluate the fitness value of chromosomes in population, \(\mathcal{S}\) is the stop condition, and \(N\) is the population size. The performance of genetic algorithms vary with the implementation parameters. Previous researchers have empirically attempted to find reasonable parameter settings for genetic algorithms\([11][12]\). The detailed discussion of these six parameters will be given in the following paragraphs.

A. Encoding Scheme

Encoding is a mapping from knowledge domain to the solution space where genetic algorithms can process. The selection of encoding scheme is varied with the design decision and also depends on the problem to be solved. It will affect the selection of genetic operators. Improper encoding scheme will produce infeasible chromosomes generated by genetic operators. Usually, chromosomes are represented as fixed length strings coded with a binary character set. Other types of encoding schemes include real number representations and permutation representations. A detailed discussion of permutation representation will be left in Section 4 because it is adopted as the encoding scheme in our approach.

B. Genetic Operators

The role of genetic operators are to create new search chromosomes by altering the composition of parent chromosomes. Fundamental genetic operators include
reproduction, crossover, and mutation. Additional advanced genetic operators have been inspired by knowledge derived from the field of genetics (e.g., inversion dominance and abeyance)[3]. In the following discussion, we will elaborate on major operators such as selection, crossover, and mutation.

**Selection.** The selection operator forms a new population by selecting chromosomes in the old population based on their fitness value. The rationale is that the chromosomes with higher fitness value should have a higher probability of surviving into next generation. Existing selection operators include proportional and ranking selections[13].

**Crossover.** Selection directs the search toward the best existing chromosomes but does not create any new chromosomes. Crossover operator, on the other hand, takes valuable information from both parent chromosomes and then combines them to find a highly fit chromosome. Current crossover operators include single-point, multi-point, and uniform crossover[3].

**Mutation.** As its name implies, the role of mutation is to keep the diversity of population. For bit representation of chromosome, the mutation operation reverses a single bit from 1 to 0 or from 0 to 1.

**C. Rate of Operator**

Most of genetic operators, excluding selection, have an occurrence rate that controls the frequency with which the operator is applied. If the crossover rate is too low, the search may stagnate due to the lower exploration rate. The higher the crossover rate, the more quickly new chromosomes will be introduced into the population. As to the mutation, it should be used sparingly because it is a random search operator. With high mutation rates, the algorithm would become little more than a random search.

**D. Fitness Function**
The fitness function gives each chromosome a fitness value which is a judgment of its surviving capability. Choosing and formulating an appropriate fitness function is crucial in obtaining efficient solution for those problems solved by genetic algorithms. A more detailed discussion on our proposed efficient fitness functions will be presented in next section.

**E. Stop Condition**

Genetic algorithms improve the whole system by eliminating the chromosomes that do not adjust to the environment among the population and by creating new chromosomes through the genetic operators. The iterative elimination and creation process proceeds and needs to stop. When to stop the evolution is dependent on the implemented domain and the design decision. Since performance is usually a critical consideration, limited time or generation number is commonly adopted as the stop condition.

**F. Population Size**

The population size affects both the ultimate performance and the efficiency of genetic algorithms. A small population size will reduce the evaluation cost in each generation. For large population size, genetic algorithms can perform a more informed search because a large population is more likely to contain representatives from a large number of hyperplanes. The advantage of large population size implies disadvantage of small population size.

### 4. Genetic-based Search For Error-Correcting Graph Isomorphism

The descriptions in previous section are the guidelines for applying genetic algorithms on an application. Some of the parameters, such as encoding scheme and fitness function, are application-specific and need special design to work on ECGI
problem. Moreover, to implement genetic algorithms in solving ECGI problem, it is a reasonable expectation to include the characteristics of graph to improve the performance of genetic algorithms. In this section, a genetic-based search algorithm for solving ECGI problem is proposed. Shown in Fig. 3 is a flowed illustration of the proposed algorithm. The grayed blocks in Fig. 3 will be explained in the following subsections.

4.1. Permutation Representation and Operators

As discussed in Section 2, permutation is the elemental operation in the solution space of ECGI problems. That is, permutation is the most appropriate encoding scheme for ECGI. However, since the chromosome (solution) represented as permutation is not the most commonly used bit string representation, it is not amenable to traditional genetic operators. Some permutation crossover operators, such as partially matched crossover (PMX), order crossover, and cycle crossover, have been introduced to preserve legality of offspring[3]. The key difference between these operators is the information which each attempts to preserve during recombination[14]. Since in our preliminary study their contribution to the convergence of the algorithm makes no apparent difference, PMX operator is adopted in our algorithm. Under PMX, two parent permutations are aligned, and two crossing sites are picked uniformly at random along the permutations. The corresponding elements between crossing sites are matched as interchanging pairs. The children are then generated by interchanging these pairs. An example of PMX is illustrated in Fig. 4(a).

The mutation operators that have been discussed in previous section operate on single bit of chromosome one at a time. When chromosome is represented as permutation, we cannot operate on a single bit of a permutation because this would produce an illegal permutation. A usually adopted permutation mutation operator, called SWAP mutation, swaps two randomly selected elements of chromosomes. Shown in Fig. 4(b) is an example illustrating SWAP mutation.
4.2. Local Search Strategies

The reason for combining genetic algorithms with local search strategies is that they compliment each other. Local search is fast but not guarantee the optimality of solution, while genetic algorithms’ global search is slower but has more chance to obtain optima. To achieve the efficiency goal, we can create a hybrid scheme that combines both of local search and global search. In our approach, two local search strategies are devised and incorporated with genetic algorithms. One is status matching which utilizes the heuristic of graph to seed a non-randomly generated candidate chromosome in initial population. Another is first-mutation which explores chromosomes in the immediate search space vicinity of the fittest chromosome in the current population.

Status Matching. Let \( G \) be an attributed relational graph. The status \( s(v) \) of a vertex \( v \) in \( G \) is defined as:

\[
s(v) = \alpha(v) + \sum_{v' \in v} \beta(v, v'),
\]

which is the sum of its attribute and the weighted values of edges which are from \( v \) to other vertices in \( G \). The status list \( S(G) \) is a list of status of vertices in \( G \). The canonical status list \( \text{CanS}(G) \) is a sorted status list in ascending order and is defined formally as below:

\[
\text{CanS}(G) = \{ s_i | \forall j > i, s_j > s_i \}
\]

where \( s_i \) and \( s_j \) are the status in positions \( i \) and \( j \). A permutation \( \sigma_{\text{CanS}(G)} \) corresponding to \( \text{CanS}(G) \) is defined as:

\[
\sigma_{\text{CanS}(G)}(i) = t, \; v_t \in G \text{ and } s(v_t) = s_i
\]

An example illustrating the concept of status is shown in Fig. 5.
Status is an attribute belonging to each vertex and is not changed even after reordering of vertices. It is an invariant which is informative for finding isomorphism. The status matching $p_{G,H}$ of graphs $G$ and $H$ is defined as:

$$p_{G,H}(i) = t, \ v_i \in G, v_i \in H, \sigma^{-1}_{\text{CanS}(G)}(i) = \sigma^{-1}_{\text{CanS}(H)}(t)$$

(7)

where $\sigma^{-1}$ is an inverse mapping of permutation $\sigma$. $p_{G,H}$ is also a permutation. However, it is a mapping of vertices between $\text{CanS}(G)$ and $\text{CanS}(H)$. The finding of $p_{G,H}$ has the time complexity of order $n \log n$. The critical step in finding $p_{G,H}$ is to sort status list into canonical form which needs at least $n \log n$ for any sorting algorithm.

For graph isomorphism problem which is the case without noise, status matching can almost obtain the optima immediately except for few situations where more than one nodes in the graph have the same status. For the case of ECGI, the result of status matching has lower chance to be an optima due to the perturbation of noise. However, the result of status matching still gives a good solution which is near the optimal solution.

Since the result of status matching is inspiring, it can substitute for a candidate chromosome in the initial population of genetic algorithms. Even if it is not the optima, the following genetic search would go toward the optima quickly from this good solution. Hence, this "seeded" chromosome can help genetic algorithms to initially search the local region that is the most probable one and thus accelerate the convergence speed.

First-mutation. The first-mutation strategy concentrates on only the best solution in the population and always takes a SWAP mutation on the best solution at each generation. It explores chromosomes in the immediate search space vicinity of the fittest chromosome in the current population. It is somewhat similar to the hill-climbing scheme except that we don't search all neighbors. The hill-climbing of a permutation needs to search $n(n-1)/2$ neighbors by swapping two elements of the permutation. On the other hand, using SWAP mutation operation in genetic algorithms is also a local search just like hill-climbing. However, it would search the
vicinity of the permutation with a predefined, usually low, mutation rate. Therefore, first-mutation is a compromise between hill-climbing and genetic algorithms' mutation scheme.

4.3 Inhibitive Selection Operator

The goal of selection operator is to select fitter chromosomes as the population of next generation. According to natural selection, individuals containing more highly fit encodings have a greater probability of reproducing than those having weaker encodings. Among current selection schemes in genetic algorithms, such as proportional and ranking selection, the principle that the fitter survive is preserved. But certain difference exists in the definition of selection probability for each chromosome in the population.

However, the principle that the fitter survive brings the problem of premature convergence in genetic algorithms. Premature convergence has been recognized as a serious problem in genetic algorithms. It occurs when the evolutionary process of genetic algorithms leads to dominant some suboptimal prior to the discovery of the optimal or satisfied solution. It has been observed that this problem is closely tied to the problem of losing diversity in the population[15][16].

In this paper, inhibitive selection operator is proposed to prevent premature convergence by keeping diversity. Suppose \( P' = (a'_1, a'_2, \ldots, a'_\lambda) \in I^\lambda \) denotes the population at generation \( t \), where \( \lambda \) is the population size and \( I \) is the space of chromosomes \( a'_i \). An inhibitive selection operator is defined as:

\[
\exists k \in \{1, 2, \ldots, \lambda - 1\} \quad \forall a'_i \in I', \quad |\text{class}(a'_i)| \leq k \quad (8)
\]

where \( \text{class}(a'_i) \) is the set of chromosomes having the same value of fitness function as \( a'_i \), and \( |\text{class}(a'_i)| \) is the number of \( \text{class}(a'_i) \).

Equation (8) does not concern the selection probability of each chromosome in the population. The probability can be determined by either existing proportional or ranking method. The parameter \( k \), called inhibition value, ensures that for every
chromosome in the population, its copies in the population is limited within $k$ no matter how it is fitted. Following the inhibition scheme, more chromosomes can be included into the population such that the diversity can be kept.

4.4. Fitness Function

In genetic algorithms, fitness function is the survival arbiter of chromosomes. In the ECGI problem, the objective is stated as finding a permutation $\sigma$ that minimizes $J(\sigma)$. However, the objective of fitness function in genetic algorithms should be to find the maximum of profit. Therefore, the fitness function $f(\sigma)$ is defined by an evaluation-to-fitness transformation which is commonly expressed as:

$$f(\sigma) = C_{\text{max}} - J(\sigma) = C_{\text{max}} - \left\| M(H) - P \cdot M(G) \cdot P^T \right\|$$

where $C_{\text{max}}$ is the maximum of $J(\sigma)$. Although the fitness function contains matrix multiplication $P \cdot M(G) \cdot P^T$, it is not difficult to see that this $O(n^3)$ operation can be reformulated as

$$P \cdot M(G) \cdot P^T = \left[ m_{\sigma(i),\sigma(j)} \right], M(G) = [m_{i,j}]$$

due to the specialty of permutation matrix. The fitness function of ECGI algorithm is then of $O(n^2)$ complexity. If the norm $\| \|$ in equation (9) is replaced with $L_1$ norm ($\| M \| = \sum \sum |m_{ij}|$), a fitness function $f'$ is obtained:

$$f'(\sigma) = C'_{\text{max}} - \sum_{i=1}^{n} \sum_{j=1}^{n} |m_{i,j} - m_{\sigma(i),\sigma(j)}|$$

Another fitness function $f''$ can be obtained by using Euclidean norm ($\| M \| = \sqrt{\sum \sum m_{ij}^2}$):

$$f''(\sigma) = C''_{\text{max}} - \left( \sum_{i=1}^{n} \sum_{j=1}^{n} (m_{i,j} - m_{\sigma(i),\sigma(j)})^2 \right)^{1/2}$$

The effectiveness of $f'$ and $f''$ will be discussed by experimental results in the next section.
5. **Experimental Results**

In this section, experiments are conducted to testify the advantages of our proposed approach. All algorithms in the experiments are implemented with C++ language and the running platform is Intel 486SX machine. The generation of graphs, adopted genetic algorithms' parameters, and experimental results will also be explained in this section.

In our experiments, the attribute values of vertices and edges are produced by a pseudo random number generator which assigns an integer value in the range from 0 to 100. Edges with attribute 0 mean that they do not exist. For each randomly generated graph $G$, a graph $H$ is created by modifying the graph $G$, i.e., by shuffling the order of vertices and adding uniformly distributed noise in the range of $-\varepsilon$ to $+\varepsilon$ to the attributes of graph $G$.

Graph pairs of different orders and different noise levels are used as input to the algorithm. The orders of 10, 15, and 20 have been tested. For each order, noises of amounts $\varepsilon=0, 5, 10, 15, 20$ are generated. 50 pairs of graphs are produced for each distinct order and noise level.

It is noted that the tuning of some genetic algorithms' parameters, such as maximum generation number and rates of operators, will influence the performance of the algorithm. However, the finding of optimal genetic algorithms' parameters was not the objective of this work. Hence, some initial experiments are needed to tune the parameters. Due to the robustness of genetic algorithms with respect to parameter values, reasonable parameter settings ensuring good results and convergence can be found very quickly. The used parameters are fixed for different graph orders and are shown in Table 1.

In addition to the parameters listed in Table 1, another criteria deciding the stop condition of the algorithm is used for different noise levels. For graph pairs with $n$ vertices and at $\varepsilon$ noise level, the expected mean of $J(o)$ is $\frac{\varepsilon^2}{2} n^2$ for L1 norm, and $\frac{\varepsilon^2}{3} n^2$ for Euclidean norm. If the proposed genetic-based algorithm proceeds to the
situation that its $J(\theta)$ is less than its expected mean value, it would also stop since the optimal or near optimal solution is obtained even if the iteration number does not go beyond the maximum.

In the experiments, results generated by different ECGI algorithms and results generated by two different fitness functions in the proposed ECGI algorithm are compared. The comparisons are achieved by recording two performance indices of each experiment: running time in seconds and correctness rate.

In the comparisons, three algorithms are included. They are branch-and-bound tree search[9], eigendecomposition method[10], and the proposed genetic-based search algorithm. They are abbreviated to Tree, Eigen, and GA respectively in Tables and Figures. Since branch-and-bound tree search is guaranteed to obtain the optimal solution, its generated results are adopted as the correct solutions. However, for noise levels 15 and 20 in order 15 and all noise levels in order 20, correct solutions cannot be obtained due to exponential running time and memory requirement of the branch-and-bound search algorithm. The solutions of these graph pairs are then obtained from the vertex-shuffling in the generation of these graph pairs. Besides, it is noted that for the proposed algorithm the results of repeated tests may be different for the same graph pair. Hence, in the experiments of the proposed algorithm, each graph pair is tested three times. The running time and the correctness rate are the average of the results through three tests.

All of the experimental results are listed in Tables 2 and 3. The illustrations in Fig. 6 and Fig. 7 are to compare these ECGI algorithms. It is obvious that for isomorphism problem which has $\varepsilon=0$, the proposed genetic-based search algorithm can find optimal solution with the least time. Even for $\varepsilon=5$, the proposed algorithm is still very fast in finding optimal solutions. The increase of running time of the proposed approach for different noise levels is satisfactory which is bounded with polynomial time. In the experiments, our approach always has higher correctness rate than eigendecomposition approach. The difference is especially apparent in higher noise levels.
It is clear that the running time and correctness rate of our algorithm can be adjusted by tuning the parameters in Table 1. For example, to elevate the correctness rate of order 10 with $\varepsilon=20$, a larger iteration number 200 can be adopted, although this will increase the running time.

Figs. 8 and 9 are illustrated to show the performance of the proposed approach with two different fitness functions, $f'$ in equation (11) and $f''$ in equation (12). It is obvious that $f'$ is superior in both running time and correctness rate. The reason why $L_1$ norm is better than Euclidean norm is that the square computation in Euclidean norm strengthens some elements too much. From equation (9), we know that the lower norm will produce higher fitness. However, for two graphs with just two exchanged vertices, the differences in exchanged rows and columns may be large. The square computation of the large differences in Euclidean norm will produce lower fitness than others which have more than two exchanged vertices. Therefore, the lower Euclidean norm doesn't mean the higher fitness. In this way, some better solutions with less permutations of rows and columns would be excluded from the population. In the theory of genetic algorithms, the fitness function that can't really reflect the phenomenon of fitness is called deceptive[3] since certain hyperplanes guide the search toward some solution that is not globally competitive. Although $f''$ acts worse than $f'$ due to its deceptive nature, it still has competitive performance with eigendecomposition approach.

6. Conclusions

In this paper, we propose a genetic-based search algorithm for finding error-correcting graph isomorphism. To solve this problem by genetic algorithms, the encoding scheme, corresponding crossover and mutation operators are discussed. Moreover, there are many proposed modifications to improve the performance of the algorithm. The characteristic of graphs, called status matching, has been adopted as a
local search strategy in our approach. It is also illustrated that fitness functions using different matrix norm have great influence on the convergence of algorithm.

By empirical comparisons, our proposed algorithm exhibits the superiority over the traditional combinatorial search algorithms and other approximation methods. Within satisfied running time, our approach can find optimal solution with higher probability than the compared eigendecomposition method.

For more pragmatic applications, the restriction of error-correcting graph isomorphism that the vertex number should be equal will seem to be impracticable. The problem of finding matching of graph pair with nonequal number of vertex is called error-correcting subgraph isomorphism[17]. The subgraph isomorphism problem has been proved as NP-complete [2] and error-correcting subgraph isomorphism is an error-correcting version of it. On the other hand, The matching between hierarchical graphs where the vertices may contain an attributed graph[18] will introduce more complicated problem. In the future, we plan to extend the proposed genetic-based search algorithm further to these topics.
REFERENCES


Fig. 1. An example of representing a Chinese character by attributed relational graph. The two graphs at the right hand correspond to the two characters respectively. Each vertex in the graph represents a stroke of the character. Attributes of vertices indicate the length of strokes. Attributes of edges denote distances between the center of stroke.

The error-correcting graph isomorphism

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>16</td>
<td>10</td>
<td>13</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>12</td>
<td>16</td>
<td>10</td>
<td>13</td>
<td>20</td>
<td>18</td>
</tr>
</tbody>
</table>

Fig. 2. An example of error-correcting graph isomorphism.
Fig. 3. The proposed genetic-based search algorithm for error-correcting graph isomorphism.
<table>
<thead>
<tr>
<th></th>
<th>2 1</th>
<th>3 4 5</th>
<th>6 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>4 3</td>
<td>1 2 5</td>
<td>7 6</td>
</tr>
</tbody>
</table>

three interchanged pairs: 3 ⇔ 1     4 ⇔ 2     5 ⇔ 5

<table>
<thead>
<tr>
<th></th>
<th>4 3 1 2 5 6 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>2 1 3 4 5 7 6</td>
</tr>
</tbody>
</table>

(a)

\[
7 6 5 4 3 2 1 \quad \Rightarrow \quad 7 3 5 4 6 2 1
\]

(b)

Fig. 4. Permutation operators. (a) PMX, (b) SWAP mutation
Fig. 5. An illustration of status. (a) Two graphs, (b) adjacency matrices of (a), (c) status lists, (d) canonical status lists and their corresponding permutations, (e) result of status matching.
Fig. 6. Running time (a) for \( n = 10 \), (b) for \( n = 15 \), (c) for \( n = 20 \).
Fig. 7. Correctness rate (a) for n=10, (b) for n=15, (c) for n=20.
Fig. 8. Running time (a) for n=10, (b) for n=15.

Fig. 9. Correctness rate (a) for n=10, (b) for n=15.
### Table 1
**Genetic Algorithms’ Parameters.**

<table>
<thead>
<tr>
<th></th>
<th>n=10</th>
<th>n=15</th>
<th>n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration no.</td>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>Population size</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Selection</td>
<td>Ranking</td>
<td>Ranking</td>
<td>Ranking</td>
</tr>
<tr>
<td>Inhibition value</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Crossover</td>
<td>PMX</td>
<td>PMX</td>
<td>PMX</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Mutation</td>
<td>SWAP</td>
<td>SWAP</td>
<td>SWAP</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

### Table 2
**Running CPU Time (Seconds).**

<table>
<thead>
<tr>
<th></th>
<th>n=10</th>
<th>n=15</th>
<th>n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>Eigen</td>
<td>GA (f')</td>
<td>GA (f'')</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0.05</td>
</tr>
<tr>
<td>5</td>
<td>11.2</td>
<td>6</td>
<td>2.74</td>
</tr>
<tr>
<td>10</td>
<td>20.4</td>
<td>6</td>
<td>5.39</td>
</tr>
<tr>
<td>15</td>
<td>35.5</td>
<td>6</td>
<td>6.73</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>6</td>
<td>8.87</td>
</tr>
</tbody>
</table>

### Table 3
**Correctness Rate.**

<table>
<thead>
<tr>
<th></th>
<th>n=10</th>
<th>n=15</th>
<th>n=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>Eigen</td>
<td>GA (f')</td>
<td>GA (f'')</td>
</tr>
<tr>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>90.0</td>
<td>98.0</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>54.0</td>
<td>82.7</td>
</tr>
<tr>
<td>15</td>
<td>100</td>
<td>26.0</td>
<td>72.0</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>12.0</td>
<td>54.0</td>
</tr>
</tbody>
</table>