参数算法的实现研究*)

张祖平　周苗苗　陈建二
(中南大学信息科学与工程学院　长沙410083)

摘　要　参数算法在工业制造和生物化学等很多领域得到了广泛的应用。在典型的参数算法中，有界搜索树和动态规划是常用技术。论文以代表性的可重构阵列瑕点覆盖参数算法为例，论述了算法基于面向对象思想的模块设计及基于Java的实现技术，详细说明了有界搜索树与动态规划的具体实现技术，对复杂参数算法从纯理论研究走向实际应用作了探索性的研究。

关键词　参数算法，有界搜索树，动态规则，点覆盖

Study of Implementation about Parameterized Algorithm

ZHANG Zu-Ping　ZHOU Miao-Miao　CHEN Jian-Er
(School of Information Science & Engineering, Central South University, Changsha 410083)

Abstract　Parameterized algorithm is widely applied in many fields such as industrial manufacture, biochemistry and so on. Bounded search tree and dynamic programming are frequently-used techniques in parameterized algorithm. Taking a typical parameterized algorithm about fault coverage in reconfigurable arrays as example, this paper describes the modules design based on Object Oriented techniques and its Java-based implementary techniques. It also explains in detail the implementation of bounded search tree and dynamic programming. It is an exploratory development for complicated parameterized algorithm from pure theoretical research to practical application.

Keywords　Parameterized algorithm, Bounded search tree, Dynamic programming, Vertex coverage

1　引言

自从20世纪90年代参数计算理论提出以来，参数算法得到了广泛的关注，并在工业制造和生物化学等很多领域得到了普遍的应用，有效地推动了包括计算机技术在内的很多技术的发展，如VLSI容错设计、生物计算的序列比对等。在典型的参数算法中，有界搜索树和动态规划是常用技术，其中有界搜索树是固定参数算法的核心部分，例如经典的点覆盖问题的算法[1,2]都是通过构造有界搜索树来解决的，查找DNA序列中Motifs的Clique算法中也通过构造有界搜索树来实现在k分图中找最大团[3]；而动态规划算法则以其准确性广泛存在于生物信息处理中，以此来进一步降低算法的时间复杂度，如通用的序列比对的Smith-Waterman算法[4]就是通过动态规划的方法在得分矩阵中回溯寻找最优的比对序列，瑕点覆盖算法通过动态规划有效地降低了算法的指数时间中的底数[6]，因此有界搜索树与动态规划在参数算法及其应用中具有很强的代表性。以往文献中提供的都是算法的提出与复杂性的理论分析，其中涉及到很多的有关图的经典结果[5,6]，而参数算法真正应用到具体的实际中还有很多实现技术值得研究，如瑞士苏黎士皇家理工学院Darwin研究小组在研究多序列比对问题时以实现点覆盖问题算法[1]为主要求解手段。

论文以典型的可重构阵列瑕点覆盖(Min-FCRA)问题[6]为实例，研究参数计算中有界搜索树和动态规划相结合的实现技术，论述了面向对象的算法实现方案，在经典图算法实现方面作了细致的研究，为参数算法真正走向实用提供桥梁。

2　典型算法及其实现技术

2.1　CVCB-Main算法

可重构阵列的瑕点覆盖(Min-FCRA)问题主要研究如何用备用行和备用列来修补VLSI中的出错元件问题。一个典型的可重构阵列包括一个$M\times N$的元件阵列和k_m个备用行，k_l个备用列，当元件阵列中出现出错元件时，我们用备用行或备用列来替代含有出错元件的行或列，从而对元件阵列进行修补。Min-FCRA问题很容易转化为二分图的受约束最小点覆盖问题：对一个$M\times N$的元件阵列A用k_m个备用行，k_l个备用列修复等价于对二分图$G=(U\cup L,E)$找一个最多k_m个U点和k_l个L点的点覆盖，其中$U=\{u_1,\cdots,u_m\}$与$L=\{v_1,\cdots,v_n\}$使得在u_i与v_j中有一条边当且仅当在阵列A中的第i行与第j列是有缺陷的。

文[6]结合经典匹配理论和参数计算技术，提出了解决Min-FCRA问题的CVCB-Main算法。同一般的固定参数算法一样，CVCB-Main算法分两步处理：首先化简问题核心，然后构造有界搜索树。

化简问题核心的目标是将原问题化简为一个规模更小且只依赖于固定参数k_m、k_l而与输入无关的一个“核心”问题。CVCB-Main算法的化简是分为两步进行的：首先删除二分图中所有度大于k_l的U点和度大于k_m的L点以及与这些顶点相关的边，接着应用Gallai-Edmonds结构定理[7]进一步缩减二分图为一个有完美匹配的二分图G'。

*)基金项目：国家自然科学基金(60373083)、长江学者奖励计划资助。张祖平　副教授，博士生，主要研究领域为算法理论、计算机网络与优化及大型数据库系统。周苗苗　研究生，研究方向：算法理论。

在构造有界搜索树之前，CVCB-Main 算法用 Dulmage-Mendelsohn 定理[7]将化简后得到的有完美匹配的二分图 G' 分解为基本块 $B_1,\cdots,B_r$，使得每个块边间联接一个低标注块的 U 点到一个高标注块的 L 点。接着是 CVCB-Main 算法的主体：构造有界搜索树，从概念上说就是用“二叉树”的形式来表示对各个块的覆盖轨迹：对某一基本块的覆盖有两个选择，覆盖其 U-部或者覆盖其 L-部，节点的两个分枝由此生成。所有的块都覆盖的节点是叶子节点，对应于图的最多 k_u 个 U 点和最多 k_l 个 L 点的点覆盖。显然树的计算量也是随节点的增多呈指数增长，为了减少树的节点数，算法总是首先覆盖权重较大的块，并且对于由某些权重较小的块组成的节点并不分枝，而是采用构造动态规划列表的方法直接找覆盖集，有效地缩短了有界搜索树的深度。

CVCB-Main 算法中将这些权重较小的块划分为若干种分枝情形（文[6]中的 T_6、T_7、T_8及 C_1、C_2、…、C_r 等），统称为有限分枝。有限分枝添加到点覆盖集 K 中的可能性都是简单且有限的，因此算法能够采用动态规划技术对这些有限分枝构造动态规划列表 L（文[6]的表1）。L 的行对应于有限分枝的一个情形，而0,1,2,…,k_u 作为列表的列名，表中的项 $L[i,j]$ 为 T（表示有覆盖）的充分必要条件为有一种方法在1,2,3,…,i 行中找到 j 个点到 K 中的 U-部分。在动态规划列表中如果最后的行有 T，则可以通过适当的数据结构记下最后一行 T 的形成，从而知道 K 中 U 部分的形成过程，由此 K 中的 L 部分也相应形成。如果列表中最后行列不为 T，则说明在 G 中找不到满足条件的约束点覆盖 K。

2.2 CVCB-Main 算法的实现

根据 CVCB-Main 算法的思想，我们用 Java 对其实现。程序主要包括4个模块，根据面向对象的思想，我们分别设计了对应的类封装其实现过程，对外只提供接口。

1. 主控模块（Min_FCRA 类） 其控制整个程序的流程。首先得到二分图的相关信息：二分图的大小、边数以及约束参数 k_u、k_l，这些参数由用户输入，系统提供简单的用户交互界面。边的位置由程序随机生成。二分图的存储采用邻接矩阵，二分图的 U 点对应矩阵的行，L 点则对应矩阵的列，用1来表示对应行和列有边。这样得到一个二维数组。

主控模块还负责对二分图作一些预处理：首先删除没有边与其他点相连的 U 点和 L 点，这些点不在点覆盖集中；然后根据算法的思想，删除二分图中所有度大于 k_l 的 U 点和度大于 k_u 的 L 点，这些点加入覆盖集中，最后得到缩减以后的存储数组 graph。然后依次调用下面的模块。

2. Gallai-Edmonds 化简模块（Gallai_ Edmonds 类） 其主要提供一个接口函数 Simplify，该函数根据 Gallai-Edmonds 结构定理来构造临界集 A[7]，独立集 D[7]，从而得到有完美匹配的二分图 $G(C)$。通过这个接口，向主控模块提供了一定程度的抽象功能，主控模块负责向该模块提供二分图的数组 graph，该模块最后只输出记录有完美匹配的二分图的数组 Cgraph 和临界集 A。临界集 A 是所有最小覆盖的交集，一定包含在最小覆盖集中。在构造 A 的过程中，一旦 A 中的 U 点数大于可以覆盖的 k_u 或者 L 点数大于可以覆盖的 k_l，函数 Simplify 返回 false，主控模块宣布找不到满足条件的点覆盖，不再进行下面的处理。

3. Dulmage-Mendelsohn 分解模块（Dulmage_Mendelsohn 类） 为了完成完美二分图的分解，此模块根据 Dulmage-Mendelsohn 分解[5]的基本思想，设计了以下几个函数：Search_ibEdge（根据完美匹配确定所有的块间边），Form_Block（将块间边去掉，根据点的连接情况，形成各基本块[6]），Form_ Bluster（根据块的连接关系划分为若干簇[6]）和 Block_Sort（对一个簇中的块排序，使得块间边联结低标号的块的 U-点到高标号的块的 L-点）。这一模块的输出是有序的块的队列，根据面向对象的设计方法，我们设计了 Block 的类来封装基本块，其成员变量包括：Degree（块的度）、uNOdes（块的 U 点集）、vNodes（块的 L 点集）、frongBlock（与块的 L-部相连的块）、backBlock（与块的 U-部相连的块）。

4. 搜索树模块（Search_Tree 类） 其是整个系统的主要部分，负责构造有界搜索树。当树中的节点属于有限分枝情形时，不再继续分枝，而是构造动态规划列表。最后输出找到的点覆盖或者宣布没有满足条件的点覆盖。这一模块的实现在下一节中详细说明。

2.3 典型算法关键部分的实现技术

1. 有界搜索树 构造有界搜索树的两个关键问题是定界和分枝规则。定界的目的是为了裁减不含可行解的分枝，减少搜索空间。对我们的搜索树，k_u，k_l 是搜索的限定条件，这里的 k_u，k_l 是每一个节点当前还可以覆盖的 U 点数和 L 点数。显然，这里“剪枝”的条件就是 $k_u<0$ 或者 $k_l<0$，即对于 $k_u<0$ 或者 $k_l<0$ 的节点从树上裁去。对于分枝的规则有两种选择，一个是广度优先，对全部节点沿宽度横向扫描，按照先生成先分枝的顺序处理；另一种是深度优先，总是对最晚生成的节点分枝，使搜索树纵向扩展。这两种分枝规则的平均效率是一样的，我们选用不需要回溯的广度优先方式进行分枝。

关键问题明确后，根据面向对象的思想，我们设计 TreeNode 类对应搜索树的节点，它的成员变量除了约束参数 k_u，k_l 和 UncoverBlock（还未覆盖的块），还包括 U_Block（已覆盖 U-部的块）和 L_Block（已经覆盖 L-部的块）。这样，当找到一个叶子节点（UncoverBlock 为空的节点）时，无须追溯其生成的过程，可以直接得到覆盖集。同时我们定义了 TreeNode 的下面两个成员函数完成分枝的过程，返回左右子节点。

```
//对本节点中的块 b 覆盖 U-部生成左子节点
    TreeNode CoverUpart (Block b);
//对本节点中的块 b 覆盖 L-部生成右子节点
    TreeNode CoverLpart (Block b);
```

采用广度优先的分枝原则，对于先生成的节点先分枝，我们用一个简单的先进先出的队列 treeNodeQueue 来记录树的中间节点。队列中的每一个元素是一个待分枝的 TreeNode 的实例。当队列不为空时，队列中的首节点出队，判断节点中块的组成连接情况属于文[6]的 $T_1\sim T_5$ 的那种情形，选取对应的块来覆盖而分枝；而对于不属于 $T_1\sim T_5$ 的有限分枝的节点则采用构造动态规划列表的方法来处理，在下一节将详细介绍其实现技术。图1是搜索树的构造流程的描述。

```
reeNodeQueue. add(Root);//Root 对应根节点
while(treeNodeQueue 不为空)
{ node=treeNodeQueue. getFirst();//取第一个节点;
  if(node 中的块不是有限分枝的情形)
  { 根据 node 中的块的情形选择要覆盖的块 b;
    //覆盖 b 的 U-部生成左子节点
    leftnode=node. CoverUpart(b);
    //左子节点满足定界条件
    if(leftnode. ku)=0)&(leftnode. kl)=0))
    { if(leftnode 是叶子节点)
        找到最小点覆盖,return(覆盖集 K);
      else
        treeNodeQueue. add(leftnode);//子节点入队}
    rightnode=node. CoverLpart(b)://生成右子节点
    同 leftnode 一样的处理……}
  else//对有限分枝情形
  { 生成动态规划列表;
    if 表中找到满足条件的覆盖集
```

```
    return(覆盖集 K);}
  treeNodeQueue.remove(node);//此结点出队}
return Null;//找不到满足条件的点覆盖
```

图1 搜索树的构造流程

2. *动态规划列表* 构造动态规划列表的首要问题是弄清这个表是如何动态逐行构造的。根据算法的描述，我们可以将表的构造过程用式(1)和(2)来描述：

$$L(1,j)=\sigma(j)\ (j=0,1,2,\cdots,k_u) \tag{1}$$

$$L(i,j+p)=L(i-1,j)\&\sigma(p)\ (j=0,1,2,\cdots,k_u;p=0,1,\cdots,k_u-j) \tag{2}$$

其中 $\sigma(j)$是第1个分枝添加 j 个 U 点到覆盖集的情形，$\sigma(p)$是第 i 个分支添加 p 个 U 点到覆盖集中的情形。只有在 $L(i-1,j)$和 $\sigma(p)$都为 T 的情况下，$L(i,j+p)$为 T。对于这些有限分支，能否添加 j 或 p 个点到覆盖集中很容易判断。因此，动态规划列表可以逐行构造。接下来的问题是如何记下最后一行中 T 的形成过程。在很多动态规划的算法中，用回溯法来寻找最优值的生成过程，而我们通过设计合适的数据结构来直接得到覆盖集，避免了回溯过程。所有的分枝都是由分解后的块(Block)组成的，因此每一个覆盖只要记下覆盖 U-部的块标号、覆盖 L-部的块标号，到最后由 Block 的 uNodes 和 vNodes 来得到覆盖的点。因此，我们构造了一个 TableNode 的对象对应表的每一项 $L[i,j]$，其成员变量包括：vNum(覆盖的 L-点数)，U_Block(覆盖 U-部的块)，L_Block(覆盖 L-部的块)和 bHasValue(该项是否为 T)。这样在动态规划列表的最后一行找到 bHasValue 为 T 的 TableNode，从其 U_Block 和 L_Block 中可以得到对应的覆盖集，从而避免了回溯，提高了效率。这时，式(2)的意义扩展了：$L(i-1,j)$和 $\sigma(p)$都为 T 的情况下，$L(i,j+p)$为 T，并且 $L(i-1,j)$与 $\sigma(p)$的覆盖集"相加"得到 $L(i,j+p)$的覆盖集。因此我们进一步定义了 TableNode 的 Add 方法：

TableNode Add (TableNode node);

该方法完成两个 TableNode 的实例的 U_Block 和 L_Block 合并，以及 vNum 相加而得到一个新的 TableNode 的实例，并且我们还将 $\sigma(p)$也对应生成一个 TableNode 的实例 $T[p]$，这样式(2)可以用如下语句实现：

```
for(j=0;j<=ku;j++)
for(p=0;p<=ku-j;p++)
  if((T[p].bHasValue) & (L[i-1,j].bHasValue)
     &(T[p].vNum+ L[i-1,j].vNum<=kl))
       L[i,p+j]=L[i-1,j].Add(T[p]);
```

另外，下面的处理使我们的程序更有效：

由于动态规划列表是逐行生成的，第 i 行的生成只需要表的第 $i-1$行，且结果直接从最后一行中得到，不需要回溯，因此我们不需要存储整个表，只使用了一个一维数组总是记录表的最后一行；

如果表的某一行中没有为 T 的项，停止构造表，宣布不可能找到满足条件的最小点覆盖。

总之，面向对象的设计和细节的处理使得我们的动态规划的实现过程无论在时间复杂度上还是空间复杂度上，都取得了较好性能。

3 实验结果及分析

对我们设计的可重构阵列的修复系统进行测试，有代表性的实验结果如表1所示。

表1 实验结果

阵列大小	缺陷数目	备用行	备用列	ku	kl	时间(ms)
500 * 500	300	50	50	不能覆盖		326
		100	100	77	91	357
600 * 600	400	60	60	不能覆盖		815
		120	120	104	92	687
700 * 700	500	80	80	不能覆盖		424
		150	150	121	139	843
800 * 800	600	120	120	不能覆盖		521
		180	180	179	151	1325

从实验结果来看，由于阵列的瑕点是随机生成的，往往较为分散，因此需要较多的备用行和备用列来修复。显然，随着阵列和备用行列的增大，修复所花的时间也越多。对于不能修复的情形，有两种可能：一种是 Gallai-Edmonds 化简后，临界集 A 中的 U 点数大于可以覆盖的 k_u 或者 L 点数大于可以覆盖的 k_l，这时可以直接断定不能修复；另一种则是构造完搜索树，判断所有叶节点都不满足约束条件，而得出不能修复的结论。显然，前一种得出结论所用的时间较少；而后者则由于构造了整个搜索树花费较多时间。总之不管哪种情形，我们的重构阵列修复系统还是相当高效实用的，其对800×800大小的阵列，用180×180的空闲行列修复或者得出不能修复的时间也不超过2秒。

结束语 很多实际应用的最优解问题都是 NP 难的，参数算法充分利用实际应用中参数的值较小这一特点，使得这些 NP 难的问题实际可解。但是，参数算法的时间复杂度仍然是指数级的，对这些算法设计高效的程序尤为重要。论文以一个典型的可重构阵列的瑕点覆盖算法为例，说明如何用面向对象的技术实现复杂参数算法。采用面向对象的思想，将算法的实现封装以便于实现的算法能更好地重用，如主控模块、搜索树模块(类)及动态规划模块(类)。下一步的目标是研究生物计算中序列比对(Sequences Alignment)及模体查找(Motif finding)等参数算法的具体实现，力求将有关图的参数算法通用化与实用化。

参 考 文 献

1 Chen J, Kanj L, Jia W. Vertex Cover: Further Observations and Further Improvements. Journal of Algorithms, 2001, 41: 280～301

2 Niedermeier R, Rossmanoth P. Upper bounds for vertex cover further improved. In: Proc. of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99), Lecture Notes in Computer Science, 1999, 1563: 561～570

3 Sze S-H, Chen J E. Find Specific motifs in DNA sequences via cliques in k-partite graphs[R]. Texas A&M University: Departments of ComputerScience and Biochemistry & Biophysics, 2003

4 Smith T, Waterman M. Identification of common molecular sequence. Journal of Molecular Biology, 1981, 147: 195～197

5 Chen J E, Kanj L A. Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms. Journal of Computer and System Sciences, 2003, 67: 833～847

6 张祖平，陈建二. 关于可重构阵列的最小瑕点覆盖算法的改进与分析. 计算机科学，2004，31(4)：184～188

7 Lovasz L, Plummer M D. Matching Theory. Annals of Discrete Mathematics, North-Holland, 1986. 29

8 Kuo S-Y, Fuchs W K. Fault diagnosis and spare allocation for yield enhancement in large reconfigurable PLAs. In: Intl. Test Conf. IEEE Computer Society Press, Sept. 1987. 944～953