International Journal of Grid and Distributed Computing
Vol. 9, No. 5 (2016), pp.241-264
http://dx.doi.org/10.14257/ijgdc.2016.9.5.21

Apply Partition Tree to Compute Canonical Labelings of Graphs

HAO Jian-Qiang[1, 2, a], GONG Yun-Zhan[1, b], Tan Li [2, c], and Duan Da-Gao[2, d]

[1]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China*
[2]*School of Computer Science and Information Engineering, Beijing Technology and Business University, 100048, China*
[a]*bshjq@vip.163.com*, [b]*gongyz@bupt.edu.cn*, [c]*tanli_913@sina.com and* [d]*duandagao@126.com*

Abstract

This paper establishes a theoretical framework by defining a set of concepts useful for classifying graphs and computing the canonical labeling Cmax(G) of a given undirected graph G, which including the partition tree PartT(G), maximum partition tree MaxPT(G), centre subgraph Cen(G), standard regular sequence SRQ(G), standard maximum regular sequence SMRQ(G), and so on. The implementations of algorithms 1 to 5 show how to calculate them accordingly. The worst time complexities of algorithms 1, 2, 4, and 5 are $O(n^2)$ respectively. The time complexity of Algorithm 3 is O(n). By Theorem 3, all leaf nodes of PartT(G) and MaxPT(G) are the regular subgraphs. By Theorem 4 and 5, there exists only one Cen(G) in G. Regular Partition Theorem 6 shows that there exists just one corresponding PartT(G), SRQ(G), MaxPT(G), and SMRQ(G). One can use Classification Theorem 7 to category graphs. Theorem 8 and 9 establish the link between the Cen(G) and the calculation of the first node u_1 added into MaxQ(G) corresponding to the canonical labeling Cmax(G) of G. Further, it utilizes the Cen(G) to calculate the first node u_1 added into MaxQ(G). The proposed methods can be extended to deal with the directed graphs and weighted graphs.

Keywords*: Degree partition, Canonical labeling, Partition tree, Regular graphs, Centre subgraph, Standard regular sequence*

1. Introduction

A canonical labeling [5, 9, 15, 20] is a unique string corresponding to a graph such that two graphs are isomorphic if and only if they have the same canonical labeling. Until now, the canonical labeling as the graph isomorphism problem yet is an unsolved problem in computational complexity theory such that no polynomial-time algorithm exists for calculating the canonical labeling of a graph.

Numerous methods have emerged to address the problem. However, Most of them use the partition refinement techniques[5, 9, 14] that play an important role for computing the canonical labeling of graphs. Moreover, different methods use distinct definitions for the canonical labeling [6, 7, 10, 12, 13]. [5] pioneered the establishment of a general group-theoretic approach to compute canonical labeling. However, combinatorial methods have worked well in various special cases. For random graphs, [4, 5] produce a canonical labeling with high probability. [2, 3] propose two corresponding logspace algorithms for partial 2- and 3-Trees.

Currently, Nauty [15–18] is the most popular and practical tools for looking the auto-morphism group and the canonical label of a graph. Nauty also requires an exponential time to compute the canonical labeling for a given Miyazaki graph [19]. [21] made improvements for dealing with the problem. Based on backtracking, individualization of vertices, and partition re finement, Bliss [8, 9] is an efficient canonical labeling tool for handling large and

ISSN: 2005-4262 IJGDC

sparse graphs. [11] can find the symmetry while calculating the canonical labeling. To fix the glitch of Nauty, Traces [20] uses the strategy of breadth-first search to find the automorphism group and the canonical labeling. Conauto [14] also uses the basic individualization/refinement technique and is very fast for random graphs and several families of hard graphs. For canonical labeling, [23] especially [18] gives a comprehensive discussion.

Nauty dominated the field for several decades. As a result, an in-depth study for the canonical labeling has been confined to the theoretical framework of Nauty. This means that people just follow the research trajectory of Nauty to extend and establish further study.

For all of theorems and lemmas not proven in this paper, we will provide the complete proofs in another article. Whether they exist or not it does not affect the establishment of the results of this paper. In the remainder, Section 2 describes the problems addressed by the present paper. Section 3 establishes some basic terminology and concepts. Section 4 describes the basic principles for computing *Cmax(G)*. Section 5 presents our algorithms for computing the *PartT(G)*, *MaxPT(G)*, *Cen(G)*, *SRQ(G)*, *SMRQ(G)*. Section 6 displays the implementation of our algorithms and evaluates our approach by giving many examples. Finally, Section 7 discusses conclusions and future work.

2. Problem Statement

This paper has two purposes. One is to establish a theoretical framework by defining a set of concepts useful for classifying graphs and computing the canonical labeling *Cmax(G)* of a given undirected graph G, which include the partition tree *PartT(G)*, centre subgraph Cen(G), standard regular sequence *SRQ(G)*, and so on. Based on the definitions, it studies how they relate to each other and the link between the *Cen(G)* and the calculation of the first node u1 added into *MaxQ(G)* corresponding to the canonical labeling *Cmax(G)* of G. The other is to apply the relationship between *Cen(G)* and the calculation of the canonical labeling *Cmax(G)* to compute the canonical labeling *Cmax(G)*.

Although the technique employed to construct *PartT(G)*, *Cen(G)*, and *SRQ(G)* belongs to degree partition technology, the degree partition technique used by the paper is essentially different from the partition refinement technique. Based on current knowledge, it is unknown whenever there exists a method that can systematically define and study the degree partition technique. In addition to, it is unknown whenever there exists a method that systematically uses the degree partition technique to calculate the canonical labeling of graphs.

In this paper, the definition of canonical labeling is completely different from the definition of Nauty. Throughout the paper, the canonical labeling *Cmax(G)* of a graph is the lexicographically largest code obtained by concatenating the rows of the upper triangular part of the associated adjacency matrix (see Definition 2). Unless by chance, the canonical labeling given by Nauty will be not a canonical labeling according to the definition offered in the paper.

Many algorithms give the definition of the canonical labeling that is the same as provided in the paper. However, their primary purpose is not to study how to create a canonical labeling, but for other purposes such as to mine the frequent subgraphs. As a result, these algorithms can only work for some restricted graph classes. Until now, based on the definition in this paper, a general algorithm for computing the canonical labeling is not available.

3. Terminology and Notation

This paper deals with finite undirected graphs with neither loops nor multiple edges. A graph consists of a set of vertices and a set of edges. For a graph $G = (V(G), E(G))$, let $V(G)$ and $E(G)$ denote the set of vertices of G and the set of edges of G, respectively. Any an edge $(u, v) \in E(G)$ connects two vertices $u \in G$ and $v \in G$.

Definition 1. Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be two undirected graphs with n nodes. If there exists a bijection function f: $V(G) \to V(H)$ such that $\forall(u,v) \in E(G)$ if and only if $(f(u), f(v)) \in E(H)$. Thus, we say f is an isomorphic map of $G \to H$. Furthermore, we say the graph G and H to be isomorphic, denoted by $G \cong H$. An isomorphic map f of G onto itself is said to be an automorphism of G.

Definition 2. Let $G = (V(G), E(G))$ be an undirected graph with n nodes whose adjacency matrix is $\mathbf{A}(G)=(a_{i,j})_{n\times n}$.To concatenate the rows of the upper triangular part of $A(G)$ according to the following order $a_{1,2}, a_{1,3}, \cdots, a_{1,n}, a_{2,3}, a_{2,4}, \cdots, a_{2,n}, \cdots, a_{i,i+1}, a_{i,i+2}, \cdots, a_{i,n}, \cdots, a_{n-1,n}$ forms a corresponding binary number $a_{1,2}a_{1,3}\cdots a_{1,n}a_{2,3}a_{2,4}\cdots a_{2,n}\cdots a_{i,i+1}a_{i,i+2}\cdots a_{i,n}\cdots a_{n-1,n}$ which is called a *canonical label* of G, denoted by $C(G)$ (see Figure.1).

$$\mathbf{A}(G) = \begin{pmatrix} 0 & \to a_{1,2} & \to a_{1,3} & \cdots & \cdots & \cdots & \cdots & \to a_{1,n} \\ a_{1,2} & 0 & \to a_{2,3} & \cdots & \cdots & \cdots & \cdots & \to a_{2,n} \\ \vdots & \cdots & \ddots & \cdots & \cdots & \cdots & \cdots & \downarrow \ \vdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \downarrow \ \vdots \\ a_{1,i} & a_{2,i} & \cdots & a_{i,i-1} & 0 \to & a_{i,i+1} & \cdots & \to a_{i,n} \\ \vdots & \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \downarrow \ \vdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \ddots & \to a_{n-1,n} \\ a_{1,n} & a_{2,n} & a_{3,n} & \cdots & \cdots & \cdots & a_{n-1,n} & 0 \end{pmatrix}$$

Figure 1. The Canonical Labeling of a graph *G*

The first row of the upper triangular part of $\mathbf{A}(G)$ is the canonical labeling piece 1 of $C(G)$, denoted by $C^1(G)$. Similarly, the second row of the upper triangular part is the canonical labeling piece 2 of $C(G)$, denoted by $C^2(G)$. $\cdots$. The $(n-1)$th row of the upper triangular part is the canonical labeling piece $n-1$ of $C(G)$, denoted by $C^{n-1}(G)$. It is clear that $C(G) = C^1(G)C^2(G)\cdots C^{n-1}(G)$. Given a permutation π of the vertices of G, one can obtain a canonical labeling $C(G)$ corresponding the π by Definition 2. Denote by $\mathbf{L}(G)$ the collection of all canonical labeling of G.

For every $C_1(G)$, $C_2(G) \in \mathbf{L(G)}$, assume that $C_1(G) = i_1 i_2 \cdots i_m$, $C_2(G) = j_1 j_2 \cdots j_n$ with $i_1, i_2, \cdots, i_m, j_1, j_2, \cdots, j_n = 0$ or 1. Let $\mathbf{X} = (i_1, i_2, \cdots, i_m)$ and $\mathbf{Y} = (j_1, j_2, \cdots, j_n)$. If $\mathbf{X} > \mathbf{Y}$ with respect to the lexicographic order (The rest is the same), then we call $C_1(G) > C_2(G)$. Otherwise, if $\mathbf{X} < \mathbf{Y}$, then we call $C_1(G) < C_2(G)$. Otherwise, if $\mathbf{X} = \mathbf{Y}$, then we call $C_1(G) = C_2(G)$.

It is clear that $(\mathbf{L}(G), \leq)$ is a well-ordered set, where $\leq$ denotes the less-than-or- equal-to binary relation on the set $\mathbf{L}(G)$ expressed as above. By the well-ordering theorem, it follows that $\mathbf{L}(G)$ has a maximum element, denoted by *Cmax(G)*.

The permutation of the n vertices of G corresponding to *Cmax(G)* are the *maximum node sequence*, denoted by *MaxQ(G)*. Likewise, the adjacency matrices of G corresponding to *Cmax(G)* are the *maximum canonical label matrix*, denoted by *Amax(G)*.

$C^1(G), C^2(G), \cdots, C^{n-1}(G)$ corresponding to *Amax(G)* are *maximum canonical label piece 1, 2, $\cdots$, $n-1$* of canonical labeling $C(G)$, denoted by $C^1_{\max}(G), C^2_{\max}(G), \cdots, C^{n-1}_{\max}(G)$, respectively. Based on the above definitions, the following equation holds.

$$C(G) = C^1_{\max}(G)C^2_{\max}(G)\cdots C^{n-1}_{\max}(G) \quad (1)$$

Theorem 1. Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be two undirected graphs with n nodes respectively. Their descending degree sequences are $d(G)$ and $d(H)$ respectively,

satisfying $|V(G)| = |V(H)|$, $|E(G)| = |E(H)|$, and $d(G) = d(H)$. Their adjacency matrices are $\mathbf{A}(G)$ and $\mathbf{A}(H)$ respectively. $G \cong H$ if and only if $Cmax(G)=Cmax(H)$.

A *MaxEm graph* is a graph with the greatest $C(G)$ that corresponds to a permutation of the vertices. Denote by $d_G(u)$ the degree of a vertex u in G, by $d(G) = (d_G(u_1), d_G(u_2),\cdots, d_G(u_n))$ the *degree sequence* of G, by $d_G(V_1) = (d_G(v_1), d_G(v_2),\cdots, d_G(v_m))$ the degree sequence of a subset $V_1 \subseteq V(G)$ with $v_i \in V_1$, $i = 1, 2,\cdots, m$, and by $d_G(H) = (d_G(w_1), d_G(w_2),\cdots, d_G(w_s))$ the *degree sequence* of a subgraph $H \subseteq G$ with $w_j \in V(H)$, $j = 1, 2,\cdots, t$, and omit the subscript G when no ambiguity can arise. Denote by $\Delta(G)$ the maximum degree of all vertices of a graph G. Throughout this paper, $S(G) = \{u|u \in V(G) \wedge d(u) = \Delta(G)\}$.

Unless otherwise specified, throughout this paper, the degree sequence is non-increasing. The distance $d(u, v)$ between any two vertices u and v is the number of edges on the shortest path from u to v. If $H \subseteq G$ is a regular subgraph of G and u is a node in H, then we call $d_G(u)$ the degree of H with regard to G, denoted by $D_G(H)$, and call $d_H(u)$ the degree of H, denoted by $D(H)$.

Definition 3. Let $G = (V(G), E(G))$ be an undirected graph with n nodes. The vertex-induced subgraph on $V_1 \subseteq V(G)$ of G is the subgraph with nodes set V_1 together with any edge whose endpoints are both in V_1, denoted by $G[V_1]$.

Definition 4. Let $G = (V(G), E(G))$ be an undirected graph with n nodes. Define the *degree partition* $\pi(G)$ of G to be the partition $\{V_1, V_2, \cdots, V_k\}$ of the nodes of G, where $V_i \subseteq V(G)$ with $1 \le i \le k$ meet all of the following conditions.

1. $\emptyset \notin \pi(G)$.
2. $V_i \cap V_j = \emptyset$ for $i \neq j$ with $1 \le i, j \le k$.
3. $V_1 \cup V_2 \cup \cdots \cup V_k = V(G)$.
4. $d(v) = d(w)$, for every $v, w \in V_i$ with $1 \le i \le k$.
5. $d(v) > d(w)$, for every $v \in V_i$, $w \in V_j$ with $1 \le i < j \le k$.

We call G the *partition father graph* of V_i for $1 \le i \le k$, denoted by $PFAG(Vi)$. We also call G the *father graph* of $G[V_i]$ for $1 \le i \le k$, denoted by $FAG(G[Vi])$. Conversely, every V_i with $1 \le i \le k$ is a *partition son* of G, denoted by $PSON(G)$. Every $G[V_i]$ with $1 \le i \le k$ is also a *Son* of G, denoted by $SON(G)$.

By Definition 4, it follows that $PFAG(V_i)$= G for $1 \le i \le k$.

Definition 5. Let $G = (V(G), E(G))$ be an undirected graph with n nodes and the *degree partition* $\pi(G)= \{V_1, V_2,\cdots, V_k\}$(see Definition 4). By Definition 4, it is clear that the vertex-induced subgraphs on $V_1, V_2,\cdots, V_k$ are $G[V_1], G[V_2],\cdots, G[V_k]$, respectively.

Define each vertex-induced subgraph $G[V_i]$ to be a *degree partition subgraph* of the first layer of G, and the nodes set V_i to be the *generating set* of $G[V_i]$ with $1 \le i \le k$. This process by which one can obtain $G[V_1], G[V_2],\cdots, G[V_k]$ is the *degree partition process* of the *first layer* of G. Define *degree partition subgraphs set* of the *first layer* of G to be the set composed by the subgraphs $G[V_1], G[V_2],\cdots, G[V_k]$, denoted by $G^{(1)}[d]$. The degree $d_H(v)$ of a node v in the *generating set* V_i is the *degree* of *generating set* V_i with $1 \le i \le k$, denoted by $d(V_i)$, where $H = PFAG(V_i)$.

Based on the *degree partition* of the first layer of G, one can recursively continue to perform the *degree partition* of $G[U_1], G[U_2],\cdots, G[U_l]$ where $G[U_1], G[U_2], \cdots, G[U_l]$ are nonregular subgraphs and $U_1, U_2, \cdots, U_l \in \{V_1, V_2, \cdots, V_k\}$, respectively. Each of the result

graphs obtained by the *degree partition* of $G[U_1]$, $G[U_2]$,$\cdots$, $G[U_l]$ is a degree partition subgraph of the second layer of G. Similar to the representation of $G^{(1)}[d]$, we denote by $G^{(2)}[d]$ the set composed by the union of the subgraphs sets $G[U_1]^{(1)}[d]$, $G[U_2]^{(1)}[d]$,$\cdots$, $G[U_l]^{(1)}[d]$. Therefore, it follows that $G^{(2)}[d] = G[U_1]^{(1)}[d] \cup G[U_2]^{(1)}[d] \cup \cdots \cup G[U_l]^{(1)}[d]$.

This *degree partition process* can be recursively continued until each vertex-induced subgraph obtained by the *degree partition* is regular. Accompanied by the processes of the *degree partitions*, one can accordingly obtain *degree partition subgraphs sets* $G^{(r)}[d]$, $G^{(r+1)}[d]$, $\cdots$. Each subgraph of $G^{(r)}[d]$ is a vertex-induced subgraph of the *rth layer* of G. Each subgraph of $G^{(r+1)}[d]$ is a vertex-induced subgraph of the *(r + 1)th layer* of G. $\cdots$.

Definition 6. Let $G = (V(G), E(G))$ be an undirected connected graph. Suppose that the *degree partition* $\pi(G) = \{V_1, V_2, \cdots, V_k\}$ and the vertex-induced subgraphs on $V_1, V_2, \cdots, V_k$ are $G[V_1], G[V_2], \cdots, G[V_k]$, respectively.

By the above conditions, one can construct the *partition tree* of G, denoted by *PartT(G)*, according to following steps:

1. Let the graph G be the root node of the tree *PartT(G)*. Let $G.Layer = 0$.
2. Starting from $G[V_1]$ until $G[V_k]$, in turn let $G[V_1]$, $G[V_2]$, $\cdots$, $G[V_k]$ be the children nodes of the root node G such that the node $G[V_i]$ is on the left side of $G[V_j]$ with respect to the root node G for $1 \le i < j \le k$. Besides, let $G[V_i].Layer = G.Layer + 1$ for $1 \le i \le k$.
3. Assume that the nodes added into *PartT(G)* in the preceding round of operations are $H_1, H_2, \cdots, H_l$. If there exists no nonregular Ht with $1 \le t \le l$, then end the algorithm and return the *PartT(G)*. Otherwise, successively take each H_t from the sequence $H_1, H_2, \cdots, H_l$ with $1 \le t \le l$. If H_t is nonregular, then perform the *degree partition* $\pi(H_t)$. Without loss of generality assume $\pi(H_t) = \{U_1, U_2, \cdots, U_m\}$. By $\pi(H_t)$, one can compute the vertex-induced subgraphs $G[U_1], G[U_2], \cdots, G[U_m]$.
4. For each nonregular graph H_t with $1 \le t \le l$, starting from $G[U_1]$ until $G[U_m]$, in turn let $G[U_1], G[U_2], \cdots, G[U_m]$ be the children nodes of the node H_t such that a node $G[U_i]$ is on the left side of $G[U_j]$ with respect to H_t for $1 \le i < j \le m$. Besides, let $G[U_i].Layer = H_t.Layer + 1$ for $1 \le i \le m$.
5. Go back to Step 3 to perform the next round recursive operations.

Algorithm 1 gives more details on how to calculate the *partition tree PartT(G)*.

Definition 7. Let $G = (V(G), E(G))$ be an undirected connected graph. Let $\boldsymbol{S}(G) = \{u | u \in V(G) \wedge d(u) = \Delta(G)\}$ and $G[\boldsymbol{S}(G)]$ be the vertex-induced subgraph on *S(G)*.

By the above conditions, one can construct the *maximum partition tree* of G, denoted by *MaxPT(G)*, according to following steps:

1. Compute $G[S(G)]$ and let $G[S(G)]$ be the root node of the tree *MaxPT(G)*. Besides, let $G[S(G)].Layer = 0$.
2. Assume that the nodes added into *MaxPT(G)* in the preceding round of operations are $H_1, H_2, \cdots, H_l$. If there exists no nonregular H_t with $1 \le t \le l$, then end the algorithm and return *MaxPT(G)*. Otherwise, successively take each H_t from the sequence $H_1, H_2, \cdots, H_l$ with $1 \le t \le l$. If H_t is nonregular, then perform the *degree partition* $\pi(H_t)$. Without loss of generality assume $\pi(H_t) = \{U_1, U_2, \cdots, U_m\}$. By $\pi(H_t)$, one can compute the vertex-induced subgraphs $G[U_1], G[U_2], \cdots, G[U_m]$.
3. For each nonregular graph H_t with $1 \le t \le l$, starting from $G[U_1]$ until $G[U_m]$, in turn let $G[U_1], G[U_2], \cdots, G[U_m]$ be the children nodes of the node H_t such that a node $G[U_i]$ is on the left side of $G[U_j]$ with respect to H_t for $1 \le i < j \le m$. Besides, let $G[U_j].Layer = H_t.Layer + 1$ for $1 \le i \le m$.
4. Go back to Step 2 to perform the next round recursive operations.

Algorithm 2 provides more details on how to calculate the *maximum partition tree MaxPT(G)*. By Definition 6 and 7, the following Proposition 1 is trivial.

Proposition 1. Let $G = (V(G), E(G))$ be an undirected graph. Suppose that the *partition tree* and *maximum partition tree* of G are *PartT(G)* and *MaxPT(G)* respectively. Then the tree *MaxPT(G)* is a subtree of the tree *PartT(G)*.

Theorem 2. Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be two undirected graphs with n nodes respectively. Their descending degree sequences are $d(G)$ and $d(H)$ respectively, satisfying $|V(G)| = |V(H)|$, $|E(G)| = |E(H)|$, and $d(G) = d(H)$. Their adjacency matrices are $\mathbf{A}(G)$ and $\mathbf{A}(H)$ respectively. If $G \cong H$, then *PartT(G)* $\cong$ *PartT(H)* and *MaxPT(G)* $\cong$ *MaxPT(H)*.

Proof. If $G \cong H$, then there exists an isomorphism mapping $f: G \rightarrow H$ satisfying conditions that for every $u, v \in V(G)$, $(u, v) \in E(G)$ if and only if $f(u), f(v) \in V(H)$ and $(f(u), f(v)) \in E(H)$. Further, it follows that G_1 is a subgraph of G if and only if $f(G_1)$ is a subgraph of H. Clearly, f is also an isomorphism mapping $G_1 \rightarrow f(G_1)$. As a result, for every node w_1, w_2 on *PartT(G)*, (w_1, w_2) is an edge on *PartT(G)* if and only if $f(w_1), f(w_2)$ are on *PartT(G)* and $(f(w_1), f(w_2))$ is an edge on *PartT(G)*. Therefore, *PartT(G)* $\cong$ *PartT(H)*. A similar proof holds for *MaxPT (G)* $\cong$ *MaxPT (H)*. □

The following Proposition 2 and 3 immediately hold by Definition 6 and 7.

Proposition 2. Let $G = (V(G), E(G))$ be an undirected graph. Suppose that the *partition tree* and *maximum partition tree* of G are *PartT(G)* and *MaxPT(G)* respectively. If G is regular, then *PartT(G)* = *MaxPT(G)* = G.

Proposition 3. Let $G = (V(G), E(G))$ be an undirected graph. Suppose that the *partition tree* and *maximum partition tree* of G are *PartT(G)* and *MaxPT(G)* respectively.

Given any two nodes M and H on *PartT(G)*, if M is an ancestor of H in *PartT(G)*, then H is a vertex-induced subgraph of M. Similarly, if M and H are on *MaxPT(G)* and M is an ancestor of H on *MaxPT(G)*, then H is a vertex-induced subgraph of M.

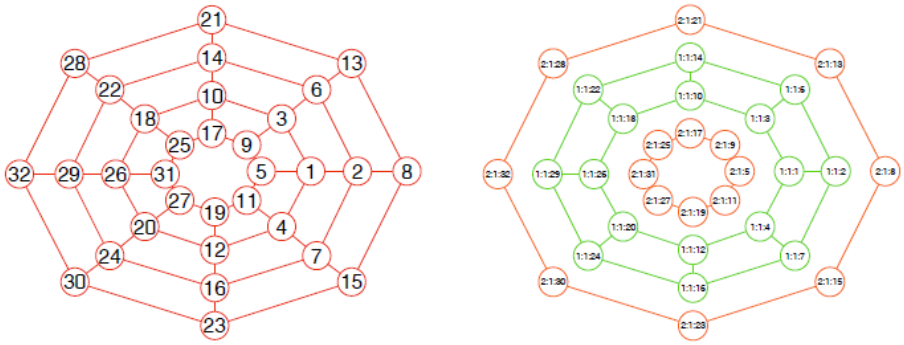
Theorem 3. Let $G = (V(G), E(G))$ be an undirected graph. Suppose that the partition tree and maximum partition tree of G are *PartT(G)* and *MaxPT(G)* respectively. Then all leaf nodes of *PartT(G)* and *MaxPT(G)* are the regular graphs.

Proof. We now prove Theorem 3 by contradiction. Assume by contradiction that there exists a leaf node H being the nonregular (vertex-induced) subgraph of G. It is clear that one can continue to perform the *degree partition* of H. As a result, the node H is no longer a leaf node. This result contradicts the condition that H is a leaf node. □

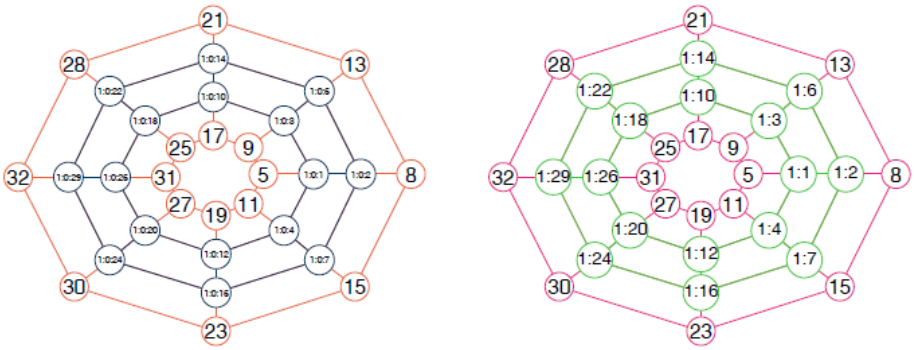
Definition 8. Let $G = (V(G), E(G))$ be an undirected graph. Suppose that the partition tree and maximum partition tree of G are *PartT(G)* and *MaxPT(G)* respectively. Following Algorithm 4 presented in Sec 5, one can create a subgraph sequence of G that contains all the leaf nodes of *PartT(G)*, and is denoted by *SRQ(G)* and called *standard regular sequence*. Similarly, Following Algorithm 5 presented in Sec 5, one can create a subgraph sequence of G that contains all the leaf nodes of *MaxPT(G)*, and is denoted by *SMRQ(G)* and called *standard maximum regular sequence*.

Definition 9. (*centre subgraph*) For a given graph G, one can compute $S(G) = \{u | u \in V(G) \wedge d(u) = \Delta(G)\}$ and then $G[S(G)]$. Let $H = G[S(G)]]$. For the vertex-induced subgraph H of G, one can again calculate $S(H) = \{u | u \in V(H) \wedge d(u) = \Delta(H)\}$ and then $G[S(H)]$. Once again, let $H = G[S(H)]$. ⋯ . This process can be recursively continued until the vertex-induced subgraph H obtained by the process is regular. We call the vertex-induced subgraph H of G the *centre subgraph* of G, denoted by *Cen(G)*.

Theorem 4. Let $G = (V(G), E(G))$ be an undirected graph. Assume that the partition tree and standard regular sequence of G are *PartT(G)* and *SRQ(G)*, respectively. Assume that the leftmost leaf node of *PartT(G)* is H_1 and the first node of *SRQ(G)* is H_2. Assume that centre subgraph of G is *Cen(G)*. Then following equations hold that $H_1 = H_2 =$ *Cen(G)*. Moreover, there exists only one *Cen(G)* in G.

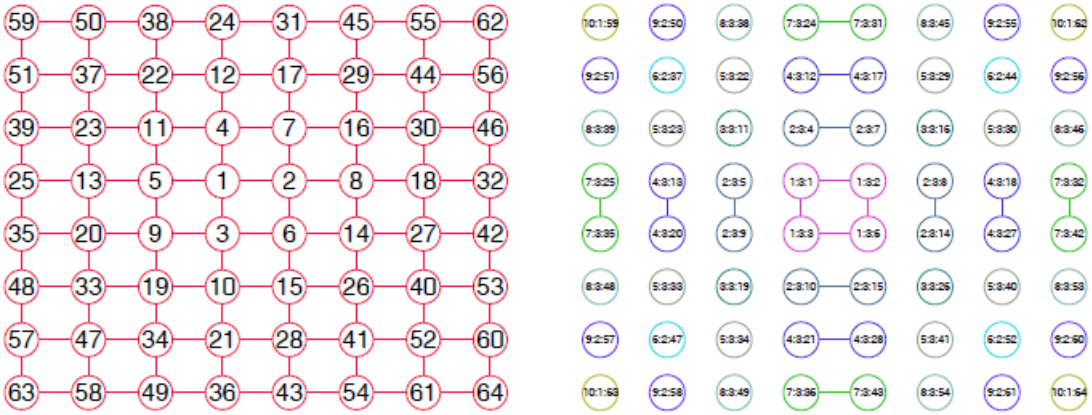


(a) The *MaxEm graphs* (b) The *SRQ(G)*

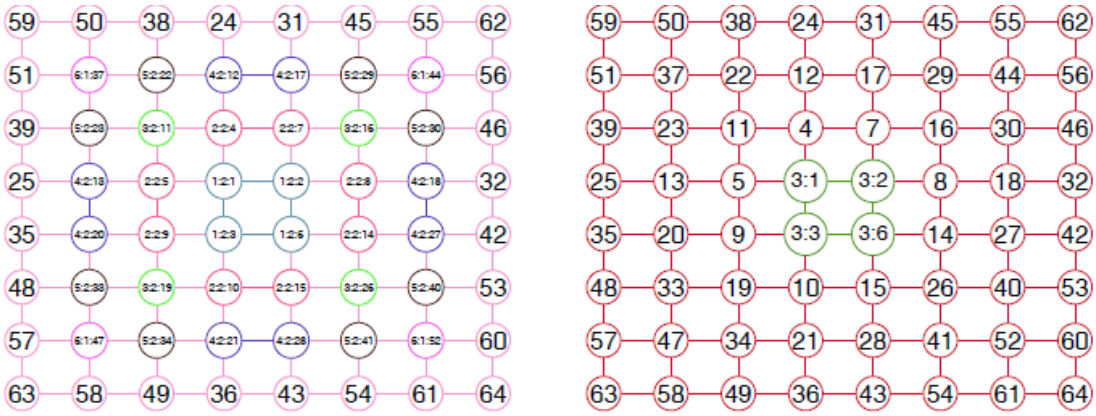


(c) The SMRQ(G) *(d) The Cen(G)*

Figure 2. A Graph *G* with 32 Nodes and 56 Edges



(a) The *MaxEm graph* $G_{8,8}$ (b) The *SRQ*($G_{8,8}$)



(c) The *SMRQ* ($G_{8,8}$) (d) The Cen($G_{8,8}$)

Figure 3. The 8 × 8 Grid Graph $G_{8,8}$ with 64 Nodes and 162 Edges

Proof. By Definition 6 and Algorithm 1, it can be seen that H_1 = *Cen(G)*. Further, by Definition 8 and Algorithm 4, it can be seen that H_2 = *Cen(G)*. Bringing together the above results, we have that equations $H_1 = H_2$ = *Cen(G)* hold. Clearly there exists only one leaf node *Cen(G)* on *PartT(G)*. Therefore, there exists only one *Cen(G)* in *G*. □

Similar to Theorem 4, we have the following Theorem 5 for maximum partition tree *MaxPT(G)* and standard maximum regular sequence *SMRQ(G)* of *G*.

Theorem 5. Let $G = (V(G), E(G))$ be an undirected graph. Assume that the *maximum partition tree* and *standard maximum regular sequence* of *G* are *MaxPT(G)* and *SMRQ(G)*, respectively. Assume that the leftmost leaf node of *MaxPT(G)* is H_1 and the first node of *SMRQ(G)* is H_2. Assume that centre subgraph of *G* is *Cen(G)*. Then following equations hold that $H_1 = H_2$ = *Cen(G)*.

Proof. By Definition 7 and Algorithm 2, it can be seen that H_1 = *Cen(G)*. Further, by Definition 8 and Algorithm 5, it can be seen that H_2 = *Cen(G)*. Bringing together the above results, we have that equations $H_1 = H_2$ = *Cen(G)* hold. □

Algorithm 3 shows how to calculate the centre subgraph *Cen(G)* of *G*. Figure. 2(d) and Figure. 3(d) are two examples obtained by performing our program, where *Cen(G)* is composed of nodes of *G*. Each node *v* in *Cen(G)* contains two attribute fields. The first field indicates the layer of *v* that is equal to *Cen(G).layer* on *PartT(G)*, and the second field specifies the string representing the label of *v*.

Theorem 6. (**Regular Partition Theorem**) For a given undirected graph *G*, if do not consider the label of the nodes that are in the same leaf nodes on *PartT(G)*, then there exists only one corresponding *partition tree PartT(G)* and *standard regular sequence SRQ(G)*. Similarly, if do not consider the label of the nodes that are in the same leaf nodes on *MaxPT(G)*, then there exists only one corresponding maximum partition tree *MaxPT(G)* and standard maximum regular sequence *SMRQ (G)*.

Proof. By Algorithm 1 and 4, it can be seen that for a given undirected graph *G*, the results obtained by performing each step of Algorithm 1 or Algorithm 4 are the exact. Further, the output obtained by performing the whole Algorithm 1 or 4 is also the exact. Therefore, if do not consider the label of the nodes that are in the same leaf nodes on *PartT(G)*, then there exists only one corresponding *partition tree PartT(G)* and standard regular sequence *SRQ(G)*. By Proposition 1 and Algorithm 2 and 5, it follows that the second part of the conclusion of Theorem 6 also holds. □

We have implemented Algorithm 4 and 5 that show in detail how to create subgraph sequences *SRQ(G)* and *SMRQ(G)* of a graph *G* respectively. Figure. 2(b), 3(b), 4, 5, 6, 7, 8

are seven sets of examples of *SRQ(G)* obtained by performing our program, where *SRQ(G)* are composed of leaf nodes. Each node *v* that belongs to a subgraph $H \in SRQ(G)$ contains three attribute fields. The first field indicates the position of *v* that is equal to *H. position* in *SRQ(G)*, the second field indicates the layer of *v* that is equal to *H.layer* on *PartT(G)*, and the third field specifies the string representing the label of *v*.

Figure. 2(c) and 3(c) are two examples of *SMRQ(G)* obtained by performing our program, where *SMRQ(G)* are composed of leaf nodes. Each node *v* that belongs to a subgraph $H \in$ *SMRQ(G)* also contains three attribute fields. The content of each field is the same as that shown in Figure. 2(b), 3(b), 4, 5, 6, 7, 8 in *SRQ(G)*.

Definition 10. Let $G = (V(G), E(G))$ be an undirected graph. Assume that the *partition tree* and *maximum partition tree* of *G* are *PartT(G)* and *MaxPT(G)*, respectively. Assume that centre subgraph of *G* is *Cen(G)*. Then we call the height of the tree *PartT(G) layer number of degree partition* of *G*, also referred to as *layer number* of *G* and denoted by *Layer(G)*. Similarly, we call the height of the tree *MaxPT(G) layer number of extreme degree partition* of G, also referred to as *extreme layer number* of *G* and denoted by *ELayer(G)*. We call the layer number of node *Cen(G)* on *PartT(G) center layer number* of *G*, denoted by *CLayer(G)*.

For every leaf node *H* on *PartT(G)* (*MaxPT(G)*), the *layer number* of *H* is equal to the length of the path from *H* to the root of the tree *PartT(G)* (*MaxPT(G)*). For every node $v \in H$, the *layer number* of node *v* is equal to the layer number of *H*.

Proposition 4. Let $G = (V(G), E(G))$ be an undirected graph. By Definition 10, then *Layer(G)*, *ELayer(G)* , and *CLayer(G)* satisfy the following relations:

$$Layer(G) \geq ELayer(G) + 1 \geq CLayer(G). \quad (2)$$

Proof. By Proposition 1, it is true that the tree *MaxPT(G)* is a subtree of the tree *PartT(G)*. Therefore, it follows that $Layer(G) \geq ELayer(G) + 1$. By Theorem 5, it can be seen that *Cen(G)* is also a leaf node of *MaxPT(G)*. As a result, we have that $ELayer(G) + 1 \geq CLayer(G)$. □

By Proposition 4, the following Definition 11 is well-defined.

Definition 11. Given a nonregular graph *G*, by Definition 10, one can classify the non-regular graph into 4 types depending on whether *Layer(G)* , *ELayer(G)* , and *CLayer(G)* are equal:

1. If $Layer(G) = ELayer(G) + 1 = CLayer(G)$, then *G* is of *type 1*.
2. If $Layer(G) > ELayer(G) + 1 = CLayer(G)$, then *G* is of *type 2*.
3. If $Layer(G) = ELayer(G) + 1 > CLayer(G)$, then *G* is of *type 3*.
4. If $Layer(G) > ELayer(G) + 1 > CLayer(G)$, then *G* is of *type 4*.

Theorem 7. (**Classification Theorem**) For a given nonregular undirected graph *G*, by Definition 11, then *G* belong to one of 4 kinds.

Proof. By Proposition 4 and Definition 11, it immediately follows that Classification Theorem 6 holds.

□

Definition 12. Let $G = (V(G), E(G))$ be an undirected graph with *n* nodes. The open 1-neighborhood subgraph of a vertex *u* in *G* is a subgraph of *G*, denoted by $N(u) = (V(N(u)), E(N(u)))$ where $V(N(u))$ is a set of all vertices adjacent to *u* ($u \notin V(N(u))$), and $E(N(u))$ is a set of all edges, each of which connects two vertices of $V(N(u))$.

The open *k*-neighborhood subgraph of *u* with $k \geq 2$ is a subgraph, denoted by $N_k(u) = (V(N_k(u)), E(N_k(u)))$ with $V(N_k(u)) = \{v \mid d(u, v) \leq k \wedge v \neq u \wedge v \in V(G)\}$, $E(N_k(u)) = \{(v, w) \mid v, w \in V(N_k(u))\}$.

Definition 13. Let $G = (V(G), E(G))$ be an undirected graph with *n* nodes. The open 1-neighborhood subgraph of a nodes set $Q \subseteq V(G)$ is a subgraph of *G*, denoted by $N(Q) = (V(N(Q)), E(N(Q)))$ where $V(N(Q))$ is a collection of all vertices each of which is adjacent to

at least one vertex in Q with $S \cap V(N(Q)) = \emptyset$, and $E(N(Q))$ is a set of all edges each of which links two nodes of $V(N(Q))$.

The open k-neighborhood subgraph of Q with $k \geq 2$ is a subgraph, denoted by $N_k(Q) = (V(N_k(Q)), E(N_k(Q)))$ with $V(N_k(Q)) = \{v \mid v \notin Q \wedge v \in V(G) \wedge \exists u \in Q \wedge d(v, u) \leq k\}$, $E(N_k(Q)) = \{(v, w) \mid v, w \in V(N_k(Q))\}$. In the following section, unless otherwise specified, each k-neighborhood subgraph is open.

4. Basic Principle for Computing *Cmax(G)*

Let $G = (V(G), E(G))$ be an undirected graph with n nodes. In the section, we will study how to compute the maximum element *Cmax(G)* of the graph G. Without loss of generality, let $MaxQ(G) = (u_1, u_2, \cdots, u_i, \cdots, u_n)$. Throughout the paper, our algorithms mentioned use an adjacency list to store the graph G.

To focus on the primary purpose of the paper, we omit the proofs of Theorems 10, 11, and 12, and Diffusion Theorem 13. We also omit the proofs of Lemmas 2 and 3. We will give the corresponding proofs in another article in detail.

4.1 Compute *Cmax(G)* of an undirected graph *G*

In this subsection, we examine how to compute the maximum element *Cmax(G)* of an undirected graph G. What approach should one take to calculate the maximum element *Cmax(G)*? From the connection between *Cmax(G)* and *Amax(G)*, any method for calculating *Cmax(G)* must first obtain the permutation *MaxQ(G)* corresponding to the adjacency matrix *Amax(G)*.

4.1.1 Compute the First Node u1 Added Into *MaxQ(G)* : In this sub-subsection, we examine how to compute the first vertex u_1 of *MaxQ(G)*. Here, assume that G is a connected graph of order $n > 1$. Note that to maximize $C(G)$ one must let $a_{1,2} = 1$ (see Figure.1). $a_{1,2} = 1$ can always be achieved since G is connected with order $n > 1$. Furthermore, to obtain *Cmax(G)*, one must select u_1 from *S(G)*. Only by so doing, can there be more "1"s in the high bits of $C(G)$ such that ensures maximum $C(G)$. Otherwise, $C(G)$ cannot reach the maximum value. From foregoing discussion, the following Proposition 5 and Lemma 1 hold.

Proposition 5. Let $G = (V(G), E(G))$ be an undirected graph with n nodes. Let $S(G) = \{u \mid u \in V(G) \wedge d(u) = \Delta(G)\}$. Then the selection of u_1 of *MaxQ(G)* is from *S(G)* for obtaining *Cmax(G)*.

Lemma 1. Let $G = (V(G), E(G))$ be an undirected graph with n nodes. Let $S(G) = \{u \mid u \in V(G) \wedge d(u) = \Delta(G)\}$. If $|S(G)| = 1$, then there exists a unique vertex $v \in S(G)$ such that $u_1 = v$ for *MaxQ(G)*.

Theorem 8. Let $G = (V(G), E(G))$ be an undirected connected graph. If condition *ELayer(G)* +1 = *CLayer(G)* holds, then u_1 of *MaxQ(G)* must belong to *Cen(G)*(see Figure. *3(c), 3(d)* and Figure. *2(c), 2(d)*).

Proof. By Proposition *5*, it follows that u_1 of *MaxQ(G)* is from *S(G)* for obtaining *Cmax(G)*. $\forall u \in V(Cen(G))$ and $\forall v \in S(G) - V(Cen(G))$ and. Then, by the construction of the *maximum partition tree MaxPT(G)* of G, there exists a leaf node H (being the regular subgraph of G) on *MaxPT(G)* such that $v \in V(H)$.

It can be shown that there always exists a most recent common ancestor node M of H and *Cen(G)* on the tree *MaxPT(G)* such that H and *Cen(G)* are the vertex-induced subgraphs of M. Let us assume that U_1 and $U_2 \in \pi(M)$, and *Cen(G)* and H are the vertex-induced subgraphs of $M[U_1]$ and $M[U_2]$ respectively.

By Theorem 5, we have that *Cen(G)* is the leftmost leaf node of *MaxPT(G)*. Therefore, although the *Cen(G)* and H are all the child nodes of M, *Cen(G)* is on the left side of H on *MaxPT(G)*. By the condition (5) of Definition 4, it follows that $d(U_1) > d(U_2)$. Therefore,

inequality $d_M(u) > d_M(v)$ holds for $\forall u \in V(Cen(G))$, $\forall v \in S(G) - V(Cen(G))$. As a result, we have that u_1 of *MaxQ(G)* must belong to *Cen(G)*. □

Theorem 9. Let $G = (V(G), E(G))$ be an undirected connected graph. If condition *Layer(G) = CLayer(G)* holds, then u_1 of *MaxQ(G)* must belong to *Cen(G)* (see Figure. 3(b), 3(d) and Figure. 2(b), 2(d)).

Proof. By Proposition 4, we have that

$$Layer(G) \geq ELayer(G) + 1 \geq CLayer(G). \quad (3)$$

By the condition of Theorem 9, we have that $Layer(G) = CLayer(G)$. Therefore, it can be seen that

$$CLayer(G) \geq ELayer(G) + 1 \geq CLayer(G). \quad (4)$$

By (4), we have that

$$ELayer(G) + 1 = CLayer(G). \quad (5)$$

By (5) and Theorem 8, it is true that u_1 of *MaxQ(G)* must belong to *Cen(G)*. □

By Lemma 1, for $|S(G)| = 1$ there exists a unique vertex $v \in S(G)$ such that $u_1 = v$. For $|S(G)| > 1$, the following Theorem 10 holds.

Theorem 10. Let $G = (V(G), E(G))$ be an undirected connected graph with n nodes. Let $S(G) = \{u | u \in V(G) \wedge d(u) = \Delta(G) = s\}$ with $|S(G)| > 1$. Let $v \in S(G)$ with $v_1 \in V(N(v))$ and $d_{N(v)}(v_1) = \Delta(N(v))$. For each $w \in S(G) \wedge w \neq v$ with $w_1 \in V(N(w))$ and $d_{N(w)}(w_1) = \Delta(N(w))$, if condition $d_{N(v)}(v_1) > d_{N(w)}(w_1)$ holds, then $u_1 = v$ for *MaxQ(G)*.

Theorem 11. Let $G = (V(G), E(G))$ be an undirected connected graph with n nodes. Let $S(G) = \{u | u \in V(G) \wedge d(u) = \Delta(G) = s\}$ with $|S(G)| > 1$. Let $v \in S(G)$ with $v_1 \in V(N(v))$ and $d_{N(v)}(v_1) = \Delta(N(v))$. For each $w \in S(G) \wedge w \neq v$ with $w_1 \in V(N(w))$ and $d_{N(w)}(w_1) = \Delta(N(w))$, if conditions $d_{N(v)}(v_1) = d_{N(w)}(w_1)$ and $d_G(v_1) > d_G(w_1)$ hold, then $u_1 = v$ for *MaxQ(G)*.

4.1.2 Calculate the Intermediate Vertices added into *MaxQ(G)* : When our algorithm has computed the first node u_1 of *MaxQ(G)*, how would it determine the subsequent nodes when calculating *Cmax(G)*? Note that an edge of G corresponds to 1 bit of the upper triangular part of the adjacency matrix $\mathbf{A}(G)$. To maximize $C(G)$ by maximizing $C^2(G)$, u_2 must belong to $N(u_1)$ so that makes $a_{1,2} = 1$ (see Figure.1). Otherwise, $u_2 \notin N(u_1)$ such that $a_{1,2} = 0$ (see Figure.1) and $C(G) \neq Cmax(G)$. The following Proposition 6 summarizes the above results.

Proposition 6. Let $G = (V(G), E(G))$ be an undirected graph with n nodes. Given the first node u1 of *MaxQ(G)*, then the selection of u_2 is from $N(u_1)$ for obtaining *Cmax(G)*.

Lemma 2. Let $G = (V(G), E(G))$ be an undirected graph with n nodes. Given the first node u_1 of *MaxQ(G)*, if there exists a unique node $v \in S(N(u_1)) = \{u | u \in V(N(u_1)) \wedge d(u) = \Delta(N(u_1))\}$, then $u_2 = v$ for *MaxQ(G)*.

Theorem 12. Let $G = (V(G), E(G))$ be an undirected connected graph with n nodes. Let $S(G) = \{u | u \in V(G) \wedge d(u) = \Delta(G) = s\}$. For computing *Cmax(G)*, suppose that *MaxQ(G)* already contains the first node $u_1 = v$ with $V(N(v)) = \{v_1, v_2, \cdots, v_s\}$, satisfying condition $d_{N(v)}(v_1) = \Delta(N(v))$. If one of the following conditions holds, then $u_2 = v_1$ for *MaxQ(G)*.

1. $d_{N(v)}(v_1) > d_{N(v)}(v_i)$ for $i = 2, \cdots, s$.
2. $d_G(v_1) > d_G(v_i)$ hold for $d_{N(v)}(v_1) = d_{N(v)}(v_i)$ with $i \in \{2, \cdots, t\}$.

Our algorithm uses an adjacency list to store a graph G. When our algorithm has computed the first i vertices $u_1, u_2, \cdots, u_i$ of *MaxQ(G)*, how does it determine the subsequent nodes u_{i+1}, $u_{i+2}, \cdots, u_n$ for calculating *Cmax(G)*? Similar to the above discussion for obtaining u_2, it can

be shown that the selections of the successor vertices u_{i+1}, u_{i+2}, $\cdots$, u_n of *MaxQ(G)* are from $N(S)$ with $S = \{u_1, u_2, \cdots, u_i\}$.

Lemma 3. Let $G = (V(G), E(G))$ be an undirected connected graph with n nodes. Let $S(G) = \{u | u \in V(G) \wedge d(u) = \Delta(G) = s\}$ with $|S(G)| > 1$. If $u_1 = v \in S(G)$ with 1-neighborhood subgraph $N(v)$, $V(N(v)) = \{v_1, v_2, \cdots, v_s\}$, then $u_2, u_3, \cdots, u_s, u_{s+1} \in \{v_1, v_2, \cdots, v_s\}$.

Theorem 13. (**Diffusion Theorem**) Let $G = (V(G), E(G))$ be an undirected connected graph with n nodes. If *MaxQ(G)* already contains the first m vertices $u_1, u_2, \cdots, u_m$ when computing *Cmax(G)*, then the following two conclusions hold.

1. the vertex-induced subgraph of the first m vertices is connected.
2. the selection of the *(m + 1)th* vertex for computing *Cmax(G)* is from the open 1-neighborhood subgraph $N(Q)$ of the nodes set $Q = \{u_1, u_2, \cdots, u_m\}$.

5. The Relevant Algorithms

In the section, based on the results of the preceding sections, we present our algorithms in detail for computing the *partition tree PartT(G)* (see Algorithm 1), *maximum partition tree MaxPT(G)* (see Algorithm 2), *centre subgraph Cen(G)* (see Algorithm *3*), *standard regular sequence SRQ(G)* (see Algorithm 4), *standard maximum regular sequence SMRQ(G)* (see Algorithm 5). To focus on the primary purpose of the paper, we omit the algorithms for computing the maximum element *Cmax(G)* of *G*, which will be present in detail in another article.

Our algorithms use an array *MaxQ* to store the nodes of *MaxQ(G)* and an array *Q* to keep the nodes to be added to MaxQ(G) temporarily. Our algorithms employ three adjacency lists to store *PartT(G)*, *MaxPT(G)*, and *Cen(G)* respectively, and utilize two arrays to save the *SRQ(G)* and *SMRQ(G)* respectively.

It is not difficult to find that the worst time complexities of algorithms 1, 2, 4, and 5 are $O(n^2)$ respectively. It can be seen that the time complexity of Algorithm 3 is $O(n)$.

6. Software Implementation

Applying the principles described in the preceding sections, we have developed a set of software tools called GraphLabel for computing canonical labelings of graphs. GraphLabel also can calculate the *partition tree PartT(G)* (see Algorithm 1), *maximum partition tree MaxPT(G)* (see Algorithm 2), *centre subgraph Cen(G)* (see Algorithm 3), standard regular sequence *SRQ(G)* (see Algorithm 4), and standard maximum regular sequence *SMRQ(G)* (see Algorithm 5). Our development environment includes an Intel(R) Core(TM)2 Quad CPU Q6600 @2.40GHz with 4.00 GB of RAM. The operating system is Microsoft Windows 8.1 Professional Edition. The graphics card is an NVIDIA GeForce 9800 GT. The display resolution is 1024 × 768 × 32 bits (RGB). The internal hard drive is 500GB. The programming environment is the Microsoft Visual C++ 2012.

GraphLabel adopts object-oriented technology to design several relevant classes including the classes *CNode*, *CNodeNeighbor*, *CEdge*, *CEdgeNeighbor*, *CGraph*, and so on. A detailed description of the software functions is outside the scope of this article. We will explain it in another paper. All the figures presented in this paper are produced by using our software system.

Figure. 2 to 8 are seven group figures obtained by performing GraphLabel. In addition, Figure. 2(a), 3(a), 4(a), 4(c), 4(e), 5(a), 5(c), 6(a), 6(c), 6(e), 7(a), 7(c), 7(e),8(a), 8(c), and 8(e) are the *MaxEm graphs*. To speed up the calculation of the *MaxEm graphs*, GraphLabel applies Theorem 8 and 9 to calculate the first node u_1 of MaxQ(G).

We selected a graph set to test the accuracy of our algorithms. Using our own software platform, we randomly produced a large number of graphs as the test cases, including

Figure.4(c), 4(e), 5(a), 5(c), 6(e), and 8(e). To increase the breadth and depth of our testing, we also select many test cases from the library of benchmarks [1] and online library [22], including Figure. 2(a), 3(a), 4(a), 6(a), 6(c), 7(a), 7(c), 7(e), 8(a), 8(c), and 8(e).

Experimental results show that our algorithm for computing *PartT(G)*, *MaxPT(G)*, *Cen(G)*, *SRQ(G)*, and *SMRQ(G)* are very effective, and our algorithm for computing canonical labelings of graphs is novel. By Theorem 8 and 9, we can improve the efficiency of the labeling algorithm.

Algorithm 1: Construct the *partition tree PartT (G)* of a graph *G*.

Input : An undirected graph *G* with *n* nodes.

Output: The *partition tree PartT (G)* of *G*.

```
1  i ← 0;  j ← 0;  k ← 0;  l ← 0;
2  List ← 0; // Initialize an empty adjacency list to store the PartT (G);
3  List.length ← 0;
4  H ← 0; // Initialize an empty subgraph;
5  H ← G;
6  H.Layer ← 0;
7  List[0] ← H; // Add H into List such that H is the root node of PartT (G);
8  List.length ← List.length + 1;
9  for ( i ← 1 to List.length ){
10     H ← List[i];
11     if ( The subgraph H is regular )then
12        H.Is_Regular ← 1;
13        List[i] ← H;
14        go back to the beginning of the for-loop;
15     Perform the degree partition π(H)={U1, U2, · · · , Ul};
16     k ← l;
17     Compute the vertex-induced subgraphs H[U1], H[U2], · · · , H[Uk];
18     for ( j ← 1 to k ){
19        H[Uj].Layer ← H.Layer + 1;
20        Let H[Uj] be the j-th child node of the subgraph node H;
21        Insert H[Uj] in the position (i + j − 1) of List;
22        List.length ← List.length + 1;
23     List[i] ← H;
24 return List; // Return the PartT (G);
```

Algorithm 2: Construct the *maximum partition tree MaxPT(G)* of a graph *G*.

Input : An undirected graph *G* with *n* nodes.

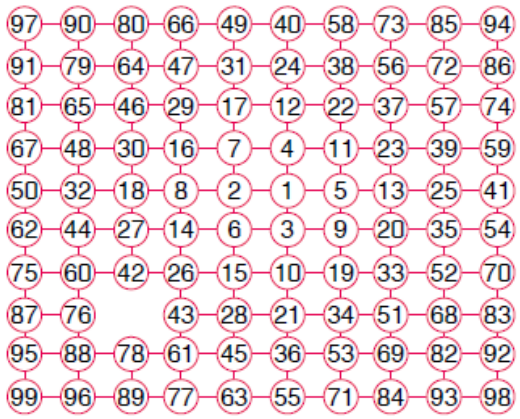
Output: The *maximum partition tree MaxPT(G)* of *G*.

```
1  i ← 0;  j ← 0;  k ← 0;  l ← 0;  H ← ∅;
2  List ← ∅ ; // Initialize an empty adjacency list to store the MaxPT(G);
```

3 $List.length \leftarrow 0$;
4 Obtain $S(G) = \{u | u \in V(G) \wedge d(u) = \Delta(G)\}$;
5 Compute the vertex-induced subgraph $G[S(G)]$;
6 $H \leftarrow G[S(G)]$;
7 $H.Layer \leftarrow 0$;
8 $List[0] \leftarrow H$; // Add H into List such that H is the root node of *MaxPT(G)*;
9 $List.length \leftarrow List.length + 1$;
10 **for** ($i \leftarrow 1$ **to** $List.length$){
11 $H \leftarrow List[i]$;
12 **if** (*The subgraph H is regular*)**then**
13 $H.Is\_Regular \leftarrow 1$;
14 $List[i] \leftarrow H$;
15 go back to the beginning of the for-loop;
16 Perform the *degree partition* $\pi(H) = \{U_1, U_2, \cdots, U_l\}$;
17 $k \leftarrow l$;
18 Compute the vertex-induced subgraphs $H[U_1], H[U_2], \cdots, H[U_k]$;
19 **for** ($j \leftarrow 1$ **to** k){
20 $H[U_j].Layer \leftarrow H.Layer + 1$;
21 Let $H[U_j]$ be the j-th child node of the subgraph node H;
22 Insert $H[U_j]$ in the position $(i + j - 1)$ of *List*;
23 $List.length \leftarrow List.length + 1$;
24 $List[i] \leftarrow H$;
25 **return** *List*; // Return the *MaxPT(G)*;

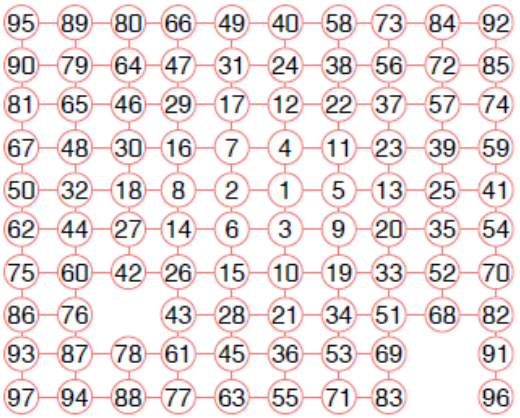
(a) The *MaxEm graph* $G_{10,10}$

(b) The $SRQ(G_{10,10})$



(c) The *MaxEm graph* G_1

(d) The *SRQ*(G_1)



(e) The *MaxEm graph* G_2

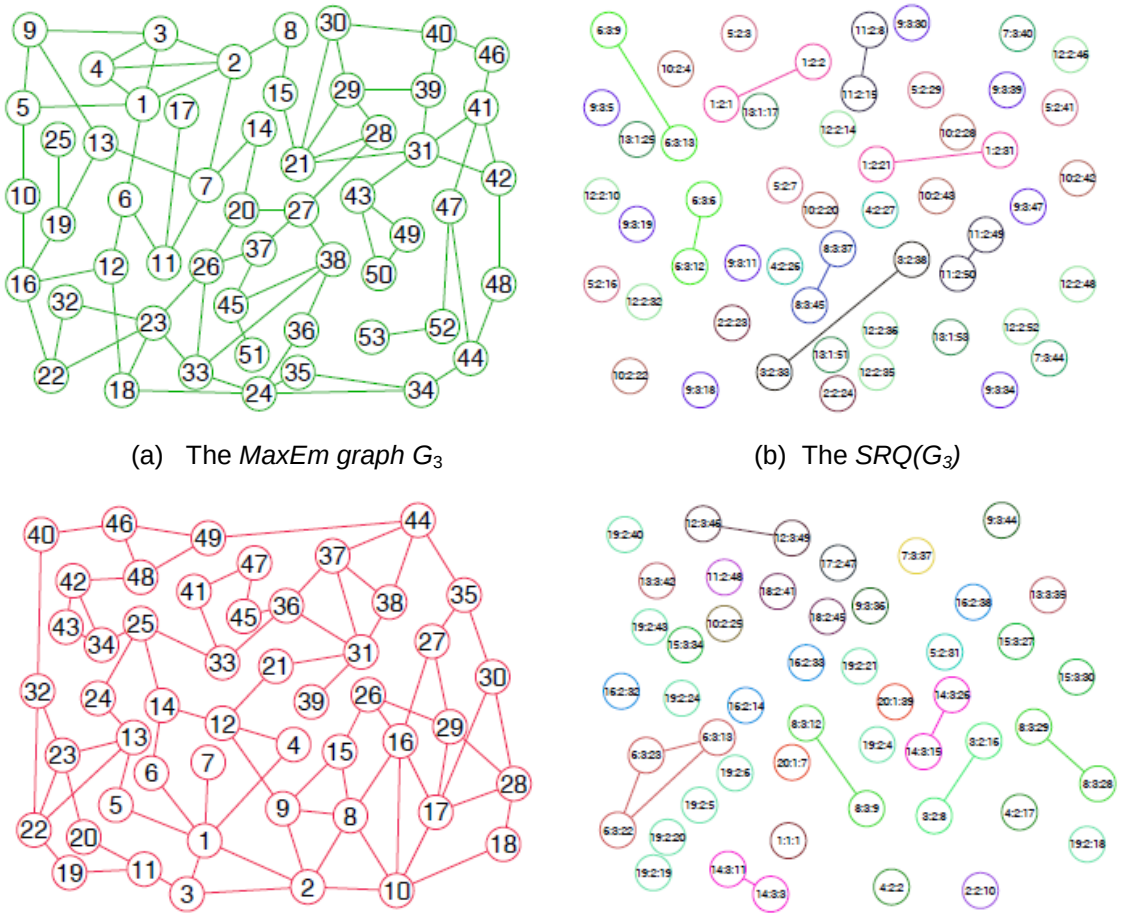
(f) The *SRQ*(G_2)

Figure 4. Three Graphs Including the 10 × 10 grid graph $G_{10,10}$ with 100 Nodes and 180 Edges, the G_1 with 99 Nodes and 176 Edges, the G_2 with 97 Nodes and 170 Edges

Algorithm 3: Compute the *centre subgraph Cen(G)* of a graph *G*.

Input : An undirected graph *G* with *n* nodes.
Output: The *centre subgraph Cen(G)*.

```
1 U ←∅; H ←∅;
2 H ← G;
3 while ( The vertex-induced subgraph H is nonregular ){
4     Compute S(H) = {u|u ∈ V(H) ∧ d(u) = Δ(H)};
5     U ← S(H);
6     Compute the vertex-induced subgraph H[U];
7     H ← H[U];
8 return H;
```



(a) The *MaxEm graph* G_3 (b) The *SRQ*(G_3)

Figure 5. Two Graphs Including G_3 with 53 Nodes and 80 Edges and G_4 with 49 Nodes and 78 Edges

Algorithm 4: Create the *standard regular sequence SRQ(G)* of a graph G.

Input : An undirected graph G with n nodes.

Output: The *standard regular sequence SRQ(G)* of G.

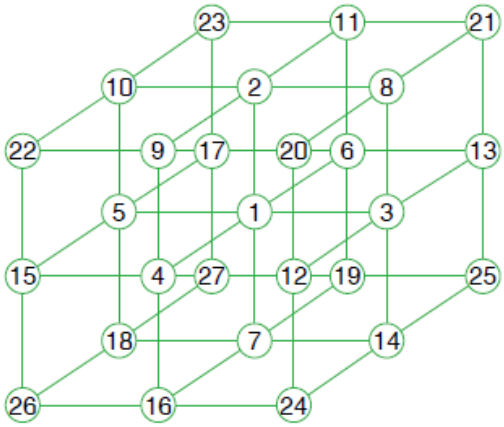
```
1  i ← 0;  j ← 0;  k ← 0;  l ← 0;
2  List ← ∅;  List.length ← 0;  //Initialize an empty adjacency list to store the PartT(G);
3  H ← ∅; // Initialize an empty subgraph;
4  H ← G;
5  H.Layer ← 0;
6  List[0] ← H; //Add H into List such that H is the root node of PartT(G);
7  List.length ← List.length + 1;
8  for ( i ← 1 to List.length ){
9      H ← List[i];
10     if ( The subgraph H is regular )then
11         H.Is_Regular ← 1;
12         List[i] ← H;
13         go back to the beginning of the for-loop;
14     Perform the degree partition π(H)={U1, U2, · · · , Ul};
15     k ← l;
16     Compute the vertex-induced subgraphs H[U1],H[U2], · · · ,H[Uk];
17     for ( j ← 1 to k ){
18         H[Uj].Layer ← H.Layer + 1;
19         Let H[Uj] be the j-th child node of the subgraph node H;
20         Insert H[Uj] in the position (i + j − 1) of List;
21         List.length ← List.length + 1;
22     List[i] ← H;
23 k ← 0;
24 for ( i ← 1 to List.length ){
25     H ← List[i];
26     if ( H.Is_Regular == 0 )then
27         k ← k + 1;
28         H.Number ← k;
29         Add the H to the back of SRQ(G);
30 return SRQ(G);
```

Algorithm 5: Create the *standard maximum regular sequence SMRQ(G)* of a graph G.

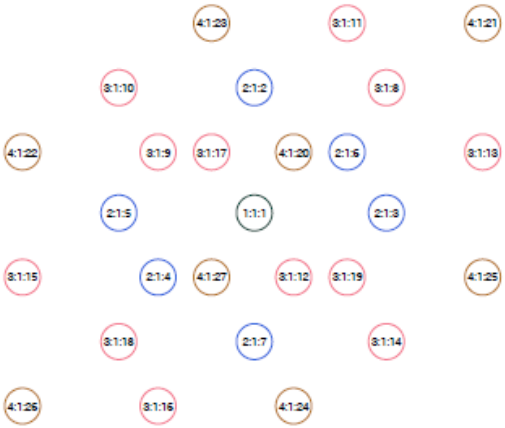
Input : An undirected graph G with n nodes.

Output: The *standard maximum regular sequence SMRQ(G)* of G.

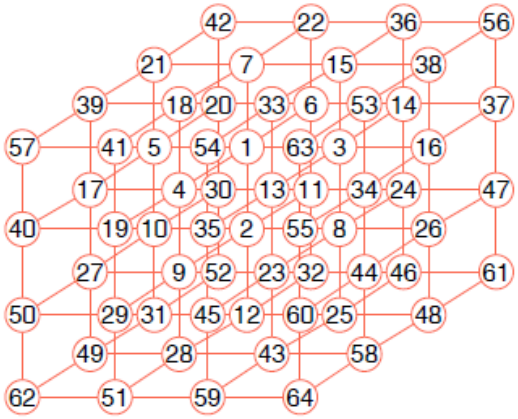
```
1  i ← 0; j ← 0; k ← 0; l ← 0; H ←∅;
2  List ←∅; List.length ← 0;// Initialize an empty adjacency list to store the MaxPT(G);
3  Obtain S(G) = {u|u ∈ V(G) ∧ d(u) = Δ(G)};
4  Compute the vertex-induced subgraph G[S(G)];
5  H ← G[S(G)]; H.Layer ←0;
6  List[0] ← H; //Add H into List such that H is the root node of MaxPT(G);
7  List.length ← List.length + 1;
8  for ( i ← 1 to List.length ){
9      H ← List[i];
10     if ( The subgraph H is regular )then
11         H.Is_Regular ← 1;
12         List[i] ←H;
13         go back to the beginning of the for-loop;
14     Perform the degree partition π(H)={U1, U2, · · · , Ul};
15     k ← l;
16     Compute the vertex-induced subgraphs H[U1],H[U2], · · · ,H[Uk];
17     for ( j ← 1 to k ){
18         H[Uj].Layer ← H.Layer + 1;
19         Let H[Uj] be the j-th child node of the subgraph node H;
20         Insert H[Uj] in the position (i + j − 1) of List;
21         List.length ← List.length + 1;
22     List[i] ←H;
23 k ← 0;
24 for ( i ← 1 to List.length ){
25     H ← List[i];
26     if ( H.Is_Regular == 0 )then
27         k ← k + 1;
28         H.Number ← k;
29         Add the H to the back of SMRQ(G);
30 return SMRQ(G);
```

(a) The *MaxEm graph* $G_{3,3,3}$

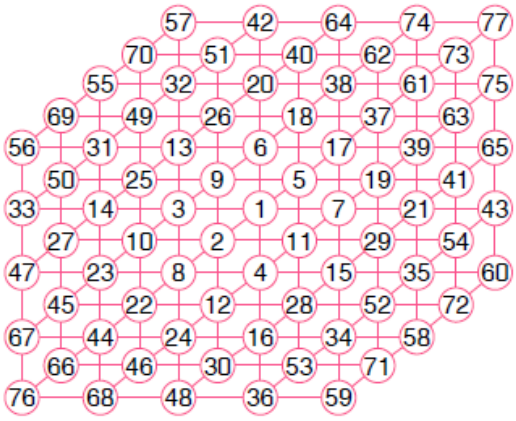


(b) The *SRQ*($G_{3,3,3}$)



c) The *MaxEm graph* $G_{4,4,4}$

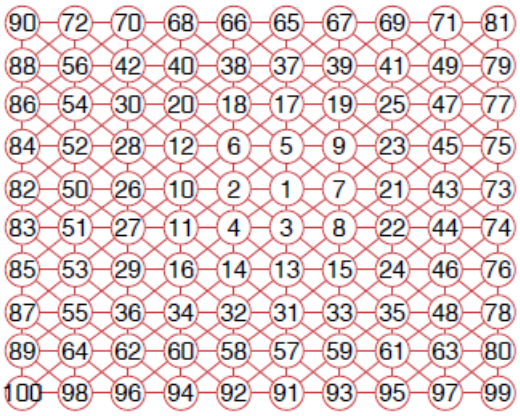
(d) The *SRQ*($G_{4,4,4}$)



(e) The *MaxEm graph* G_5

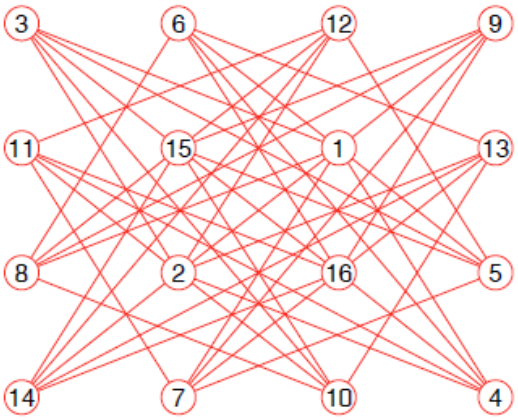
(f) The *SRQ*(G_5)

Figure 6. Three Graphs Including the 3 × 3 × 3 Grid Graph $G_{3,3,3}$ with 27 Nodes and 54 Edges, the 4 × 4 × 4 Grid Graph $G_{4,4,4}$ with 64 Nodes and 144 Edges, and G_5 with 77 Nodes and 196 Edges

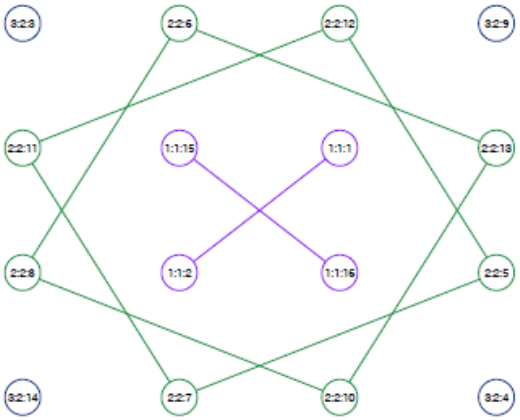


(a) The *MaxEm graph* G_6

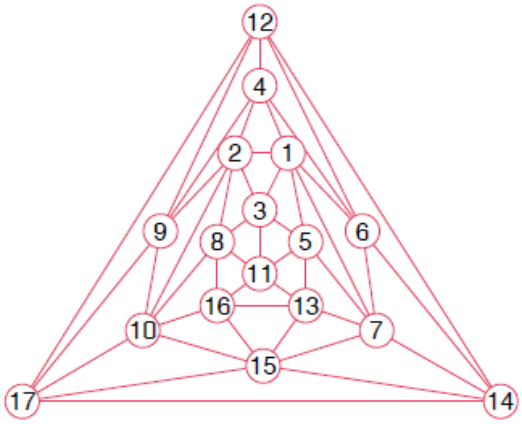
(b) The *SRQ*(G_6)



(c) The *MaxEm graph* G_7



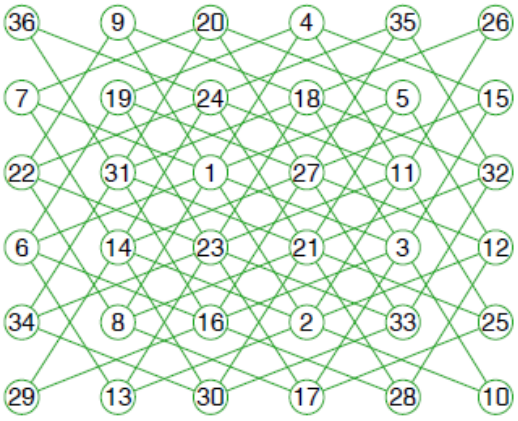
(d) The *SRQ*(G_7)



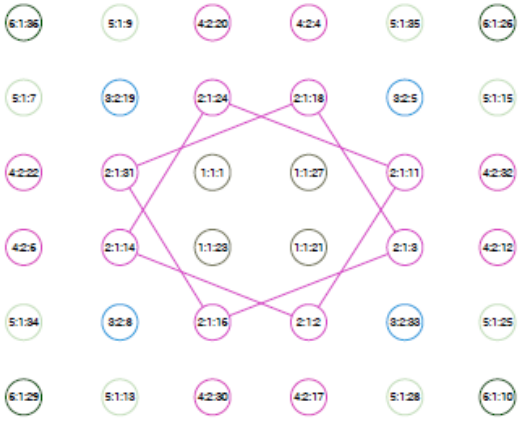
(e) The *MaxEm graph* G_8

(f) The *SRQ*(G_8)

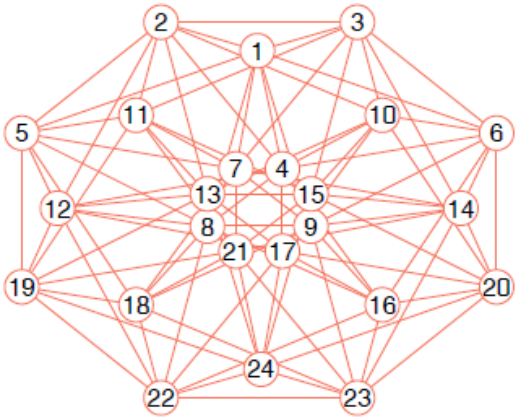
Figure 7. Three Graphs Including the 10 × 10 King Graph G_6 with 100 Vertices and 342 Edges, the Errera Graph G_7 with 17 Nodes and 45 Edges, and the 4-Dimensional Keller Graph G_8 with 16 Nodes and 46 Edges



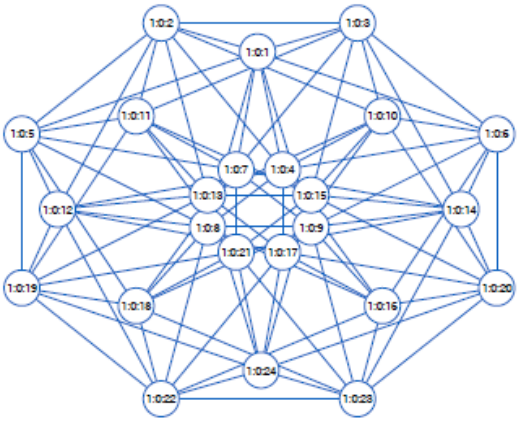
(a) The MaxEm graph G9



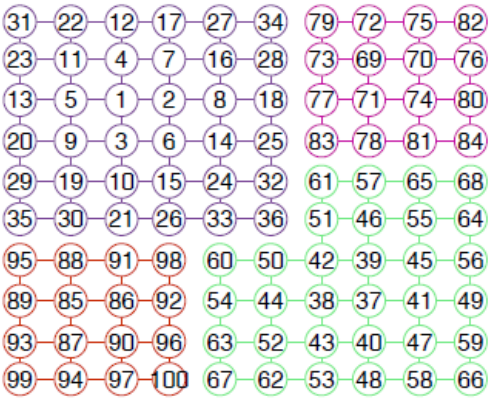
(b) The SRQ(G9)



(c) The MaxEm graph G10



(d) The SRQ(G10)



(e) The *MaxEm graph* G_{11}

(f) The *SRQ(G_{11})*

Figure 8. Three Graphs Including the 6 × 6 Knight Graph G_9 with 36 Vertices and 80 Edges, the 24-Cell Graph G_{10} with 24 Nodes and 94 Edges, and a Disconnected Graph G_{11} that has Four Connected Components and a Total of 100 Nodes and 160 Edges

7. Conclusions and Future Work

In summary, we obtain the following conclusions: By Theorem 2 to 9, this paper establishes a theoretical framework by which one can classify graphs and create the *PartT(G)*, *MaxPT(G)*, *Cen(G)*, *SRQ(G)*, and *SMRQ(G)* for a given graph *G*. By Theorem 3, all leaf nodes of *PartT(G)* and *MaxPT(G)* are the regular graphs. By Theorem 4 and 5, there exists only one *Cen(G)* in *G*. Regular Partition Theorem 6 shows that there exists just one

corresponding *PartT(G)*, *SRQ(G)*, *MaxPT(G)*, and *SMRQ(G)*. One can use Classification Theorem 7 to category graphs. By Theorem 8 and 9, one can use *Cen(G)* to calculate the first node u_1 added into *MaxQ(G)* corresponding to the canonical labeling *Cmax(G)* of *G*. Our algorithm for computing canonical labelings of graphs is novel.

The implementations of algorithms 1 to 5 show how to calculate them accordingly. The worst time complexities of algorithms 1, 2, 4, and 5 are $O(n^2)$ respectively. The time complexity of Algorithm 3 is $O(n)$. The proposed methods can be extended to deal with the directed graphs and weighted graphs. This is the goal of our future work. In addition to, in future we will study the interconnections among *SRQ(G)*, *SMRQ(G)*, and *Cmax(G)*. We will study how to use *SRQ(G)* and *SMRQ(G)* to compute *Cmax(G).*

Acknowledgements

The work described in this paper was supported by Key Project of the National Natural Science Foundation of China (No.91318301), National Natural Science Foundation of China (No.61202080), China Postdoctoral Science Foundation (No.2015M581032).

References

[1] Alenex 2007 submission: source code, benchmark instances, and summary results (**2015**). URL http://www.tcs.hut.fi/Software/benchmarks/ ALENEX-2007/.
[2] Arnborg, S., Proskurowski, A.: Canonical representations of partial 2- and 3-trees. Bit Numerical Mathematics 32(2), 197–214 (**1998**) .
[3] Arvind, V., Das, B., KBler, J.: A logspace algorithm for partial 2-tree canonization. Lecture Notes in Computer Science 5010, 40–51 (**2008**) .
[4] Babai, L., Kucera, L.: Canonical labelling of graphs in linear average time. IEEE pp. 39 – 46 (**1979**) .
[5] Babai, L., Luks, E.M.: Canonical labeling of graphs. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC '83, pp. 171– 183. ACM, New York, NY, USA (**1983**). DOI 10.1145/800061.808746 .
[6] He, P.R., Zhang, W.J., Li, Q.: Some further development on the eigensystem approach for graph isomorphism detection. Journal of the Franklin Institute- Engineering and Applied Mathematics 342(6), 657–673 (**2005**) .
[7] Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraphs in the presence of isomorphism. In: Data Mining, 2003. ICDM 2003. Third IEEE International Conference on, pp. 549–552 (**2003**) .
[8] Junttila, T., Kaski, P.: Conflict Propagation and Component Recursion for Canonical Labeling. Springer Berlin / Heidelberg (**2011**) .
[9] Junttila, T.A., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. Alenex pp. 135–149 (**2007**) .
[10] Kashani, Z.R.M., Ahrabian, H., Elahi, E., Nowzari-Dalini, A., Ansari, E.S., Asadi, S., Mohammadi, S., Schreiber, F., Masoudi-Nejad, A.: Kavosh: a new algorithm for finding network motifs. Bmc Bioinformatics 10(41), 1–12 (**2009**) .
[11] Katebi, H., Sakallah, K., Markov, I.: Graph symmetry detection and canonical labeling: Differences and synergies. Proc. Turing-100, Manchester, UK (**2012**) .
[12] Kuramochi, M., Karypis, G.: An efficient algorithm for discovering frequent sub- graphs. Knowledge and Data Engineering, IEEE Transactions on 16(9), 1038–1051 (**2004**). DOI 10.1109/TKDE.2004.33.
[13] Kuramochi, M., Karypis, G.: Finding frequent patterns in a large sparse graph*. Data Mining and Knowledge Discovery 11(3), 243–271 (**2005**) .
[14] López-Presa, J.L., Anta, A.F., Chiroque, L.N.: Conauto-2.0: Fast isomorphism testing and automorphism group computation. CoRR abs/1108.1060 (**2011**). URL http://arxiv.org/abs/1108.1060.
[15] McKay, B.: Computing automorphisms and canonical labellings of graphs Combinatorial Mathematics, Lecture Notes in Mathematics, vol. 686, pp. 223–232. Springer Berlin / Heidelberg (**1978**) .
[16] McKay, B.: Practical graph isomorphism (**1981**) .
[17] McKay, B.D.: Isomorph-free exhaustive generation. Journal of Algorithms 26(2), 306–324 (**1998**) .
[18] Mckay, B.D., Piperno, A.: Practical graph isomorphism, ii. Journal of Symbolic Computation 60(1), 94âAˇ S¸ 112 (**2013**) .
[19] Miyazaki, T.: The complexity of mckayâAˇ Z´ s canonical labeling algorithm (**1997**) .
[20] Piperno, A.: Search space contraction in canonical labeling of graphs. Arxiv preprint arXiv:0804.4881 (**2008**) .
[21] Tener, G., Deo, N.: Efficient isomorphism of miyazaki graphs. algorithms 5, 7 (**2008**) .
[22] Weisstein, E.W.: Simple graphs – from wolfram mathworld (**2015**). URL http: //mathworld.wolfram.com/topics/SimpleGraphs.html.

[23] Yan, X., Han, J.: gspan: graph-based substructure pattern mining. In: Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, pp. 721– 724 (**2002**). DOI 10.1109/ICDM.2002.1184038.

Authors

HAO Jian-Qiang is a PhD student at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China. His research interests include Graph Theory, Computational Geometry, Computer Graphics, and Software Testing.

GONG Yun-Zhan is a Professor and Doctoral Tutor at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China. His research interests include Network and Service Test.

Tan Li is an Assistant Professor at the School of Computer Science and Information Engineering, Beijing Technology and Business University, China. His research interests include Wireless Sensor Networks, Intelligent Information Network and Machine Learning.

Duan Da-Gao is an Assistant Professor at the School of Computer Science and Information Engineering, Beijing Technology and Business University, China. His research interests include Intelligent Robotics and Machine Vision, Network Communications, Embedded Systems, Internet Data Mining.