IOP Conf. Series: Materials Science and Engineering **231** (2017) 012050 doi:10.1088/1757-899X/231/1/012050

An Improved Heuristic Method for Subgraph Isomorphism Problem

Yingzhuo Xiang[1], Jiesi Han[1], Haijiang Xu[2], Xin Guo [3]

[1] National Key Laboratory of Science and Technology on Blind Signal Processing, Chengdu, China
[2]Jiangnan Institute of Computing Technology, Wuxi, China
[3]School of Mathematical Sciences of UESTC, Chengdu, China

Abstract. This paper focus on the subgraph isomorphism (SI) problem. We present an improved genetic algorithm, a heuristic method to search the optimal solution. The contribution of this paper is that we design a dedicated crossover algorithm and a new fitness function to measure the evolution process. Experiments show our improved genetic algorithm performs better than other heuristic methods. For a large graph, such as a subgraph of 40 nodes, our algorithm outperforms the traditional tree search algorithms. We find that the performance of our improved genetic algorithm does not decrease as the number of nodes in prototype graphs.

1 Introduction

Subgraph isomorphism plays an important role in computer vision, pattern recognition, bioinformatics [2], and even in chemistry [1]. Formally, two graphs *G* and *G'* are said to be isomorphic if there is a one-to-one correspondence between their vertices and between their edges such that incidence relationship is preserved [4]. If the isomorphism is encountered between a graph and a subgraph of another larger graph, then it is called subgraph isomorphism or graph monomorphism [3].

Subgraph isomorphism problem has been studied for decades. [5] proposed a tree search algorithm and cut down the search space to accelerate the speed. [6] is considered the most efficient subgraph isomorphism algorithm which set lots of rules in order to cut the searching tree smaller. VF2 algorithm has many advantages, such as the cost function is strongly related to the size of the subgraph. However, when the subgraph is very large, the performance decreased seriously. Subgraph isomorphism is a generalization of both the maximum clique problem and the problem of testing whether a graph contains a Hamiltonian cycle, and is therefore NP-complete [3]. While tree search based algorithms are dedicated to solve the small subgraph isomorphism problem, some heurist method can deal with this NP-complete problems, such as simulated annealing idea of Kirkpatrick [7] and genetic search [8] devised based on the nature genetic processes. Rather than being motivated by the heat-bath analogy of simulated annealing, genetic search appeals to ideas concerning chromosomal evolution, which offers certain attractive computational features.

In the next section we will give a SI problem modelling. We will introduce the improved genetic algorithm base on this model in section 'Improved Algorithm'. Experiments and analysis will be arranged in the succeed section. At the end of this paper, we make a conclusion and summarize all this work.

2 Problem Modelling

We model the graph as an adjacency matrix [9], as this is a very effective approach to describe a graph. The matrix contains only '1's and '0's. Each row and column represents a node in the graph. Moreover, the '1' in a row represent the node connect to the other node, while the '0's indicate no connections. For a graph $G=(V_g, E_g)$ and a subgraph $H=(V_h, E_h)$, the adjacency matrix of G and H are $G=[gi, j]$ and $H=[hi, j]$, respectively. The number of nodes in G and H are N_g and N_h. We define a transfer matrix [5] P to be $N_g(rows)\times N_h(columns)$ matrix whose elements are only '1's and '0's, such that each row contains exactly one '1'. No column of matrix P contains more than one '1'. Thus if there exists a subgraph isomorphism, the adjacency matrix G and H satisfied Eq. 1.

$$P^T * G * P = H \qquad (1)$$

Therefore, we transfer the SI problem to the problem of finding the transfer matrix P.

Improved Algorithm

Genetic algorithm contains encoding, population generating, crossover, mutation, selection and evaluation. In this section, we introduce the improved algorithm emphasizing the crossover part and evaluation part.

2.1 Encoding

In order to use GA algorithm to solve the problem, the first step is to encode the target. We have modelled the SI problem as finding a P matrix that satisfies Eq. 1. Here we need to encode the P matrix to be easier to calculate. We encode P as a vector V of length N_h, and a value i at position t indicating a node number, is an integer less than or equal to N_g. For example, $V[i]=j$ is equivalent to $P^T[i, j]=1$. Obviously, the permutation vector V and transfer matrix P can derive from each other. They are equivalent. Thus, we can say that encoding as vector V satisfies completeness, soundness and non-redundancy [10].

2.2 Population Generation

Each individual of the population encodes as a permutation vector V. We use a random function to select N_h numbers of a permutation of N_g in order to guarantee the diversity of the population. Repeat this process until the population size big enough. In this paper, we set the population size as 9500.

2.3 Crossover

The crossover operation swaps parts of two parents in the population to generate children, which constitute the next generation. Generally, we hope that the children we derive inherit good parts of their parents. There have been many algorithms proposed for crossover. Such as partially matched crossover [11], ordered crossover [12], uniform partially matched crossover [13], and other algorithms based on these three. However, these excellent crossover algorithms do not work well in this situation, because they expect sequence individuals of indices while the permutation vector V is partial sequence. For clarity, let us consider an example with N_h=5, N_g=8. Fig. 1 demonstrates the improper of the UPMX algorithm in SI problem. This algorithm leads to that there are two '8' in 'Child2', which does not satisfy the permutation.

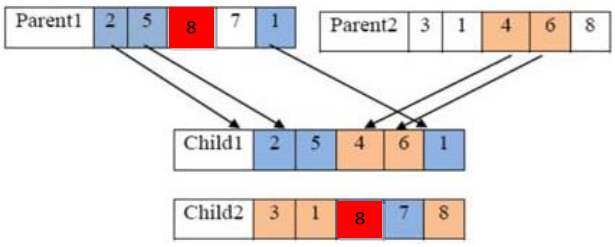

Figure 1: the UPMX crossover process

We propose a non-sequence uniform partially matched crossover (NUPMX) algorithm to solve this problem. The algorithm is depicted in Table 1.

2.4 Mutation

When designing a genetic algorithm, mutation appears to be among the most important operations to escape local optima since it preserves the diversification in the population [14]. We randomly shuffle the attributes of the individual to get the mutant. [15]

2.5 Evaluation and Selection

To evaluate the fitness of each individual, we need to give each individual a comparable fitness score. Here we proposed Eq. 2 as the fitness.

$$fitness = Vo*(P^T * G * P - H)*Vo^T \quad (2)$$

Vo is a $1\times N_h$ vector composed with all '1's. Fitness is a nature number that means the number of nodes that does not matched. If fitness equals zero, we say that *H* is a subgraph of *G*, and transfer matrix is the mapping function. The smaller the fitness is, the more similar graph *H* is to the subgraph of *G*. Therefore, in the selection algorithm, we try to select individuals with smaller fitness. In this paper, we use a *K* tournaments method [16] according to the fitness function Eq. 2.

Table 1: NUPMX Algorithm

Input parameters: ind1: The first individual participating in the crossover. ind2: The second individual participating in the crossover. Indpb: the probability to execute the swap
1. *for* each indices in ind1 and ind2: 2. *if* random() < indpb: # execute the swap 3. *if* ind1(i) and ind2(i) both exist in ind1 and ind2: 4. swap ind1(position(ind2(i)), ind2(position(ind1(i)) 5. swap ind1(i) , ind2(i) 6. *elseif* ind1(i) does not exist in ind2, ind2(i) exist in ind1: 7. ind1(position(ind2(i))=ind1(i) 8. swap ind1(i), ind2(i) 9. *elseif* ind2(i) does not exist in ind1, ind1(i) exist in ind2: 10. Ind2(position(ind1(i))=ind2(i) 11. swap ind1(i), ind2(i) 12. *elseif* ind1(i) and ind2(i) both do not exist in ind1 and ind2: 13. swap ind1(i), ind2(i) algorithm end
Output parameters: ind1: The first child after the crossover. ind2: The second child after the crossover.

Now we have introduced the genetic algorithm. We encode this problem as optimizing a permutation vector to minimize Eq. 2. Moreover, we propose a NUPMX crossover algorithm to solve this problem. In the next section, we perform some experiment to test this improved genetic algorithm.

3 Experiment

We evaluate our improved genetic algorithm with some randomly generated graphs and make a cooperation to the tree search algorithm VF2 and another genetic algorithm ILS proposed in [17]. We compare the average deviation of solution value to the optimal solution (in percent). Results are shown in Table 2.

Table2: The compare of three algorithms

	VF2	GA*ILS	GA*NUPMX
Nh=13 Ng=18	0	0.71%	0
Nh=19 Ng=24	Inf	2.06%	0.91%
Nh=25 Ng=30	Inf	4.50%	2.71%

As VF2 can only deal with small graphs, when the number of node is larger than 19, VF2 takes unendurable time. The GA*NUPMX algorithm performs better than GA*ILS, in every dataset. Although our GA*NUPMX algorithm takes a little more time to derive the optimal solution, the error rate is much lower than GA*ILS.

We also test our algorithm on a larger graph dataset with more than 100 nodes in a graph. The results are shown in Fig. 2.

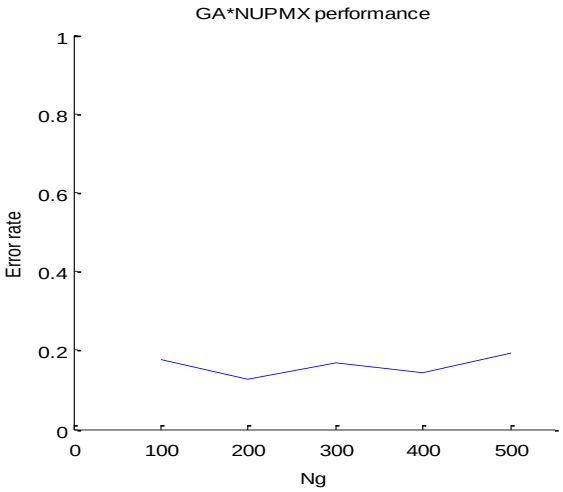

Figure 2: Performance with different size of Ng

The Fig.2 shows that the performance of GA*NUPMX is stable with the size of N_g less than 500. The error rate is no more than 20%. When the graph size is large enough, computation time takes longer than that when the size is smaller. Experiment also shows that the size of N_g has little influence to the performance of our algorithm.

4 Summary

In this paper, we propose GA*NUPMX algorithm to solve the subgraph isomorphism problem. We use the adjacency matrix to model the graph, and encode the problem as a permutation vector to make the problem easier for genetic searching. We also define an efficient fitness function for the optimization of this problem. Moreover, a dedicated crossover algorithm is proposed to raise the performance of genetic algorithm.

The proposed GA*NUPMX algorithm is tested on three randomly generated problem sets. Results show that our algorithm can test subgraph isomorphism in a large graph with 500 nodes. Though the cost of time is larger than small graphs, but it can still be far faster than VF2 and can give an answer of this NP-complete problem.

We will do some further experiment to figure out the influence of the size of initial population. Investigations to the other parameters of our algorithm should be carried out in the next work. The performance of our algorithm is expected to be improved in the future.

Acknowledgements
Thanks for the cooperation of the Jiangnan of Computing Technology and UESTC. Dr. Xu and Dr. Han help a lot in the experiment environment construction.

References

[1] Raymond, John W., and Peter Willett. "Maximum common subgraph isomorphism algorithms for the matching of chemical structures." Journal of computer-aided molecular design 16.7 (2002): 521-533.
[2] Huan, Jun, Wei Wang, and Jan Prins. "Efficient mining of frequent subgraphs in the presence of isomorphism." Data Mining, 2003. ICDM 2003. Third IEEE International Conference on. IEEE, 2003.
[3] Garey, Michael R., David S. Johnson, and Larry Stockmeyer. "Some simplified NP-complete graph problems." Theoretical computer science 1.3 (1976): 237-267.
[4] Fortin, Scott. "The graph isomorphism problem." (1996).
[5] Ullmann, Julian R. "An algorithm for subgraph isomorphism." Journal of the ACM (JACM) 23.1 (1976): 31-42.
[6] Cordella, Luigi P., et al. "A (sub) graph isomorphism algorithm for matching large graphs." IEEE transactions on pattern analysis and machine intelligence 26.10 (2004): 1367-1372.
[7] Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. "Optimization by simulated annealing." science 220.4598 (1983): 671-680.
[8] Fogel, David B. "An introduction to simulated evolutionary optimization." IEEE transactions on neural networks 5.1 (1994): 3-14.
[9] Cvetković, Dragoš M., Michael Doob, and Horst Sachs. Spectra of graphs: theory and application. Vol. 87. Academic Pr, 1980.
[10] Ono, Isao, Masayuki Yamamura, and Shigenobu Kobayashi. "A genetic algorithm for job-shop scheduling problems using job-based order crossover." Evolutionary Computation, 1996., Proceedings of IEEE International Conference on. IEEE, 1996.
[11] Goldberg, David E., and Robert Lingle. "Alleles, loci, and the traveling salesman problem." Proceedings of an International Conference on Genetic Algorithms and Their Applications. Vol. 154. Lawrence Erlbaum, Hillsdale, NJ, 1985.
[12] Golberg, D. E. "Genetic algorithms in search, optimization and machine learning reading." MA: Addisonn-Wisley, USA (1989).
[13] Cicirello, Vincent A., and Stephen F. Smith. "Modeling GA performance for control parameter optimization." Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Inc., 2000.
[14] Mühlenbein, Heinz. "How Genetic Algorithms Really Work: Mutation and Hillclimbing." PPSN. Vol. 92. 1992.
[15] Whitley, Darrell. "A genetic algorithm tutorial." Statistics and computing 4.2 (1994): 65-85.
[16] Deb, Kalyanmoy. "An efficient constraint handling method for genetic algorithms." Computer methods in applied mechanics and engineering 186.2 (2000): 311-338.
[17] Farahani, Mina Mazraeh, and Seyed Kamal Chaharsoughi. "A genetic and iterative local search algorithm for solving subgraph isomorphism problem." Industrial Engineering and Operations Management (IEOM), 2015 International Conference on. IEEE, 2015.