实验报告

子图同构算法：VF2算法

谭鑫 1601111294

1.实验环境

操作系统	OS X EI Capitan
处理器	2.9 GHz Intel Core i5
内存	8 GB 1867 MHz DDR3

2. 算法设计

所谓的子图同构任务，即给定一个target graph，G1=(N1,B1)和一个query graph，G2=(N2,B2)。其中N是点集，B是边集。我们希望找到一组映射(n,m)（其中n ∈ G1,m ∈ G2），使得两个图的对应点label相同，query graph中的边，在target graph中对应点之间都存在，且label相同。

算法伪代码如下：

```
PROCEDURE Match(s)
    INPUT:   an intermediate state s; the initial state s0 has M(s0)=∅
    OUTPUT:  the mappings between the two graphs

    IF M(s) covers all the nodes of G2 THEN
      OUTPUT M(s)
    ELSE
      Compute the set P(s) of the pairs candidate for inclusion in M(s)
      FOREACH p in P(s)
        IF the feasibility rules succeed for the inclusion of p in M(s) THEN
          Compute the state s´ obtained by adding p to M(s)
          CALL Match(s')
        END IF
      END FOREACH
      Restore data structures
    END IF
END PROCEDURE Match
```

起初，状态是s0，M(s0)是空集，即还没有任何匹配。之后递归的进行搜索。如果当前状态s代表的部分匹配M(s)包含了G2（query graph）中的所有节点，则已经找到了G2在G1中同构的子图，搜索结束。否则，在当前的局部匹配基础上，再匹配一个点。首先，找出所以可能进行匹配点对集合P(s)。然后，对于每一个匹配对p，检查加入匹配p是否可行。即加入p后，两个图还是否同构。以及加入p之后，是否还有就扩展的可能性（即实行一些剪枝策略）。

如果加入匹配p可行，则将p加入s，递归调用Match(s)，继续搜索。如果刚才若干次调用Match(s)后都没有找到同构的子图，则说明当前从状态不可能扩展出可行的子图同构匹配。所以，将生成改状态时加入的两点匹配p从s中删除，回溯到上一个状态。

检查新匹配的可行性：

$$
\begin{aligned}
&R_{\text{pred}}(s,n,m) \Longleftrightarrow \\
&(\forall n' \in M_1(s) \cap \text{Pred}(G_1,n) \exists m' \in \text{Pred}(G_2,m) \mid (n',m') \in M(s)) \wedge \\
&(\forall m' \in M_2(s) \cap \text{Pred}(G_2,m) \exists n' \in \text{Pred}(G_1,n) \mid (n',m') \in M(s)),
\end{aligned} \tag{3}
$$

$$
\begin{aligned}
&R_{\text{succ}}(s,n,m) \Longleftrightarrow \\
&(\forall n' \in M_1(s) \cap \text{Succ}(G_1,n) \exists m' \in \text{Succ}(G_2,m) \mid (n',m') \in M(s)) \wedge \\
&(\forall m' \in M_2(s) \cap \text{Succ}(G_2,m) \exists n' \in \text{Succ}(G_1,n) \mid (n',m') \in M(s)),
\end{aligned} \tag{4}
$$

$$
\begin{aligned}
&R_{\text{in}}(s,n,m) \Longleftrightarrow \\
&(\text{Card}(\text{Succ}(G_1,n) \cap T_1^{\text{in}}(s)) \geq \text{Card}(\text{Succ}(G_2,m) \cap T_2^{\text{in}}(s))) \wedge \\
&(\text{Card}(\text{Pred}(G_1,n) \cap T_1^{\text{in}}(s)) \geq \text{Card}(\text{Pred}(G_2,m) \cap T_2^{\text{in}}(s))),
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
&R_{\text{out}}(s,n,m) \Longleftrightarrow \\
&(\text{Card}(\text{Succ}(G_1,n) \cap T_1^{\text{out}}(s)) \geq \text{Card}(\text{Succ}(G_2,m) \cap T_2^{\text{out}}(s))) \wedge \\
&(\text{Card}(\text{Pred}(G_1,n) \cap T_1^{\text{out}}(s)) \geq \text{Card}(\text{Pred}(G_2,m) \cap T_2^{\text{out}}(s))),
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
&R_{\text{new}}(s,n,m) \Longleftrightarrow \\
&\text{Card}(\tilde{N}_1(s) \cap \text{Pred}(G_1,n)) \geq \text{Card}(\tilde{N}_2(s) \cap \text{Pred}(G_2,n)) \wedge \\
&\text{Card}(\tilde{N}_1(s) \cap \text{Succ}(G_1,n)) \geq \text{Card}(\tilde{N}_2(s) \cap \text{Succ}(G_2,n)).
\end{aligned} \tag{7}
$$

3. 算法实现

定义了Edge、Graph和State三个类：

Edge		
类属性	数据类型	含义
u	int	边的起点
v	int	边的终点
label	int	边的标签，属性
next	int	边在邻接表中的下一条边的编号
prev	int	边在邻接表中的前一条边的编号

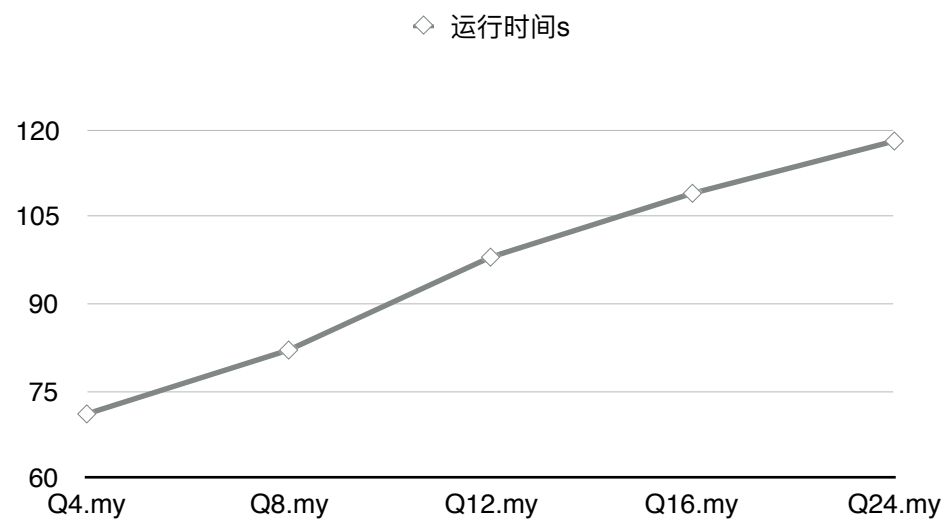
Graph		
类属性	数据类型	含义
vertex_count	int	图中点的总数
edge_count	int	图中边的总数
vertex	vector<VLabel>	每个点的标签，属性
edge	vector<Edge>	存储图中的边
head_edge	vector<EIndex>	邻接表的顶点数组
rev_head_edge	vector<EIndex>	邻接表的反向顶点数组，用于寻找所有以某一点为终点的边
pred	vector<set<VIndex>>	存储每个点的前继节点
succ	vector<set<VIndex>>	存储每个点的后继节点
类方法		含义
void addVertex(int label)		向图中增加一个点
void addEdge(int u, int v, int label)		向图中添加一条从u到v，标号为label的边
void initial()		初始化，清空一个图
void printGraphInfo() const		输出图的结构信息
State		
类属性	数据类型	含义
vertex_count	int	查询图的点数
subisomorphism	bool	当前查询为子图同构还是图同构
M1	set<VIndex>	G1（或G2）中已成功匹配的点
M2		
core_1	vector<VIndex>	G1（或G2）中每个点在G2（或G1）所匹配的点，如果当前还没有匹配，设成NULL_VIndex
core_2		
in_1	set<VIndex>	能直接连接M1（或M2）中的点，且不属于M1（或M2）的点的集合
in_2		
out_1	set<VIndex>	能被M1（或M2）中的点直接连接，且

out_2	不属于M1（或M2）的点的集合
类方法	含义
vector<pair<VIndex, VIndex>> genCandiPairSet()	生成当前状态下所有可能的匹配节点对
void addNewPair(VIndex n, VIndex m…)	添加G1中的点n到G2中的点m的匹配到当前状态
bool checkPredRule(const Graph &G1, const Graph &G2, VIndex n, VIndex m)	类似功能的类方法还有：checkSuccRule，checkInRule，checkOutRule，checkNewRule， 分别检查是否满足论文中提到的五项语法条件
set<VIndex> genComplementary(…)	检查规则的辅助函数，用于计算两个集合在第三个集合的补集
int set_intersection_size(const set<VIndex> &a, const set<VIndex> &b)	检查规则的辅助函数，用于计算两个集合交集的大小
bool checkSynRules(const Graph &G1, const Graph &G2, VIndex n, VIndex m)	论文中提及的“语法”规则，即检查候选的映射(n->m)是否破坏了两图结构的匹配
bool checkSemRules(const Graph &G1, const Graph &G2, VIndex n, VIndex m)	论文中提及的“语义”规则，即检查候选的映射(n->m)是否破坏了两图性质的匹配

4. 实验结果

实验结果在graphDB/output.txt中给出

图同构：



子图同构：

mygraphdb.data中的10000个图，由于数据量太大，每个查询文件只用了前500个图。