Hans Journal of Data Mining 数据挖掘, 2011, 1, 21-25
http://dx.doi.org/10.12677/hjdm.2011.12006 Published Online October 2011 (http://www.hanspub.org/journal/hjdm/)

Using C++ Object-Oriented Technology to Realize the Restore of Complex Biological Network Set

Xiangzhen Zan, Biyu Xiao, Peng Xu, Wenbin Liu
Department of Physics and Electronic Information Engineering, Wenzhou University, Wenzhou
Email: xiangcheng2436@163.com; wbliu6910@126.com
Received: Sep. 6th, 2011; revised: Sep. 18th, 2011; accepted: Sep. 23rd, 2011.

Abstract: The development of high-throughput biotechnology emerged in a mass of biological network data. How to dig out the conservative frequent pattern effectively from complex biological networks is one of the hot issues in contemporary systems biology. For the reasons that the biology data tends to have the features of large sacle and high demension, to develop a software that can mine the frequent patterns from these biological networks is apt to face a storage prolem. Therefore, we use the method of C++ object-oriented to tackle this problem and discuss its application of frequent pattern mining from biological networks in the end of our paper.

Keywords: Biological Networks; Frequent Patterns; Storage

用 C++面向对象技术实现复杂生物网络集存储

昝乡镇，肖碧玉，许　鹏，刘文斌
温州大学物理与电子信息工程学院，温州
Email: xiangcheng2436@163.com; wbliu6910@126.com
收稿日期：2011 年 9 月 6 日；修回日期：2011 年 9 月 18 日；录用日期：2011 年 9 月 23 日

摘　要：高通量生物技术的发展涌现出了海量的生物网络数据。如何有效的从复杂的生物网络集中挖掘出保守的频繁模式是当代系统生物学研究的热点问题之一。由于生物数据规模大、维数高等的原因，开发出从复杂生物网络集中挖掘出保守频繁模式的应用软件往往面临着一个存储表示的难题。本文用C++面向对象的相关技术，研究了生物网络集的存储表示，并在文中最后讨论了该方法在频繁模式挖掘中的应用。

关键词：生物网络集；频繁模式；存储

1. 引言

近年来，随着 DNA 测序技术和高通量技术的发展，生物学研究的热点已经逐步把分子生物学推入到系统生物学时代。生物网络作为一种描述生物分子间相互作用关系的研究模型，在揭示生物体的生长、发育、衰老和疾病等生命系统的基本分子过程和规律中受到越来越多的重视[1-4]。生物网络主要包括基因调控网络、蛋白质网络、代谢网络及信号传导网络。由于生物系统通常具有保守性，利用网络集中的频繁模式挖掘可以排除生物数据中的各种干扰，挖掘出有生物意义的模式。通常，共表达网络中的频繁稠密子图可能表示一簇紧密的共表达聚类，通过这种方法得到的基因簇可以进行基因功能的注释和疾病基因的发现等。

鉴于这一研究的重要性，很多生物信息学家开发出了一些应用软件，如 CODENSE[5]、NeMo[6]等。传统的邻接矩阵或邻接表方法在处理大规模网络集(网络数在几十到上百，基因个数在 5000~10,000)时在时间和空间方面都存在很大的局限，本文用 C++面向对

象技术实现了一种存储表示生物网络集的方法。

2. 生物网络集的逻辑存储结构

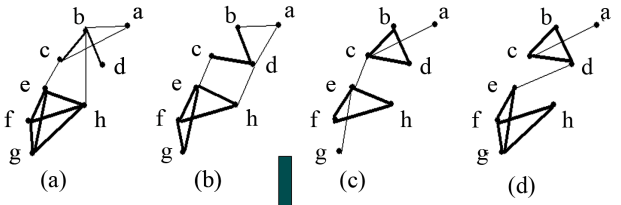
本文对生物网络集的逻辑存储结构采用了表的形式。我们以图 1 为例来具体讲述网络集的逻辑存储结构。图 1 中的(a)、(b)、(c)和(d)分别代表由四个生物芯片数据构建的四个生物网络。

该生物网络集最后以表的形式存储起来，如图 1 中的表格所示。在这个表中，第一列表示边相关联的点，第二列至第四列分别表示图集中的各张图。如果一条边在一个图中存在，则在该边在该表中对应的位置设为 1，否则设为 0。这样我们就得到了一条边在一个图集中的支持向量。

3. 图集中边的支持向量物理存储结构

常见的物理存储方法有两种：1) 用一个连续的二维数组，这种方式可以实现随机访问，因而具有很好的操作性。缺点是浪费了大量的空间，适应不了规模大的生物网络图集。2) 用链表来实现存储，这种方法可以适应大规模图集，但是因为链表具有顺序访问的特点，因此，频繁操作性很差。对于规模很大的生物网络数据，往往因为过度的耗时无法接受而失败。

为了最大限度的节省存储空间而又不失去良好操作性的前提下，我们采用了 C 语言的位段和 C++面向对象的技术，实现了边的支持向量的存储。设计的边支持向量的代码如下：



边	图1	图2	图3	图4
a b	1	1	0	0
b c	1	0	1	1
b d	1	1	1	1
a c	1	0	1	1
a d	0	1	0	0
c d	0	1	1	1
e f	1	1	1	1
e h	1	1	1	0
e g	1	1	1	1
e c	1	1	1	0
f h	1	1	1	1
f g	1	1	0	1
h d	0	1	0	0
h g	1	0	0	1
e d	0	0	0	1

Figure 1. The restore of biological networks
图 1. 生物网络集的存储

```
typedef struct
{
  unsigned one:1;  unsigned two:1;
  unsigned three:1;unsigned four:1;
  unsigned five:1;  unsigned six:1;
  unsigned seven:1;      unsigned eight:1;
  int returnValue(int i)//返回第 i 个元素的值
  {
    switch(i)
    {
        case 1:return one;   case 2:return two;
        case 3:return three;    case 4:return four;
        case 5:return five;   case 6:return six;
            case 7:return seven;case 8:return eight;
            default: return -1;
    }
  }
  void setBitValueOne(int i)//对第 i 个元素赋值为 1
  {
    switch(i)
    {
            case 1:one=1;break;          case
    2:two=1;break;
            case 3:three=1;break;     case
    4:four=1;break;
            case 5:five=1;break;          case
    6:six=1;break;
            case 7:seven=1;break;   case
    8:eight=1;break;
            default: return;
    }
   }
  void setBitValueZero(int i)//对第 i 个元素赋值为 0
  {
     switch(i)
    {
          case 1:one=0;break;          case
2:two=0;break;
          case 3:three=0;break;   case
```



```
4:four=0;break;
            case 5:five=0;break;    case 6:six=0;break;
                case 7:seven=0;break;
            case 8:eight=0;break;
            default: return;
      }
    }
} EdgeSupportVector;
```

采用了这种方式，边支持向量每个元素的值仅占一个 bit。需要说明的是，在 C/C++语言中，这里的 unsigned 类型 默认等于 unsigned int[7]。这里 Edge Support Vector 类型变量在内存中跟 int 类型变量占据的字节数相同，均是四个字节。这里，为了进一步节省空间 Edge Support Vector 位段数可增加至三十二个，本文为了节省篇幅，仅用八个位段表示。方法同上述定义的相同。

4. 生物网络集的物理存储结构

上述方法仅仅实现了边支持向量的存储表示，但是边的其他一些重要相关的信息，如边相关联的点，却并没有包含进去。另外，这种方式还不能适应图集规模扩大的情况。为了解决这个问题，我们用 C++面向对象定义了边的类。定义的代码如下：

```
class Edge
{
    private:
       int headvex,tailvex;
       float weight;//存放边的权值
       int hanming;//存放该边支持向量的汉明值
            EdgeSupportVector * edgeVector;//存放该
      //边对应的支持向量
   public:
       static int edgeVecPacNum;//记录边向量字段个
       //数
       static int graphnum;//记录图集大小
       static int genenum;//记录基因个数
       Edge();//申请空间以及必要的初始化操作
       ~Edge();//释放动态申请的空间
};
int Edge::graphnum=0;
int Edge::genenum=0;
int Edge::edgeVecPacNum=0;
Edge::Edge()
{
    edgeVector=new EdgeSupportVector
    [Edge::graphnum/8+1];
    weight=1;
}
Edge::~Edge()
{
     delete []edgeVector;
}
```

上述类的定义仅列出了最基本的功能，至于其他的功能可以以成员函数的形式加进去即可。这样，我们就实现了边的存储。在此基础上，我们用 C++定义的生物网络图集的存储代码如下：

```
class Graphlet
{
   private:
       Edge * edgearray;//存放该图集所有的边的信
   //息
   public:
       static int size;//记录图集大小
            static int edgenum;//记录该图集边数
       Graphlet();
       ~Graphlet();
};
int Graphlet::size=0;
int Graphlet::edgenum=0;
Graphlet::Graphlet()
{
    edgearray=new Edge[Graphlet::edgenum];
}
Graphlet::~Graphlet()
{
      delete [] edgearray;
}
```

5. 生物网络集物理存储结构的应用

我们以美国斯坦福大学的微阵列数据库(Stanford Microarray Database，http://smd.stanford.edu/)以及美国国立生物技术信息中心的基因表达谱数据(NCBI Ex-

pression Omnibus，http://www.ncbi.nlm.nih.gov/geo/)中的有关酵母(Saccharomyces cerevisiae)的 10 组数据为例。根据实验条件将这些数据划分为 20 个文件，每一个文件包含相似或相关的实验条件；然后分别对每个文件中的数据进行预处理；最后对每个文件，计算所有基因之间的相关性，建立得到 20 个共表达网络。每个网络的大小均为 5672 个基因。经过摘要图处理后(这里频繁度取值为 0.2)[5]，该生物网络集含有的边数为 820,976 条边。

根据上述存储方法，我们构建了一个频繁模式挖掘软件，该软件主要采取迭代循环逐渐去除网络中与稠密频繁模式无关的噪声边的策略。这种用 C++面向对象技术实现的生物网络集的存储结构除了具有面向对象便于程序设计的优点外，还具有如下的一些优点：

首先，从占用的内存空间上来说，这种存储方式节省了大量的存储空间。而这些节省的空间可以用做程序运行中其他数据的存储。该网络集的存储结构采用了 C 语言位段和 C++类相结合的技术，来达到了这一目的。在 Visual Studio C++ 2008 中，一个 int 类型数据变量占据四个字节，我们定义的边支持向量(Edge Support Vecotr)类型数据也是占据四个字节。以 C++中简单数据类型占据空间最小的 bool 变量类型为例，假设我们要存储的图集大小为 32，则用 bool 类型变量数组来存储一条边的支持向量需要占用内存 32 个字节。但是用我们定义的边支持向量类型来存储这条边的支持向量的话，仅需要一个字节就够了。我们以上述提到的酵母菌生物网络集数据为例，来看看这种方式节省的存储空间。用这种方式存储该网络集(需要占用内存约 0.8 M)要比使用 bool 类型数组存储该网络集(需占用内存约 15.6 M)节约大约 95%的空间。对于数据量更大的生物网络数据这种存储方式优势更明显。

其次，从可操作性方面来说，边支持向量可以随机访问，节省了访问的时间。由于处理的边集的信息是一个一维的连续数组，因此可以随机的访问任意一条边的信息。对于一条边的信息，我们可以随机的访问它在图集中第 i 个图的出现情况。例如，假设处理的网络集的大小是 8，假定我们要读一条边在第 5 个图中出现的情况，我们可以调用该边对象的 return Value ()成员函数。调用该函数的实参在本例中就可以设置为(5 mod 8) + 1。在我们基于此存储方式开发的软件运行上述构建好的酵母菌网络数据所花费的时间(需花费一天)来看，消耗的时间是可以接受的。而如果改用链表的存储方式来运行酵母菌网络数据，到两个星期程序还没有结束。(注：运行此软件的计算机配置环境为操作系统 Windows XP，CPU 英特尔酷睿 2 E6550@2.33 GHz，内存 2 G，硬盘 160 G)。

最后，这种存储方式能够根据生物网络图集的大小动态的增删空间。由于该网络图集的存储结构的实现是用 C++语言实现的，又由于 C++语言本身具有动态申请空间和释放空间等特点，因此，这种存储能够根据生物网络集的大小动态的调整空间。此外，这种存储方式具有很高的执行效率。这是因为该存储结构是用 C++语言实现的，而 C++语言本身就具有很高的执行效率。

6. 结论

高通量生物技术的发展涌现出海量的生物网络数据。如何有效的从复杂生物网络集提取出保守的频繁模式在当代系统生物学中具有十分重要的意义。由于生物网络集数据具有维数高、高噪声等特点，如何高效的表示生物网络集从而达到在存储空间和时间上最佳结合，是开发此类生物信息学软件需解决的难题。本文主要探讨了一种用 C++面向对象和 C 语言位段相结合的网络集存储表示方法。该方法在内存占用方面一方面节省内存，同时又能满足算法中对边的频繁操作。通过对酵母表达数据的分析，也验证了该方法的有效性。

7. 致谢

本论文受到了国家自然科学基金(项目号 60970065)、浙江省杰出青年自然科学基金(项目号 R1110261)和温州大学研究生创新基金项目(项目号 3160601010951)的资助，在此一并表示感谢。

参考文献 (References)

[1] L. H. Hartwell, et al. From molecular to modular cell biology. Nature, 1999, 402(2): C47-C52.

[2] L. Hood, et al. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. Science, 2001, 292 (5518): 929-934.

[3] E. Ravasz, et al. Hierarchical organization of modularity in metabolic networks. Science, 2002, 297(5586): 1551-1555.

[4] A. W. Rives, T. Galitski. Modular organization of cellular networks. Proceeding of the National Academy Science, 2003, 100(3): 1128-1133.

[5] H. Hu, X. Yan, et al. Mining coherent dense subgraphs across massive biological networks for functional discovery. BMC Bioinformatics, 2005, 21(1): 213-221.

[6] X. Yan, M. Mehan, Y. Huang, et al. A Graph based approach to systematically reconstruct human transcriptional regulatory modules. Bioinformatics, 2007, 23(13): i577-i586.

[7] 谭浩强. C 程序设计教程[M]. 北京: 清华大学出版社, 2005: 305-308.

