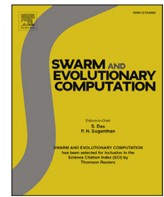
Swarm and Evolutionary Computation 49 (2019) 75–86

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo



Investigation of incremental hybrid genetic algorithm with subgraph isomorphism problem

HyukGeun Choi [a], Jinhyun Kim [b], Yourim Yoon [c], Byung-Ro Moon [a,*]

[a] School of Computer Science and Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, Republic of Korea
[b] Samsung Electronics, 129 Samsung-ro, Yeongtong-gu, Suwon-si, Gyeonggi-do, 16677, Republic of Korea
[c] Department of Computer Engineering, College of Information Technology, Gachon University, 1342 Seongnam-daero, Sujeong-gu, Seongnam-si, Gyeonggi-do, 13120, Republic of Korea

ARTICLE INFO

Keywords:
Incremental genetic algorithm
Hybrid genetic algorithm
Subgraph isomorphism problem

ABSTRACT

Graph pattern matching is a key problem in many applications which data is represented in the form of a graph, and this problem is generally defined as a subgraph isomorphism. In this paper, we analyze an incremental hybrid genetic algorithm for the subgraph isomorphism problem considering various design issues to improve the performance of the algorithm. An incremental hybrid genetic algorithm was previously suggested to solve the subgraph isomorphism problem and have shown good performance. It decomposes the problem into a sequence of consecutive subproblems which has an optimal substructure. Each subproblem is solved by the hybrid genetic algorithm and the solutions obtained are extended to be applied to the next subproblem as initial solutions. We examine a wide range of schemes that determine the overall performance of the incremental process and make a number of experiments to verify the effectiveness of each scheme with the synthetic dataset of random graphs. We show that the performance of incremental approach can be significantly improved compared to the previous representative studies by applying appropriate schemes found by the experiments. In addition, we also investigate the effect of different genetic parameters and identify the scalability of our method by conducting experiments using real world dataset with large-sized graphs.

1. Introduction

Graph is a simple and universal data representation to model pairwise relationships among a set of objects. One of the interesting problems encountered when handling graph data is a graph pattern matching arising from pattern recognition, knowledge discovery, biology, cheminformatics, dynamic network traffic and intelligence analysis [1–4]. And this matching is typically formulated in terms of the subgraph isomorphism problem.

Given two graphs G and H, the subgraph isomorphism problem is to determine whether H contains a subgraph that is isomorphic to G and this decision problem is well-known NP-Complete [5]. Many algorithms have been proposed to solve this problem starting with the backtracking algorithm by Ullmann [6]. VF2 [7], QuickSI [8], GraphQL [9], GADDI [10] and SPath [11] improved the performance by exploiting different join orders, pruning rules, and auxiliary information from the Ullmann algorithm [12]. The maximum common subproblem, the generalized problem of the subgraph isomorphism problem, also has been tackled by many algorithms [13–15]. However, these algorithms for both problems have exponential time complexity, their scalability are limited and they only work with auxiliary information such as vertex or edge labels.

On the other hand, metaheuristic algorithms, especially a genetic algorithm, have been used to address this problem [16–22]. They can usually find good quality solutions within a reasonable amount of time, but most algorithms does not have enough search capability to cover large and complex problem space of the subgraph isomorphism problem.

In order to address this problem effectively through a metaheuristic algorithm, it is necessary to improve the search capability of the algorithm. Generally, it is done by newly designing effective search operators or local heuristics to directly improve the search capability. However, this is very complex work. There is another way to improve the performance by reforming the search strategy to handle the algorithm instead. If we divide the whole problem space into subspaces with suitable size for the search capability of the algorithm, sequentially apply the method of searching one subspace and expanding the

* Corresponding author.
E-mail addresses: hgchoi@soar.snu.ac.kr (H. Choi), jh@soar.snu.ac.kr (J. Kim), yryoon@gachon.ac.kr (Y. Yoon), moon@snu.ac.kr (B.-R. Moon).

https://doi.org/10.1016/j.swevo.2019.05.004
Received 12 September 2018; Received in revised form 11 March 2019; Accepted 20 May 2019
Available online 23 May 2019
2210-6502/© 2019 Elsevier B.V. All rights reserved.

solutions obtained for the next space search, the solutions to the original problem can be obtained efficiently. This is the motivation of introducing the incremental genetic algorithm [23], and it has been successfully applied to the real world problem [24].

In this paper, we investigate the characteristics and performance of the incremental hybrid genetic algorithm (IHGA). First, the structure of the algorithm is described by a rigourous representation, and the theoretical background and conditions for the algorithm to work well are inspected. We define the process of dividing the original problem as a sequence of successive subproblems with optimal substructure, and clarify the solving process, that is, solving subproblems and extending the solutions. In addition, the conditions for finding high quality solutions in IHGA are revealed through the characteristics of the subgraph isomorphism problem and the population-based algorithm. The schemes for designing a sequence of subproblems, which is an important factor in determining the performance of IHGA, are also discussed. Second, for the synthetic dataset, we experimentally analyze the effect of designing a sequence of subproblems and extending solutions, which are the key elements of IHGA, on the performance of the algorithm. We also compare the performance and the running time of IHGA with those of the previous algorithms. Finally, we confirm the performance of IHGA with various genetic parameter settings and real world dataset with large-sized graphs. From the experiments with two datasets, we will show that our algorithm has better performance and scalability than previous algorithms.

The paper is organized as follows: Section 2 provides the formal definition of the subgraph isomorphism problem and the overview of previous studies. In Section 3, we describe the incremental process and related design issues. Genetic framework is introduced in Section 4. Section 5 provides the experimental results and discussions. Conclusions are in Section 6.

2. Backgrounds

2.1. Subgraph isomorphism

Definition 1 (Subgraph Isomorphism). *Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the subgraph isomorphism is an injective function $g : V_G \to V_H$ such that $(u, v) \in E_G$ if and only if $(g(u), g(v)) \in E_S$ where $S = (V_S, E_S) \subseteq H$. g is an induced subgraph isomorphism in addition if $(u, v) \notin E_G$, then $(g(u), g(v)) \notin E_S$.*

The difference between subgraph isomorphism and induced subgraph isomorphism is that, in induced subgraph isomorphism, the absence of an edge in G implies that the corresponding edge in H must also be absent.

2.2. Subgraph isomorphism problem

Given two graphs G and H, the subgraph isomorphism problem is to determine whether there exists a subgraph $S \subseteq H$ such that $f : V_G \to V_H$ is an isomorphism from G to S. This decision problem is a well-known NP-Complete problem [5].

The first practical algorithm for the subgraph isomorphism problem was proposed in Ullmann's research [6]. It is a recursive backtracking algorithm to find all subgraph isomorphisms between two graphs. After this study, several algorithms [7–11] have been proposed to enhance and to improve Ullmann's algorithm. These algorithms commonly improved the vertex join order and pruning rules to lop off infeasible candidates as early as possible. The subgraph isomorphism problem can be generalized to the maximum common subgraph problem, which finds the largest subgraph of two given graphs that are isomorphic to each other. The McGregor [13], Durand and Pasari [14] and Balas and Yu [15] proposed representative algorithms. However these algorithms have exponential time complexity, they have limited scalability or only work with auxiliary information such as vertex or edge labels.

Messmer and Bunke [25] introduced the method of decomposing model graphs into a set of subgraphs in advance. And then, when given the input graph, they recombined these small subgraphs into the complete subgraph isomorphism. They showed recombining the answers of small size problems recursively is helpful to build answers of the original problem.

2.3. Genetic algorithm for subgraph isomorphism problem

In order to apply the genetic algorithm to the subgraph isomorphism problem, it is necessary to introduce a fitness function and convert it to the form of an optimization problem, rather than addressing the decision problem of the subgraph isomorphism directly.

First, we describe the formal definition of the optimization problem for the subgraph isomorphism.

Definition 2 (Subgraph Isomorphism Problem). *Given two directed graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ where $|V_G| \leq |V_H|$, the subgraph isomorphism problem, denote by $\mathbf{SIP}(G, H)$, is to find an injective function $g : V_G \to V_H$ that minimizes the fitness function f. The optimal solution, with the fitness value of 0, is the subgraph isomorphism from G to H.*

The fitness function has been commonly defined as a number of edges that match or mismatch over the mapping. And real world applications have been studied by applying this fitness function into the subgraph isomorphism problem and solving it with a genetic algorithm. Brown et al. [16] applied it to 2D Chemical structure matching. Zhong et al. [21] used it for the resource assignment in the real time digital simulator and Kim and Moon [19] proposed the malware detection system by solving the subgraph isomorphism problem with this fitness function.

Because these algorithms showed limitation in terms of performance or scalability, several studies were proposed to improve performance of a genetic algorithm for the subgraph isomorphism problem. Choi et al. [26] introduced the multi-objective fitness function. They added the degree comparison result of the matched vertices between two graphs to the fitness function as a penalty function and showed that the multi-objective function is more globally convex than the previous single objective function. After then, Choi et al. [23] proposed a new hybrid genetic algorithm using an incremental process. Kim et al. [24] applied this methods to measure the source similarity and showed the similarity measure of this approach reflects the actual likeliness between the codes.

3. Incremental genetic algorithm

3.1. Overview

The incremental genetic algorithm (IGA) is a method of dividing a problem into successive subproblems and solving them sequentially to obtain solutions to the original problem [23]. In this process, each subproblem is solved by a genetic algorithm and the obtained solutions are extended as the initial solutions of the next subproblem. We explain in detail how this algorithm works for the subgraph isomorphism problem.

Let a substructure of a graph $G = (V, E)$ be a subset of V and their connected edges in the graph G. In this problem, it means a subgraph. On the same problem structure, if the input is changed from the original graph to its substructure, we call it a subproblem. For example, in the subgraph isomorphism problem, $\mathbf{SIP}(G', H)$ is a subproblem of $\mathbf{SIP}(G, H)$ where $G' \subseteq G$.

Consider a finite subsequence of n consecutive subproblems $\{\mathbf{SIP}(G_1, H), \mathbf{SIP}(G_2, H), \ldots, \mathbf{SIP}(G_n, H)\}$, where $G_1 \subseteq G_2 \subseteq \ldots \subseteq G_n(= G)$. Let G_1 be a sufficiently small subgraph of G and G_n be the same graph as G. In this case, if we add extra mappings to the solutions obtained after solving the subproblem $\mathbf{SIP}(G_i, H)$, these extended solutions are likely to be good-quality initial solutions for the next subprob-

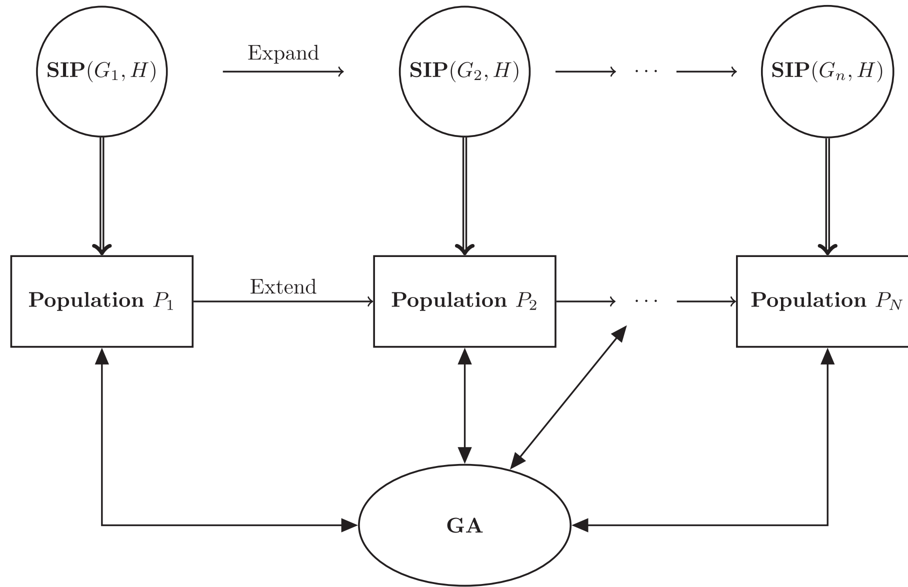


Fig. 1. Overview of the process of an incremental genetic algorithm.

lem **SIP**(G_{i+1}, H). If we repeat this task sequentially, we will eventually get the solutions of the original problem. Fig. 1 shows an overview of an incremental genetic algorithm.

The rationale behind IGA is that high-quality solutions of the one subproblem probably provide good initial points of the next subproblem [23,27–30]. If the order of the subproblems in the sequence are sorted by the graph size of the subproblems, the optimal solutions of the subproblem with small size graph can be found easily and these are expected to be extended as good solutions for the next subproblem.

To precisely illustrate these aspects, we say that a sequence of subproblems has an optimal substructure, if one of the optimal solutions of each subproblem could be extended to the optimal solution of the next subproblem. Existence of an optimal substructure conceptually explains that we are able to solve the original problem by enumerate all of the optimal solutions for each of the subproblems. Of course, the presence of an optimal structure does not guarantee that IGA can always find optimal solutions. Not every sequence having an optimal substructure leads to a good solution. If there exist a large number of optimal solutions for intermediate subproblems, and only a few of them are extendable to optimal solutions for the original problem, then IGA is less likely to find promising solutions.

Therefore, in order to increase the probability that IGA obtains an optimal solution, the solutions of one subproblem must be extended to a good initial population of next subproblem, and the following two conditions are required. The first is that the quality should be maintained even if the solution is extended. In the subgraph isomorphism problem, every sequence of the subproblems has the optimal substructure because the subgraph relation is transitive, i.e, if G_1 is a subgraph of G_2 and G_2 is a subgraph of G_3, then G_1 is a subgraph of G_3. This means that the quality of the existing part does not change when the solution is extended in this problem. This property makes the application of an incremental approach to the subgraph isomorphism problem more appropriate than other combinatorial optimization problems such as MAX-CUT or TSP. Second, the diversity of solutions must be ensured. If the solutions are converged in a specific space, it may become difficult to cover the increased space when the problem space is expanded for the next subproblem. If we intentionally maintain the population diversity, the solutions are more uniformly distributed over the expanded problem space, and the exploration and exploitation become much easier and this is why the population-based metaheuristic is suitable to solve these subproblems.

3.2. The structure of the algorithm

The key to determining the performance of IHGA is how to design a sequence of subproblems. First, the order in which vertices are added to expand a graph, determines the overall search path, which has already been identified in previous backtracking methods. Second, the expansion size between the subproblems, that is, a number of vertices to expand a graph from one subproblem to the next, determines the size of the problem space searched by GA. If we set the expansion size to be small, GA will be able to exploit the problem space in detail for each problem, but it takes a lot of time and is inefficient in terms of overall execution time. On the other hand, too large expansion size makes it difficult for GA to evolve previous solutions into good solutions. Therefore, it is necessary to set the appropriate expansion size and the order of subproblems considering the search ability of GA for each subproblem.

Algorithm 1 Incremental process for the subgraph isomorphism problem.
Input: $G = (V_G, E_G), H = (V_H, E_H)$
Output: An injective function $g : V_G \to V_H$
1: $V' \leftarrow$ Reordering (V_G)
2: $n \leftarrow$ the number of subproblems
3: $\mathbf{m} = \{m_1, \ldots, m_n\} \leftarrow$ the sequence of expansion size
4:
5: $G_0 \leftarrow \emptyset$
6: $P_0 \leftarrow$ random initial population of **SIP**(G_0, H)
7: **for** $i = 1$ **to** n **do**
8: $V_{curr} \leftarrow \{V'_1, \ldots, V'_{m_i}\}$
9: $V' \leftarrow V' - V_{curr}$
10: $G_i \leftarrow$ `inducedsubgraph`$G[V_{G_{i-1}} \cup V_{curr}]$
11: $P_i \leftarrow$ initial population generated by P_{i-1}
12: $P_i \leftarrow$ hybrid GA (P_i)
13: **end for**
14: **return** the best in P_n

We present the structure of IHGA for the subgraph isomorphism problem in Algorithm 1. The first half of the algorithm sets up the incremental process by decomposing the original problem into the sequence of consecutive subproblems. The sequence is determined by the number of subproblems, n, the number of vertices to be added at each subproblem, $\mathbf{m}$, and the order of vertices, V'. In line 1, the order of vertices

added is determined by the vertex reordering scheme. According to previous study [23], the vertex join order is one of the most important factors for the performance of an incremental genetic algorithm. The reordering schemes we used are described in detail in the next subsection. The number of subproblems and the expansion size of each subproblem are determined in line 2 and 3, which decide the problem space size of each subproblem. If we improve the performance of hybrid genetic algorithm, we can increase the size of problem space of each subproblem.

In the second half of the algorithm, it is shown how to solve the subproblems and extending solutions to obtain the original solution problems. We starts from the problem $\mathbf{SIP}(G_0, H)$ with an empty domain graph, $G_0 = \emptyset$. For every ith subproblem, we expand G_{i-1} into G_i by adding m_i vertices and edges between $V_{G_{i-1}}$ and new added vertices. And then, hybrid genetic algorithm takes the results of the previous subproblem to build an initial population of $\mathbf{SIP}(G_i, H)$ and evolves it over generations. The solution of the original problem $\mathbf{SIP}(G, H)$ is obtained from the results of the $\mathbf{SIP}(G_n, H)$.

Fig. 2 shows how the incremental process works. Consider two graphs in Fig. 2(a). In this case, the number of subproblems is 3 and the expansion size for each subproblem, m_i, is 1. Fig. 2(b) describes the second subproblem $\mathbf{SIP}(G_2, H)$ and the population P_2 evolved by hybrid GA. From the second subproblem $\mathbf{SIP}(G_2, H)$ and the population P_2, Fig. 2(c) shows how to obtain the graph G_3 from G_2 and the population P_3 from P_2 for initializing the third subproblem $\mathbf{SIP}(G_3, H)$. Graph G_2 is expanded into G_3 by adding one vertex 3 and two edges $(1, 3)$ and $(2, 3)$. And the population P_3 is initialized from P_2 by adding extra mappings for a new added vertex 3, which are denoted in bold.

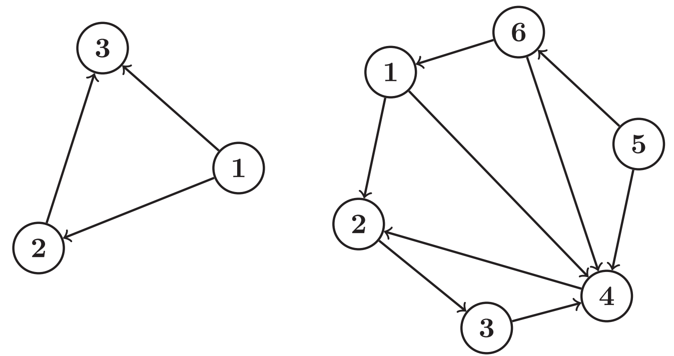
3.3. Vertex reordering

The vertex join order is one of the most important design issues to determine the performance of an incremental genetic algorithm. Well-designed ordering scheme can prune out infeasible problem space early in the search process and allow to select an efficient overall search path. Information on vertex adjacency and the degree of vertices has been used in the ordering method of graph based problem [12,31] and we applied three reordering schemes for the vertices of G from Ref. [23] and considered an ordering newly.

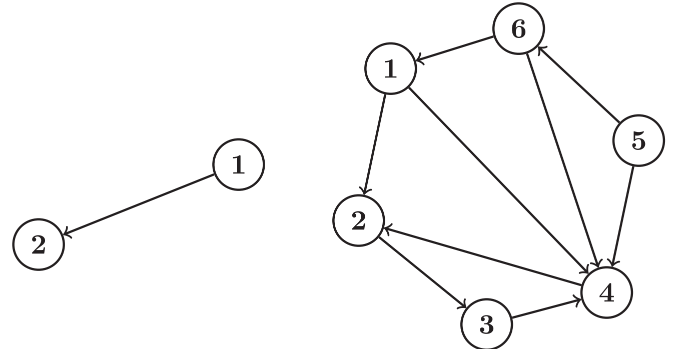
- Max-degree ordering (MD)
  We sort the vertices in non-increasing order of degree. The degree of a vertex is the sum of both ingoing and outgoing degrees.
- BFS ordering (BFS)
  We randomly select a starting vertex, and then run the breadth-first search on G. When the graph is disconnected and not all of the vertices are visited, we randomly choose another unvisited vertex and continue the traverse.
- Max-adjacency ordering (MA)
  We randomly select a starting vertex, and repeatedly select one of the most attractive vertex in a greedy manner. The attractiveness of a vertex v is the number of adjacent vertices that are already ordered. Two vertices are adjacent to each other if there is an edge in any direction.
- Max-degree-adjacency ordering (MDA)
  We add a tie-breaking rule to Max-degree ordering; in a case of a tie, the vertex having more adjacency to vertices in previous subproblem comes earlier in the ordering. This is a combination of the Max-degree ordering and Max-adjacency ordering for a synergy effect.

3.4. Stopping criterion

Basically, we use a fixed number of generations for all of the subproblems. But this may lead to an excessive number of generations in earlier subproblems, because hybrid GA may converge very fast for relatively small problems. Moreover, keeping some solutions that are not converged in the population may preserve the diversity of solutions. Both the quality and the diversity of population in the one subproblem can have a decisive effect on the next subproblem. So, if a certain percentage of the population for the one subproblem converges to the optimal solution, we terminate GA and move on the next subproblem. Initially, we regard $|V_H|$ generations as the unit of time and completely distributed a fixed number of

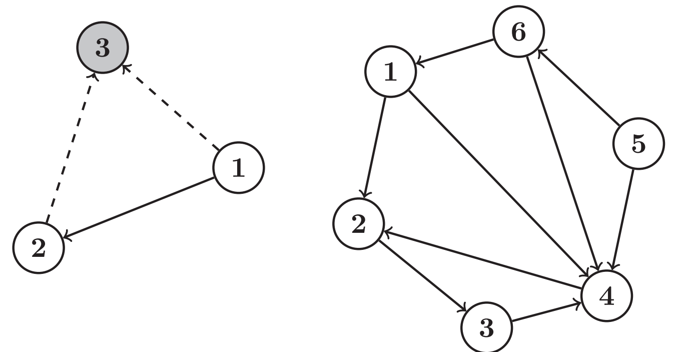


(a) Two graphs G and H



P_2^1	P_2^2	P_2^3	$\cdots$
$1 \to 2$	$1 \to 6$	$1 \to 4$	$\cdots$
$2 \to 3$	$2 \to 1$	$2 \to 2$	

(b) Population P_2 evolved in subproblem $\mathbf{SIP}(G_2, H)$



P_3^1	P_3^2	P_3^3	$\cdots$
$1 \to 2$	$1 \to 6$	$1 \to 4$	
$2 \to 3$	$2 \to 1$	$2 \to 2$	$\cdots$
$\mathbf{3 \to 4}$	$\mathbf{3 \to 5}$	$\mathbf{3 \to 3}$	

(c) Initial population P_3 of $\mathbf{SIP}(G_3, H)$ extended from P_2

Fig. 2. Description of expanding graph G_i and initializing population P_i in the incremental algorithm.

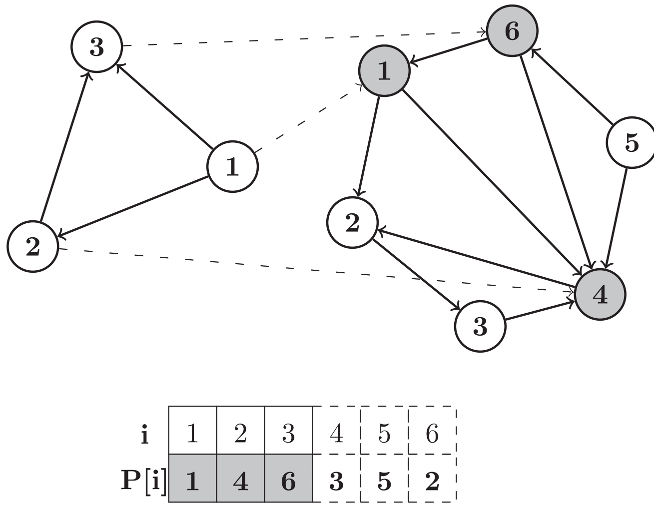


Fig. 3. Description of chromosome representation as a permutation. Each vertex i in G is mapped by a vertex $P[i]$ in H, drawn by dashed line.

generations equally to each subproblem. Before starting each subproblem, we redistribute the remaining generations equally to the remaining subproblems. By this procedure, we expect that more generations are assigned to the later larger subproblems to be evolved longer.

3.5. Expansion size

The number of vertices to be added also determines the problem space of each subproblem. A naive method is to add a single vertex at each step. But it will be a waste of time to run hybrid GA when the graph expanded is too simple. Adding more vertices at a step enables efficient space search, but an immoderate expansion size may cause the problem space to be too large for hybrid GA to cover. It is required to strike a balance between efficiency and difficulty by selecting an appropriate expansion size.

4. Genetic framework

The hybrid genetic algorithm we used in the incremental process for the subgraph isomorphism problem is described below.

- Representation
  Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ where $|V_G| \leq |V_H|$, a chromosome represents a permutation of V_H as an integer array. A mapping $g : V_G \rightarrow V_H$, a solution of SIP(V_G, V_H), is decoded by first $|V_G|$ genes in the chromosome. A vertex $v_{G,i} \in V_G$ is mapped to $v_{H,p[i]} \in V_H$ and an edge $(v_{G,i}, v_{G,j}) \in E_G$ is mapped to $(v_{H,p[i]}, v_{H,p[j]}) \in E_H$. Fig. 3 shows an example. The main advantage of this representation is the flexibility toward the problem size expansion. Since a chromosome already has a full permutation of V_H, we can easily crossover and mutate without extend the mapping at each subproblem without modifying values of the genes.
- Fitness Function
  We use the fitness function introduced in Ref. [26]. It combines a number of mismatched edges defined as

$$f_1 = \sum_{e \in E_G} I(e, E_S) + \sum_{e \in E_S} I(e, E_G)$$

where

$$I(e, E) = \begin{cases} 0 & \text{if } e \in E \\ 1 & \text{otherwise} \end{cases}$$

and a number of mismatched vertices

$$f_2 = \sum_{v \in V_G} J(v)$$

where

$$J(v) = \begin{cases} 0 & \text{if } d^+(v) \leq d^+(g(v)) \text{ and } d^-(v) \leq d^-(g(v)) \\ 1 & \text{otherwise} \end{cases}$$

and d^+ denotes outgoing-degree and d^- denotes incoming-degree. The fitness function is defined as

$$f = w \cdot f_1 + (1 - w) \cdot f_2$$

where $w = 10/|V_H|$. The optimal fitness value is zero and it means that GA has found a subgraph isomorphism from G to H.
The function f_1, typically used in previous studies [16,19,21], can not accurately evaluate invalid mappings which violate the degree constraint of subgraph isomorphism. Therefore, by introducing a penalty function, f_2, and using a multi-objective fitness function combined f_1 and f_2, a solution can be evaluated more precisely. Choi et al. [26] showed that the multi-objective fitness function f facilitates optimizing the fitness function and improves the performance of local heuristics and genetic algorithms, because this function transforms the fitness landscape more globally convex.
- Population management
  The population size of each subproblem in the incremental process is fixed to 100. The only initial population for the first subproblem is randomly generated. From the second subproblem, GA takes the population evolved in the previous subproblem as initial population of the current subproblem.
- Selection
  The tournament selection is used. We pick two chromosomes randomly and return better one with 80 percent of chance, otherwise return the worse one.
- Crossover and Mutation
  We used cycle crossover [32]. For the mutation, we select a number of genes to shuffle them in random order. Each of the gene independently has 40 percent of mutational chance.
- Local Heuristics
  We hybridize the local optimization algorithm with the GA, by applying the algorithm to the initial solutions at the first subproblem, and to the offsprings after crossover and mutation. We randomly swap two vertices when there is an improvement; this operation is repeated until there is no way to improve. The details are described in Algorithm 2.
- Replacement
  We generate 50 offsprings per generation and take 100 best solutions out of the existing solutions and the offsprings.
- Stopping Criterion
  The hybrid GA is terminated when a certain ratio of the solutions in the population becomes optimal solutions. We use the ratio values (*TH*) of 1%, 50%, and 100%. Regardless of this criterion, in the last step, in which, the subproblem is the same as the original one, the algorithm stops if it finds the optimal solution. If we set *TH* as ∞, the hybrid GA unconditionally executes a fixed number of generations and ends.

Algorithm 2 Vertex swap local optimization algorithm.
Input: A chromosome C of **SIP**(G,H)
1: $L \leftarrow \{(i,j) \mid 1 \leq i \leq |V_G|, i < j \leq |V_H|\}$
2: **repeat**
3: *flag* ← *false*
4: **for all** $(i,j) \in L$ in random order **do**
5: *swap*$(C[i], C[j])$
6: calculate the difference
7: **if** improved **then**
8: *flag* ← *true*
9: **else**
10: *swap*$(C[i], C[j])$ //cancel
11: **end if**
12: **end for**
13: **until** flag

5. Experimental results

We had experiments with the proposed incremental hybrid genetic algorithm for two kinds of datasets. The effects of design schemes proposed in Section 3 were verified and our algorithm was compared with previous algorithms on synthetic data consisting of random graphs. We also tested our algorithm for the real world data with large size graphs. All algorithms implemented with in C++ language, compiled by g++ 4.8.4 with -O3 option and executed with Intel Xeon CPU E5-2660 v3 @ 2.60 GHz and 1 GB memory.

5.1. Synthetic data

5.1.1. Dataset and evaluation

We generated random graphs by following a widely-used graph generation process for the subgraph isomorphism problem [26,33,34]. First, a graph H is generated by randomly selecting $\eta|V_H|^2$ directed edges among $|V_H|$ vertices without any other constraint, where η denotes the edge density of a graph H. And then, a smaller graph G is generated by sampling $|V_G|$ vertices from H and selecting the induced subgraph $H[V_G]$. This means that there is always a subgraph isomorphism from G to H and the optimal fitness function value is always zero.

We selected 20 classes using 4 kinds of η values of H and 5 kinds of $|V_G|$ values, and independently generated 10 pairs of graph instances for each class, so that a total 200 pairs of graph instances were used for our experiments. We chose 0.01, 0.05, 0.1 and 0.5 for η and 10, 30, 50, 70 and 90 for $|V_G|$. The number of vertices of the larger graph, $|V_H|$, was fixed to 100.

We conducted 1000 runs for each instance to test the algorithms and averaged the results of all instances in each class. We measured the average fitness value, the average running time, and the ratio of finding optimal solutions.

We compared the proposed incremental hybrid genetic algorithm (IHGA) with the conventional hybrid genetic algorithm without the incremental process (BASE).

5.1.2. Effect of vertex reordering

Table 1 shows the average fitness value of the conventional hybrid GA (BASE) and IHGA with five different reordering schemes. For each subproblem, the expansion size is set to 1, and the stopping criterion threshold TH is set to ∞, which means the hybrid GA is set to run for a fixed number of generations in each subproblem. The best results are shown in bold. Adding the incremental process to the hybrid GA with randomized vertex reordering degraded the performance in all of the 9 classes. On the other hand, the other schemes reflecting the information of the vertices show improved results, and we have found that the vertex ordering, which determines the order of expanding the graph G, is very important in the incremental process. The best reordering schemes for the incremental process is max-degree-adjacency (MDA), even showed better performance than the previous work [23]. We therefore will fix the reordering scheme as MDA in the rest of our experiments for the incremental process.

5.1.3. Effect of partial random initialization

Through the experiment in 5.1.2, we could conclude that performance improvement can be achieved by only applying the basic incremental process with the reordering scheme. In fact, in order to verify the effectiveness of the incremental process itself, we randomly initialized a certain ratio of solutions before running the hybrid GA in each subproblem. We used 0%, 10%, 20%, 30%, 40% and 50% for the ratios. Table 2 denotes the average fitness value of each partial randomization. For most of the classes, the random initialization degraded the quality of solutions. And through correlation coefficients between random initialization ratio and the average fitness value, it can be shown that the quality of solutions is declined more if more solutions are randomly initialized in each subproblem. Therefore, it is useful to evolve and extend solutions gradually and we will extend and reuse all solutions of the one subproblem as the initial solutions for the next subproblem.

5.1.4. Stopping criterion

Table 3 denotes the average fitness value of IHGA with three different stopping criteria for each subproblem. The stopping criteria were applied to the algorithm with two reordering schemes, RAND and MDA. For each reordering scheme, we also denoted the result of an algorithm when threshold value is ∞. We marked the best result in bold for each class.

In general, reducing the threshold value showed better performance for both reordering schemes. In the case with random reordering scheme, the case of lowest threshold value gave the best performance in all classes. When MDA reordering scheme was applied, there was no

Table 1
Comparative analysis of IHGA with different reordering schemes.

$\|V_G\|$	η	f average					
		BASE	RAND	BFS	MD	MA	MDA
10	0.05	0.0002	0.0026	0.0009	**0.0001**	0.0005	**0.0000**
	0.1	0.0002	0.0042	0.0006	**0.0002**	0.0007	**0.0002**
	0.2	0.0427	0.0962	0.0691	**0.0256**	0.0495	**0.0229**
30	0.05	0.2412	0.8088	**0.2106**	**0.0959**	0.0903	**0.0560**
	0.1	0.4723	1.1702	0.6226	**0.2622**	**0.2151**	**0.1973**
	0.2	0.0047	0.1227	0.0973	0.0049	0.0224	0.0062
50	0.01	0.0043	0.0899	**0.0005**	**0.0001**	**0.0006**	**0.0000**
	0.05	0.0015	0.1758	0.0483	0.0209	0.0112	**0.0000**
70	0.01	0.0126	0.0914	**0.0069**	**0.0013**	**0.0068**	**0.0001**

Table 2
Comparative analysis of IHGA with different partially random initialization ratio values.

$\|V_G\|$	η	f average						Corr.
		0%	10%	20%	30%	40%	50%	
10	0.05	0.0000	0.0001	0.0003	0.0004	0.0005	0.0003	0.8448
	0.1	0.0002	0.0002	0.0002	0.0002	0.0002	0.0003	0.2015
	0.2	0.0229	0.0225	0.0242	0.0234	0.0259	0.0274	0.8956
30	0.05	0.0560	0.0892	0.1445	0.2586	0.4503	0.6462	0.9573
	0.1	0.1973	0.2417	0.3454	0.5938	0.8790	1.2312	0.9619
	0.2	0.0062	0.0050	0.0165	0.0280	0.0702	0.1766	0.8537
50	0.01	0.0000	0.0002	0.0026	0.0101	0.0202	0.0318	0.9421
	0.05	0.0000	0.0057	0.0113	0.0458	0.1774	0.3129	0.8898
70	0.01	0.0001	0.0151	0.0374	0.0647	0.0847	0.1023	0.9972

Table 3
Results of di_erent stopping criteria.

$\|V_G\|$	η	f average				Running time (s)		
∞	100%	50%	1%	100%	50%	1%		
a RAND reordering								
10	0.05	0.0026	0.0003	0.0003	**0.0001**	0.20	0.19	0.16
0.1	0.0042	0.0006	0.0004	**0.0003**	0.33	0.28	0.20	
0.2	0.0962	0.0652	0.0583	**0.0478**	1.18	1.16	0.90	
30	0.05	0.8088	0.6528	0.6304	**0.5350**	6.40	6.45	6.35
0.1	1.1702	1.0699	1.0533	**0.9319**	6.81	6.79	6.16	
0.2	0.1227	0.1335	0.1483	**0.1034**	6.33	6.26	6.00	
50	0.01	0.0899	0.0464	0.0396	**0.0189**	12.89	13.21	11.26
0.05	0.1758	0.0973	0.1027	**0.0935**	15.00	14.86	14.07	
70	0.01	0.0914	0.0915	0.0915	**0.0731**	28.61	28.54	25.99
b MDA reordering								
10	0.05	0.0000	**0.0000**	0.0000	0.0001	0.17	0.16	**0.14**
0.1	0.0002	0.0001	**0.0000**	0.0001	0.25	0.23	**0.18**	
0.2	0.0229	0.0157	**0.0127**	0.0137	0.69	0.70	**0.40**	
30	0.05	0.0560	0.0406	0.0388	**0.0347**	3.96	3.74	**3.36**
0.1	0.1973	0.1718	0.1695	**0.1680**	4.25	4.19	**4.04**	
0.2	0.0062	0.0061	**0.0049**	0.0088	4.63	**4.58**	4.72	
50	0.01	0.0000	0.0000	0.0000	0.0000	9.89	9.04	**7.71**
0.05	0.0000	0.0008	0.0008	**0.0000**	11.96	11.92	**11.81**	
70	0.01	0.0001	**0.0001**	0.0001	0.0002	25.10	24.064	**22.27**

significant difference in the average fitness value between the case of 1% threshold value and that of 50% threshold value, but the case of 1% threshold value executed slightly more rapidly than the case of 50% threshold value. Since reducing the threshold value increases the diversity of solutions, focusing on exploration in intermediate steps seems to be more helpful than focusing on exploitation. Than evolving from a population full of local optima, it was better to evolve from a diverse population where only one of the solutions is locally optimal.

5.1.5. Expansion size

Table 4 shows the average fitness value and the average running time when different expansion sizes were applied. Since there are classes of graphs with $|V_G|$, we used expansion sizes less than or equal to five. We used MDA reordering scheme and set the stopping criterion threshold as 1%. Among the five different sizes, the best results were marked in bold for each class.

Although it takes a long time to run, decreasing the expansion size showed better results for the average fitness value. There was no big difference in the results of size 1 and size 2, but we chose size 1 because we saw a tendency to improve performance as the overall expansion size decreased.

Table 4
Comparative analysis of IHGA with different expansion sizes.

$\|V_G\|$	η	f average					Running time(s)				
		1	2	3	4	5	1	2	3	4	5
10	0.05	0.0001	**0.0000**	0.0001	0.0001	0.0002	0.14	0.09	0.09	0.08	**0.07**
	0.1	0.0001	0.0001	0.0001	**0.0000**	0.0002	0.18	0.11	0.12	**0.09**	0.10
	0.2	**0.0137**	0.0174	0.0233	0.0220	0.0375	**0.398**	0.42	0.53	0.49	0.83
30	0.05	0.0347	**0.0297**	0.0315	0.0386	0.0322	3.36	2.19	1.89	1.95	**1.78**
	0.1	0.1680	**0.1603**	0.1712	0.1685	0.1745	4.04	2.55	2.07	1.95	**1.69**
	0.2	0.0088	0.0024	0.0037	0.0025	**0.0012**	4.72	3.16	2.45	2.30	**1.81**
50	0.01	**0.0000**	0.0000	0.0000	0.0002	0.0002	7.71	4.34	3.33	2.92	**2.41**
	0.05	**0.0000**	0.0032	0.0025	0.0008	0.0016	11.81	6.34	4.83	4.14	**3.38**
70	0.01	**0.0002**	0.0006	0.0008	0.0011	0.0010	22.27	12.78	10.18	8.48	**7.09**

5.1.6. Overall result

We tested the incremental hybrid genetic algorithm using the schemes that showed the best results in the previous experiments. We selected MDA reordering, the stopping criterion with 1% threshold value, and the expansion size value of 1. All of the 20 classes were tested and the results of IHGA were compared to those of three metaheuristic algorithms, Kim et al. [19], Choi et al. [26], and Li et al. [22], which did not apply the incremental process.

Table 5 denotes the overall results. The average fitness value, the standard deviation of the values, the average running time in seconds and the ratio of runs in which an optimal solution has been found are shown in the table.

For the nine relatively difficult classes, the results of IHGA that were better than those of previous studies are shown in bold. The minimum ratio of finding an optimal solution, in the class that $|V_G|$ is 30 and η is 0.05 was dramatically increased to 95.61% by using IHGA. Compared with the result of Choi et al. [26] which showed the best performance among the previous algorithms, the overall ratios for the 9 difficult classes were also improved from 92.36% to 98.37% and from 96.56% to 99.27% for all classes by IHGA. The experimental results showed that the well-designed incremental process is helpful to improve the performance of the hybrid genetic algorithm and IHGA outperforms the previous studies.

In addition, we applied Wilcoxon's signed rank test at 5% significance level, one of the nonparametric statistical hypothesis tests, to compare the performance among the previous algorithms with the proposed algorithm, as guided in Ref. [35]. The test results are shown in Table 6. Except for the classes in which the previous algorithms find the optimal solution, we found that the p-value is smaller than 0.05 in 12, 8, and 11 classes, respectively. This confirms that IHGA improves performance in general, not only in certain examples.

One disadvantage of IHGA revealed through the experimental results was that it takes longer time to execute than the conventional hybrid GA. However, if we look at the results by class, we can see that the running times of typical GA were short for easy classes, but for the difficult classes, the running times of IHGA were shorter. Therefore, we performed two additional experiments to analyze the running time of the Choi et al. [23] and IHGA and the results are shown in Table 7.

First, we set 1% threshold value for the stopping criterion and counted the number of generations the two algorithms would terminate. In most classes, both algorithms found the optimal solution in a very short generations. Since even if the subproblems are easy, at least one generation should be executed for all subproblems. For this reason, the running time of IHGA seems to be longer. However, as the number of generations increases for difficult classes, there was a different tendency in running time. For difficult classes where the number of generations of the typical GA exceeded 30, such as the class (30, 0.05), IHGA found the optimal solution with a relatively small number of generations. In this case, IHGA is more effective in terms of running time, and the results reflecting these are shown in Table 5. This trend can be confirmed by the running time obtained when *TH* was set to ∞ so that the both algorithms run the same number of generations. Thus, although there are some penalties for running time for easy problems, we can conclude that the more difficult the problem is, IHGA shows better performance in terms of accuracy and running time than the previous works.

5.2. Real world data

5.2.1. Used dataset and common parameter setting

In previous experiments, we tested several design schemes to verify the effectiveness of the incremental process and compared the performance of IHGA with other algorithms. In this subsection, we tested the

Table 5
Performance comparison among four metaheuristic algorithms and IHGA.

$\lvert V_G \rvert$	η	Kim et al. [19]				Choi et al. [26]				Li et al. [22]				IHGA			
		f average	*f* SD	Time(s)	Ratio	*f* average	*f* SD	Time(s)	Ratio	*f* average	*f* SD	Time (s)	Ratio	*f* average	*f* SD	Time (s)	Ratio
10	0.01	0.0000	0.0000	0.03	100.00%	0.0000	0.0000	0.03	100.00%	0.0000	0.0000	0.0004	100.00%	0.0000	0.0000	0.09	100.0%
	0.05	0.0020	0.0447	0.05	99.80%	0.0002	0.0042	0.06	99.82%	0.0042	0.0201	0.0818	95.76%	**0.0001**	**0.0024**	**0.14**	**99.94%**
	0.1	0.0023	0.0479	0.08	99.77%	0.0002	0.0048	0.11	99.77%	0.0068	0.0252	0.1911	93.19%	**0.0001**	**0.0026**	**0.18**	**99.93%**
	0.2	0.4256	0.9002	0.55	79.17%	0.0427	0.0895	0.93	78.85%	0.1352	0.1353	1.5192	39.85%	**0.0137**	**0.0549**	**0.40**	**93.49%**
30	0.01	0.0059	0.0766	1.13	99.41%	0.0000	0.0000	0.39	100.00%	0.0029	0.0167	0.5067	97.14%	0.0000	0.0000	1.80	100.00%
	0.05	5.0088	7.0046	6.60	60.96%	0.2412	0.5020	7.09	78.73%	1.2355	0.6716	8.5546	19.61%	**0.0347**	**0.1818**	**3.36**	**95.61%**
	0.1	5.9865	16.2949	4.16	87.99%	0.4723	1.4629	6.31	90.43%	3.8778	2.3659	12.9296	25.65%	**0.1680**	**0.8954**	**4.04**	**96.56%**
	0.2	0.0226	1.5980	1.04	99.98%	0.0047	0.2350	2.52	99.96%	2.4753	4.9639	10.1112	80.04%	0.0088	0.3307	4.72	99.93%
50	0.01	2.2322	1.3002	15.19	4.80%	0.0043	0.0203	7.29	95.69%	0.0353	0.0718	6.3916	74.59%	**0.0000**	**0.0000**	**7.71**	**100.00%**
	0.05	8.6474	25.8336	10.50	89.78%	0.0015	0.1093	3.51	99.98%	1.5321	3.1726	8.1765	80.91%	**0.0000**	**0.0000**	**11.81**	**100.00%**
	0.1	0.0875	4.3795	3.72	99.96%	0.0000	0.0000	3.65	100.00%	2.6643	7.0309	11.8881	87.42%	0.0021	0.2070	13.95	99.99%
	0.2	0.0000	0.0000	1.51	100.00%	0.0000	0.0000	3.86	100.00%	0.6049	5.1455	7.9019	98.64%	0.0000	0.0000	18.08	100.00%
70	0.01	16.0792	5.2860	32.32	0.00%	0.0126	0.0351	14.48	88.01%	0.0000	0.0000	8.3505	100.00%	0.0002	0.0041	22.27	99.86%
	0.05	0.9251	14.4306	9.41	99.59%	0.0000	0.0000	2.76	100.00%	0.0000	0.0000	0.0842	100.00%	0.0000	0.0000	28.88	100.00%
	0.1	0.0000	0.0000	3.49	100.00%	0.0000	0.0000	3.92	100.00%	0.0000	0.0000	0.1969	100.00%	0.0000	0.0000	34.57	100.00%
	0.2	0.0000	0.0000	2.37	100.00%	0.0000	0.0000	6.52	100.00%	0.0000	0.0000	0.9863	100.00%	0.0000	0.0000	45.12	100.00%
90	0.01	8.9344	8.0415	52.60	14.52%	0.0000	0.0000	6.75	100.00%	0.0000	0.0000	0.6940	100.00%	0.0000	0.0000	46.09	100.00%
	0.05	0.0000	0.0000	4.93	100.00%	0.0000	0.0000	3.52	100.00%	0.0000	0.0000	0.0422	100.00%	0.0000	0.0000	56.26	100.00%
	0.1	0.0000	0.0000	3.63	100.00%	0.0000	0.0000	4.55	100.00%	0.0000	0.0000	0.0548	100.00%	0.0000	0.0000	68.98	100.00%
	0.2	0.0000	0.0000	3.43	100.00%	0.0000	0.0000	7.67	100.00%	0.0000	0.0000	0.0929	100.00%	0.0000	0.0000	91.07	100.00%

Table 6
Performance comparison by Wilcoxon's signed rank test among the previous algorithms with IHGA.

$\|V_G\|$	η	Kim et al. [19]	Choi et al. [26]	Li et al. [22]
10	0.01	–	–	–
	0.05	3.214e – 05	7.125e – 03	≈ 0
	0.1	5.672e – 06	1.732e – 03	≈ 0
	0.2	1.026e – 12	≈ 0	≈ 0
30	0.01	7.225e – 15	–	≈ 0
	0.05	≈ 0	≈ 0	≈ 0
	0.1	≈ 0	≈ 0	≈ 0
	0.2	0.7928	0.8174	≈ 0
50	0.01	≈ 0	≈ 0	≈ 0
	0.05	≈ 0	7.865e – 03	≈ 0
	0.1	2.274e – 03	0.8414	≈ 0
	0.2	–	–	≈ 0
70	0.01	≈ 0	≈ 0	5.141e – 08
	0.05	7.299e – 11	–	–
	0.1	–	–	–
	0.2	–	–	–
90	0.01	≈ 0	–	–
	0.05	–	–	–
	0.1	–	–	–
	0.2	–	–	–

Table 7
Running time analysis between the hybrid GA and IHGA.

$\|V_G\|$	η	The number of generations		Running time (s)	
		(When $TH = 1$)		(When $TH = \infty$)	
		Choi et al. [23]	IHGA	Choi et al. [23]	IGA
10	0.01	1.00	10.00	0.88	0.42
	0.05	3.11	10.37	1.21	0.61
	0.1	5.03	10.47	1.47	0.73
	0.2	42.64	21.38	2.10	0.97
30	0.01	4.13	30.02	6.16	3.79
	0.05	67.15	40.67	10.90	6.53
	0.1	37.34	38.21	14.36	7.30
	0.2	7.01	31.72	17.28	8.65
50	0.01	41.49	50.14	17.06	11.11
	0.05	8.53	51.07	29.28	16.89
	0.1	4.75	52.39	34.11	19.29
	0.2	2.02	51.66	46.40	24.78
70	0.01	34.92	70.33	40.71	30.80
	0.05	1.00	70.00	51.38	39.02
	0.1	1.00	70.00	59.25	46.42
	0.2	1.00	70.00	83.56	60.85
90	0.01	6.42	90.00	72.46	52.82
	0.05	1.00	90.00	77.13	63.38
	0.1	1.00	90.00	94.26	77.68
	0.2	1.00	90.00	128.24	102.89

performance and scalability of IHGA for the real world data consisting of large graphs. We also observed the performance of IHGA according to different genetic parameter settings. We collected 7 kinds of biological networks from Network Repository[1] [36], which have hundreds to thousand number of vertices. Table 8 shows the statistics of these biological networks.

The one whole network was set to H. And for one problem case of H and $|V_G|$, we independently generated 10 problem instances by sampling G in the same way as Section 5.1.1. We chose 50, 100 as $|V_G|$ and conducted 100 runs for each instance using IHGA and averaged the results of all instance in each problem case. We set MDA reordering for vertex ordering scheme and 1% threshold value for stopping criterion. And we measured the average fitness value and running time with 5 different expansion size, S, from 1 to 5.

[1] Network repository: biological networks, http://networkrepository.com/bio.php.

5.2.2. Result on biological networks

To verify the performance of IHGA on the realworld graphs, we compared the performance with VF2 algorithm [7], one of the representative algorithm for the subgraph isomorphism problem. Since VF2 algorithm is deterministic, it always finds the optimal solution no matter how long it takes. For each instance, the running time of VF2 was limited to twice the average running time of IHGA and the ratio of finding the optimal solutions and average execution time of VF2 algorithm were measured within this time. Table 9 shows the performance comparison between the two algorithms.

First of all, in terms of the average fitness value, IHGA found the optimal solution in most cases, even though the graphs G and H become larger. However, in terms of running time, increasing the expansion size shortened the time by 3–4 times. The instances of previous synthetic data were relatively easy to solve, so that reducing the expansion size and exploiting the subspace in detail improved the quality of solu-

Table 8
Statistics of 7 biological networks.

	Name	$\|V\|$	$\|E\|$	η	$d_{\max}$	d_{avg}	Assortativity
H_1	celegans	453	2025	0.0197796	237	8	−0.225821
H_2	diseasome	516	1188	0.00894107	50	4	0.0666456
H_3	SC-TC	636	3959	0.0196058	66	12	0.921112
H_4	DM-LC	658	1129	0.00522315	50	3	−0.121817
H_5	CE-GT	924	3239	0.00759569	151	7	−0.159339
H_6	grid-mouse	1455	3272	0.00311463	222	4	−0.153014
H_7	yeast	1458	1948	0.00183401	56	2	−0.209541

Table 9
The ratio of finding the optimal solutions and the average running time of IHGA and VF2 on biological networks.

$\|V_G\|$	ALG.	S	Ratio							Running time (s)						
			H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_1	H_2	H_3	H_4	H_5	H_6	H_7
50	IHGA	1	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	44.49	43.67	59.08	55.67	88.93	126.58	138.82
		2	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	25.16	24.34	33.30	30.65	49.42	66.87	73.58
		3	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	18.98	18.26	24.95	22.93	37.07	48.98	54.17
		4	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	15.88	15.06	20.55	18.93	30.68	40.04	44.29
		5	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	12.83	11.67	15.84	14.81	23.94	30.78	34.16
	VF2	–	30.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	17.98	0.012	2.43	0.0017	0.67	0.0022	0.0026
100	IHGA	1	29.90%	86.50%	99.80%	100.00%	100.00%	100.00%	100.00%	848.50	485.57	458.57	420.22	761.45	991.44	1077.79
		2	33.20%	85.80%	99.90%	100.00%	100.00%	100.00%	100.00%	738.14	359.40	261.41	233.04	457.14	527.73	577.75
		3	33.60%	86.30%	100.00%	100.00%	100.00%	100.00%	100.00%	717.83	313.23	194.85	173.07	360.90	388.41	427.29
		4	33.30%	85.40%	100.00%	100.00%	100.00%	100.00%	100.00%	677.70	288.77	148.49	130.57	288.89	289.60	321.39
		5	30.50%	85.60%	99.90%	100.00%	100.00%	100.00%	100.00%	686.64	272.82	125.71	108.24	258.06	241.78	267.37
	VF2	–	0.00%	80.00%	60.00%	90.00%	70.00%	100.00%	100.00%	–	218.34	233.77	43.75	344.04	0.0026	0.0026

tion. However, in this experiment, we found that it is efficient to set the search space larger for each subproblem because the instances are difficult and the problem space size is large. Although it took a long time to execute as $|V_G|$ grows in size, but we can shorten the running time while maintaining the quality of solutions well by increasing the expansion size.

Next, the execution of VF2 algorithm was divided into the cases where the optimal solution was found in a short time, or the case where the solution was not found even by using twice the IHGA running time. Experiments have shown that IHGA can find the optimal solution better and more stable than VF2 when the graph is large and complex. As a result, we have confirmed that our algorithm works well and has scalability for the real world data.

5.2.3. Investigation on different genetic parameter settings

Based on the previous experiments, we observed the relation between genetic parameters, such as population size, a number of offspring per generation, expansion size, and a number of generations, and the performance of IHGA through additional experiments. On the biological networks H_1 and H_2 with $|V_G| = 100$, we measured the ratio of finding the optimal solutions and the average running time with three different parameter settings based on the previous experiments.

First, we set the population size to 100, 200, and $|V_H|$, and generate half of population size offspring at each generation. Table 10 shows the result. We found that increasing the population size and hence a number of offspring resulted with respect to a slight improvement in average fitness value, but on the contrary, it needed a much longer computation time.

Second, performances of various settings were observed while changing a number of generations to 100, 200, and $|V_H|$. Table 11 shows the performance of IHGA according to a number of generation changes. In the first three parameter settings, we confirmed that reducing the number of generations shortens the running time, but reduces the performance as well. In the last three parameter settings, we recognized that there is a difference in the performance improvement of IHGA according to population change and generation change. Even if we created the same number of offspring during the whole process, increasing a number of generations resulted in more performance improvement than increasing population size.

To clarify this tendency, we observed performance by changing an expansion size and a number of generations to produce the same number of solutions during the whole incremental algorithm. As shown in Table 12, we could more reliably confirm the characteristics of IHGA found in Table 11 through experiments with settings that produce the same number of solutions. Rather than expanding each subproblem to a larger scale through a large-sized population, it has been shown that expanding the subproblem to a smaller scale and evolving the population gradually over generations will improve performance.

Table 10
Comparative analysis of IHGA with different population sizes.

	$\|V_H\|$	Population	offspring	Expansion size	Generation	Time (s)	Ratio
H_1	453	100	50	5	453	795.72	34.90%
		200	100			1400.78	49.10%
		453	226			2773.61	63.70%
H_2	516	100	50	5	516	588.82	84.50%
		200	100			616.55	87.00%
		516	258			1345.30	90.00%

Table 11
Comparative analysis of IHGA with different numbers of generations.

	$\|V_H\|$	Population	Offspring	Expansion size	Generation	Time (s)	Ratio
H_1	453	100	50	5	453	795.72	34.90%
		100	50	11	200	519.89	9.40%
		100	50	22	100	270.00	0.00%
		100	50	22	200	496.67	9.00%
		200	100	22	100	540.56	0.00%
H_2	516	100	50	5	516	588.82	84.50%
		100	50	12	200	218.33	74.00%
		100	50	25	100	138.41	65.30%
		100	50	25	200	206.76	73.80%
		200	100	25	100	262.09	67.30%

Table 12
Comparative analysis of different parameter settings which generate the same number of offspring during the whole process.

	$\|V_H\|$	Population	Offspring	Expansion size	Generation	Time (s)	Ratio
H_1	453	100	50	5	453	795.72	34.90%
		200	100	10	226	933.14	20.90%
		226	113	11	200	1118.42	14.70%
		453	226	22	100	1213.38	0.10%
H_2	516	100	50	5	516	588.82	84.50%
		200	100	10	258	411.79	81.90%
		258	129	12	200	510.38	77.60%
		516	258	25	100	618.87	70.30%

6. Conclusion

In this paper, we analyzed various aspects of incremental hybrid genetic algorithm for the subgraph isomorphism problem and improved the performance. The incremental algorithm starts from decomposing the original problem into a sequence of consecutive subproblems that satisfies the optimal substructure property. The each problem is solved by the hybrid genetic algorithm and the solutions obtained are extended as initial solutions for the next subproblem. And then we sequentially solve a sequence of subproblems in this way, finally obtain a solution of the original problem.

We noticed that how to design a sequence of subproblems is most important in the performance of the incremental algorithm, so that we introduced several design schemes. Vertex reordering determines the entire search path of the incremental genetic algorithm. Experimental results showed that the maximal adjacency-degree reordering scheme combining adjacency information and degree information has the best performance. Stopping criterion and expansion size determine the size of the problem space in which the hybrid genetic algorithm searches for each subproblem. The schemes to preserve the diversity of solutions in solving successive subproblems showed the best in the experiments. Extending the solutions from the previous subproblem showed better performance than initializing solutions partially in each subproblem. We confirm that it is important to continue succession of solutions in the incremental algorithm by this experiment. The reason why this approach takes more running time than the conventional hybrid genetic algorithm is that each subproblem must be solved at least on one generation, regardless of how easy the subproblem is. Based on this analysis, we have shown through experiments that the well-designed incremental hybrid genetic algorithm outperforms the previous algorithms. Furthermore, the proposed algorithm has also been confirmed that it has scalability through the experiments based on the real world data with a large-sized graph.

Although we traced the several design schemes for the incremental algorithm, there are still other issues that have to be figured out for better performance. As shown in the experiments, our methods for building a sequence of subproblems and tuning parameters are manually designed. In addition, scalability of the algorithm can be further extended by parallel processing using GPUs. Generally, if we generate and evaluate many offspring in parallel by using many GPUs in each generation, we can shorten the execution time and increase the scale of the problem accordingly. If we modify the algorithm in a way with divide and conquer method and expands the solutions in parallel rather than expanding our problems in sequence, we can expect additional performance improvement through parallel processing.

In the future, we are considering the adaptive features which are able to set parameters automatically and the dynamic configurations which can change the schemes in the process of solving the subproblems. Our final goal is to design a generic incremental algorithm that can be applied to arbitrary metaheuristic algorithms. Based on this paper, we will apply the incremental process to other metaheuristic algorithms in the future and will upgrade it in general form.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science ICT and Future Planning) (No. 2017R1C1B1010768), and the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2018-2017-0-01630) supervised by the IITP (Institute for Information & communications Technology Promotion).

References

[1] C.C. Aggarwal, H. Wang, Graph data management and mining: a survey of algorithms and applications, in: Managing and Mining Graph Data, Springer, 2010, pp. 13–68.
[2] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, Int. J. Pattern Recognit. Artif. Intell. 18 (03) (2004) 265–298, https://doi.org/10.1142/S0218001404003228, https://www.worldscientific.com/doi/pdf/10.1142/S0218001404003228, https://www.worldscientific.com/doi/abs/10.1142/S0218001404003228.

[3] B. Gallagher, Matching structure and semantics: a survey on graph-based pattern matching, AAAI FS 6 (2006) 45–53.
[4] D. Shasha, J.T.L. Wang, R. Giugno, Algorithmics and applications of tree and graph searching, in: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '02, ACM, New York, NY, USA, 2002, pp. 39–52, https://doi.org/10.1145/543613.543620.
[5] S.A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, ACM, New York, NY, USA, 1971, pp. 151–158, https://doi.org/10.1145/800157.805047.
[6] J.R. Ullmann, An algorithm for subgraph isomorphism, J. ACM 23 (1) (1976) 31–42, https://doi.org/10.1145/321921.321925.
[7] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE Trans. Pattern Anal. Mach. Intell. 26 (10) (2004) 1367–1372, https://doi.org/10.1109/TPAMI.2004.75.
[8] H. Shang, Y. Zhang, X. Lin, J.X. Yu, Taming verification hardness: an efficient algorithm for testing subgraph isomorphism, Proc. VLDB Endow. 1 (1) (2008) 364–375, https://doi.org/10.14778/1453856.1453899.
[9] H. He, A.K. Singh, Graphs-at-a-time: query language and access methods for graph databases, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, ACM, New York, NY, USA, 2008, pp. 405–418, https://doi.org/10.1145/1376616.1376660.
[10] S. Zhang, S. Li, J. Yang, Gaddi: distance index based subgraph matching in biological networks, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '09, ACM, New York, NY, USA, 2009, pp. 192–203, https://doi.org/10.1145/1516360.1516384.
[11] P. Zhao, J. Han, On graph query optimization in large networks, Proc. VLDB Endow. 3 (12) (2010) 340–351, https://doi.org/10.14778/1920841.1920887.
[12] J. Lee, W.-S. Han, R. Kasperovics, J.-H. Lee, An in-depth comparison of subgraph isomorphism algorithms in graph databases, Proc. VLDB Endow. 6 (2) (2012) 133–144, https://doi.org/10.14778/2535568.2448946.
[13] J.J. McGregor, Backtrack search algorithms and the maximal common subgraph problem, Software Pract. Ex. 12 (1) (1982) 23–34, https://doi.org/10.1002/spe.4380120103, https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.4380120103, https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380120103.
[14] P.J. Durand, R. Pasari, J.W. Baker, C.-c. Tsai, An efficient algorithm for similarity analysis of molecules, Internet J. Chem. 2 (17) (1999) 1–16.
[15] E. Balas, C.S. Yu, Finding a maximum clique in an arbitrary graph, SIAM J. Comput. 15 (4) (1986) 1054–1068.
[16] R.D. Brown, G. Jones, P. Willett, R.C. Glen, Matching two-dimensional chemical graphs using genetic algorithms, J. Chem. Inf. Comput. Sci. 34 (1) (1994) 63–70.
[17] M. Wagener, J. Gasteiger, The determination of maximum common substructures by a genetic algorithm: application in synthesis design and for the structural analysis of biological activity, Angew. Chem. Int. Ed. 33 (11) (1994) 1189–1192.
[18] H. Frhlich, A. Koir, B. Zajc, Optimization of fpga configurations using parallel genetic algorithm, Inf. Sci. 133 (34) (2001) 195–219.
[19] K. Kim, B.-R. Moon, Malware detection based on dependency graph using hybrid genetic algorithm, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10, ACM, New York, NY, USA, 2010, pp. 1211–1218, https://doi.org/10.1145/1830483.1830703.
[20] M.M. Farahani, S.K. Chaharsoughi, A genetic and iterative local search algorithm for solving subgraph isomorphism problem, in: 2015 International Conference on Industrial Engineering and Operations Management (IEOM), 2015, pp. 1–6, https://doi.org/10.1109/IEOM.2015.7093815.
[21] Q. Zhong, Z. Wu, L. Lin, Y. Zhang, J. Zhang, Computing resources assignment in rtds simulators with subgraph isomorphism based on genetic algorithm, in: 2011 4th International Conference on Electric Utility Deregulation and Restructuring and Power Technologies (DRPT), 2011, pp. 1144–1149, https://doi.org/10.1109/DRPT.2011.5994067.
[22] Z. Li, B. Chen, D. Che, Solving the subgraph isomorphism problem using simulated annealing and evolutionary algorithms, in: Proceedings on the International Conference on Artificial Intelligence (ICAI), the Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), Athens, 2016, pp. 293–299 copyright - Copyright The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) 2016; Document feature - Tables; Graphs; Equations; Diagrams; Last updated - 2016-07-26, https://search.proquest.com/docview/1806562933?accountid6802.
[23] H. Choi, J. Kim, B.-R. Moon, A hybrid incremental genetic algorithm for subgraph isomorphism problem, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14, ACM, New York, NY, USA, 2014, pp. 445–452, https://doi.org/10.1145/2576768.2598382.
[24] J. Kim, H. Choi, H. Yun, B.-R. Moon, Measuring source code similarity by finding similar subgraph with an incremental genetic algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, ACM, New York, NY, USA, 2016, pp. 925–932, https://doi.org/10.1145/2908812.2908870.
[25] B.T. Messmer, H. Bunke, Efficient subgraph isomorphism detection: a decomposition approach, IEEE Trans. Knowl. Data Eng. 12 (2) (2000) 307–323, https://doi.org/10.1109/69.842269.
[26] J. Choi, Y. Yoon, B.-R. Moon, An efficient genetic algorithm for subgraph isomorphism, in: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12, ACM, New York, NY, USA, 2012, pp. 361–368, https://doi.org/10.1145/2330163.2330216.
[27] G. Bakrl, D. Birant, A. Kut, An incremental genetic algorithm for classification and sensitivity analysis of its parameters, Expert Syst. Appl. 38 (3) (2011) 2609–2620. https://doi.org/10.1016/j.eswa.2010.08.051.
[28] N. Mansour, M. Awad, K. El-Fakih, Incremental genetic algorithm, Int. Arab J. Inf. Technol. 3 (1) (2006) 42–47.
[29] P. Vivekanandan, R. Nedunchezhian, A new incremental genetic algorithm based classification model to mine data with concept drift, J. Theor. Appl. Inf. Technol. 21 (2010) 36–42.
[30] A.S. Wu, H. Yu, S. Jin, K.C. Lin, G. Schiavone, An incremental genetic algorithm approach to multiprocessor scheduling, IEEE Trans. Parallel Distrib. Syst. 15 (9) (2004) 824–834, https://doi.org/10.1109/TPDS.2004.38.
[31] I. Hwang, Y.-H. Kim, B.-R. Moon, Multi-attractor gene reordering for graph bisection, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, ACM, New York, NY, USA, 2006, pp. 1209–1216, https://doi.org/10.1145/1143997.1144188.
[32] I.M. Oliver, D.J. Smith, J.R.C. Holland, A study of permutation crossover operators on the traveling salesman problem, in: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1987, pp. 224–230. http://dl.acm.org/citation.cfm?id42512.42542.
[33] P. Foggia, C. Sansone, M. Vento, A database of graphs for isomorphism and sub-graph isomorphism benchmarking, in: Proc. of the 3rd IAPR TC-15 International Workshop on Graph-Based Representations, 2001, pp. 176–187.
[34] D. Conte, P. Foggia, M. Vento, Challenging complexity of maximum common subgraph detection algorithms: a performance analysis of three algorithms on a wide database of graphs, J. Graph Algorithms Appl. 11 (1) (2007) 99–143.
[35] J. Derrac, S. Garca, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evolut. Comput. 1 (1) (2011) 3–18. https://doi.org/10.1016/j.swevo.2011.02.002.
[36] R.A. Rossi, N.K. Ahmed, The network data repository with interactive graph analytics and visualization, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 4292–4293. http://networkrepository.com.