Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心

频繁子图挖掘算法gSpan

李蕊　　硕士研究生

2019年09月15日

内容提要

- 背景简介
- 基本概念
- 算法原理
- 优劣分析
- 应用总结
- 参考文献

背景简介

- 预期收获
  - 1. 熟悉频繁子图挖掘算法gSpan（Graph-Based Substrcture Pattern Mining ）的历史现状及应用场景
  - 2. 理解频繁子图挖掘算法gSpan的算法原理
  - 3. 了解频繁子图挖掘算法gSpan在安卓恶意软件中的应用

背景简介

- 算法提出的原因
  - 传统的数据挖掘任务（关联规则挖掘、聚类等），都是试图从一个具有单一关系的独立实例中寻找模式，缺乏样本之间连接的潜在关系
  - 图可以挖掘样本之间的复杂关系
  - 频繁子图是图挖掘的重要分支，可以寻找在图数据集中频繁出现的子结构

背景简介

- 算法提出的原因
  - 基于Apriori算法缺点：

1）检验子图是否同构耗费很多时间

2）子图候选生成复杂且效率低

gSpan 能解决这些问题：

1）消除候选生成

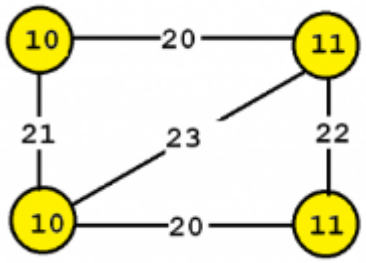
2）用深度优先DFS(depth-first search)不需要多次扫描

3）引入新的词典规范标签系统，不需要检查子图同构

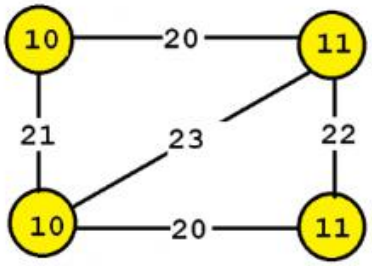
4）以上三点使gSpan效率高

基本概念

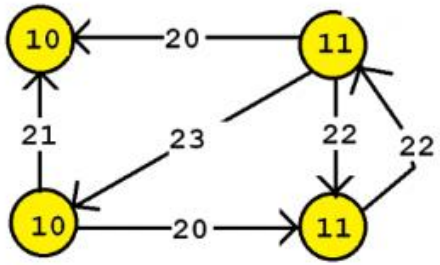
- 基本概念
  - 标记图:四个元素（顶点，边，标签，顶点和边的映射函数）



  - 无向图和有向图



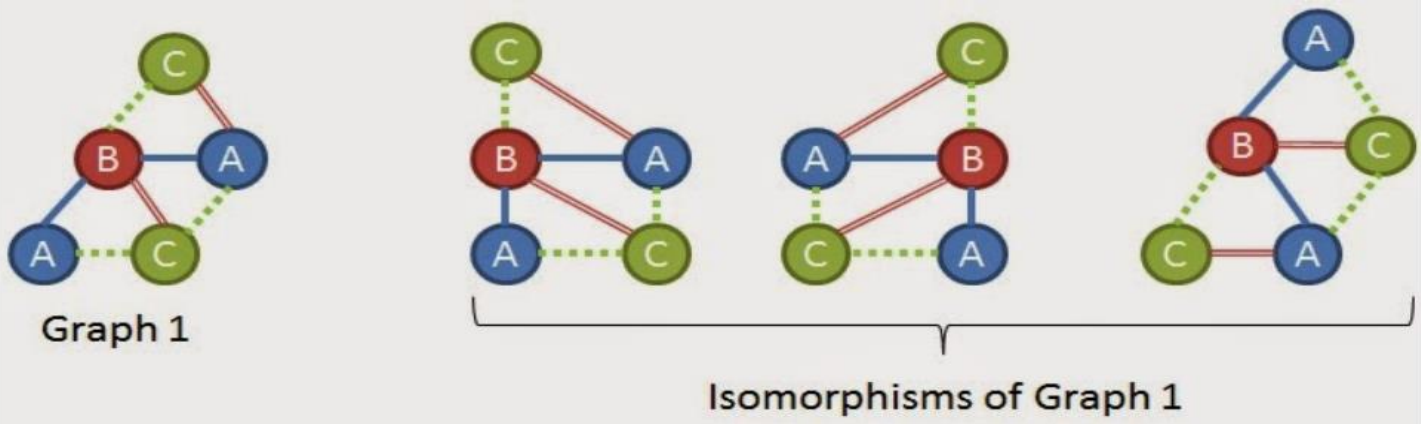
An undirected graph



A directed graph

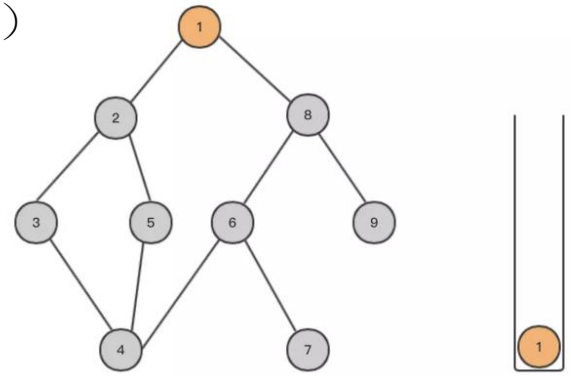
基本概念

- 同构:两个图在拓扑结构上是相同的

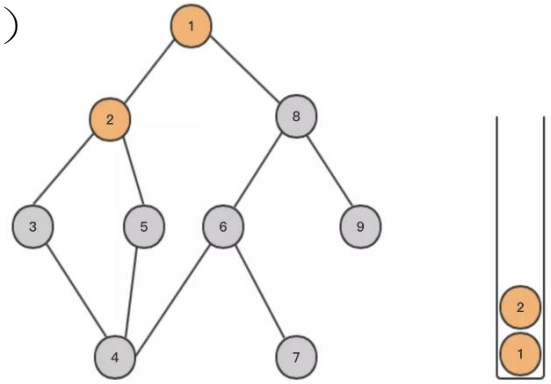


- 深度优先搜索DFS:对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次，用堆来实现

（1）

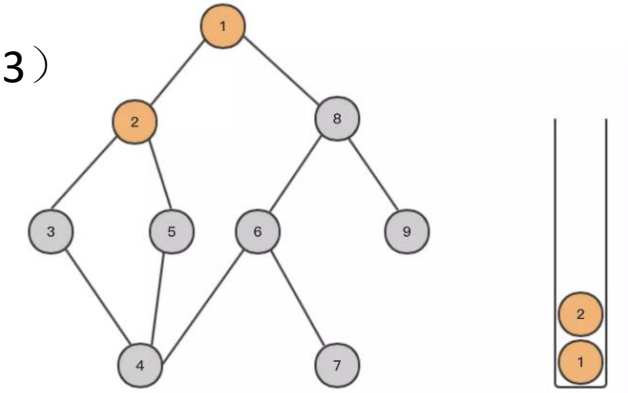


（2）



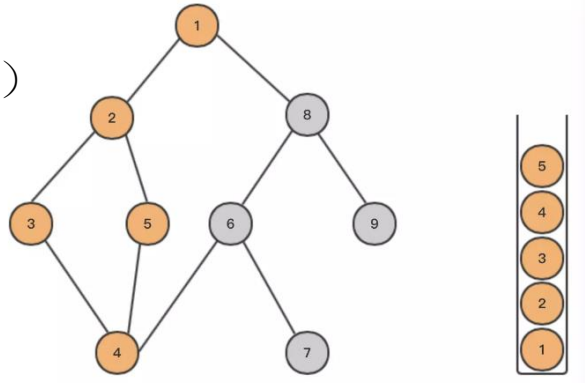
基本概念

（3）



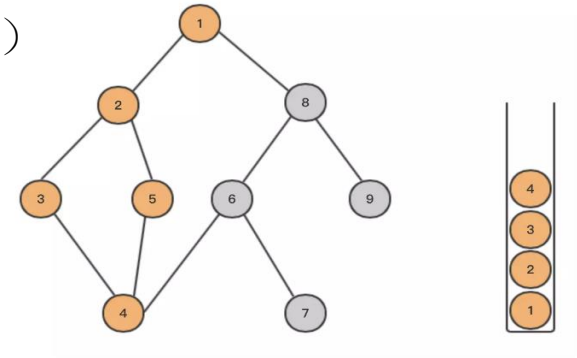
找出与此点邻接的且尚未遍历的点，进行标记，然后放入堆中，
依次进行

（4）



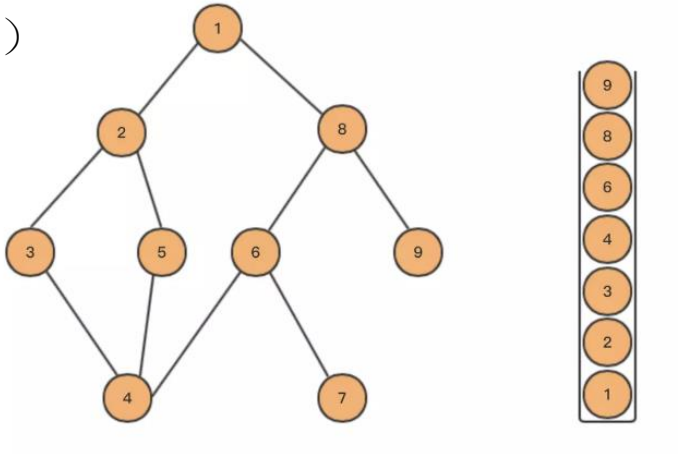
如果此点没有尚未遍历的邻接点，则将此点从堆中弹出，再按照（3）依次进行

（5）

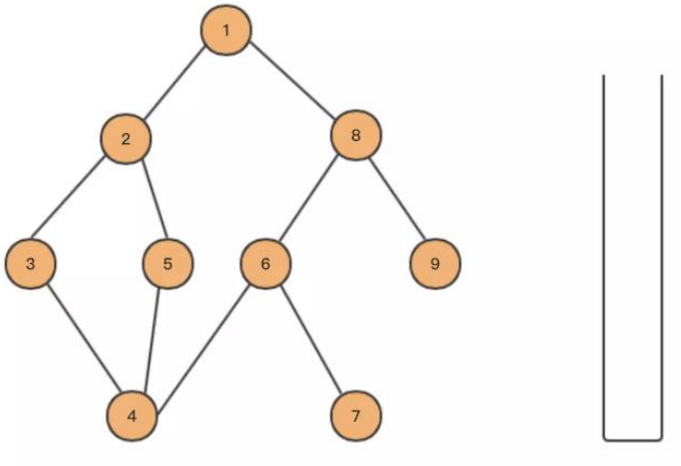


基本概念

（6）



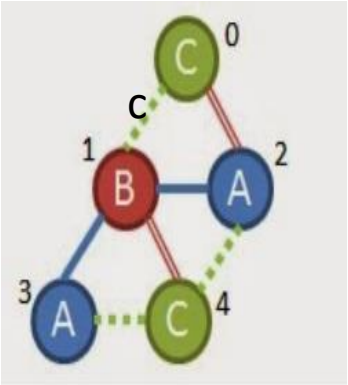
直到遍历完整个树，
堆里的元素都将弹出，
最后栈为空，DFS遍历完成



基本概念

– DFS 编码

Edge Number	DFS Code
0	(0,1,C,c,B)
1	(1,2,B,a,A)
2	(2,0,A,b,C)
3	(1,3,B,a,A)
4	(3,4,A,c,C)
5	(4,1,C,b,B)
6	(4,2,C,c,A)

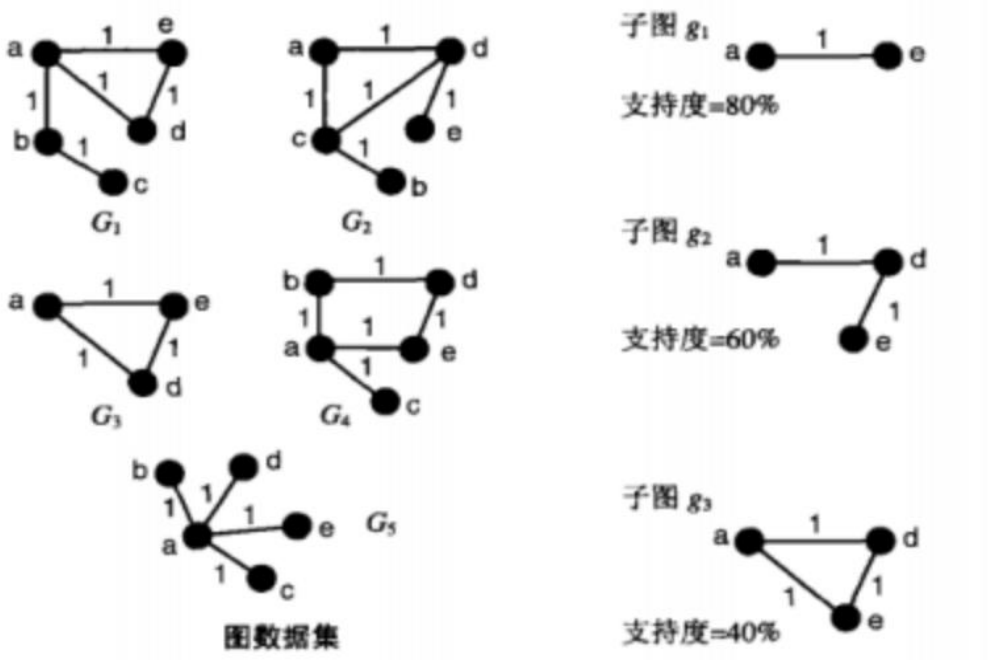


Vertex Symbol	Vertex Label	Edge Symbol	Edge Label
A	A	—	a
B	B	=	b
C	C	·····	c

– 支持值 Support

$$\text{Support} = \frac{\text{子图出现次数}}{\text{子图总数量}}$$

基本概念



– 频繁子图挖掘

给定集合ς和支持度阈值$minsup$，频繁子图挖掘的目标是找出使得所有$s(g) \geq minsup$ 的子图g

算法原理

T	基于有标签数据寻找频繁子图
I	有标签无向图数据，最小支持值(min Support)

P	1.获取多个同构图的正确DFS字典顺序
	2.获取多个同构图的最小DFS编码
	For {
	1. 计算DFS码的最小支持值
	2. 更新频繁子图集
	}

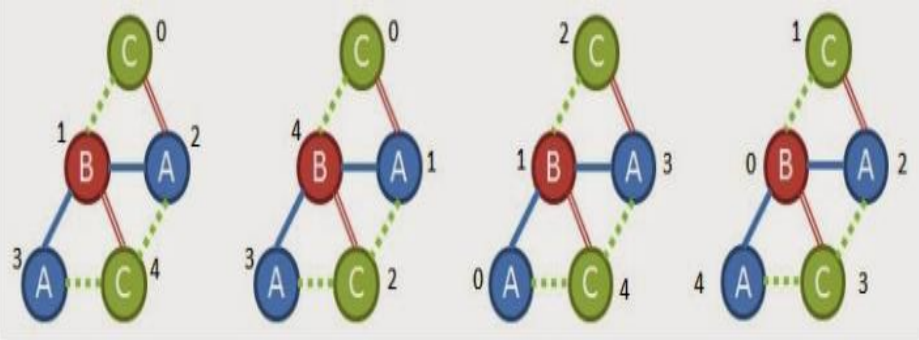
O	频繁子图集

P	尽可能提高遍历效率
C	频繁子图是连通图
D	如何找到最小DFS序列
L	CCF B类期刊

算法原理

- 算法步骤
  - 1. 多个同构图的正确DFS字典排序

Vertex Symbol	Vertex Label	Edge Symbol	Edge Label
A	A	——	a
B	B	——	b
C	C	·····	c



Edge Number	Example 1	Example 2	Example 3	Example 4
0	(0,1,C,c,B) ✓	(0,1,C,c,B) ✓	(0,2,C,b,A) ✗	(0,1,C,c,B) ✗
1	(1,2,B,a,A) ✓	(1,3,B,a,A) ✓	(2,1,A,a,B) ✗	(1,3,B,a,A) ✓
2	(2,0,A,b,C) ✓	(3,4,A,c,C) ✓	(1,4,B,b,C) ✗	(3,4,A,c,C) ✓
3	(1,3,B,a,A) ✓	(4,2,C,c,A) ✗	(4,3,C,c,A) ✗	(4,1,C,b,B) ✗
4	(3,4,A,c,C) ✓	(2,0,A,b,C) ✗	(3,1,A,a,B) ✗	(1,2,B,a,A) ✗
5	(4,1,C,b,B) ✓	(1,2,B,a,A) ✗	(1,0,B,c,C) ✗	(2,4,A,c,C) ✗
6	(4,2,C,c,A) ✓	(1,4,B,b,C) ✗	(2,4,A,c,C) ✗	(2,0,A,b,C) ✗

$$a_k = (i_k, j_k, l_{i_k}, l_{(i_k,j_k)}, l_{j_k}),\ and\ a_{k+1} = (i_{k+1}, j_{k+1}, l_{i_{k+1}}, l_{(i_{k+1},j_{k+1})}, l_{j_{k+1}}).$$

rule 1. if a_k is a backward edge, then either of the following holds.

(i) if a_{k+1} is a forward edge, $i_{k+1} \leqslant i_k$ and $j_{k+1} = i_k + 1$;

(ii) if a_{k+1} is a backward edge, $i_{k+1} = i_k$ and $j_k < j_{k+1}$.

rule 2. if a_k is a forward edge, then either of the following holds.

(i) if a_{k+1} is a forward edge, $i_{k+1} \leqslant j_k$ and $j_{k+1} = j_k + 1$;

(ii) if a_{k+1} is a backward edge, $i_{k+1} = j_k$ and $j_{k+1} < i_k$.

算法原理

- 算法步骤
  - 2. 获取多个同构图的最小DFS编码

$$a_t = (i_a, j_a, l_{i_a}, l_{(i_a,j_a)}, l_{j_a}) \text{ and } b_t = (i_b, j_b, l_{i_b}, l_{(i_b,j_b)}, l_{j_b})$$

(i) *for some* $t, 0 \leqslant t \leqslant min\{m,n\}$, *we have* $a_k = b_k$ *for* $k < t$, *and**

$$a_t < b_t = \begin{cases} true\ if\ a_t \in E_{\alpha,b}\ and\ b_t \in E_{\beta,f}. \\ true\ if\ a_t \in E_{\alpha,b},\ b_t \in E_{\beta,b},\ and\ j_a < j_b. \\ true\ if\ a_t \in E_{\alpha,b},\ b_t \in E_{\beta,b},\ j_a = j_b,\ and\ l_{(i_a,j_a)} < l_{(i_b,j_b)}. \\ true\ if\ a_t \in E_{\alpha,f},\ b_t \in E_{\beta,f},\ and\ i_b < i_a. \\ true\ if\ a_t \in E_{\alpha,f},\ b_t \in E_{\beta,f},\ i_a = i_b,\ and\ l_{i_a} < l_{i_b}. \\ true\ if\ a_t \in E_{\alpha,f},\ b_t \in E_{\beta,f},\ i_a = i_b,\ l_{i_a} = l_{i_b},\ and\ l_{(i_a,j_a)} < l_{(i_b,j_b)}. \\ true\ if\ a_t \in E_{\alpha,f},\ b_t \in E_{\beta,f},\ i_a = i_b,\ l_{i_a} = l_{i_b},\ l_{(i_a,j_a)} = l_{(i_b,j_b)},\ and\ l_{j_a} < l_{j_b}. \end{cases}$$

(ii) $a_k = b_k$ *for* $0 \leqslant k \leqslant m$, *and* $n \geqslant m$.

Edge Number	Example 1	Example 5	Example 6
0	(0,1,C,c,B)	(0,1,A,a,B)	(0,1,B,c,C)
1	(1,2,B,a,A)	(1,2,B,c,C)	(1,2,C,b,A)
2	(2,0,A,b,C)	(2,3,C,b,A)	(2,0,A,a,B)
3	(1,3,B,a,A)	(3,1,A,a,B)	(2,3,A,c,C)
4	(3,4,A,c,C)	(3,4,A,c,C)	(3,0,C,b,B)
5	(4,1,C,b,B)	(4,0,C,c,A)	(3,4,C,c,A)
6	(4,2,C,c,A)	(4,1,C,b,B)	(4,0,A,a,B)

Example 1 < Example 5 < Example 6

算法原理

- 算法步骤
  - 3. 计算图数据集D中顶点和边出现的频率并排序
  - 4. 去掉出现频率低的顶点和边
  - 5. 按照降序为保留下来的顶点和边重新打标签
  - 6. 将数据集中所有1条边的频繁图放到 $\mathbb{S}^1$ 中
  - 7. 将 $\mathbb{S}^1$ 按照DFS词典顺序排序
  - 8. $\mathbb{S} \leftarrow \mathbb{S}^1$;

第一次减少数据量

算法原理

- 算法步骤
  - 9. for 每一条边 $e \in \mathbb{S}^1$ do
    用e初始化s（一个DFS码）
    如果s不是最小dfs码
    return;
    $\mathbb{S} \leftarrow \mathbb{S} \cup \{s\}$;
    For每一个c，c属于s do
    if $support(c) \geq minSup$
    $s \leftarrow c$;
    重复p过程

遍历边，
从最小的边开始

p过程

算法原理

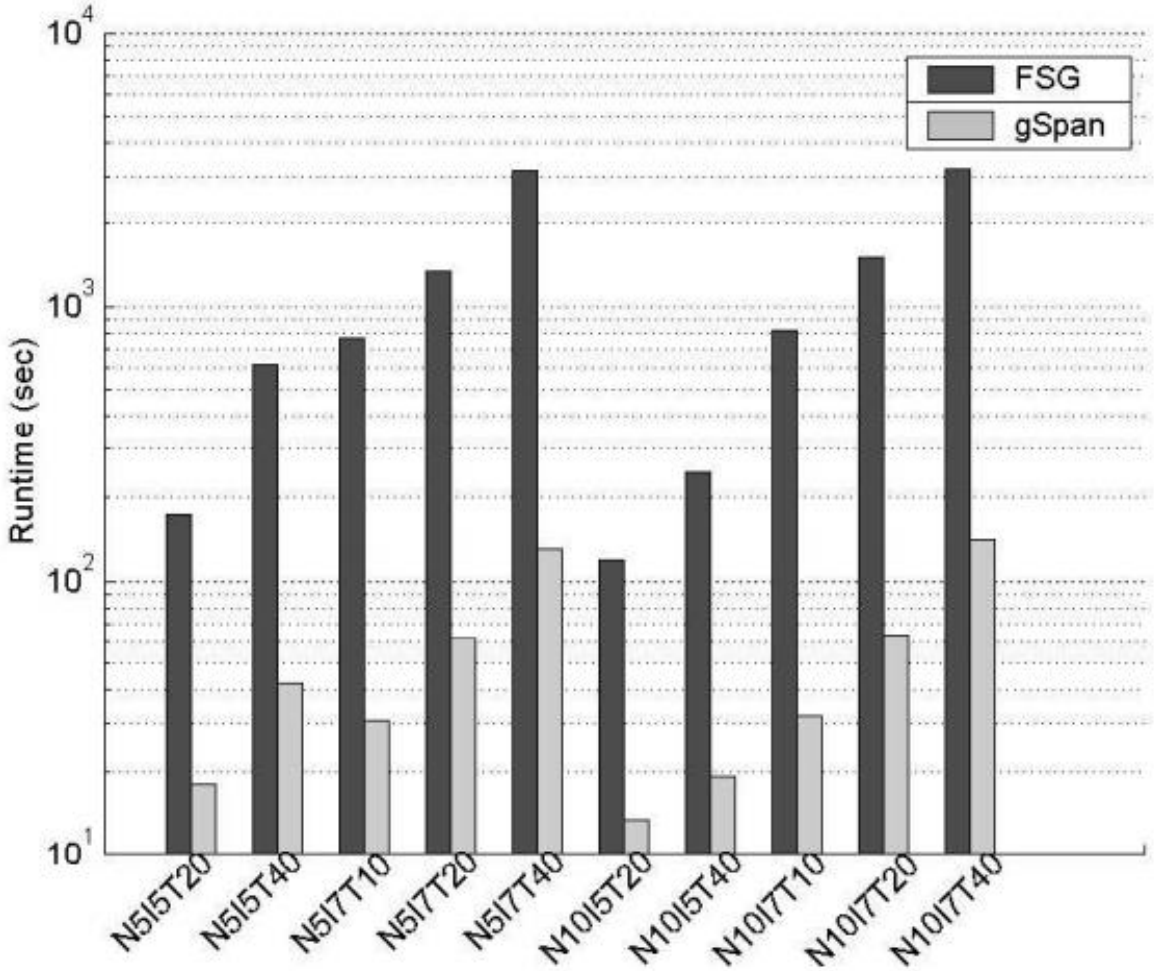
- 算法步骤

10: $\mathbb{D} \leftarrow \mathbb{D} - e$;
11: **if** $|\mathbb{D}| < minSup$;
12: **break**;

- 因为e已经被搜索过了，所以从图数据集中去掉边e的所有实例
- 使后期迭代的挖掘过程花费很少的时间
- 再次减少遍历数据，提高效率

算法原理

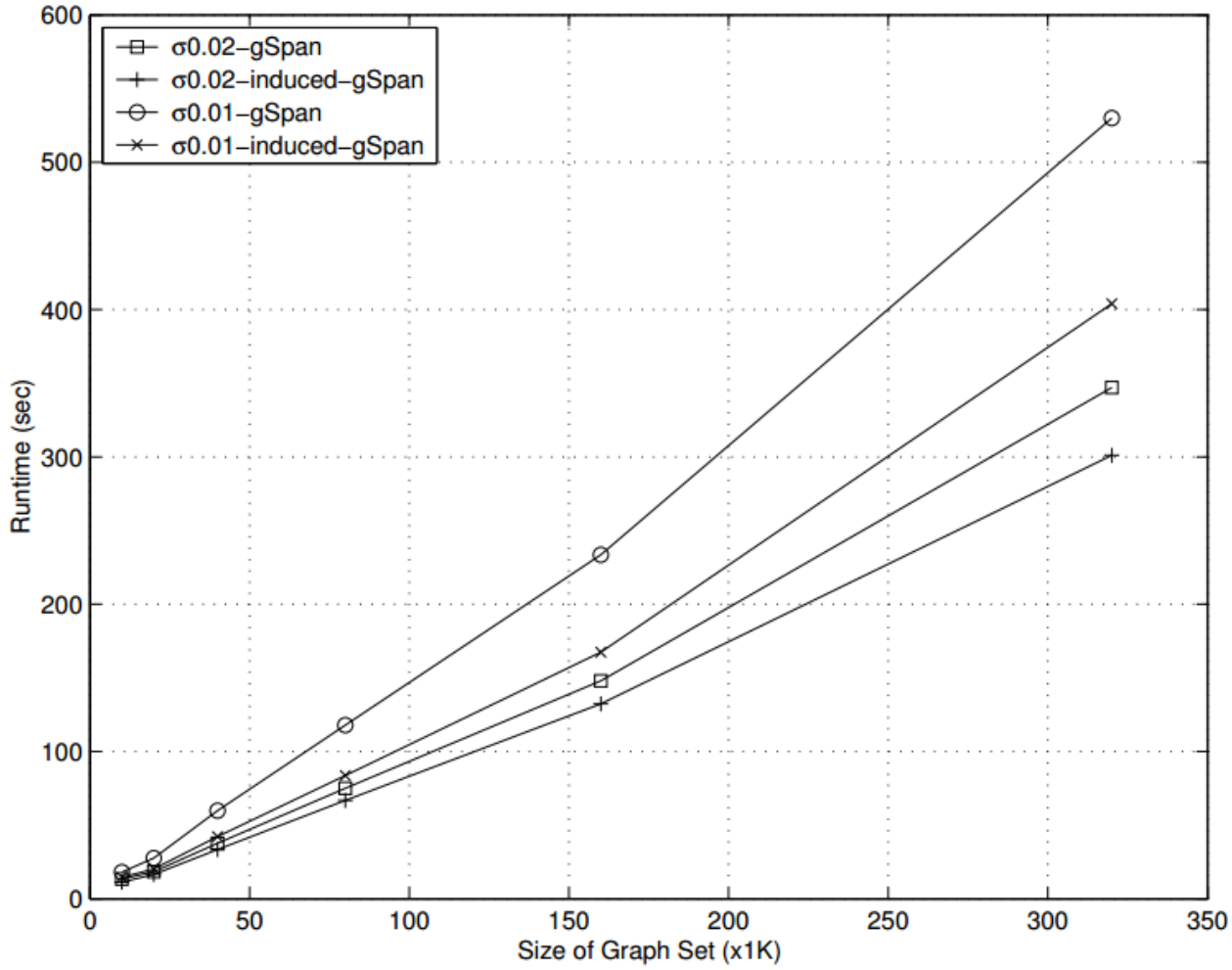
- 算法执行结果



6-15倍

算法原理

- 算法执行结果


σ0.02–gSpan
σ0.02–induced–gSpan
σ0.01–gSpan
σ0.01–induced–gSpan
Runtime (sec)
Size of Graph Set (x1K)
0
100
200
300
400
500
600
0
50
100
150
200
250
300
350

优劣分析

- 横向对比
  - 优势：高效
  - 不足：挖掘连通的频繁子图
- 纵向对比

  1）消除候选生成

  2）用深度优先DFS(depth-first search)不需要多次扫描

  3）引入新的词典规范标签系统，不需要检查子图同构

  4）效率高

应用总结

- 应用领域
  - 社交网络挖掘，计算机视觉，欺诈分析
  - 安卓恶意软件检测

  （关联分析）
- 未来的发展
  - 挖掘闭合频繁子图，极大频繁子图，考虑对挖掘的结果增加一些条件限制。有些条件限制的单调性或者反单调性还可以用于搜索空间的剪枝，加快程序的运行。还可以从挖掘到的结果中，根据条件限制过滤出有用的结果，或者根据统计学的方法过滤出更符合真实情况的结果。

参考文献

gSpan: Graph-Based Substructure Pattern Mining

Android Malware Familial Classification and Representative Sample Seclection via Frequent Subgraph Analysis

道德经

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！