

VF2算法实验报告

海量图数据的管理和挖掘

2015 年 12 月 13 日

作者：马凌霄

学号：1501111302

院系：信息科学技术学院

E-Mail: xysmlx@gmail.com

Repository (Private): <https://bitbucket.org/xysmlx/vf2>

Git (Remote): <https://xysmlx@bitbucket.org/xysmlx/vf2.git>

The screenshot shows the Bitbucket web interface for a private repository named 'VF2' by user 'xysmlx'. The interface includes a left sidebar with navigation links (Overview, Source, Commits, Branches, Pull requests, Downloads, Settings) and a main content area. The 'Overview' tab is active, displaying repository statistics: 1 Branch, 0 Tags, 0 Forks, and 1 Watcher. It also shows the last updated time (3 minutes ago) and the language (C++). A section titled 'VF2' describes the repository as a C++ implementation of a graph isomorphism algorithm. Below this, the 'Build' section lists requirements for Visual Studio 2015 and C++ 11 support. The 'Test Data Set' section is also visible. On the right, there is a 'Recent activity' section showing the latest commit pushed by 'Lingxiao Ma'.

Repository	Language	Access level	Branches	Tags	Forks	Watchers
xysmlx VF2	C++	Admin	1	0	0	1

VF2
A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs written in C++.
Course assignment of Massive Graph Data Management of Peking University.

- Author: xysmlx
- E-Mail: xysmlx@gmail.com

Build
Provide Visual Studio Project

Require:

- Microsoft Visual Studio 2015
- C++ 11 support

Test Data Set

Recent activity

- 1 commit
Pushed to xysmlx/vf2
eedf545 Add timer
Lingxiao Ma · 3 minutes ago
- 1 commit
Pushed to xysmlx/vf2
a809caf Format codes
Lingxiao Ma · 38 minutes ago
- 1 commit
Pushed to xysmlx/vf2
6d7cfe7 Bug fixed! Finished!
Lingxiao Ma · 41 minutes ago
- 1 commit
Pushed to xysmlx/vf2
49b1bbb Add all features

目 录

1	VF2算法简介	3
2	程序实现	4
2.1	图-类: class Graph	4
2.1.1	定义代码	4
2.1.2	设计	6
2.2	VF2算法-类: VF2	7
2.2.1	定义代码	7
2.2.2	设计	9
2.3	程序主驱动-类: Solver	11
2.3.1	定义代码	11
2.3.2	设计	11
3	实验	12
3.1	测试数据	12
3.2	测试环境	12
3.2.1	硬件软件配置	12
3.2.2	程序使用说明	13
3.3	实验结果	13
3.3.1	正确性测试	13
3.3.2	运行速度测试	15
4	总结	16

§ 1 VF2算法简介

VF2是子图匹配的算法，子图匹配的定义如定义1所示。VF2的流程用一句话概括，就是搜索+剪枝。其算法的主框架如算法1所示。其剪枝的方法如式1-5所示。

定义1. 给定图 $Q = (V(Q), E(Q), L_V, F)$ 和图 $G = (V(G), E(G), L_V, F)$ ，当且仅当存在一个映射 $g : V(Q) \rightarrow V(G)$ 使得

$$\forall x \in V(Q), F(x) = F'(g(x))$$

且

$$\forall v_1, v_2 \in V(Q), \overrightarrow{v_1 v_2} \in E(Q) \Rightarrow \overrightarrow{g(v_1)g(v_2)} \in E(G)$$

$$R_{pred}(s, n, m) \Leftrightarrow$$

$$\begin{aligned} & (\forall n' \in M_1(s) \cap Pred(G_1, n), \exists m' \in Pred(G_2, m) | (n', m') \in M(s)) \wedge \\ & (\forall m' \in M_2(s) \cap Pred(G_1, m), \exists n' \in Pred(G_1, n) | (n', m') \in M(s)) \end{aligned} \quad (1)$$

$$R_{succ}(s, n, m) \Leftrightarrow$$

$$\begin{aligned} & (\forall n' \in M_1(s) \cap Succ(G_1, n), \exists m' \in Succ(G_2, m) | (n', m') \in M(s)) \wedge \\ & (\forall m' \in M_2(s) \cap Succ(G_1, m), \exists n' \in Succ(G_1, n) | (n', m') \in M(s)) \end{aligned} \quad (2)$$

$$R_{in}(s, n, m) \Leftrightarrow$$

$$\begin{aligned} & (Card(Succ(G_1, n)) \cap T_1^{in}(s) \geq Card(Succ(G_2, m)) \cap T_2^{in}(s)) \wedge \\ & (Card(Pred(G_1, n)) \cap T_1^{in}(s) \geq Card(Pred(G_2, m)) \cap T_2^{in}(s)) \end{aligned} \quad (3)$$

$$R_{out}(s, n, m) \Leftrightarrow$$

$$\begin{aligned} & (Card(Succ(G_1, n)) \cap T_1^{out}(s) \geq Card(Succ(G_2, m)) \cap T_2^{out}(s)) \wedge \\ & (Card(Pred(G_1, n)) \cap T_1^{out}(s) \geq Card(Pred(G_2, m)) \cap T_2^{out}(s)) \end{aligned}$$

(4)

$$R_{new}(s, n, m) \Leftrightarrow$$

$$(Card(\tilde{N}_1 \cap Pred(G_1, n)) \geq Card(\tilde{N}_2 \cap Pred(G_2, n))) \wedge (5)$$

$$(Card(\tilde{N}_1 \cap Succ(G_1, n)) \geq Card(\tilde{N}_2 \cap Succ(G_2, n)))$$

其中，公式1-2保证了两图同构，公式3-5为剪枝条件。

VF2算法流程如算法1所示。

算法 1: VF2

Input: 图 Q , 图 G

Output: 匹配关系

```

1 Function VF2( $Q, G$ ):
2   | Match( $s_0$ );
3 end
4 Function Match( $s$ ):
5   | if  $M(s)$ 覆盖了 $Q$ 的所有点 then
6   |   | Output  $M(s)$ ;
7   |   | return ;
8   | end
9   | 根据 $s$ 计算候选节点匹配集合 $P(s)$ ;
10  | for  $p \in P(s)$  do
11  |   | if 满足公式1-5 then
12  |   |   | 更新状态 $s$ 得到新状态 $s'$ ;
13  |   |   | Match( $s'$ );
14  |   | end
15  | end
16 end

```

§ 2 程序实现

2.1 图-类: class Graph

2.1.1 定义代码

```

1 struct Vertex
2 {
3     int id;
4     int label;

```

```
5     int seq;
6     bool del;
7
8     Vertex(int _id = 0, int _label = 0) : id(_id), label(_label), seq(-1),
        del(0) {}
9     ~Vertex() {}
10 };
11
12 struct Edge
13 {
14     int u;
15     int v;
16     int label;
17     int next;
18     bool del;
19
20     Edge(int _u = 0, int _v = 0, int _label = 0, int _next = -1) : u(_u),
        v(_v), label(_label), next(_next), del(0) {}
21     ~Edge() {}
22
23     bool operator == (const Edge &o) const
24     {
25         return u == o.u&&v == o.v&&label == o.label;
26     }
27 };
28
29 class Graph
30 {
31 public:
32     Graph()
33     {
34         memset(head, -1, sizeof(head));
35         vn = 0;
36         en = 0;
37     }
38     ~Graph() {}
39
40     void init ();
41     void addv(int id, int label);
42     void addse(int u, int v, int label);
43     void adde(int u, int v, int label);
44     void delse(int u, int v, int label);
45     void dele(int u, int v, int label);
46
47 public:
```

```
48     const static int maxv = 250;
49     const static int maxe = 510;
50
51 public:
52     int head[maxv];
53     int vn;
54     int en;
55     Vertex vtx[maxv]; // 0 to vn-1
56     Edge edge[maxe]; // 0 to en-1
57 };
```

2.1.2 设计

图的存储使用链式前向星来存储。链式前向星的效率高于使用`vector`写的邻接表。

链式前向星的标准设计是：

- `head[]`数组：大小为顶点数，存这个点的对应的第一条边在`edge[]`数组的下标
- `edge[]`数组：用数组存储边
- `Edge`边的结构：边的节点 u, v ，边的标号 $label$ ，删除标记 del ，下一个访问的边的下标 $next$
- 添加边：

```
void Graph::addv(int id, int label)
{
    vtx[id] = Vertex(id, label);
    vn++;
}
void Graph::addse(int u, int v, int label)
{
    edge[en] = Edge(u, v, label, head[u]);
    head[u] = en++;
}
```

- 访问一个定点的所有边

```

for (int i = head[u]; ~i; i = edge[i].next)
{
    Edge e = edge[i];
    // Solve this edge
}

```

- 删除边：令边的 $del = 1$ ，由于边 i 和边 i^1 互为反向边，所以直接遍历一次即可。

```

void Graph::dele(int u, int v, int label)
{
    for (int i = head[u]; ~i; i = edge[i].next)
        if (edge[i].u == u && edge[i].v == v && edge[i].label == label)
        {
            edge[i].del = 1;
            edge[i ^ 1].del = 1;
            return;
        }
}

```

图里面用数组存储节点的访问顺序和点的标号，用链式前向星存储图的结构。

2.2 VF2算法-类：VF2

2.2.1 定义代码

```

1  class VF2
2  {
3  public:
4      struct State // State of dfs matching
5      {
6          vector<pii> s; // Store matched pairs
7          // Same means with the paper
8          int core1[Graph::maxv]; // core1[n] = m or -1
9          int core2[Graph::maxv]; // core2[m] = n or -1
10         bool in1[Graph::maxv];
11         bool in2[Graph::maxv];
12         bool out1[Graph::maxv];
13         bool out2[Graph::maxv];
14         State()

```

```

15     {
16         s.clear();
17         memset(core1, -1, sizeof(core1));
18         memset(core2, -1, sizeof(core2));
19         memset(in1, 0, sizeof(in1));
20         memset(in2, 0, sizeof(in2));
21         memset(out1, 0, sizeof(out1));
22         memset(out2, 0, sizeof(out2));
23     }
24 };
25
26 struct Match // Store match
27 {
28     vector<prii> s; // match
29     int id; // graphDB id
30     Match() {}
31     Match(vector<prii> _s, int _id) : s(_s), id(_id) {}
32 };
33
34 public:
35     VF2() {}
36     void init(const vector<Graph> &db); // Init the VF2 class
37     int vf2(const Graph &QG, const int &QID, bool isOutput = 0);
38     // Run VF2 on QueryGraph & DBGraph
39     // (engine)
40
41 private:
42     void GenRevGraph(const Graph &src, Graph &dst); // Generate
43     // reversed graph
44     bool CheckPrev(const State &s, int a, int b); // Prev
45     bool CheckSucc(const State &s, int a, int b); // Succ
46     bool CheckIn(const State &s); // In
47     bool CheckOut(const State &s); // Out
48     bool CheckNew(const State &s); // New
49     void CalIDFSVec(const State &s); // Cal all temp vec per
50     // dfs
51     void CalCheckVec(const State &s, int a, int b); // Cal all temp vec per
52     // check
53     bool check(const State &s, int a, int b); // Check feasibility
54     void GenPairs(const State &s); // Generate all pairs
55     // to var allPairs
56     void CheckPairs(const State &s); // Check allPairs,
57     // return candiPairs
58     void UpdateState(State &s, int a, int b); // Update state ns
59     // with pair(a,b)
60     bool FinalCheck(const State &s); // Final check for
61     // answer
62     bool dfs(const State &s); // VF2 dfs (recursive)

```



```

54 |     bool query();                                // Run VF2 on pat &
      |         g (main procedure)
55 |
56 | public:
57 |     vector<Graph> DBGraph; // Graph database
58 |     Graph QueryGraph;     // Graph for query, pattern
59 |     int QueryID;          // Store the Query ID
60 |
61 |     vector<int> match; // Store the matched graph info
62 |
63 | private:
64 |     // Temp var
65 |     vector<prii> tlist; // Store temp list of match
66 |     vector<prii> allPairs; // Generated pairs
67 |     vector<prii> candiPairs; // Candidate pairs pass check()
68 |     bool flagIn, flagOut, flagAll; // Flags
69 |     vector<int> pred1, pred2; // Pred of a in pat & b in g
70 |     vector<int> succ1, succ2; // Succ of a in pat & b in g
71 |     vector<int> m1, m2; // M_1, M_2
72 |     vector<int> tin1, tin2; // Tin_1, Tin_2
73 |     vector<int> tout1, tout2; // Tout_1, Tout_2
74 |     vector<int> n1, n2; // N_1, N_2
75 |     vector<int> ns1, ns2; // Point set of pat & g
76 |     vector<int> t1, t2; // tin+tout
77 |
78 |     Graph pat, g; // Pattern & DBGraph
79 |     Graph revpat, revg; // Reversed Pattern & DBGraph
80 | };

```

2.2.2 设计

变量 变量命名按照论文[1]的变量名称进行命名。为了使得程序更快，本文选择用开全局变量的方式代替传递参数的方式，所以VF2类中有大量的临时变量，临时变量就不介绍了，这里仅介绍有意义的主要变量。

- *DBGraph*: 图的数据库，使用链式前向星存储
- *QueryGraph*: 当前进行query的图模式
- *QueryID*: 当前进行query的图模式的编号
- *match*: query的匹配结果（包括匹配的数量，以及点-点映射的结果）
- *pat*: 模式图

- g : 目标图
- $revpat$: 模式图的反向图
- $revg$: 目标图的反向图

方法

- $init$: 初始化VF2
- $vf2$: VF2算法的主控
- $GenRevGraph$: 根据传入的图, 生成这个图的反向图 (把边反向), 反向图用于在链式前向星中快速查询反向边
- $CheckPrev$: 判定状态是否满足公式1
- $CheckSucc$: 判定状态是否满足公式2
- $CheckIn$: 判定状态是否满足公式3
- $CheckOut$: 判定状态是否满足公式4
- $CheckNew$: 判定状态是否满足公式5
- $CalDFSVec$: 计算用于判定公式1-5的各个变量数组, 离线计算, 减小时间复杂度
- $CalCheckVec$: 计算用于判定公式1-5的各个变量数组, 离线计算, 减小时间复杂度
- $check$: 判定状态是否满足公式1-5, 以及点的label是否匹配
- $GenPairs$: 生成所有候选点对集合
- $CheckPairs$: 筛选GenPairs函数生成的集合, 生成最终的候选点对集合 $P(s)$, 离线计算
- $UpdateState$: 加入一个新的匹配, 更新状态
- $FinalCheck$: 判定两个图的边label是否匹配
- dfs : 算法1的Match函数
- $query$: 算法1的VF2函数

2.3 程序主驱动-类：Solver

2.3.1 定义代码

```
1  class Solver
2  {
3  public:
4      void init (bool _isOutput = 0); // init
5      void input(); // input
6      void solve(); // solve
7      void output(); // output
8
9  private:
10     void ReadFile(string path, vector<Graph> &vec); // Read file from
        path, write graph data to vec
11     void ReadDB(string path); // Read DB file
12     void ReadQuery(string path); // Read Query file
13
14     void PrintQueryAns(int id, int cnt); // Print query ans, id = query file
        id, cnt = match num
15
16 private:
17     bool isOutput; // Is output the matching ans?
18     VF2 vf2; // VF2 main component
19     string dbPath; // DB file path
20     vector<string> queryPath; // Query file path vector
21     vector<string> outputPath; // Output ans file path
22
23     vector<Graph> DBGraph; // Store DB
24     vector<Graph> QueryGraph; // Store Query
25 };
```

2.3.2 设计

Solver类是整个程序的主驱动，包含了读取输入文件，运行VF2，输出运行时间和运行结果。从Data文件夹读取输入数据，将匹配结果输出到Output文件夹中，对每个query文件的匹配结果输出一个文件。并将运行时间输出至time.txt中。

变量

- *isOutput*: 选择是否输出匹配结果，1表示是，0表示否
- *vf2*: VF2类的对象，VF2算法

- *dbPath*: 图的数据库文件路径
- *queryPath*: query的模式图的文件路径集合
- *outputPath*: query输出的文件路径集合
- *DBGraph*: 读入的图的数据库, 使用链式前向星存储
- *QueryGraph*: 读入的一个query的模式图

方法

- *init*: 初始化
- *input*: 调用ReadDB从文件读取整个10000的图数据,
- *solve*: 主控, 调用vf2, 并计时
- *output*: 用于输出, 目前无用
- *ReadFile*: 用ifstream从文件读取图数据
- *ReadDB*: 调用ReadFile读取DB文件
- *ReadQuery*: 调用ReadFile读取Query文件
- *PrintQueryAns*: 打印一个Query和整个DB的匹配结果

§ 3 实验

3.1 测试数据

实验选择graphDB数据集进行运行时间测试, 选择graphDB数据集的一个子集进行正确性测试。

graphDB数据集包括大小为10000个图的图数据库文件"mygraphdb.data", 以及6个query图数据文件"Q4.my", "Q8.my", "Q12.my", "Q16.my", "Q20.my", "Q24.my"。

3.2 测试环境

3.2.1 硬件软件配置

使用的计算机硬件和软件配置如表1所示。

表 1 测试环境（计算机配置和编译环境）

项目	详细信息
CPU	Core i7-2630QM (2.0GHz, 4 Cores, 6MB L3 Cache)
内存	16GB DDR3 1600MHz
测试所用磁盘	480GB Sandisk Extreme Pro SSD (Read: 550MB/s)
操作系统	Windows 10 Professional x64 Build 10240
C/C++编译器	Microsoft Visual Studio 2015
C++版本	需支持C++11

该代码已上传至BitBucket的私有仓库，课程结束后会开源。

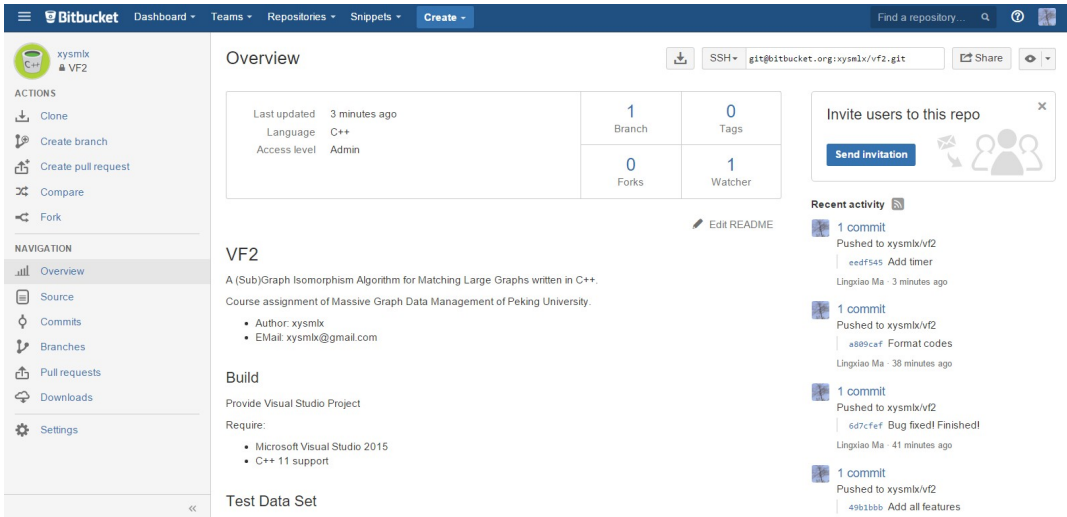


图 1 BitBucket私有仓库

3.2.2 程序使用说明

程序从/Data读取”mygraphdb.data”, ”Q4.my”, ”Q8.my”, ”Q12.my”, ”Q16.my”, ”Q20.my”, ”Q24.然后把运行时间输出到”time.txt”, 可选是否输出匹配结果(代码中选项vf2的isOutput选项)

3.3 实验结果

3.3.1 正确性测试

正确性验证使用Q4的第一个和DB的一些数据进行匹配，下面展示Q4的第一个数据和DB的第一个数据的匹配结果，如下所示：

Q4第一个

```
t # 0
v 0 2
v 1 2
v 2 2
v 3 2
v 4 2
e 0 1 2
e 1 2 2
e 2 3 2
e 3 4 2
```

DB第一个

```
t # 0
v 0 2
v 1 2
v 2 2
v 3 3
v 4 2
v 5 4
v 6 2
v 7 3
v 8 3
v 9 3
v 10 2
v 11 2
v 12 2
v 13 2
v 14 2
v 15 2
v 16 2
v 17 2
v 18 2
e 0 1 2
e 0 2 2
e 2 3 3
e 2 4 2
e 3 5 2
```

```
e 4 6 2
e 5 7 2
e 5 8 2
e 5 9 2
e 6 10 2
e 6 9 3
e 7 11 3
e 8 12 3
e 10 13 2
e 11 14 2
e 11 15 2
e 12 16 2
e 12 15 2
e 14 17 2
e 16 18 2
```

匹配结果

```
0 0
1 2
2 4
3 6
4 10
```

显然匹配正确。由于篇幅所限，就不把所有的测试结果展出了。测试一共进行了15组，均正确。

3.3.2 运行速度测试

本文对程序做了以下运行速度测试：

- "Q4.my", "Q8.my", "Q12.my", "Q16.my", "Q20.my", "Q24.my" 文件的一个query在数据库中匹配一个图的时间，如图2所示。
- "Q4.my", "Q8.my", "Q12.my", "Q16.my", "Q20.my", "Q24.my" 文件的一个query在数据库中匹配所有图的时间，如图3所示。
- 由于每个query文件全跑完的时间太久，所以本文仅将"Q4.my"完整的跑完，并给出了运行时间，共花费了24617.9s。

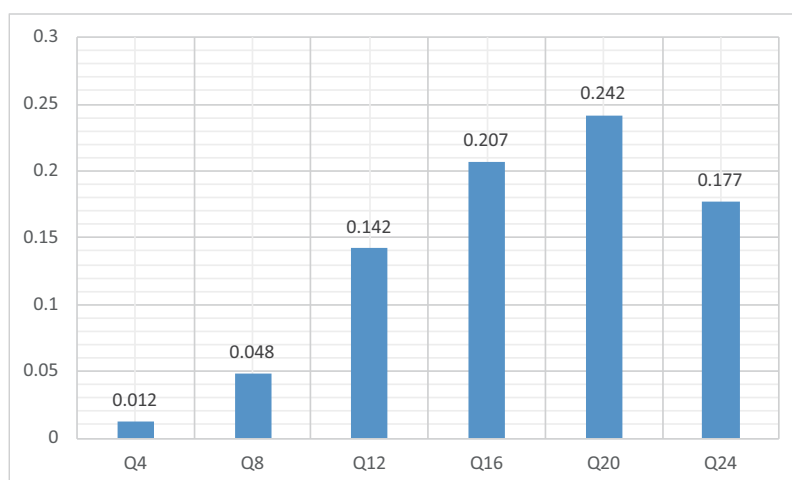


图 2 一个query在数据库中匹配一个图的时间(单位: s)

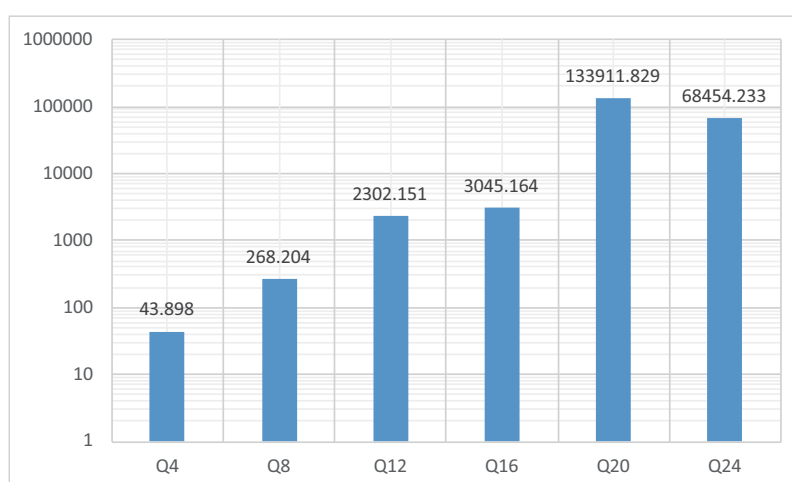


图 3 一个query在数据库中匹配所有图的时间(单位: s)

§ 4 总结

本文用C++实现了子图匹配算法VF2，通过加入大量优化，比如：尽可能离线计算数据、用链式前向星代替STL实现的邻接表、尽可能保留后面会用到的数据、用尽可能快的库函数，使得实现的算法在某些情况下（查询的图比较小的时候）速度比开源库boost的VF2算法更快。通过正确性实验，跑了15组数据，并手动验证，发现15组数据均正确，可以证明算法的正确性。

参考文献

- [1] Cordella L P, Foggia P, Sansone C, et al. A (sub) graph isomorphism algorithm for matching large graphs[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2004, 26(10): 1367-1372.