第 7 章 多模块划分方法

絜矩之道 《礼记·大学》

智能嵌入式系统通常会含有众多任务，这些任务之间通过通信进行消息/数据/事件传输,协同完成整个系统需要完成的任务。为了满足时间性能的要求，人们会把这些任务划分成多个模块，每个模块由一个或多个处理器或软件与硬件融合的异构平台进行处理。模块内依据任务的性能指标（如时间、功耗、成本、硬件面积等）进行软硬件优化划分。这样处理，一方面可以缩短设计周期，极大地提高设计效率，另一方面可以根据系统各个部分的特点和设计约束选择软件或硬件实现方式,从而得到高性能、低成本的优化设计方案。

本章将以任务间通信代价为基础进行智能嵌入式系统模块划分，通信代价大的两个任务将划分在一个模块，实现整个系统的通信代价极小。智能嵌入式系统模块划分是智能嵌入式系统软硬件优化配置的关键技术之一，它对整个系统的设计结果有着重要的影响。

第 7.1 节 多模块划分方法

多模块是当前智能嵌入式系统实现复杂系统优化设计的一个重要手段，其目的是将一个复杂系统中任务划分成多个模块，每个模块相对独立，模块间通过通信实现数据传输，完成整个系统的并行同步与分布处理。多模块划分的基本算法是依据任务间的通信代价通过贪心算法进行聚类，两个任务通信代价越高或越频繁，越应该把这两个任务划分到一个模块（簇，类），依据划分结果，计算模块间的通信代价，而模块内的通信代价量规定为零，划分目的是使模块间通信代价之和极小。

7.1.1 模块化划分问题

模块化划分试图解决组合问题的一个基本问题：给定一个边上有权的图 G，其权值是通信代价，把图 G 节点集划分成若干个元素个数不超过最大规模的子集块，并使得块间通信代价最小化。

为了数学上建立多模块划分问题， 我们需要下面的一些定义。

7.1.2 可许划分

设 G 是一个含有 n 个节点的图，其中每个节点都有尺寸（权重） $w_i > 0$（ $i = 1, ...,n$）。设 p 是一个正数并且对于所有的 i 都有 $0 < w_i \leqq p$. 设 $C = (c_{ij})(i, j = 1, ...,n)$ 是图 G 边的权值邻接矩阵（Adjacency Matrix）。

注：图 G 中节点尺寸（权重）可以是指节点（构件）开发的成本或功耗，图 G 中边的权值可以是节点（构件）间的通信代价。

定义 7.1（划分）设 k 是一个正整数，图 G 的一个 k-式划分是将图 G 的节点集划分成 k 个互不相交的模块 $V_1, V_2, ..., V_k$ 使得 $\cup^k_{i=1} V_i = G$ 节点集合，即其集合并等于图 G 的节点集合。

定义 7.2（划分成本）划分中两个模块 Y_1，Y_2 之间成本 $C(Y_1, Y_2)$ 是这两个模块的所有元素 i, j 成本 C_{ij} 之和，即

$$C(Y_1, Y_2) = \sum_{i \in Y1, j \in Y2} C_{ij} .$$

图 G 的一个划分 $\{V_1, V_2, ..., V_k\}$ 成本 $C(V_1, V_2, ..., V_k)$ 是该划分中两两不同模块之间成本之和，即，$C(V_1, V_2, ..., V_k) = \sum_{i=1..k, j=1..k(i \neq j)} C(V_i, V_j)$。

定义 7.3（可许划分）一个划分 $\{V_1, V_2, ..., V_k\}$ 是可许的（admissible），若对于所有的 i 都有 $|V_i| \leqq p$，其中符号 | X |表示集合 X 的尺寸，等于 X 元素尺寸之和。

所考虑的划分问题实际上是寻找图 G 的一个极小成本的可许划分。

本章介绍两种多模块划分算法：基于聚类的多模块划分方法和基于 KL 算法的多模块划分方法。

第 7.2 节 基于通信代价的聚类算法

聚类是数据处理一个最基本方法，其目的是将数据划分成若干组（称为块/类/簇）。基于通信代价的聚类算法是将智能嵌入式系统的任务按照通信代价进行聚类，把通信代价高的两个元素聚集到一个类中，而把通信代价低的两个元素划分到两个不同类中。这样聚类的结果是类内元素通信代价高，而类间元素通信代价低。

聚类算法考虑的问题是 7.1 节划分问题中图 G 的节点权重都等于 1 的简单情形。本节主要参考文献是[36]。

7.2.1 层次聚类算法

经典层次聚类算法（Hierarchical Clustering）实际上是通过迭代产生嵌套的类集，算法的整体流程如下：

1）算法开始时，每个成员都组成一个单独类；

2）在以后迭代过程中，把相邻的类合并成一个类；

3）直到所有的成员组成一个类为止。

层次聚类算法最核心的步骤是 2）：把相邻的类合并，相邻类是其关键。相邻类的定义不同，产生了不同的层次聚类算法：

1）单链接算法（Single Link）：类间元素中的最大通信代价大于等于通信代价阈值，即有一对元素的通信代价大于等于通信代价阈值；

2）全链接算法（Complete Link）：类间元素中的最小通信代价大于等于通信代价阈值，即类间所有元素对的通信代价都大于等于通信代价阈值；

3）均链接算法（Average Link）：类间元素中的平均通信代价大于等于通信代价阈值，此时有一部分元素对的通信代价大于等于阈值，而另外一部分元素对的通信代价小于阈值。

设类 $A=\{a_i|1\le i\le n\}$，类 $B=\{b_j|1\le j\le m\}$，则类 A 与类 B 间的平均通信代价:

$$\text{avec}(A,B)=\frac{\sum_{1\le i\le n,1\le j\le m}C(a_i,b_j)}{n*m}$$

其中 $C(a_i，b_j)$是元素 a_i 与 b_j 的通信代价。

7.2.2 谱系图（Dendrogram）

层次聚类算法实际上是在谱系图（树）上产生的。

（1）谱系图定义

谱系图是一个树型数据结构，演示了层次聚类技术，见图 7-1。每一层表示该层的类集，其中：

1）叶结点：每个叶子都是一个类

2）中间结点：表示由其子结点合并而成的新类

3）根结点：聚类成一个类

假设有如下例子：

五层：有 1 个类－{A,B,C,D,E,F}

四层：有 2 个类－{A,B,C,D,E}，{F}

三层：有 3 个类－{A,B},{C,D,E}，{F}

二层：有 4 个类－{A,B},{C,D},{E},{F}

一层：有 6 个类－{A},{B},{C},{D},{E},{F}

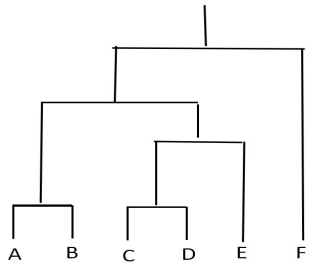


图 7-1 谱系图

（2）谱系图表示

谱系图用有序三元组<c,k,K>表示：其中 c 是通信代价阈值，k 是类的数目，K 是类的集合，如：

{

<7, 6, {{A},{B},{C},{D},{E},{F}}> 通信代价阈值=7，有 6 个类-{A}, {B},{C},{D},{E},{F}

<6, 4, {{A,B},{C,D},{E},{F}}> 通信代价阈值=6，有 4 个类-{A,B},{C,D},{E},{F}

<5,3, {{A,B},{C,D,E},{F}}> 通信代价阈值=5，有 3 个类-{A,B},{C,D,E},{F}

<4,2, {{A,B,C,D,E},{F}}> 通信代价阈值=4，有 2 个类-{A,B,C,D,E},{F}

<3,2, {{A,B,C,D,E},{F}}> 通信代价阈值=3，有 2 个类-{A,B,C,D,E},{F}

<1,1,{{A,B,C,D,E,F}> 通信代价阈值=1，有 1 个类-{A,B,C,D,E,F}

}

（3）谱系图生成

给定一组节点，根据节点之间的通信代价形成邻接矩阵 A，矩阵元素 C_{ij} 的值是第 i 个节点和第 j 个节点之间的通信代价，其中 C_{ii} 表示节点 i 自身的通信代价，实际情况应为 0。邻接矩阵 A 是对称阵，即 $C_{ij}=C_{ji}$。 符号 Max(A)表示邻接矩阵 A 中的最大元素， 即 $Max(A)=Max\{C_{ij}| 1<=i，j<=n\}$。 通过一定的策略逐步将样本合并为同一类，建立起对应的谱系图。

7.2.3 层次聚类算法

设 D 表示含有 n 个元素集合，$A=(C_{ij})_{n\times n}$ 是元素间邻接矩阵，c 表示类间通信代价阈值，每次聚类通信代价减少 1，k 表示类的数目，K_c 表示通信代价阈值 c 对应的类集合，即 $k=|K_c|$，算法从 k=n（n 个类）开始直到 k=1 结束，即聚成一个类。

算法 7.1 聚类算法(AA, Agglomerative Algorithm)

Input: D={t1,t2,...,tn} // Set of elements
A // Adjacency matrix showing communication cost between elements
Output：DE // Dendrogram represented as a set of ordered triples

```
c=1+Max(A) ;
k=n;
K={{t1},{t2},...,{tn}};
DE={<c,k,Kc>} // Initially dendrogram contains each element in its own cluster
repeat
   <k,Kc>=NewClusters(A,D);
   DE=DE∪{<c-1,k,Kc>}; // New set of clusters added to dendrogram
   c=c-1;
until k=1
```

调用过程 NewCluster 是合并类运算，不同的算法采用不同的合并策略，常用合并策略，形成了单链接聚类算法、全链接聚类算法和均链接聚类算法。

7.2.4 单链接聚类算法

定义两个类间元素最大通信代价，若最大通信代价大于等于通信代价阈值 c，则合并这两个类。因此，两类间只要有一个元素对的通信代价大于等于给定的阈值 c，则合并这两个类。

算法 7.2 单链接聚类算法(SAA, Single Agglomerative Algorithm)

Input: $D=\{t_1,t_2,...,t_n\}$ // Set of elements
$A=(C_{ij})$ //Adjacency matrix showing communication costs between elements
Output：DE // Dendrogram represented as a set of ordered triples

```
c=1+Max(A);
k=n;
Kc={{t1},{t2},...,{tn}};
DE={<c,k,Kc>} //Initially dendrogram contains each element in its own cluster
repeat
    for each Ki, Kj ∈Kc do
      {
     larc=largest communication cost between all ti∈Ki and tj∈Kj ;
     if larc≥c then Kc=Kc-{ki}-{Kj}∪{Ki∪Kj};
      }
```

```
        k=|Kc|;
        DE=DE∪{<c,k,Kc>}; // New set of clusters added to dendrogram
      c=c-1;
    until k=1
```

7.2.5 全链接聚类算法

定义两个类间元素最小通信代价，若最小通信代价大于等于通信代价阈值 c，则合并这两个类，这意味着两个类间所有元素对的通信代价都要大于等于通信代价阈值。

算法 7.3 全链接聚类算法(CAA, Complete Agglomerative Algorithm)

```
Input: D={t1,t2,...,tn}  // Set of elements
  A=(Cij) // Adjacency matrix showing communication costs between elements
Output
  DE // Dendrogram represented as a set of ordered triples

  c=1+Max(A);
  k=n;
  Kc={{t1},{t2},...,{tn}};
  DE={<c,k,Kc>} //Initially dendrogram contains each element in its own cluster
  repeat
    for each Ki, Kj ∈Kc do
      {
      smac=small communication cost between all ti∈Ki and tj∈Kj ;
      if smac≥c then Kc=Kc-{ki}-{Kj}∪{Ki∪Kj};
      }
      k=|Kc|;
      DE=DE∪{<c,k,Kc>}; // New set of clusters added to dendrogram
    c=c-1;
  until k=1
```

7.2.6 均链接聚类算法

定义两个类间平均通信代价，若平均通信代价大于等于通信代价阈值 c，则合并这两个类。相关的技术：平均链接技术。

算法 7.4 均链接聚类算法(AAA, Average Agglomerative Algorithm)

```
Input: D={t1,t2,...,tn}  // Set of elements
  A=(Cij) // Adjacency matrix showing communication costs between elements
Output:  DE // Dendrogram represented as a set of ordered triples
```

c=1+Max(A);
k=n;
K_c={{t_1},{t_2},...,{t_n}};
DE={<c,k,K_c>} //Initially dendrogram contains each element in its own cluster
repeat
  for each K_i, K_j ∈K_c do
    {
    avec=Average communication cost between all t_i∈K_i and t_j∈K_j ;
    if avec≥c then K_c=K_c-{k_i}-{K_j}∪{K_i∪K_j};
    }
    k=|K_c|;
    DE=DE∪{<c,k,K_c>}; // New set of clusters added to dendrogram
  c=c-1;
until k=1

例 7.1 设智能系统含有 5 个任务：A，B，C，D，E，其间通信及通信代价如图 7-2：

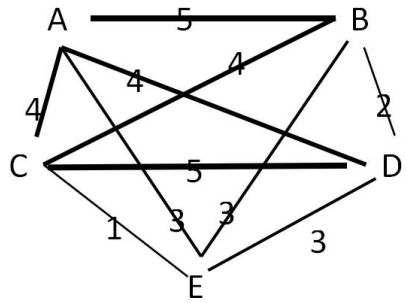


图 7-2 例 7.1 任务图

依据聚类方法求得系统的谱系图。

解：任务间的通信代价邻接矩阵：

	A	B	C	D	E
A	0	5	4	4	3
B	5	0	4	2	3
C	4	4	0	5	1
D	4	2	5	0	3
E	3	3	1	3	0

依据这个通信代价矩阵，按照三种聚类方法，产生谱系图，其中均链接算法会产生两个结果：

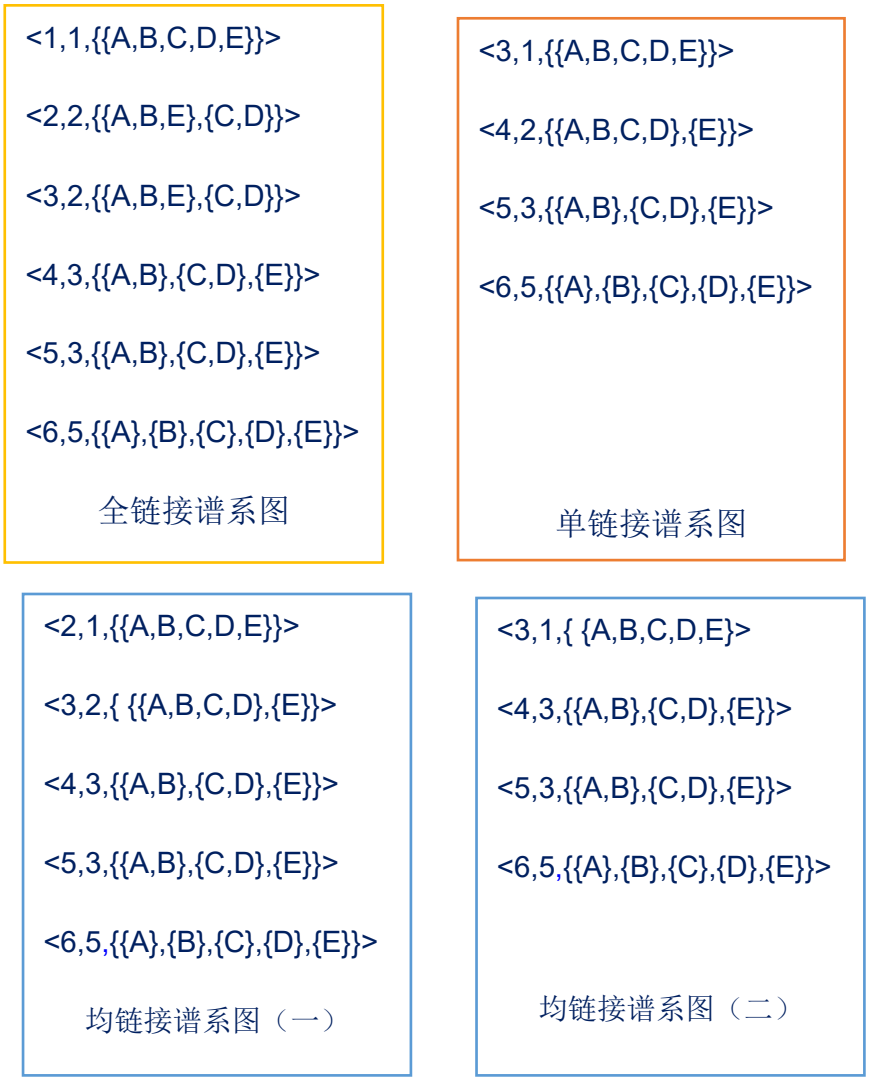


图 7-3 例 7.1 谱系图

第 7.3 节 基于聚类的多模块划分算法

基于聚类的多模块划分方法是将聚类的思想和方法应用于带有多任务的智能嵌入式系统的多模块划分中。

7.3.1 多模块聚类算法

多模块划分的目的是将任务集划分到若干个模块，因此划分成块的总数是已知的，譬如说 M。这样多模块划分中的模块实际上是聚类谱系图中的类，因此，聚类算法就可以直接应用于多模块划分。

算法 7.5 多模块聚类算法(MuMAA, Agglomerative Algorithm for Mulitmodules)

```
Input:D={t1,t2,...,tn}  // Set of elements
      A=(Cij)// Adjacency matrix showing communication cost between elements
      M // The number of modules
Output:  DE // Dendrogram represented as a set of ordered triples

  c=1+Max(A) ;
  k=n;
  K={{t1},{t2},...,{tn}};
  DE={<c,k,Kc>} // Initially dendrogram contains each element in its own module
  repeat
    <k,Kc>=NewClusters(Ac,D);
    DE=DE∪{<c,k,Kc>}; // New set of modules added to dendrogram
    c=c-1;
  until k=M
```

修改聚类算法可以得到基于聚类的多模块划分算法：多模块单链接划分算法、多模块全链接划分算法和多模块均链接划分算法。

算法 7.6 多模块单链接聚类算法(MuSAA, Single Agglomerative Algorithm for Multimodules)

```
Input: D={t1,t2,...,tn}  // Set of elements
  A=(Cij)  // Adjacency matrix showing communication costs between elements
  M  // The number of modules
Output: DE // Dendrogram represented as a set of ordered triples

  c=1+Max(A);
  k=n;
  Kc={{t1},{t2},...,{tn}};
  DE={<c,k,Kc>} //Initially dendrogram contains each element in its own module
  repeat
    for each Ki, Kj ∈Kc do
      {
      larc=largest communication cost between all ti∈Ki and tj∈Kj ;
      if larc≥c then Kc=Kc-{ki}-{Kj}∪{Ki∪Kj};
      }
      k=|Kc|;
```

DE=DE∪{<c,k,K_c>}; // New set of modules added to dendrogram
c=c-1;
until k=M

算法 7.7 多模块全链接聚类算法(MuCAA, Multimodular Complete Agglomerative Algorithm for multimodules)

Input: D={t_1,t_2,...,t_n} // Set of elements
A=(C_{ij}) // Adjacency matrix showing communication costs between elements
M // The number of modules
Output：DE // Dendrogram represented as a set of ordered triples

c=1+Max(A);
k=n;
K_c={{t_1},{t_2},...,{t_n}};
DE={<c,k,K_c>} //Initially dendrogram contains each element in its own module
repeat
for each K_i, K_j ∈K_c do
{
smac=small communication cost between all t_i∈K_i and t_j∈K_j ;
if smac≥c then K_c=K_c-{k_i}-{K_j}∪{K_i∪K_j};
}
k=|K_c|;
DE=DE∪{<c,k,K_c>}; // New set of modules added to dendrogram
c=c-1;
until k=M

算法 7.8 多模块均链接聚类算法(MuAAA, Average Agglomerative Algorithm for Multimodules)

Input: D={t_1,t_2,...,t_n} // Set of elements
A=(C_{ij}) // Adjacency matrix showing communication costs between elements
M // The number of modules
Output：DE // Dendrogram represented as a set of ordered triples
c=1+Max(A);
k=n;
K_c={{t_1},{t_2},...,{t_n}};
DE={<c,k,K_c>} //Initially dendrogram contains each element in its own module
repeat

```
for each Ki, Kj ∈Kc do
    {
    avec=average communication cost between all ti∈Ki and tj∈Kj ;
    if avec≥c then Kc=Kc-{ki}-{Kj}∪{Ki∪Kj};
    }
    k=|Kc|;
    DE=DE∪{<c,k,Kc>}; // New set of modules added to dendrogram
  c=c-1;
until k=M
```

例 7.2 将例 7.1 中样本节点化成 3 个模块和 2 个模块。

解： (1) 化成 3 个模块： 由例 7.1 的系统谱系图获得，三种聚类算法结果都是{A, B}，{C, D}，{E}。

(2) 化成 2 个模块： 由由例 7.1 的系统谱系图获得，多模块全链接划分算法：{{A,B,E}，{C,D}}，多模块单链接划分算法：{{A,B,C,D},{E}}，多模块均链接划分算法： {{A,B,C,D},{E}}。

7.3.2 多模块划分代价

多模块划分代价实际上是定义 7.2 的划分成本。

定义 7.4 多模块划分代价是指任务划分完成后所有模块间通信代价的总和。

多模块划分的目的是使划分代价极小。基于聚类划分使用贪心策略，即将通信代价大的两个任务划分到一个块内，因此划分结果可能会是局部极优，为了使划分整体极优，可以多次划分，比较划分代价，决定划分结果。

例 7.3 将例 7.1 中样本节点化成 2 个模块，并计算划分代价。

解：由例 7.2 得到，多模块全链接划分算法的划分结果为{{A，B，E}，{C，D}}，其划分代价=18，多模块单链接划分算法与多模块均链接划分算法划分结果均是{{A，B，C，D}，{E}}，其划分代价=10。

注：从划分代价来看，划分结果{{A，B，C，D}，{E}}比较好，但块{A，B，C，D}包含的任务太多，整体来看不太均匀。而划分结果{{A,B,E},{C,D}}的块中任务相对平均。

这个简单例子告诉我们使用不同聚类策略，划分结果不同。

7.3.3 规定模块最大任务数算法

在已知通信代价邻接矩阵的基础上，聚类方法是使用贪心算法，将邻接矩阵中值最大的两个节点聚集在一个块中。但这种方法不能满足在划分中块尺寸的要求。

规定单个模块包含任务数的上限，使得划分到每个块中任务个数相对均匀。在具体划分过程中，加上条件：每个块的元素个数小于等于 EN (Max of the number of elements in Modules)，当块中元素个数等于 EN 时，不能再往上加元素了，即不能再与其他块合并了。规定单个模块最大任务数的多模块划分算法 Multi-Modules-Max (MMM)：

算法 7.9 MMM 单链接聚类算法(MMMSAA, Single Agglomerative Algorithm for Multi-Modules-Max)

Input: $D=\{t_1,t_2,...,t_n\}$ // Set of elements

$A=(C_{ij})$ // Adjacency matrix showing communication costs between elements

M // The number of modules

EN // Max of the number of elements in Modules

Output： DE // Dendrogram represented as a set of ordered triples

$c=1+Max(A)$;

$k=n$;

$K_c=\{\{t_1\},\{t_2\},...,\{t_n\}\}$;

$DE=\{<c,k,K_c>\}$ //Initially dendrogram contains each element in its own module

repeat

for each $K_i, K_j \in K_c$ do

{

larc=largest communication cost between all $t_i \in K_i$ and $t_j \in K_j$;

if larc≥c and $|K_i \cup K_j| \leq EN$ then $K_c=K_c-\{k_i\}-\{K_j\} \cup \{K_i \cup K_j\}$;

}

$k=|K_c|$;

$DE=DE \cup \{<c,k,K_c>\}$; // New set of modules added to dendrogram

$c=c-1$;

until c=0 or k=M

If c=0 and k>M then calling again MMMSAA for K_1 as the input.

注：对于不太稀疏的通信代价链接矩阵，MMM 单链接聚类算法可以终止并给

出聚类结果。若不能给出符合要求的聚类结果， 则对最后的聚类 K_1 再次调用 MMMSAA 算法，在调用时需要以 K_1 中的类为元素，计算类间的通信代价，形成新的通信代价邻接矩阵，在类进行合并时注意到新类元素个数不超过 M。

MMM 全链接聚类算法和 MMM 均链接聚类算法，对不太稀疏的通信代价链接矩阵可能不会终止，也就是说聚类阈值等于 0，聚类结果也到不了希望的模块数，即模块数大于规定的模块数 M。当出现这种情况时需要使用 MMM 单链接聚类算法对聚类结果进行再次聚类，直到聚类的模块数等于规定的模块数。

算法 7.10 MMM 全链接聚类算法(MMMCAA, Complete Agglomerative Algorithm for moltimodules)

Input: $D=\{t_1,t_2,...,t_n\}$ // Set of elements
$A=(C_{ij})$ // Adjacency matrix showing communication costs between elements
M // The number of modules
EN // Max of the number of elements in Modules
Output：DE // Dendrogram represented as a set of ordered triples

```
c=1+Max(A);
k=n;
Kc={{t1},{t2},...,{tn}};
DE={<c,k,Kc>} //Initially dendrogram contains each element in its own module
repeat
  for each Ki, Kj ∈Kc do
       {
      smac=small communication cost between all ti∈Ki and tj∈Kj ;
      if smac≥c and |Ki∪Kj|≤EN then Kc=Kc-{ki}-{Kj}∪{Ki∪Kj};
       }
       k=|Kc|;
      DE=DE∪{<c,k,Kc>}; // New set of modules added to dendrogram
     c=c-1;
  until c=0 or k=M
If c=0 and k>M then calling MMMSAA for K1.
```

算法 7.11 MMM 均链接聚类算法(MMMAAA, Average Agglomerative Algorithm for Multimodules)

Input: $D=\{t_1,t_2,...,t_n\}$ // Set of elements

A=(C_{ij}) // Adjacency matrix showing communication costs between elements
M // The number of modules
EN // Max of the number of elements in Modules
Output：DE // Dendrogram represented as a set of ordered triples

c=1+Max(A);
k=n;
K_c={{t_1},{t_2},...,{t_n}};
DE={<c,k,K_c>} //Initially dendrogram contains each element in its own module
repeat
for each K_i, K_j ∈K_c do
{
avec=Average communication cost between all t_i∈K_i and t_j∈K_j ;
if avec≥c and |K_i∪K_j|≤EN then K_c=K_c-{k_i}-{K_j}∪{K_i∪K_j};
}
k=|K_c|;
DE=DE∪{<c,k,K_c>}; // New set of modules added to dendrogram
c=c-1;
until c=0 or k=M
If c=0 and K>M then call MMMSAA for K_1.

例 7.4 设一智能嵌入式系统，由任务 T1，T2，…，T10 组成，任务间的通信代价见图 7-4。使用 MMM 算法对此系统进行 3 模块划分，使得每个模块内任务数不超过 4，并计算划分的通信代价。

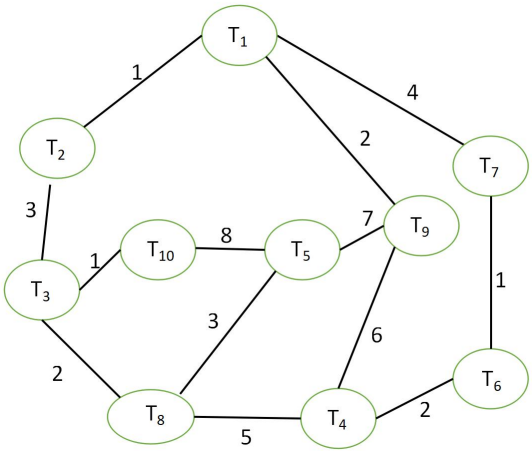


图 7-4 例 7.4 任务通信代价图

解：分四步完成。

第一步，建立任务通信代价邻接矩阵(0 省略)；

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10

T1		1					4		2	
T2	1		3							
T3		3						2		1
T4						2		5	6	
T5								3	7	8
T6				2			1			
T7	4					1				
T8			2	5	3					
T9	2			6	7					
T10			1		8					

第二步：调用多模块聚类算法，当聚类过程产生的块数为 3 时，则聚类过程终止。

第三步：输出划分结果--把这 10 个任务划分到 3 个模块上。计算划分后的通信代价。

采用不同的策略得到不同的聚类结果，单链接策略的结果如表 7-1，全链接策略的结果如表 7-2，均链接算法划分过程及结果如表 7-3 。

表 7-1 例 7.4 单链接算法划分过程及结果

阈值 c	块目 k	块 K	larc>=
9	10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	
8	9	{5, 10}, 1, 2, 3, 4, 6, 7, 8, 9	(5, 10)=8
7	8	{5, 9, 10}, 1, 2, 3, 4, 6, 7, 8	(5, 9)=7
6	7	{4, 5, 9, 10}, 1, 2, 3, 6, 7, 8	(4, 9)=6
5	7	{4, 5, 9, 10}, 1, 2, 3, 6, 7, 8	(4, 8)=5, 模块 |{4, 5, 9, 10}|=4

4	6	{4, 5, 9, 10}, {1, 7}, 2, 3, 6, 8	(1, 7)=4
3	5	{4, 5, 9, 10}, {1, 7}, {2, 3}, 6, 8	(2, 3)=(5, 8)=3，模块 |{4, 5, 9, 10}|=4
2	4	{4, 5, 9, 10}, {1, 7}, {2, 3, 8}, 6	(1, 9)=(3, 8)=(4, 6)=2，模块|{4, 5, 9, 10}|=4
1	3	{4, 5, 9, 10}, {1, 6, 7}, {2, 3, 8}	(1, 6)=1

划分后通信代价计算：

C({4,5,9,10},{1,6,7},{2,3,8})=C({4,5,9,10},{1,6,7})+C({4,5,9,10},{2,3,8})+C({1,6,7},{2,3,8})=4+9+1=14。

表 7-2 例 7.4 全链接算法划分过程及结果

阈值 c	块目 k	块 K	smac>=
9	10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	
8	9	{5, 10}, 1, 2, 3, 4, 6, 7, 8, 9	(5, 10)=8
7	9	{5, 10}, 1, 2, 3, 4, 6, 7, 8, 9	（5, 9）=7, 但 smac({5, 10}, 9)=0
6	8	{5, 10}, {4, 9}, 1, 2, 3, 6, 7, 8	(4, 9)=6
5	8	{5, 10}, {4, 9}, 1, 2, 3, 6, 7, 8	
4	7	{5, 10}, {4, 9}, {1, 7}, 2, 3, 6, 8	(1, 7)=4
3	6	{5, 10}, {4, 9}, {1, 7}, {2, 3}, 6, 8	(2, 3)=3
2	6	{5, 10}, {4, 9}, {1, 7}, {2, 3}, 6, 8	
1	6	{5, 10}, {4, 9}, {1, 7}, {2, 3}, 6, 8	
调用 MMMSAA	3	{4, 5, 9, 10}, {1, 6, 7}, {2, 3, 8}	计算出这些模块间的通信代价，得到模块间通信代价矩阵见表 7-3，使用单链接算法再次进行聚类，聚类模块大小不能超过 4。

表 7-3 表 7-2 中模块间通信代价矩阵

	6	8	{2,3}	{1,7}	{4,9}	{5,10}
6				1	2	

8			2		5	3
{2,3}				1		1
{1,7}					2	
{4,9}						7
{5,10}						

划分后计算通信代价：

C({4,5,9,10},{1,6,7},{2,3,8})=C({4,5,9,10},{1,6,7,8})+C({4,5,9,10},{2,3})+C({1,6,7,8},{2,3})=12+1+3=16.

表 7-4 例 7.4 均链接算法划分过程及结果

阈值 c	块目 k	块 K	avec>=
9	10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	
8	9	{5, 10}, 1, 2, 3, 4, 6, 7, 8, 9	avec(5, 10)=8
7	9	{5, 10}, 1, 2, 3, 4, 6, 7, 8, 9	
6	8	{5, 10}, {4, 9}, 1, 2, 3, 6, 7, 8	avec(4, 9)=6
5	8	{5, 10}, {4, 9}, 1, 2, 3, 6, 7, 8	
4	7	{5, 10}, {4, 9}, {1, 7}, 2, 3, 6, 8	avec(1, 7)=4
3	6	{5, 10}, {4, 9}, {1, 7}, {2, 3}, 6, 8	avec(2, 3)=3
2	5	{5, 10}, {4, 8, 9}, {1, 7}, {2, 3}, 6	avec({4, 9}, 8)=2.5
1	5	{5, 10}, {4, 8, 9}, {1, 7}, {2, 3}, 6	虽然 avec({4, 8, 9}, {5, 10})=1.67>1，但由于|{4, 8, 9, 5, 10}|=5>4，所以这两个模块不能合并成一个模块。
调用 MMMSAA	3	{4, 6, 8, 9}， {2, 3, 5, 10}， {1, 7}	计算出这些模块间的通信代价矩阵，其中×表示这两个模块不能合并，因为若合并则得到的模块尺寸超过了规定的模块尺寸。依据这个通信代价矩阵进行单链接聚类。

划分后通信代价计算：

C({4,6,8,9},{2,3,5,10},{1,7})=C({4,6,8,9)+{2,3,5,10})+C({4,6,8,9},{1,7})+C({2,3,5,10},{1,7})=12+3+1=16。

表 7-5 表 7-4 中模块间通信代价矩阵

	{5,10}	{4,8,9}	{1,7}	{2,3}	6
{5,10}		×		1	
{4,8,9}			×	×	2
{1,7}					1
{2,3}					
6					

划分结果统计：

表 7-6 例 7.4 模块划分结果统计表

	单链接			全链接			均链接		
划分结果	①{4, 5, 9, 10} ②{1, 6, 7} ③{2, 3, 8}			①{4, 5, 9, 10} ②{1, 6, 7} ③{2, 3, 8}			①{4, 6, 8, 9} ②{2, 3, 5, 10} ③{1, 7}		
通信代价	14			14			16		
均值	4.67			4.67			5.33		
方差	16.34			16.34			34.44		
块间通信代价	① ②	① ③	② ③	① ②	① ③	② ③	① ②	① ③	② ③
	4	9	1	4	9	1	12	3	1
块间通信代价所占的比例	29%	64%	7%	29%	64%	7%	75%	19%	6%

从划分结果统计来看使用不同链接算法，划分结果是有差异的。单链接算法与全链接算法结果比较好，一是通信代价小，二是块的大小比较均匀，三是块间通信代价的方差小。因此，此智能系统 10 个任务模块化比较好的结果为：{4, 5, 9, 10}，{1, 6, 7}，{2, 3, 8}。但是经过大规模数据训练，发现均链接算法划分结果是比较好的[37]。

7.3.4 多处理器任务分配

多处理器任务分配问题描述如下：

考虑 n 个任务 T_1，T_2， …，T_n 在 m 个处理器 P1，P2，…，Pm 上执行，任务之间存在通信，通信会产生通信代价。如何在 m 个处理器上划分这 n 个任务，每个处理器处理的任务数不超过最大值 Max，使得 m 个处理器间通信代价达到极小？

解决方案：每个处理器可以看做一个模块，m 个处理器就是 m 个模块，使用规定单个模块最大任务数的多模块划分算法 MMM，就可以给出多处理器任务分配问题。

第 7.4 节 基于 KL 算法的多模块划分

KL 算法是 Kernighan 与 Lin 在 1970 年设计的一个图划分算法，其目的是在寻找图 G 的一个极小成本可许划分[38]。KL 算法是一种基于贪心策略的划分算法，通过交换两模块中的元素实现两模块间通信代价极小化。由于贪心算法会导致局部最优解，KL 算法在交换模块中元素时把所有可能交换后产生的代价增益都计算出来，从中选择一个最好的交换元素进行交换。这样在某种程度上降低了局部最优的风险，但会造成算法时间复杂性的增加。不过，KL 算法在很大程度上会给出全局最优解。实验表明 KL 算法给出的划分通信代价极小化是最好的[37]。

假设图 G 含有尺寸为 1 的 n 个节点，将 G 划分成尺寸为 p 的 k 子集块，其中 kp = n。则选第一个块有 C_n^p 方式，选第二个块有 C_{n-p}^p 方式，选第三个块有 C_{n-2p}^p 方式，一直下去。由于这些块排序是不需要的，因此，共有 $C_n^p \times C_{n-p}^p \times C_{n-2p}^p \times \ldots \times C_{n-(k-1)p}^p / k!$ 方式。这种算法的复杂性是很高的。

对于较大的 n， k 和 p, 这个表达式产生了非常大的数字，如当 n = 40，p = 10 (k= 4)，至少有 10^{20} 选择方式[38]。

Kernighan 与 Lin 提出了一个基于贪心算法的 2 式划分算法，将待划分的图（含有 2n 个）节点集合划分成两个含元素个数一样多的两个块（如 A 和 B，都含有 n 个节点）。

7.4.1 1-优化与 2 式划分

设有向图 G 含有 2n 节点，S 为 G 的节点集，带有通信代价邻接矩阵 $C = (C_{ij})$，$i, j = 1, \ldots, 2n$. 不失一般性，假定矩阵 C 是对称的并且对于所有的 i 都有 $C_{ii} = 0$。

希望把 G 的节点集 S 划分成集合 A 和 B，每个含有 n 个节点，使得 "外部成本" $T = \Sigma_{(a,b)\in A\times B} C_{ab}$ 是极小的。

定义 7.4 （2 式划分）给定 2n 节点的图 G，其中每个节点都拥有相同的尺寸，寻找一个极小成本划分将 2n 节点划分成都含有 n 个节点的两个块 A*和 B*，称 A*和 B*为极小成本 2 划分。

这个方法原则上是这样的：

从 S 的任何划分 A 和 B 开始，为了减少这初始的外部成本 T，可以安排一系列的 A，B 子集间的互换。当没有进一步的改进时，划分结果 A'和 B'是这个算法的局部极小。这个过程可以从另外任何一个初始划分 A，B 开始，生成许多局部极小划分结果。

设 A，B 是 S 的任意的 2 划分，则找子集 $X\subseteq A$，$Y \subseteq B$，具有 $|X| = |Y| \leqq n/2$，使得互换 X 和 Y 产生划分 A*和 B*， 如图 7-5：

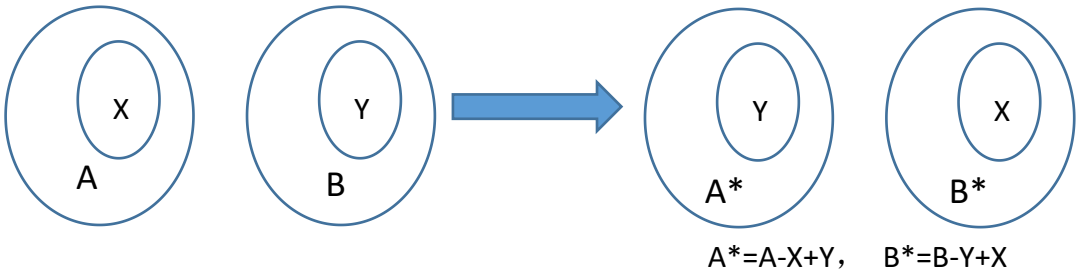


图 7-5 子集互换示意图

问题是如何从 A 和 B 中精选 X 和 Y， 而不是考虑所有的可能选择。 精选 X 和 Y 是通过一个过程逐步选定。

对于每个 $a \in A$，定义一个外部成本 $E_a = \Sigma_{y\in B} C_{ay}$， 一个内部成本 $I_a = \Sigma_{z\in A} C_{az}$。类似地， 对于每个定义 $b \in B$ 定义 E_b 和 I_b。

对于给定集合 S 和 S 的一个 2 划分 A 和 B，对于所有的 $z \in S$， 定义外部成本与内部成本之差 $D_z = E_z - I_z$，进一步地有，若 $z \in A$ 则 $D_z = (\Sigma_{y\in B} C_{zy}) - (\Sigma_{y\in A} C_{zy})$，若 $z \in B$ 则 $D_z = (\Sigma_{y\in A} C_{zy}) - (\Sigma_{y\in B} C_{zy})$。

定义 7.5（λ**互换**）两个集合间的λ点互换是指交换这两个集合的λ个元素。

定义 7.6（1 **互换**）1 互换是将一个集合的单点与另一个集合的单点进行互换。

定义 7.7 (互换收益) 设 a ∈ A 和 b∈ B, 若 a 和 b 互换 则交换 a 和 b 收益定义为交换前成本与交换后成本之差，并记作 g。

引理 7.8 考虑任何 a ∈ A 和 b∈ B. 若 a 和 b 互换，则收益 g 恰好是 $D_a+D_b-2C_{ab}$。

证明：设 F 是关于 A 和 B 间所有邻接成本之和，但不包括 a 和 b。再设 T 是 A 和 B 之间所有邻接成本之和，则 $T = \Sigma_{(x,y)\in A\times B} C_{xy} =F + E_a + E_b-C_{ab}$ （因为 C_{ab} 在 E_a 和 E_b 中都出现了，因此出现了两次）。现交换 a 和 b，再设 T'是交换 a 和 b 之后模块 A 和 B 之间的新成本，则

$$T' =F + E_a'+E_b'-C_{ba}=F+(I_a +C_{ab})+ (I_b +C_{ba})- C_{ab}。$$

进而收益 g=老成本-新成本 $= T-T'=D_a+D_b-2C_{ab}$，这是因为 $T-T'=F + E_a + E_b-C_{ab} -(F+(I_a +C_{ab})+ (I_b +C_{ba})- C_{ab})=E_a-I_a+E_b-I_b-2C_{ab}=D_a+D_b-2C_{ab}$。

注：在 a 和 b 互换时，若 g>0，则表明这次互换使得新模块间的通信成本减少了，可以接受这次互换，即把用 b 替换集合 A 中的 a，同时用 a 替换集合 B 中的 b。这次互换得到的新集合 A'和 B'间通信成本低于 A 和 B 间的通信成本。若 g<=0，则可以不接受这次 a 与 b 的互换，因为交换后得到的模块间通信成本没有降低甚至提升了。

定义 7.9 （1-优化）若一划分 A'和 B'，不再存在 1−互换导致划分成本下降，则称这个划分 A'和 B'为 1−优化。

7.4.2 KL 1 优化 2 式划分算法

本段介绍 KL 算法的基础算法：1 优化 2 式划分算法。算法的核心是计算 1 交换产生的划分成本增益，若不再存在 1 互换导致划分成本下降，则算法终止。Kernighan 与 Lin 在文献[38]表明 KL 算法有极大的可能性获得整体极小划分。

算法 7.12 KL 1 优化 2 式划分算法

Input: 含 2n 节点的有权图 G, 以及 G 的通信代价邻接矩阵 $C=(C_{ij})$

Output: 通信代价代价极小的 2 式划分 A 和 B

第 1 步：对 G 的节点进行任意 2 式划分：A 和 B。

第 2 步：对于初始划分 A 和 B，计算 G 中每个节点 a 的 D_a。

第 3 步：从 A 中选 a，从 B 中选 b 使得收益 $g_1=D_a+D_b-2C_{ab}$ 最大，将 a 记为 a1，将 b 记为 b1。

第 4 步：对 A-{a1}和 B-{b1}使用第 2 步；在第 3 步中选定的 a∈A-{a1}和 b∈B-{b1}分别记为 a2 和 b2 以及对应的收益 gg2；对 A-{a1，a2}，B-{b1，b2}再次使用第 2 步，依次下去，在 n 次重复后会停止，并会选定元素对：(a1，b1)，(a2，b2)， …，(an，bn)和对应的收益 g_1，g_2，…，g_n。

第 5 步：选 k 使得 G= $g_1+\ldots+g_k$ 最大。若 G>0 则令 X={a1，…，ak}，Y={b1，…，bk}并交换 X 与 Y 即，令 A=A-X+Y，B=B-Y+X；转到第 2 步；若 G<=0 则，输出局部极小划分 A 和 B。

第 6 步：算法终止。

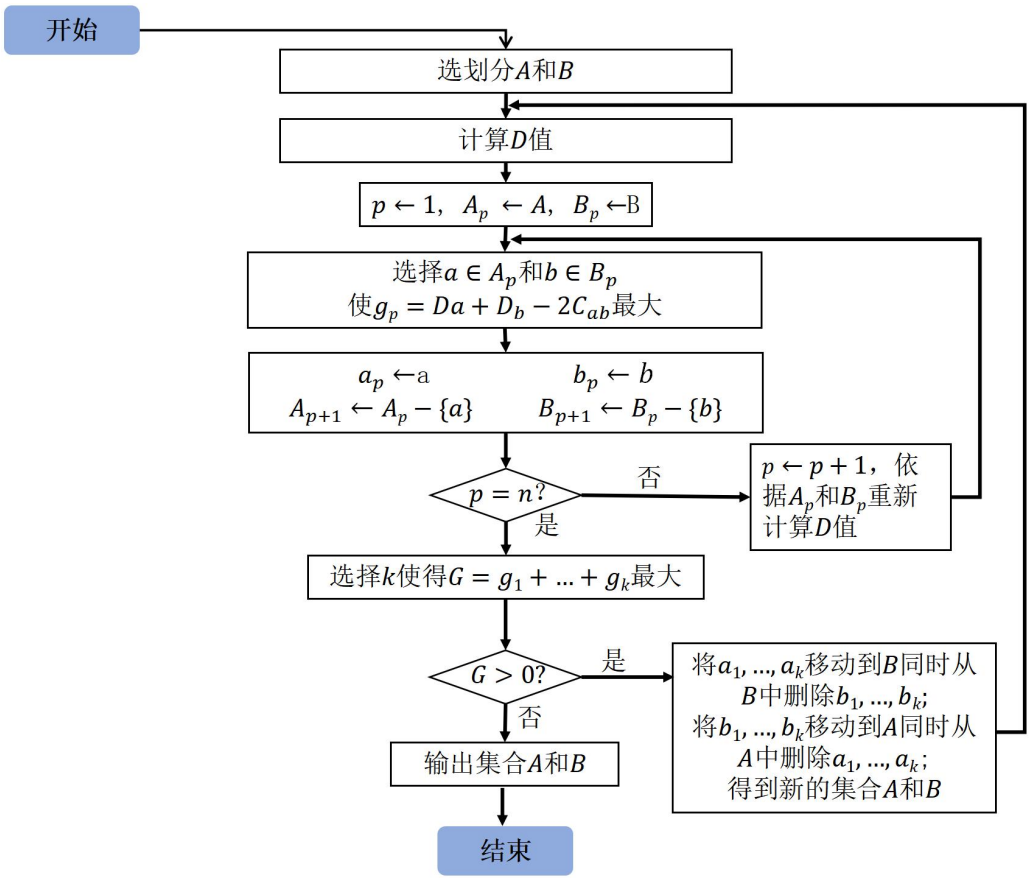


图 7-6 KL 的 1 优化 2 式划分算法框架图

7.4.3 KL 划分算法例子

例 7.5 图 7-7 是一个含有 8 个元器件（节点）的电路图[39]，元器件边的权重是通信代价，使用 KL 算法，给出比较优的 2 式划分。

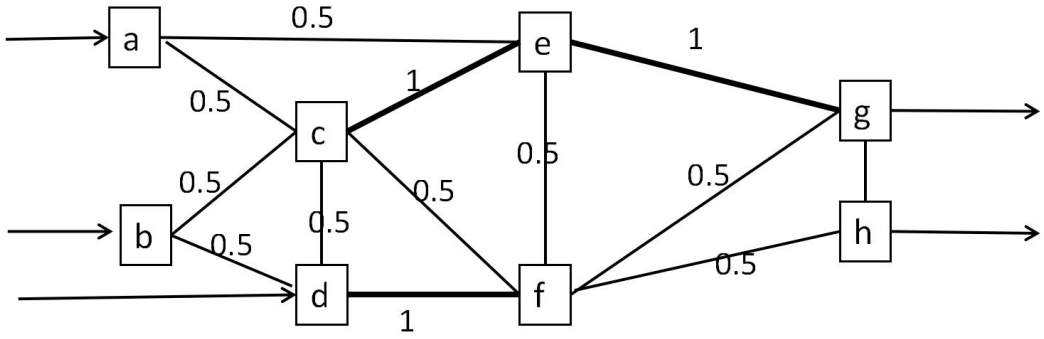


图 7-7 例 7.5 电路示意图

表 7-7 例 7.5 通信代价矩阵

Cxy	a	b	C	d	e	f	g	h
a	0	0	0.5	0	0.5	0	0	0
b	0	0	0.5	0.5	0	0	0	0
c	0.5	0.5	0	0.5	1	0.5	0	0
d	0	0.5	0.5	0	0	1	0	0
e	0.5	0	1	0	0	0.5	1	0
f	0	0	0.5	1	0.5	0	0.5	0.5
g	0	0	0	0	1	0.5	0	0.5
h	0	0	0	0	0	0.5	0.5	0

解：使用 KL 算法（算法 7.12）。

第一步：任意选出初始划分集合 A={a，c，d，f}，B={b，e，g，h}，记 A_1=A，B_1=B；

第二步：依据这个初始划分 A_1 和 B_1，计算 D 的值

按照公式 $D_z=E_z-I_z$ 依次计算出 D_z 的值，如下表：

表 7-6 D 值

D 的值	
D_a	0
D_b	1
D_c	0
D_d	-1
D_e	1
D_f	0
D_g	-1
D_h	0

第三步：计算收益 g 的值：

按照公式 $g_p=D_x+D_y-2C_{xy}$ 计算待交换产生的收益

p=1 情形：

表 7-9 收益 值(p=1)

交换对(x, y)	Dx	Dy	Cxy	收益 g_p
(a, b)	0	1	0	1
(a, e)	0	1	0.5	0
(a, g)	0	−1	0	−1
(a, h)	0	0	0	0
(c, b)	0	1	0.5	0
(c, e)	0	1	1	−1

(c, g)	0	-1	0	-1
(c, h)	0	0	0	0
(d, b)	-1	1	0.5	-1
(d, e)	-1	1	0	0
(d, g)	-1	-1	0	-2
(d, h)	-1	0	0	-1
(f, b)	0	1	0	1
(f, e)	0	1	1	0
(f, g)	0	-1	0.5	-2
(f, h)	0	0	0.5	-1

g_1 最大值是 1，对应的交换对是(a，b)，a_1<--a，b_1<--b，或者 a_1<--f，b_1<--b。更新后得到 A_2={c，d，f}和 B_2={e，g，h}，或者 A_2={a，c，d}和 B_2={e，g，h}。

p=2 情形：

选择计算初始组为 A_2={c，d，f}和 B_2={e，g，h}。

依据这个初始组，重新计算 D 值，如下表：

表 7-10 D 值(p=2)

D 的值	
D_c	0
D_d	-1.5

D_e	0.5
D_f	0
D_g	-1
D_h	0

计算待交换对发生后产生的收益 g_2

表 7-11 收益 g_p 值(p=2)

交换对(x, y)	Dx	Dy	Cxy	收益 g_p
(c, e)	0	0.5	1	-1.5
(c, g)	0	-1	0	-1
(c, h)	0	0	0	0
(d, e)	-1.5	0.5	0	-1
(d, g)	-1.5	-1	0	-2.5
(d, h)	-1.5	0	0	-1.5
(f, e)	0	0.5	0.5	-0.5
(f, g)	0	-1	0.5	-2
(f, h)	0	0	0.5	-1

g_2 最大值是 0，对应的交换对是(c，h)， a_2<--c，b_2<--h。计算的结果选为(c，h)得集合 A_3={d，f}，B_3={e，g}。

p=3 情形:

计算的初始组为： A_3={d，f}和 B_3={e，g}。重新计算 D 值为下表：

表 7-12 D 值(p=3)

D 的值	
D_d	-1
D_e	-0.5
D_f	0
D_g	-0.5

计算交换对发生后产生的收益 g_3

表 7-13 收益 g_p 值(p=3)

交换对(x, y)	D_x	Dy	Cxy	收益 g_p
(d, e)	-1	-0.5	0	-1.5
(d, g)	-1	-0.5	0	-1.5
(f, e)	0	-0.5	0.5	-1.5
(f, g)	0	-0.5	0.5	-1.5

g_3 最大值是-1.5，对应的交换对是(d，e)， a_3<--d，b_3<--e。计算结果交换对为(d，e)得分组集合 A_4={f}，B_4={g}。

p=4 情形：

计算的初始组为： A_4={f}和 B_4={g}

再次重新计算 D 值，为下表：

表 7-14 D 值(p=4)

D 的值	
D_f	0.5
D_g	0.5

计算交换对发生后产生的收益 g_4

表 7-15 收益 值(p=4)

交换对(x, y)	D_x	D_y	Cxy	收益 g_p
(f, g)	0.5	0.5	0.5	0

收益 $g_4=0$。

各步计算收益为： $g_1=1$，$g_2=0$，$g_3=-1.5$，$g_4=0$。

第四步：计算最大的 k 使得收益和 G 最大

计算 $G=g_1+g_2+g_3+g_4$。得到结果为 k=1 或 2 取最大值 1，但由于 $g_2=0$，即即使交换 c 与 h，没有产生增益，因此，只取 k=1 情形。

第五步：移动元素，得到划分结果：

初始划分：A={a，c，d，f} ， B={b，e，g，h}

p=1: 交换对是(a，b)-a_1<--a，b_1<--b

p=2: 交换对是(c，h)- a_2<--c，b_2<--h

p=3: 交换对是(d，e)-a_3<--d，b_3<--e

p=4: 交换对是(f，g)-a_4<--f，b_4<--g

取 k=1：交换一对(a，b)结果为 A'={b，c，d，f}，B'={a，e，g，h}。

由于 G=1>0，因此依据 KL 算法进入循环，以 A'={b，c，d，f}，B'={a，e，g，h}作为划分初始分组，进行第二次计算和分组。

第二次计算：以 A'={b，c，d，f}和 B'={a，e，g，h}为初始划分，使用 KL 算法（算法 7.12）计算过程省略，计算收益结果为： $g_1=0$，$g_2=-0.5$，$g_3=-1.5$，$g_4=0$。计算最大的 k 使得收益和 $G=g_1+g_2+g_3+g_4$ 最大，得到结果为 k=1 并且 G

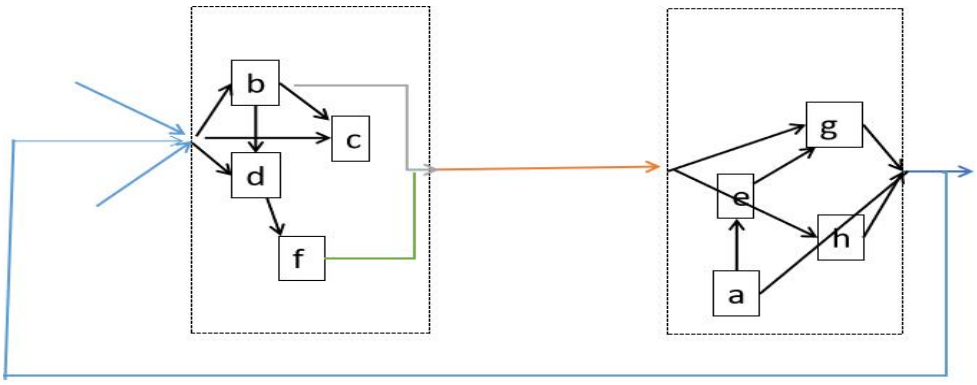
取得最大值 0。由于此时 G=0，因此算法终止。输出最后分组结果为（同第二次初始组）：A*={b, c, d, f}，B*={a, e, g, h}。

第五步：计算通信代价见表 7-16。

表 7-16 通信代价

初始分组			KL 算法	划分结果		
A	B	通信代价		A^*	B^*	通信代价
{a, c, d, f}	{b, e, g, h}	3. 5		{b, c, d, f}	{a, e, g, h}	3

在选出初始划分集合 A={a，c，d，f}和 B={b，e，g，h}时，通过使用 KL 算法，算法终止于划分 A*={b，c，d，f}和 B*={a，e，g，h}，为一个候选划分结果。



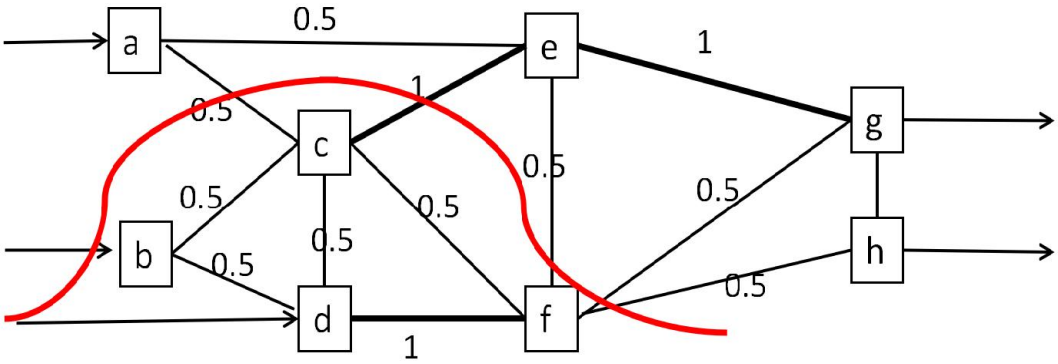


图 7-8 划分结果电路图:划分组{b, c, d, f}和{a, e, g, h}

从电路图设计角度来看， 这个划分结果不太理想。我们需要再次选定一个初始划分，使用 KL 算法，计算出划分结果。

现在选 A''={a，b，c，d}和 B''={e，f，g，h}为初始分组，使用 KL 算法再进行划分，看看结果如何？经过使用 KL 算法（算法 7.12），各步计算收益为：g_1=0，

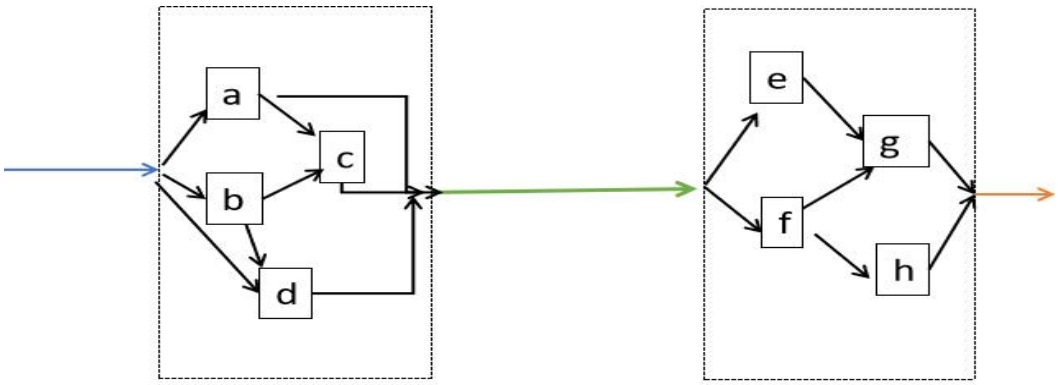
g_2=-0.5，g_3=-1.5，g_4=0。由此可得：k=1 时使得各步收益和 $G=g_1+g_2+g_3+g_4$ 取得最大值 0。

由于 G=0，因此算法分组终止。输出最后分组结果为：A’={a，b，c，d}，B’={e，f，g，h}，与初始划分 A’’和 B’’相同。其计算通信代价见表 7-17。

表 7-17 通信代价

划分结果	A’	B’	通信代价
结果	{a, b, c, d}	{e, f, g, h}	3

因此，划分 A’={a，b，c，d}和 B’={e，f，g，h}也是一个解。至此， 得到了两个解：划分组{b，c，d，f}和{a，e，g，h}与划分组{a，b，c，d}和{e，f，g，h}。两个划分代价都是 3，即两个组间通信代价都是 3。这样这两个划分组都是解的候选者。但从电路设计来看，划分组{a，b，c，d}和{e，f，g，h}更合理和优化。从原电路图图 7-8 来看，元件 a 要直接给元件 c 和 e 送数据。若选择划分组{b，c，d，f}和{a，e，g，h}，则 a 送数据给 e 是在模块内完成，但 a 给 c 送数据是要通过模块间通信完成。若选择划分组{a，b，c，d}和{e，f，g，h}，则电路图为图 7-9，这样 a 给 c 送数据是在模块内完成，当然 a 要给 e 送数据是要通过模块间通信完成，但这种数据传输可以封装在模块间通信进行。



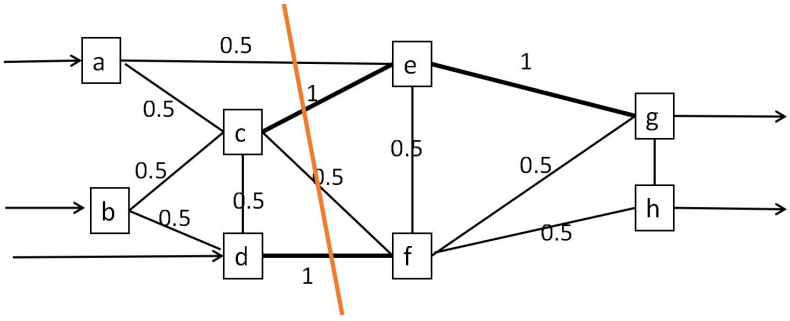


图 7-9 划分结果电路图：划分组{a, b, c, d}和{e, f, g, h}

7.4.4 KL 划分算法拓展

7.4.2 节介绍的 KL 算法 1 优化 2 划分情况，特别地，2 划分是比较特殊的情形，本段介绍一般情形的划分。整体思想是把一般情形转换成 7.4.2 节的特殊情形，调用 2 式划分算法， 最后进行处理，实现一般情形的划分。解决这个问题的关键是引入零元素。

定义 7.10 （零元素）若一个元素与其他所有元素没有邻接，即与其他所有元素邻接（边）权重值都是数值零，则称该元素为零元素。

（1）奇数个元素集合的划分

设 S 含有奇数个元素，比如 2n+1，则增加一个零元素 ε，得到一个新的含有偶数个元素 2(n+1)集合 S*，对 S*进行 2 式划分，从划分结果中删掉零元素，这样就得到了集合 S 的一个 2 划分，其中一个集合含有 n+1 个元素，而另外一个含有 n 个元素。

（2）模块尺寸不相等划分

设 S 含有 n 个元素，将 S 划分成两个模块 A 和 B，分别含有 n_1 个元素和 n_2 个元素($n_1+n_2=n$)。

假定 $n_1 < n_2$，则加入 $2n_2-n$ 个零元素到 S 中，得到新的集合 S*，共有 $2n_2$ 个元素，使用 KL 算法把 S*进行 2 划分，初始划分 S*成均含有 n_2 的模块 A 和 B，其中一个模块不含有零元素，譬如模块 A 不含有零元素。对这个初始划分使用 KL 算法进行交换对求解，在进行交换对时，规定零元素不参与交换，求出的结果划分 A*和 B*，集合 A*不含有零元素，而模块 B*含有所有的零元素。把零元素去掉，最后两个集合是所需要的划分，即满足元素个数要求的划分。

（3）节点尺寸不相等划分

2 式划分是假定节点的尺寸相等。若节点的尺寸不相等，比如节点的尺寸为 k(>1)。

可以将尺寸为 k 的节点分成 k 个尺寸为 1 的节点，这样节点尺寸是相等的（为 1）。然后调用 2 式划分。

7.4.5 KL 多式划分算法

设集合 S 含有 n 个元素，把集合 S 划分成 k 个模块 S_1，…，S_k，使得

$$S=S_1 \cup S_2 \cup \ldots \cup S_k,$$

即 k 式划分，这 k 个模块未必要求含有相同的元素个数。为了调用 KL 算法的 2 式划分，可以增加零元素，使之初始划分的模块含有相同元素个数。

情形 1: 若 $k=2^L$ 则进行 L 个 2 式划分，就可以得到结果。

L 个 2 式划分是：

第 1 次 2 式划分：将集合 X 划分成 A^1，B^1；

第 2 次 2 式划分：将集合 A^1 和 B^1 分别划分成 $A^2 1$ 和 $A^2 2$， 和 $B^2 1$ 和 $B^2 2$；

第 3 次 2 式划分：将集合 $A^2 1$，$A^2 2$，和 $B^2 1$，$B^2 2$ 分别划分成 $A^3 1$ 和 $A^3 2$，$A^3 3$ 和 $A^3 4$，以及，$B^3 1$ 和 $B^3 2$，$B^3 3$ 和 $B^3 4$；

依次下去

第 L 次 2 式划分：得到划分模块 $A^L 1$，$A^L 2$，…，$A^L 2^{L-1}$ 以及 $B^L 1$，$B^L 2$，…，$B^L 2^{L-1}$。

例 7.6 设集合 S 含有 16 个元素，划分成 4 个模块，每个模块合含有 4 个元素。

解：需要进行 4=（2^2）式划分，即只需 2 次 2 式划分。

第一次 2 式划分 A1,B1(每个集合含有 8 个元素)，第二次 2 式划分，得到模块 $A^2 1$，$A^2 2$ 和 $B^2 1$，$B^2 2$，每个模块含有 4 个元素。两次调用 2 式划分，完成了 4 式划分。

例 7.7 设集合 Y 含有 15 个元素，进行 4 式划分，即划分成 4 个模块，每个模块最多含有 4 个元素。

解：需要增加 1 个零元素，得到集合 Y'含有 16 个元素，然后进行 4 式划分。

例 7.8 设集合 Z 含有 13 个元素，进行 4 式划分，即划分成 4 个模块，每个模块最多含有 4 个元素。

解：这个例子需要增加 3 个零元素，得到集合 Z'含有 16 个元素，然后进行 4 式划分。

例 7.9 集合 U 含有 11 个元素，进行 4 式划分，即划分成 4 个模块，每个模块最多含有 4 个元素。

解：这个例子需要增加 5 个零元素，得到集合 U'含有 16 个元素，然后进行 4 式划分。

一般：设集合 X 含有 N 个元素，进行 $k=2^L$ 式划分，每个模块最多含有 n 个元素，增加 kn-N 个零元素，得到含有 kn 个元素的新集合 X'，对 X'进行 L 个 2 式划分，就可以得到所要的划分结果。

情形 2: 若 k 不是 2^L，则存在 m 使得 $2^{m-1}<k<2^m$。先将集合进行 2^m 式划分，即 m 个 2 式划分，然后进行聚类合并，聚类合并时要使得聚类结果集满足小于等于 n，同时通信代价最小。

例 7.10 设集合 X 含有 15 个元素，进行 3 式划分，每个集合含有 5 个元素。

解：由于 $2^1<3<2^2$，则对集合 X 先进行 2^2 式划分：增加 5 个零元素(=5×4-15)到集合 X 中得到含有 20 个元素的集合 X'，对集合 X'进行 2^2 式划分，得到划分模块 A,B,C,D（每个模块含有 5 个元素，也包括了零元素）。去掉增加的 5 个零元素，得到 4 个模块 A'，B'，C'和 D'或 3 个模块 A'，B'，C'。若是 3 个模块，则就不需要聚类了，这 A'，B'，C'就是所要的划分结果。若是 4 个模块 A'，B'，C'和 D'，使用聚类把 A'，B'，C'和 D'进行合并,合并时注意到合并的模块元素个数为 5，形成 3 式划分 A''，B''，C''。

算法 7.13　2^L 式划分算法

Input: 有权图 G, 以及 G 的通信代价邻接矩阵 $C=(C_{ij})$，正整数 L 和 n

Output:通信代价代价极小的 2^L 式划分 A1，A2，…，$A2^L$

// L 次调用 KL　1 优化 2 式划分

第 1 步： 初始步， 令 S 是 G 的节点集;

第 2 步： 判断 $k=2^L$? 若等号成立则转向第 3 步，否则转向第 5 步；

第 3 步：调用 KL 1 优化 2 式划分，分 L 次进行：

　第 1 次 2 式划分：将集合 S 划分成 $A^1 1$ 和 $B^1 1$；

　第 2 次 2 式划分：将集合 $A^1 1$ 和 $B^1 1$ 分别划分成 $A^2 1$ 和 $A^2 2$，$B^2 1$ 和 $B^2 2$；

　第 3 次 2 式划分：将集合 $A^2 1$，$A^2 2$　以及 $B^2 1$, $B^2 2$ 分别划分成：

　$A^3 1$ 和 $A^3 2$，$A^3 3$ 和 $A^3 4$，以及，$B^3 1$ 和 $B^3 2$，$B^3 3$ 以及 $B^3 4$；

　依次下去

　第 L 次 2 式划分：得到划分模块 $A^L 1$，$A^L 2$，…，$A^L 2^{L-1}$，以及，$B^L 1$，$B^L 2$，…，$B^L 2^{L-1}$。

第 4 步：输出划分模块 $A^L 1$，$A^L 2$，…，$A^L 2^{L-1}$，以及，$B^L 1$，$B^L 2$，…，$B^L 2^{L-1}$。

第 5 步：算法结束

算法 7.14 多式划分算法

Input: 有权图 G, 以及 G 的通信代价邻接矩阵 C=(Cij)，正整数 k 和 n

Output:通信代价代价极小的 k 式划分 A1，A2，…，Ak

第 1 步：令 m=|G|，判断 m=k×n？若=成立，则转向第 2 步，否则转向第 3 步；

第 2 步：判断是否有正整数 L 使得 $k=2^L$？若=成立则转向第 4 步，否则转向第 5 步；

第 3 步：增加 k×n－m 个零元素，构造新的有权图 G，和 G 的通信代价链接矩阵 C=(Cij)，转向第 1 步；

第 4 步：对 G 进行 2^L 式划分，得到模块划分结果 A1，…，Ak(去过零元素)，转向第 6 步；

第 5 步：找正整数 T 使得 $2^{T-1}<m<2^T$，并进行 2^T 式划分，得到划分模块 A1，…，$A2^T$（去过零元素），将这些 2^T 模块聚类合并成 k 个模块 A1，…，Ak（聚类合并时模块的元素个数不超过 n），转向第 6 步；

第 6 步：输出 A1，…，Ak，算法终止。

多式划分算法流程图如图 7-10。

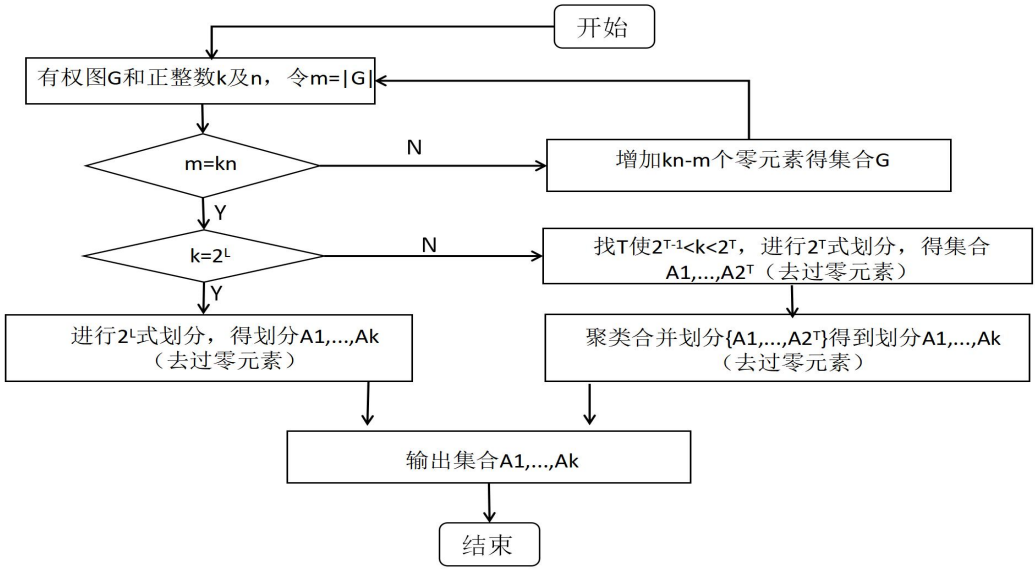


图 7-10 多式划分算法流程图

第 7.5 节 本章小结

本章介绍了基于聚类算法的多模块划分方法和基于 KL 算法的多模块划分方法。划分目的是将整个系统的任务进行模块化，每个模块可以安排给 FPGA

或 ASIC 或微处理器进行实现执行，划分的优化体现在模块间通信代价极小化。从例子可以看出，不同划分方法会产生不能的划分结果，因而需要从这些不同的结果中选择恰当结果，成为最终的解决方案。

文献[37]提出一种异构分布式嵌入式系统的优化设计方法，该方法能在满足通信代价、能耗、硬件资源以及时间等系统属性约束的前提下，给出系统任务划分的最优策略，实现系统任务的合理分配。优化方法主要分为 3 步：第一步，将系统按照一定粒度大小划分为多个任务，并获取系统的任务属性；第二步，依据通信代价对系统任务进行模块化聚合，将每个模块分配到系统的某个异构嵌入式设备中运行；最后一步，对每个模块依据任务属性和设备资源进行软硬件划分，使各模块在满足其异构设备资源约束的前提下运行时间最短。该优化方法充分运用系统软硬件的资源和能效，得到高性能、低成本的优化设计方案，极大地提高系统设计的合理性和效率。该方法可以提高异构分布式嵌入式系统中任务分配的合理性，在微电子技术和通信技术有着重要应用。该文献系统地比较了基于模块划分的软硬件划分方法以及 KL 划分算法的性能指标。从运行执行性能来看，基于单链接聚类算法有优越的时间性能。从划分结果上看，改进的 KL 算法通过多次迭，最后总能得到通信代价最小的划分方案。基于均链接聚类算法得到的划分方案仅次于 KL 算法，但运行时间远小于 KL 算法，因而是综合性能最好的算法。

习题

7.1 编程实现多模块化划分算法 MMM（单链接、全链接、均链接），输入是通信代价、模块数、每个模块最大任务数，输出为划分好的模块，并给出划分代价。

(1) 使用实现的程序将例 7.4 的 10 个任务划分成 4 个模块，每个块内任务个数不超过 3，任务图见图 7-4。

(2) 使用实现的程序将图 7-7 中的 8 个任务 a，b，c，d，f，g，h 分成 2 个模块，每个模块含有 4 个任务，使得模块间通信代价最小，任务间通信代价见表 7-5。

7.2 编程实现 KL 算法，对表 7-16 中的任务 a，b，c，d，f，g，h 选取至少 3 组不同的初始划分进行 2 式划分，并计算划分代价，从中选择一个最优划分方案，其中表中数字为通信代价。

表 7-18 习题 7.2 的任务通信代价表

Cxy	a	b	c	d	e	f	g	h
a	0	1	0.5	0	0.5	1	0	0.5
b	1	0	0.5	0	0	0	0	0
c	0.5	0.5	0	0.5	1	0.5	0	0
d	0	0	0.5	0	0	1	0	0.5
e	0.5	0	1	0	0	0.5	1	0
f	1	0	0.5	1	0.5	0	0.5	0.5
g	0	0	0	0	1	0.5	0	0.5
h	0.5	0	0	0.5	0	0.5	0.5	0

7.3 编程实现 KL 多式划分算法，将图 7-11 任务划分成 3 组，使得每个组中元素不超过 4 个，且分组后组间通信成本最小，图中边的权值是通信代价。

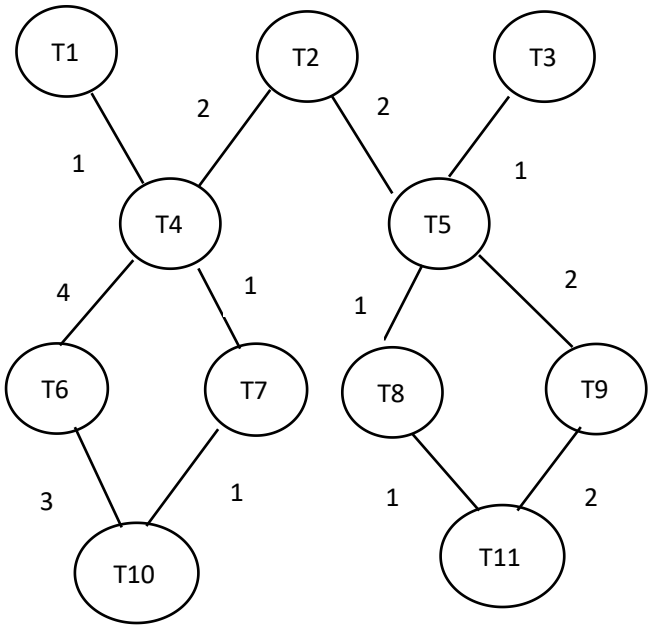


图 7-11 习题 7.3 任务图