稀疏矩阵

就像在上一节描述的一样，标准的离散化的偏微分方程往往会伴随着一个庞大的且稀疏的矩阵。稀疏矩阵可以被模糊的描述为一个具有非常少的非零元的矩阵。但是，事实上，当特殊的技巧需要利用到大量的非零元以及它们的位置时，一个矩阵是可以被稀疏化的。这些稀疏化矩阵的技巧是从不储存零元的想法开始的。一个关键的问题是制定能够适合于高效地使用不论是直接还是迭代的标准计算方法的存储稀疏矩阵的数据结构。这一章节将简介稀疏矩阵，它们的属性、呈现，以及用以存储它们的数据结构。

3.1 介绍

利用一个矩阵中的零元以及它们的位置的自然的想法最初是由在不同学科的工程师们提出的。在涉及带状矩阵的简单地例子中，特殊的技巧直接的被发明了。在 20 世纪 60 年代研发电子网络的电子工程师们是最早的去利用稀疏性来对于具有特殊结构的矩阵解决一般稀疏线性系统。对于稀疏矩阵技巧而言，最主要也是最早需要解决的问题是去设计一个在线性系统中得直接求解算法。这些算法需要是可以接受的，在存储和计算效率上。直接的稀疏算法可以被用于计算那些庞大的难以被稠密算法来实现的问题。

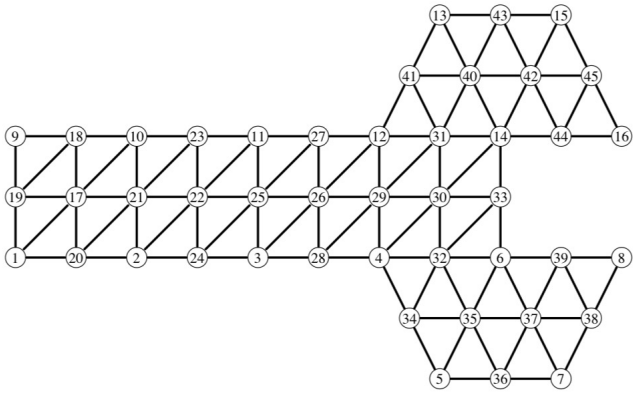

Figure 3.1 *A finite element grid model.*

基本上，有两个明显的类别的稀疏矩阵，结构化的和非结构化的。一个结构

化的矩阵是指一个非零元的位置形成某个规律的矩阵，通常这些非零元在对角线附近。要不然，这些非零元会在相同大小的块内（稠密子矩阵），而这也会形成一个规律，通常这些非零元在对角线（块）附近。一个具有着不规则位置的非零元的矩阵会被称作是非结构化的。最好的一个结构化的矩阵的例子是一个只有着少量对角元的矩阵。网格上的有限差分矩阵，就像上一节中提到的，是典型的具有着规律结构的例子。大部分的对于复杂几何的有限元和有限体积技巧会导致非结构化的矩阵。图 3.2 展示了一个与图 3.1 所呈现的有限元网格问题的一个小规模的非结构化的矩阵。

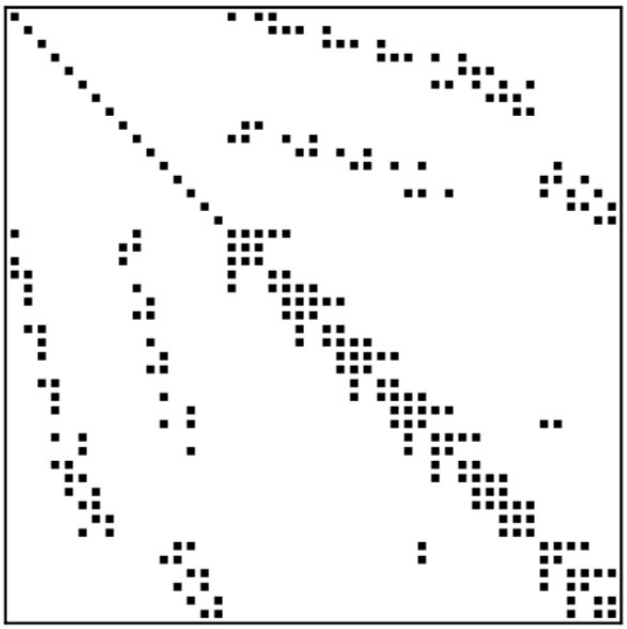

Figure 3.2 *Sparse matrix associated with the finite element grid of Figure 3.1.*

3.2 图论

图论是用来表示稀疏矩阵结构的一个理想的工具，因此，在稀疏矩阵技巧中，它扮演着一个主要的角色。例如，图论是用于解决并行稀疏高斯消除和预处理技术的关键。在下一节中，将讨论图的一般特性，以及它们在有限元和有限差分矩阵中得应用。

3.2.1 图与邻接图

记住一个图由两个集合定义，一个顶点集合 $V = \{v_1, v_2, \cdots, v_n\}$ 和一个边的集合 E, E 是由点对 (v_i, v_j) 组成的，v_i, v_j 都是 V 中的元素，换而言之，$E \subseteq V \times V$。这个图 $G = (V, E)$ 通常被平面内的一系列的被边联系的点的向量来表示。这个图被用来描述集合 V 中元素间的关系。例如，V 可以被用来描述世界上的主要城市。线就是两个城市间的直达航线。那么这个图就会描述这样一个关系“在城[illegible]元关系很可能有一条 B 到 A 的有向图相对。

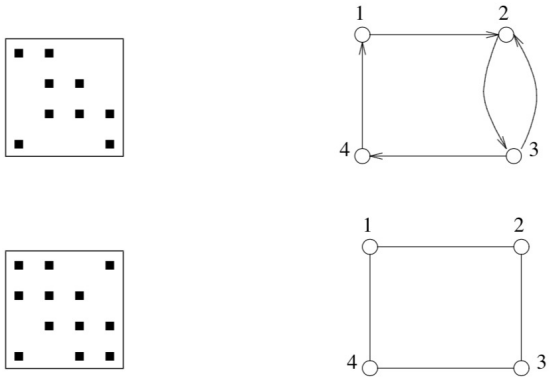

Figure 3.3 *Graphs of two* 4×4 *sparse matrices.*

回到稀疏矩阵，稀疏矩阵的邻接图是一个图 $G = (V, E)$，V 中得 n 个顶点代表 n 个未知数。它的边是按照以下规则建立的方程式建立的二元关系：当 $a_{ji} \neq 0$ 时，有一条从节点 i 指向节点 j 的边。而这条边将因此描述包含未知量 j 的二元关系方程式 i。注意，这个图是有向的，除非这个矩阵是有对称结构的（对任意的 $1 \leq i, j \leq n$，若 $a_{ji} \neq 0$，那么 $a_{ij} \neq 0$ ）。

当一个矩阵的非零元总有一个对称非零元，换而言之 a_{ij} 和 a_{ji} 总是同时为非零元，那么这图就是无向的。因此，对于无向图，每条边都有两个方向。因此，无向图可以用无向边来表现。

作为利用图模型的例子，并行高斯消去法可以通过寻找在指定消去阶段的未知数来获得。根据以上的二元关系，这些未知数两两独立。这些与未知数一致的行可以被用作基。因此，在一个极端情况下，当一个矩阵是对角阵，那么所有的未知数是独立的。与之相反的是，当一个矩阵是稠密的，那么每一个未知量都与其他未知量相关。稀疏矩阵则介于这两种极端情况之间。

邻接图有一些有趣的简单性质。A^2 的图可以被解释成一个 n 顶点图，对每条边的点对 (i,j)，表示在原图 A 中至少存在一条长度确切的说是 2 的从节点 i 到节点 j 的路径。与之相似的时，A^k 的图包含的时用以描述从节点 i 到节点 j 的至少存在一条长度为 k 的路径的二元关系的边。欲知详情，请看练习 4.

3.2.2PDE 矩阵的图

对于在每个网格点只涉及一个屋里未知量的偏微分方程，离散矩阵的邻接图通常就是用来描述网格的图。但是，在每个网格点上有着多个未知量是很常见的。例如，模拟流体流动的方程可能涉及流体的两个速度分量（二维）以及在每个网格点的能量和动量。在这样的情况下，有两种用来标记未知量的选择。在每个网格点，它们可以被连续的标记。因此，在刚才的例子中，我们可以在一个指定的网格点例如 u(k),$\cdots$,u(k+3) 上标记所有的四个未知量（两个速度的分量，动量以及压力）。另外，所有的与一类变量相关的未知量可以最先被标记（比如，第一个速度分量），接下来是第二类的变量（比如，第二个速度分量）等等。在任意情况下，很明显邻接矩阵是有冗余信息的。物理网格的商图可以被用来替代使用。这将节约大量的存储量和计算量。在上述的流体流动的例子中，用以描述图的整数数组的存储可以被缩小到接近 1/16。这是因为边的数量被所见到了大约这么多，但是通常很小的顶点数却保持着不变。

3.3 置换和重新排序

对于稀疏矩阵而言，重排序行或列，或者行和列是一个常见操作。事实上，重排序行和列是一个用于直接求解法和迭代法并行实现的一个最重要的部分。本节介绍这些重排技术和矩阵的邻接图的相关思想的关系。记得在第一章中，矩阵的第 j 列记作 a_{*j}，第 i 行则记作 a_{i*}。

3.3.1 基础概念

我们先开始一个定义与符号。

定义 3.1 有一个矩阵 A，以及 $\pi = \{i_1, i_2, \cdots, i_n\}$ 的一个交换集合 $\pi = \{i_1, i_2, \cdots, i_n\}$。那么矩阵 $A_{\pi,*} = \{a_{\pi(i),j}\}_{i=1,\cdots,n;j=1,\cdots,m}, A_{*,\pi} = \{i, a_{\pi(j)}\}_{i=1,\cdots,n;j=1,\cdots,m}$ 就分别被称作 A 的行 π-交换和列 π-交换。

广为周知的时，最多 n 个交换（换而言之，就是只互换两项的基本排列）可以产生集合 $\{1, 2, \cdots, n\}$ 的任意置换。一个交换矩阵就是一个两行互换了得单位矩阵。用 X_{ij} 来表示第 i 和 j 行交换了的单位矩阵。注意到，为了交换矩阵 A 的第 i 和 j 行，我们可以用矩阵 X_{ij} 来左乘矩阵 A。让 $\pi = \{i_1, i_2, \cdots, i_n\}$ 为一个任意序列。这个置换就是一系列连续的交换矩阵 $\sigma(i_k, j_k), k = 1, \cdots, n$ 的乘积。那么，我们就可以通过交换矩阵的 i_1 和 j_1 行，然后再结果矩阵的基础上交换 i_2 和 j_2 行，以此类推，最后交换 i_n 和 j_n 行。每一步我们都可以通过左乘矩阵 X_{i_k,j_k} 来实现。同样的，对于矩阵的列也是一样的：为了交换矩阵的第 i 和 k 列，通过右乘矩阵 X_{i_k,j_k} 可以实现。从上我们可以得到下述命题。

命题 3.1 让 π 是交换 $\sigma(i_k, j_k), k = 1, \cdots, n$ 的乘积得到的置换。那么，$A_{\pi,*} = P_\pi A$，$A_{*,\pi} = AQ_\pi$，当

$$P_\pi = X_{i_n,j_n} X_{i_{n-1},j_{n-1}} \cdots X_{i_1,j_1} \tag{3.1}$$

$$Q_\pi = X_{i_1,j_1} X_{i_2,j_2} \cdots X_{i_n,j_n} \tag{3.2}$$

这些交换矩阵的乘积被称作置换矩阵。显然，一个置换矩阵只不过是进行了行列交换的单位矩阵。
注意到 $X_{i,j}^2 = I$，换而言之，置换矩阵的平方是一个单位矩阵，或者等价地，置换矩阵的逆等于它本身，这是很显然的一个属性。易见，矩阵（3.1）和（3.2）满足

$$P_\pi Q_\pi = X_{i_n,j_n} X_{i_{n-1},j_{n-1}} \cdots X_{i_1,j_1} \times X_{i_1,j_1} X_{i_2,j_2} \cdots X_{i_n,j_n}$$

表示了矩阵 Q_π 和 P_π 都是非退化的，且互为另一个的逆。换而言之，用同一个置换矩阵来交换一个矩阵的行和列事实上做了类似的变换。因为定义 (3.1) 和（3.2）中得 P_π 和 Q_π 的乘积是相反的顺序，另一个推论就显而易见了。由于每一个基矩阵 XX_{i_k,j_k} 是对称的，那么 Q_π 是 P_π 的转置。因此

$$Q_\pi = P_\pi^T = P_\pi^{-1}$$

因为矩阵 P_π 的逆矩阵是它的转置，置换矩阵就是唯一的。
另一个用来推出上述关系的方法是用置换矩阵 P_π 和 P_π^T 来代表行列交换了的单位矩阵。（在练习 3 中）显而易见

$$P_\pi = I_{\pi,*}\ P_\pi^T = I_{*,\pi}$$

那么，接下来就可以直接验证

$$A_{\pi,*} = I_{\pi,*}A = P_\pi A\ A_{*,\pi} = AI_{*,\pi} = AP_\pi^T$$

这对于在线性系统中解释置换操作很重要。当矩阵的行交换了，方程的顺序就改变了。换而言之，当列交换了，那么未知量就会对应的改变标记或者改变顺序。
例子 3.1 思考，比如，线性系统 $Ax = b$，当

$$A = \begin{pmatrix} a_{11} & 0 & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & a_{42} & 0 & a_{44} \end{pmatrix}$$

以及 $\pi = \{1, 3, 2, 4\}$，那么（列）交换线性系统是

$$\begin{pmatrix} a_{11} & a_{13} & 0 & 0 \\ 0 & a_{23} & a_{22} & a_{24} \\ a_{31} & a_{33} & a_{32} & 0 \\ 0 & 0 & a_{42} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

注意到，不只是未知量交换了，方程也是，特别的，右边没有变。
在上述例子中，只有 A 的列交换了。在稀疏矩阵技术中，这样的单侧变换不如两侧变换寻常。事实上，这通常与线性系统中得对角元起着一个明显且重要的角色的事实有关。比如，在偏微分应用中，对角元通常很大，而且，在交换矩阵中可能很需要去保留这一性质。为了达到这一目的，很典型的是去同时对 A 的行和列进行相同的交换。这样的操作被叫做对称置换，若果用 $A_{\pi,\pi}$ 来表示，那么，这样的对称置换的结果就满足这一关系

$$A_{\pi,\pi} = P_\pi^T A P_\pi$$

对称置换的解释很简单。由此产生的矩阵用相同的方式重命名，或重标记，或重排序未知量和重排序等式。

例子 3.2 对于前面的例子，如果行和列用相同的置换矩阵来置换，那么线性系统可以这样来获得

$$\begin{pmatrix} a_{11} & a_{13} & 0 & 0 \\ a_{31} & a_{33} & a_{32} & 0 \\ 0 & a_{23} & a_{22} & a_{24} \\ 0 & 0 & a_{42} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

注意到，对角元是原来的矩阵的对角元在主对角线上德一个不一样的顺序。

3.3.2 与邻接图的关系

从图论的角度，另一个对于对称置换的重要解释是这相当于不不改变边来重新标记顶点。事实上，(i,j) 是原矩阵 A 的邻接图的边，A' 是交换了得矩阵。当且仅当 $(\pi(i), \pi(j))$ 是原始矩阵 A 的图中的一条边时，那么 $a'_{ij} = a_{\pi(i),\pi(j)}$，结果 (i,j) 是交换了的矩阵 A' 的邻接图中一条边。因此，交换了的矩阵的图没有改变；甚至，顶点的标记也是。与之相对的是，不对称置换就不会保存好图。事实上，它们可以将一个无向图转换为一个有向图。尽管邻接矩阵的一般图是相同的，对称置换可能会对矩阵的结构造成一些重大的影响。

例子 3.3 考虑图 3.4 描述的矩阵和它的邻接图。因为它们的形状，这样的矩阵又是被叫做“箭头”矩阵，但是，因为它们图的结构可能更适合把它们叫做“星”矩阵。

如果用置换 $9, 8, \cdots, 1$ 来重排序等式，图 3.5 所描述的矩阵和图就得到了。尽管两张土建的区别看起来很小，但是矩阵可能会有一个对于算法有着重要影响的完全不同的结构。以此为例，如果用高斯消去法来重排序矩阵，那么填充就不会发生。换而言之，LU 分解的 L 和 U 部分会与 A 的下和上两部分有着一样的结构。在另一方面，在原始矩阵上作高斯消去法会导致灾难性的填充。特别的，在高斯消去法第一部以后，LU 分解的 L 和 U 部分是稠密矩阵。用直接稀疏矩阵技术，找到在高斯消去过程中对于较少填充有作用的矩阵的置换很重要。最后这一段，总的来说，两侧非对称置换也可能在实践中出现。然而，在直接法中它们很常见。

3.3.3 常用重排序

在实践中，重排序和置换的种类七绝与直接或者迭代法是否被考虑。下面是一个这样的对赌迭代法更为有用的一个重排序的例子。

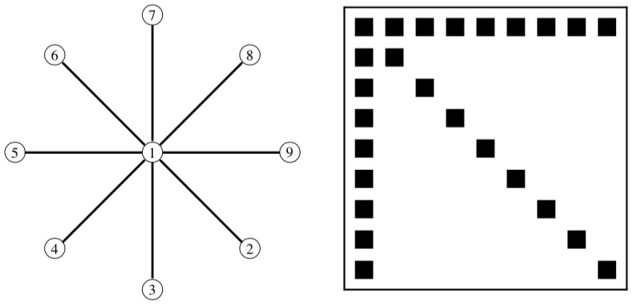

Figure 3.4 *Pattern of a 9 × 9 arrow matrix and its adjacency graph.*

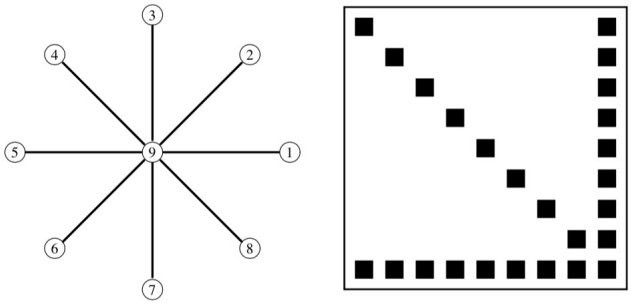

Figure 3.5 *Adjacency graph and matrix obtained from above figure after permuting the nodes in reverse order.*

水平集序这种顺序类型包含了许多基于水平集的图的便利的技巧。水平集是递归定义的上一级的所有节点的所有未标记的邻集。最初，一个水平集有一个节点，虽然有几个将来会被讨论的也重要的起始点。当一个水平集被遍历完成，它的节点就被标记了且排了编号。比如，它们可以按照遍历的顺序来编号。另外，遍历顺序的不同会产生不同的顺序。例如，某一个水平集中的节点可以按照它们列出的自然序访问。然后，可以检查它们每一个的邻节点。每一次，当遇到一个访问过的顶点有一个没有编号的邻节点，那么它就被添加到列表中并标记为下一水平集的下一元素。在图论中，这个简单地策略被称为广度优先搜索。在每个水平集中，顺序取决于节点遍历的方式。在广度优先搜索中，水平集中元素总是以它们列出的自然序来遍历。在 Cuthill-McKee 排序中，水平集的元素被以从最低到最高的顺序来遍历。

算法 3.1：Cuthill-McKee 排序

```
输入：初始节点 i1；输出：排序数组 iperm
开始：levset:=i1；next=2；
marker(i1)=1;operm(1)=i1
while (next < n)
      do Next_levset=φ
          遍历 levset 来提高次数，同时：
          for 每一个访问到的节点
               do for 每一个 j 的邻节点 i, 有 marker(i)=0
                        do 向集合 Next_levset 加入 i
                           marker(i):=1； operm(next)=i
                           next=next+1
                   EndDo
          EndDo
          levset := Next_levset
EndWhile
```

从程序列出的数组 iperm 列出了它们被访问的顺序。在实际应用中，它们可以被存储在一个连续水平集中。在每一个集合开始的时候，指针需要被声明。因此，数组 iperm 就代表了之前定义的置换数组 π。

在 1971 年，乔治 [103] 注意到，对于稀疏高斯消去法，反 Cuthill-McKee 排序是一个更好的方案。去理解这一事实的最简单的方法是去看这两个排序方法产生的两张图。在示例有限元问题中，标准和反向的 Cuthill-McKee 排序的结果就像之前图 3.6 和 3.7 所呈现的，当厨师节点是 $i_1 = 3$（与图 2.10 中的原始排序标记有关）。图中的情况下，对应于一个在行 6 的遍历的 CMK 算法的变量，是做一个随机排序而不是根据次数排序。与例子 3.3 相似的，两个矩阵的结构的大部分包含着“箭头”矩阵。在常规 CMK 排序中，这些箭头是向上的，就像在图 3.4 中标记了的水平集这样。这些块与图 3.4 中的星矩阵类似。因此，高斯消去法会在本质上填充跨越了的块。就像在例子 3.3 中显示的这样，一个补救方法是重新排列节点后，在全局使用反 CMK 策略。对于反 CMK 排序，箭头是向下的，就像在图 3.5 中这样，而且高斯消去会大大地提高稀疏度。

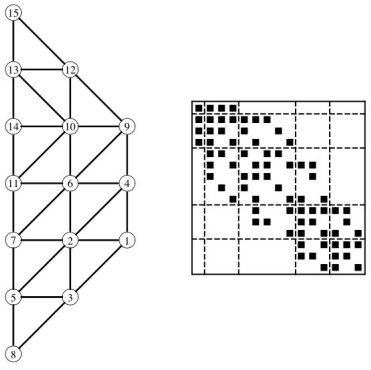

Figure 3.6 *Cuthill-McKee ordering.*

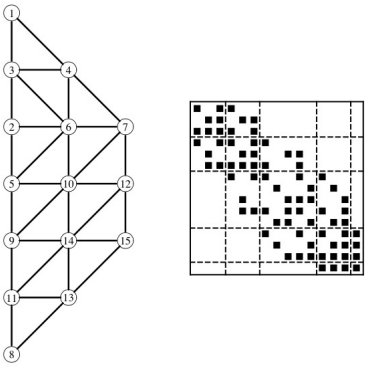

Figure 3.7 *Reverse Cuthill-McKee ordering.*

例子 3.4 CMK 和 RCMK 排序的初始节点的选择是重要的。从图 2.10 额原来的排序可以看到，原来使用了 $i_1 = 3$。但是，很明显，对于有着小块或者轮廓的矩阵而言，这是一个糟糕的选择。如果取而代之的用 $i_1 = 1$，那么，反 CMK 算法的乘积就是图 3.8 中的矩阵，更适合于块状或者有着轮廓线的求解。

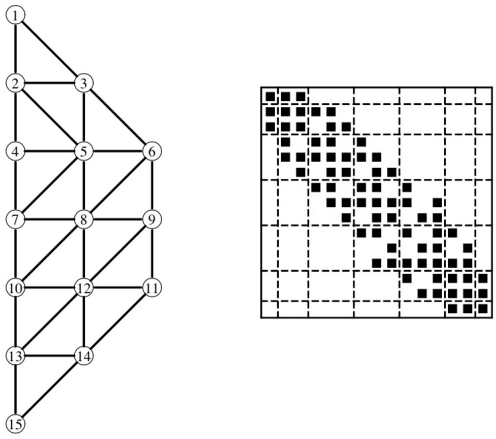

Figure 3.8 *Reverse Cuthill-McKee ordering starting with* $i_1 = 1$.

独立集排序在图 2.7，2.10 和 3.2 中的有限元问题的模型产生的矩阵以左上角块是对角的为特征，它们有结构

$$A = \begin{pmatrix} D & E \\ F & C \end{pmatrix} \tag{3.3}$$

其中 D 是对焦的，C、E 和 F 是稀疏矩阵。由于方程使用顺序，上对角块对应于未知量先前的消去以及它的出现。作为细化网格新创建的顶点，它们被赋予了新的编号，而初始顶点的编号是不变的。既然旧的连接节点是用新的“切”的，那么它们就不再是相关方程。这样的集合被称为独立集。在并行计算中，不论是直接法还是迭代法，独立集尤其有用。
参考矩阵的邻接图 G=(V,E)，用 (x,y) 来表示从点 x 到点 y 得边，独立集 S 是

顶点集 V 的子集，满足，

$$\text{如果} x \in S\text{，那么} \{(x,y) \in E \text{或者} (y,x) \in E\} \rightarrow y \notin S$$

为了解释它：S 中的元素不被允许去与 S 中其他的不论是传入还是传出边连接。如果一个独立集不能增加补充元素来形成一个更大的独立集，那么称它是最大的。注意到，最大独立集不一定是一个可以找到的最大的最大独立集。事实上，寻找最大独立集的基是一个非确定性多项式问题 [132]。下面，一直用独立集来代指最大独立集。

有许多简单且廉价的用于寻找大规模最大独立集的启发式算法。一个贪婪的启发式算法按给定的顺序来遍历，如果一个节点没有标记，那么就选择 S 的一个新的元素。那么一个节点就沿它最近的邻节点标记。现在，节点 x 的最近的节点就意味着与 x 用传入或者传出边来的任意节点。

算法 3.2：ISO 贪心算法

1 S=ϕ
2 **for** j=1,2,$\cdots$,**Do**:
3 如果节点 j 没被标记，那么
4 $S = S \cup \{j\}$
5 标记 j 以及它的所有邻近节点
6
7 **EndDo**

在上述算法中，节点按照自然序 $1, 2, \cdots, n$ 的顺序遍历，但是，它们也可按照 $\{1, 2, \cdots, n\}$ 的置换 $\{i_1, \cdots, i_n\}$ 来遍历。因为减小了的系统的大小是 n-|S|，为了取得一个小的减小了的系统，有理由去尝试增加 S 的大小。可以给出一个大致的 S 的大小。假设每个节点的最大次数不超过 v。当上述算法接受一个新节点作为 S 的一个新元素时，它潜在的输出了它所有的邻节点，换而言之，在 S 的补集中，最多 v 个节点。因此，如果 S 的大小是 s，那么它的补集的大小事 n-s，有 $n - s \leq vs$，因此

$$s \geq \frac{n}{1+v}$$

可以通过用构成 S 的所有顶点的最大次数 v_s 来替换 v 来稍微提高下界。这得到了下面的不等式

$$s \geq \frac{n}{1+v_s}$$

这暗示了先访问小的节点是一个很好的方案。事实上，这个观察导致了一个很好的遍历顺序的通用启发算法。该算法可以被视作如下：每当访问一个节点，从图中移除它以及它的邻节点，然后访问图剩下部分的一个节点。直到所有节点被用完，重复相同的方法。每个访问了的节点是 S 的元素，它的最近邻节点是 $\bar{S}$ 的元素。因此，在第 i 步，如果 v_i 是被访问了的节点的次数，那么调整所有的之前的访问步骤中删除了的边，然后，在第 i 步剩下的节点的数量 n_i 就满足关系

$$n_i = n_{i-1} - v_i - 1$$

这个程序在每一步中向集合 S 添加一个新的元素，并当 $n_i = 0$ 时停止。为了最大化 |S|，程序必须最大化步骤的数量。因为与被移除的节点相关的边的移除，

在每一步 i，次数更新的复杂度会提高。如果这个过程被延长了，一个访问节点的经验法则是先访问最小次数的节点。

算法 3.3：ISO 提高次数遍历法

1 S=ϕ，按照节点次数升序得到一个顺序 $i_1, \cdots, i_n$
2 **for** j=1,2,$\cdots$,**Do**:
3 如果节点 i_j 没有被标记，那么
4 $S = S \cup \{i_j\}$
5 标记 i_j 以及它的所有邻近节点
6 **EndDo**

一个对于上述算法的改进是在移除中更新所有相关节点的次数，并动态选择最小次数的作为下一个访问的节点。这可以用最小堆数据结构来高效地实现。一个不同的启发式算法是通过决定动态遍历排序的局部优化来最大化元素的数量。下面，从图中移除一个顶点就意味着移除从这个顶点传入或者传出的所有边和这个顶点。

例子 3.5 本节中描述的算法在之前相同的例子中进行了测试，即图 2.10 的有限元网格问题。