# gSpan实验报告

海量图数据的管理和挖掘

2015 年 10 月 31 日

**作者：马凌霄**
**学号：1501111302**
**院系：信息科学技术学院**
**EMail：xysmlx@gmail.com**
**Private Repository:** https://bitbucket.org/xysmlx/gspan

# 目　　录

# § 1 gSpan算法简述

# § 2 程序设计

## 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：
-
-

## 2.2 输入和重排序-类：class InputFilter

### 2.2.1 定义代码

```
class InputFilter
{
public:
    struct Node
    {
        int label;
        int cnt;
        Node(int _label = 0, int _cnt = 0) : label(_label), cnt(_cnt) {}
        bool operator < (const Node &o) const // greater
        {
            return cnt > o.cnt;
        }
    };

public:
    void init();
    void addv(int id, int label);
    void adde(int u, int v, int label);
    void filterV();
    void filterE();
    void filter();

public:
    const static int maxv = 1010;
    const static int maxe = 5010;

public:
    int cntv[maxv], cnte[maxe];
    int mpv[maxv], mpe[maxe];
    vector<Vertex> vecv;
    vector<Edge> vece;
    vector<int> listv, liste;
    vector<Node> filterv, filtere;
    vector<string> inputStr;
};
```

### 2.2.2 设计

## 2.3 DFS Code五元组节点-结构体：struct DFSCodeNode

### 2.3.1 定义代码

```
struct DFSCodeNode
```

```
2  {
3      int a, b;
4      int la, lab, lb;
5
6      DFSCodeNode(int _a = -1, int _b = -1, int _la = -1, int _lab = -1,
           int _lb = -1) : a(_a), b(_b), la(_la), lab(_lab), lb(_lb) {}
7      ~DFSCodeNode() {}
8
9      bool isForward() const
10     {
11         return a < b;
12     }
13     bool isBackward() const
14     {
15         return a > b;
16     }
17
18     bool operator < (const DFSCodeNode &o) const
19     {
20         if (this->isBackward() && o.isForward()) return 1;
21         else if (this->isBackward() && o.isBackward() && b < o.b)
               return 1;
22         else if (this->isBackward() && o.isBackward() && b == o.b&&
               lab < o.lab) return 1;
23         else if (this->isForward() && o.isForward() && a > o.a) return
               1;
24         else if (this->isForward() && o.isForward() && a == o.a&&la
               < o.la) return 1;
25         else if (this->isForward() && o.isForward() && a == o.a &&la
               == o.la&&lab < o.lab) return 1;
26         else if (this->isForward() && o.isForward() && a == o.a &&la
               == o.la&&lab == o.lab&&lb < o.lb) return 1;
27         return 0;
28     }
29     bool operator == (const DFSCodeNode &o) const
30     {
31         return a == o.a&&b == o.b&&la == o.la&&lab == o.lab&&lb
               == o.lb;
32     }
33 };
```

### 2.3.2 设计

## 2.4 DFS Code-类：class DFSCode

### 2.4.1 定义代码

```
1  class DFSCode
2  {
3  public:
4      DFSCode()
5      {
6          dfsCodeList.clear();
7          rightPath.clear();
8      }
9      bool operator < (const DFSCode &o) const
10     {
11         int minsize = min(dfsCodeList.size(), o.dfsCodeList.size());
12         for (int i = 0;i < minsize;i++)
13             if (dfsCodeList[i] < o.dfsCodeList[i]) return 1;
14         return dfsCodeList.size() < o.dfsCodeList.size();
15     }
16     bool operator == (const DFSCode &o) const
17     {
18         if (dfsCodeList.size() != o.dfsCodeList.size()) return 0;
19         for (int i = 0;i < (int)dfsCodeList.size();i++)
20             if (!(dfsCodeList[i] == o.dfsCodeList[i])) return 0;
21         return 1;
22     }
23
24 public:
25     void init();
26     void output(); // Output this dfscode
27     Graph Convert2Graph();
28     bool GenMinDFSCode(Graph &g, DFSCode &ret, int now); //
           Generate the min dfscode from now
29     DFSCode FindMinDFSCode(); // Find the min dfscode of this pattern
30     bool isMinDFSCode(); // Is this dfscode the min dfscode?
31
32 public:
33     vector<DFSCodeNode> dfsCodeList;
34     vector<pair<int, int> > rightPath;
35 };
```

### 2.4.2 设计

## 2.5 gSpan算法-类：class GSPAN

### 2.5.1 定义代码

```
1  class GSPAN
2  {
3  public:
4      struct FreqEdgeSortNode
5      {
6          Edge e;
7          int cnt;
8          FreqEdgeSortNode(Edge _e = Edge(), int _cnt = 0) :e(_e), cnt(_cnt)
             {}
9          bool operator < (const FreqEdgeSortNode &o) const // greater
10         {
11             return cnt > o.cnt;
12         }
13     };
14
15 public:
16     GSPAN() {}
17     void init();
18     void input(const InputFilter &_inputFilter, double _minSup); // Build
           relabeled graph
19     void output();
20     void GenSeedSet(); // Generate the seed edge set
21     void DeleteEdgeFlag(const Edge &e); // Label deleted edge
22     void DeleteEdge(const Edge &e); // Delete edge from graph
23     void DeleteUnFreqEdge(); // Delete unfreq edge
24     void RebuildGraph(int id); // Rebuild graph with id
25     bool JudgePatternInGraph(Graph &graph, const DFSCode &dfscode,
           int ith, int now); // DFS, ith = dfscode.dfsCodeList[ith], now =
           now vertex
26     bool isPatternInGraph(Graph graph, const DFSCode &dfscode); // Is
           this pattern in this graph?
27     void SolveFreqPattern(const DFSCode &dfscode); // Work when
           dfscode is freq pattern
28     bool isFreqPattern(const DFSCode &dfscode); // Is dfscode a freq
           pattern?
29     void BuildPattern(DFSCode &dfscode, int loc, int backloc, int maxseq);
           // DFS build pattern and test, loc = now extend location in
           rightpath, backloc = -1(forward) or backward location in rightpath,
           maxseq = max sequence id
30     void SubMining(const Edge &base); // Sub-Mining Procedure
31     void gSpan(); // Run gSpan
32
```

```
33      void debug(); // For debug
34
35  public:
36      const static int maxGraph = 10010; // Maximum graph number of
            graph set
37
38  public:
39      ofstream out; // Output to file
40      DFSCode tmpDFSCode; // Temp dfscode
41
42      double minSup; // minimum support
43      int minSupDeg; // minSup * cntGraph
44
45      Graph graph[maxGraph]; // 0 to cntGraph-1
46      int cntGraph; // Num of graphs in the graph set
47
48      map<Edge, int, EdgeCMP> freqEdgeCnt; // Count edge's frequency
49      set<Edge, EdgeCMP> freqEdgeVis; // Visit or not in a graph
50
51      vector<Edge> freqEdge; // Freq edge set
52      vector<Edge> unFreqEdge; // Unfreq edge set
53
54      vector<DFSCode> freqPattern; // Freq pattern, the answer
55  };
```

#### 2.5.2 设计

## § 3 程序测试

### 3.1 测试环境

测试环境如表1所示：

表 1 实验环境

| 项目 | 详细信息 |
|---|---|
| CPU | AMD Opteron 8380 (2.5GHz, 4 Cores) × 16 |
| 内存 | 64GB ECC DDR2 |
| 测试所用磁盘 | 2TB 7200RPM HDD (Read: 96.5MB/s) |
| 操作系统 | Ubuntu 12.04.2 LTS x64 |
| C/C++编译器 | GNU C++ 4.8 |

## 3.2 正确性测试

### 3.2.1 测试数据

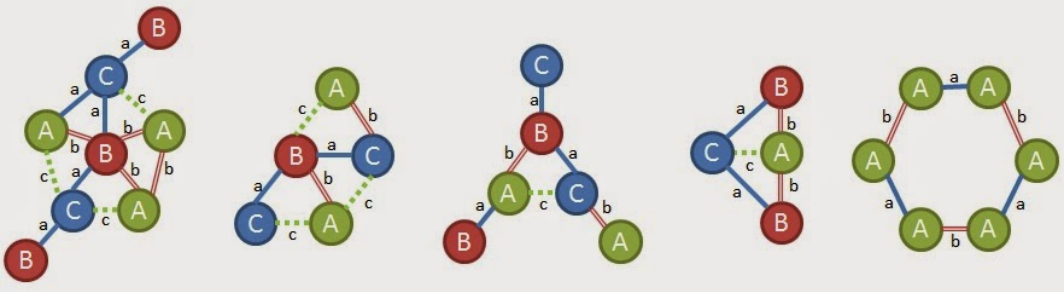

图 1 正确性测试的测试数据

```
1  t # 0
2  v 0 3
3  v 1 4
4  v 2 2
5  v 3 3
6  v 4 2
7  v 5 2
8  v 6 4
9  v 7 3
10 e 0 1 2
11 e 1 2 2
12 e 2 3 3
13 e 1 3 2
14 e 3 4 3
15 e 1 4 4
16 e 4 5 3
17 e 3 5 3
18 e 5 6 4
19 e 3 6 2
20 e 6 2 4
21 e 6 7 2
22 t # 1
23 v 0 2
24 v 1 3
25 v 2 4
26 v 3 2
27 v 4 4
28 e 0 1 4
29 e 1 2 2
30 e 0 2 3
```

```
31 e 2 3 4
32 e 1 3 3
33 e 3 4 4
34 e 1 4 2
35 t # 2
36 v 0 4
37 v 1 3
38 v 2 2
39 v 3 3
40 v 4 4
41 v 5 2
42 e 0 1 2
43 e 1 2 3
44 e 2 3 2
45 e 2 4 4
46 e 1 4 2
47 e 4 5 3
48 t # 3
49 v 0 3
50 v 1 2
51 v 2 3
52 v 3 4
53 e 0 1 3
54 e 1 2 3
55 e 2 3 2
56 e 3 0 2
57 e 3 1 4
58 t # 4
59 v 0 2
60 v 1 2
61 v 2 2
62 v 3 2
63 v 4 2
64 v 5 2
65 e 0 1 2
66 e 1 2 3
67 e 2 3 2
68 e 3 4 3
69 e 4 5 2
70 e 5 0 3
71 t # -1
```

### 3.2.2 输出结果

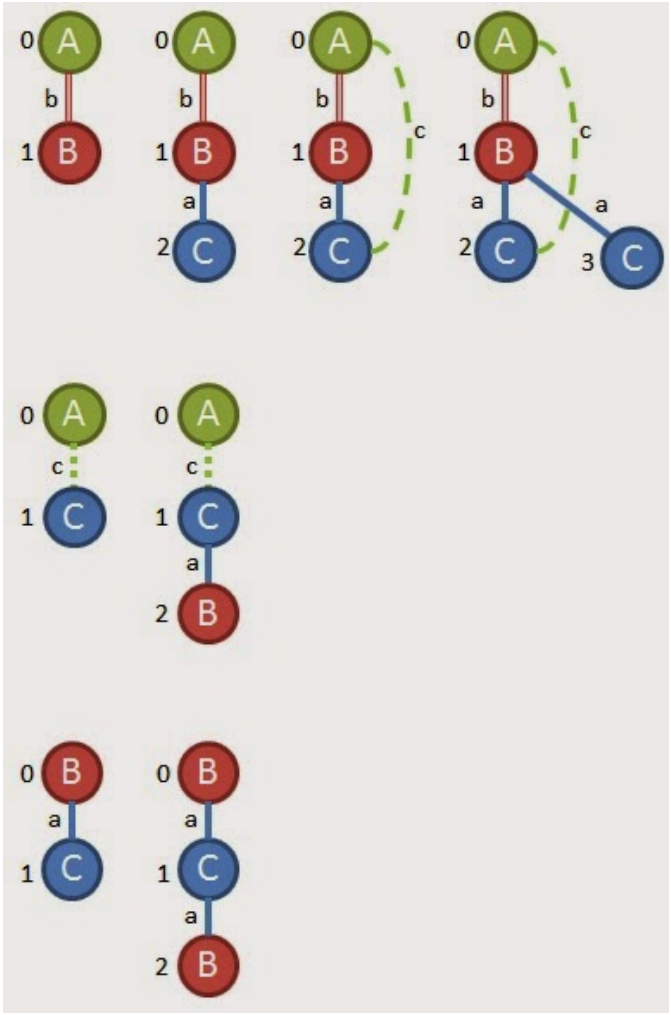


图 2 正确性测试的结果

```
1  Seed Set:
2  2 3 3
3  2 4 4
4  3 4 2
5  Pattern #1:
6  0 1 2 3 3
7  Pattern #2:
8  0 1 2 3 3
9  1 2 3 2 4
10 Pattern #3:
11 0 1 2 3 3
12 1 2 3 2 4
13 2 0 4 4 2
14 Pattern #4:
```

```
0 1 2 3 3
1 2 3 2 4
2 0 4 4 2
1 2 3 2 4
Pattern #5:
0 1 2 4 4
Pattern #6:
0 1 2 4 4
1 2 4 2 3
Pattern #7:
0 1 2 4 4
1 2 4 2 3
2 2 3 2 4
Pattern #8:
0 1 3 2 4
```

### 3.3 运行速度测试

#### 3.3.1 测试数据

测试数据为graph.data：图集合中有10000个图，每个图最大点数不超过250，边数不超过250。

#### 3.3.2 对照程序

使用Xifeng Yan, Jiawei Han的原作者的程序[1]进行运行速度测试的对比，*minSup*阈值选取和运行时间数据如表2所示。运行时间作图如图3所示。

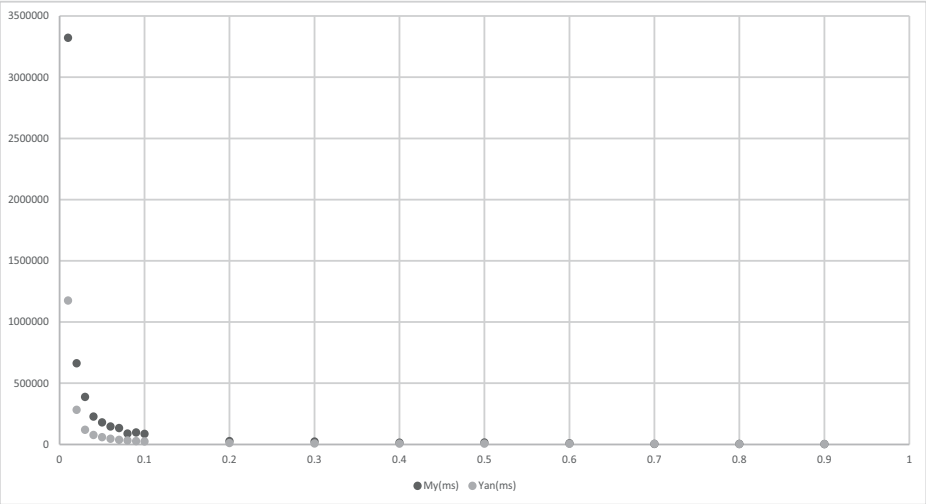


图 3 运行时间测试

[1] http://www.cs.ucsb.edu/~xyan/software/gSpan2009-02-20/gSpan6.tar.gz

表 2 运行时间测试

| minSup | My(ms) | Yan(ms) |
|---|---|---|
| 0.9 | 2050 | 1837 |
| 0.8 | 2790 | 2299 |
| 0.7 | 3520 | 4298 |
| 0.6 | 8510 | 4554 |
| 0.5 | 14920 | 5749 |
| 0.4 | 13690 | 6001 |
| 0.3 | 22620 | 7318 |
| 0.2 | 25940 | 11164 |
| 0.1 | 85820 | 23648 |
| 0.09 | 96650 | 26878 |
| 0.08 | 87780 | 31235 |
| 0.07 | 133020 | 36047 |
| 0.06 | 146220 | 45141 |
| 0.05 | 178760 | 58311 |
| 0.04 | 227410 | 75880 |
| 0.03 | 387420 | 119003 |
| 0.02 | 772316 | 282170 |
| 0.01 | 4421358 | 1175110 |

从表??和图3看出，本程序的性能不如原作者的程序的性能。还有一些优化可以加入，但是因为时间的原因，所以没有加入程序。

### 优化策略

- 链式前向星写的图的点数和边数开的较大，可以根据图的大小开；
- 将枚举构造频发模式改为在图集合中查找构造频繁模式；
- gSpan有很好的并行性，所以可以使用多线程：将每个任务放入一个队列，然后让每个线程从队列中提取任务执行。

## § A 备注

### A.1

```
1 #include<cstdio>
2 int main()
3 {
4     return 0;
5 }
```