ISSN1000-9825 Journal of Software 软件学报 2000,11(7):984~989

立体堆与分枝界限算法*

武继刚[1,2] 陈国良[1] 吴明[1]

[1](中国科学技术大学计算机科学技术系 合肥 230027)

[2](烟台大学计算机科学与工程系 烟台 264005)

E-mail: glchen@ustc.edu.cn

摘要 分枝界限算法是解决组合优化问题的常用方法之一.对于给定的问题和分枝策略,算法的运行时间取决于实现算法的数据结构.该文讨论了立体堆及其上的插入、删除算法;通过将分枝界限算法的运作过程与排序过程建立对应关系,给出了一般分枝界限算法的复杂度下界 $\Omega(m+h\log h)$,其中 m 为评估的结点数,h 为扩展的结点数;得出了立体堆为实现一般分枝界限算法的几乎最优数据结构;并对具体的作业分派问题实现了一个使用立体堆的分枝界限算法;提出了改善立体堆平衡性的措施.

关键词 分枝界限,组合搜索,算法,计算复杂度.

中图法分类号 TP301

分枝界限算法是解决人工智能、运筹学、组合数学中许多NP难题的常用技术.它由4条基本规则组合而成,分别为分枝规则、选择规则,淘汰规则与终止条件的测试.分枝规则用来完成在活结点集(liveset)中选出一个子问题以便继续扩展的工作;淘汰规则为利用问题的界限值完成删除无必要扩展的分枝;而终止条件测试则用来完成活结点是否为问题可行解乃至最优解的测试工作.对于具体的问题将导出具体的算法策略与规则.关于串行算法已有丰富的研究成果[1~4],近年来,这一算法在全局最优化等组合问题上的应用更是方兴未艾.自80年代至今又有更多的并行分枝界限算法出现,它们中的代表为文献[5~10].通过对分枝界限算法实现环节的分析可知,每种具体的分枝界限算法在具体实现时均需要一种合适的数据结构来快速实现其4条规则,特别是选择规则.这是因为在同一问题和同一策略的前提下,使用不同的数据结构来处理活结点集将导出不同的算法运行速度.基于这一点,文献[11]提出了称作立体堆的一种数据结构,它便于选择规则的执行,但仅仅粗略地给出了基本设计思想与算法.本文对其进行细化之后做了以下工作:给出了基于数据结构的一般分枝界限算法的复杂度下界;得出立体堆结构为实现一般分枝界限算法的几乎最优的数据结构;结合具体的作业调度问题,实现了一个使用此结构的具体的分枝界限算法;提出了改善立体堆平衡性(对应于多机处理的负载平衡)的措施.

1 立体堆及算法

堆(min-heap)是集合的一种抽象数据类型[12].它可用一棵二叉树来描述,其中的每个内结点均小于其子结点,堆的大小为二叉树的结点总数.这一结构在数据排序、最小生成树、作业调度以及其他一些网络问题最优化处理中都有广泛的应用.立体堆是堆结构的一种进化形式,可用如下定义来表达.

定义.设有 m 个堆 $H_1,H_2,\ldots,H_m$,将它们的堆顶元素也组织成一个堆 H_0,则由 $H_0,H_1,H_2,\ldots,H_m$ 组成的结构 H 称作立体堆,其中称 $H_1,H_2,\ldots,H_m$ 为原子堆,H_0 为堆顶堆.

图1给出了一个简单立体堆的逻辑结构,此结构可用数组这一简单而方便的形式给予实现.

* 本文研究得到教育部博士点基金(No.9703825)资助.作者武继刚,1963年生,副教授,主要研究领域为算法设计与分析,并行分布计算,计算智能.陈国良,1938年生,教授,博士生导师,主要研究领域为并行分布计算,计算机网络,神经计算.吴明,1973年生,博士,主要研究领域为并行分布计算,遗传算法.

本文通讯联系人:陈国良,合肥 230027,中国科学技术大学计算机科学技术系

本文 1999-04-01 收到原稿,1999-06-21 收到修改稿

立体堆上的基本运算主要为插入一个原子堆与删除一个立体堆上的堆顶元素,这两个基本运算的基本过程与复杂度如下.文中的对数函数均以2为底.

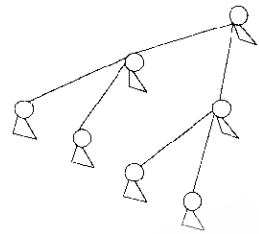

Fig. 1 Scheme of a simple cubeheap
图1 一个简单立体堆框架

· 插入一个原子堆

设数组 A 中存放的原子堆依次为 $H_1,H_2,\ldots,H_m,H_{m+1}$,而 H_{m+1} 为准备插入到堆顶堆 H_0 中的原子堆.首先将 H_{m+1} 的堆顶元素与堆尾元素的地址存入 $B[m+1].addr1$ 与 $B[m+1].addr2$ 中,再重新调整数组 B 的状态使其构成堆顶堆.这等同于在堆顶堆 H_0 中插入一个元素,使用已成熟的堆上的插入算法[13]可在 $O(\log\log m)$ 时间内完成这一插入工作.为使全文表达简洁且不失正确性,设 INSERT($C[i:j],a$)为一个将 a 插入大小为 $j-i+1$ 的堆 $C[i,j]$中的过程,其复杂度为 $O(\log\log(j-i+1))$.这一算法的具体实施与常用的在堆 $C[1:j-i+1]$上插入的算法的唯一区别在于事先作一个映射:$f(x)=x+i-1:[1,j-i+1]\to[i,j]$.则在 H_0 上插入一个原子堆的过程可记为 INSERT($A[B[1:m].addr1],A[B[m+1].addr1]$).

· 在立体堆上删除一个元素

首先对原子堆 $A[B[1].addr1:B[1].addr2]$进行删除,然后再将新的 $A[B[1].addr1]$在堆顶堆 H 上堆化,同样使用已成熟的在大小为 n 的堆上花费 $O(\log n)$时间的删除算法和堆化算法来实现.

设 DELETE($A[i:j]$)为对原子堆 $A[i,j]$进行删除堆顶元素的过程.当仅仅由于堆顶元素使得 $A[i:j]$不构成一个堆时,将堆顶元素沿其较小子结点的路径下降到适当的位置,这一过程记作 HEAPIFY($A[i:j]$),其实质(复杂度)等同于 DELETE($A[i:j]$).则在立体堆上删除一个堆顶元素的过程可形式化为:

```
PROCEDURE CUBEHEAPDEL(B[1:m])
BEGIN
    DELETE (A[B[1].addr1:B[1].addr2]);
    IF B[1].addr1≤B[1].addr2
    /* B[1]原子堆不空 */
        THEN HEAPIFY(B[1:m])
        ELSE BEGIN
                B[1]:=B[m],HEAPIFY(B[1:m-1])
                END
END.
```

令 $r=B[1].addr2-B[1].addr1$,对上述删除过程略作分析便知,在立体队上删除一个堆顶元素的复杂度为 $O(\log r)+O(\log m)-O(\log r+\log m)$.

2 分枝界限算法及复杂度下界

在状态空间树上,分枝界限搜索过程的搜索目标可理解成寻找从根结点到最优叶结点的一条可行路径.这种一般的分枝界限过程,用自然语言可简单描述如下.

(1) 将原问题看作一棵树的根结点,整个分枝界限算法生成的这棵树称作状态空间搜索树.按照分枝规则将原问题分解成若干个子问题,即根结点被分解成若干个子结点.

(2) 对这些子结点,根据当前的上界判定它是否为活结点,以便确定能否被进一步扩展.

(3) 在当前的所有活结点中选出一个最优结点($g(\cdot)$值最小的结点)进行继续扩展.

(4) 根据上下界限,判定生成的叶结点是否为所求的最优解.

整个状态空间搜索树上的所有结点被称作生成的结点,而内点则被称为被扩展的结点.设删除最小元的操作在一个称作 liveset 的表上实现,T 为状态空间树,根结点表示源问题.$f(\cdot)$为代价函数,而 $g(\cdot)$为下界的估价函数.输出的最优解为 Z.一般分枝界限算法的形式描述请见文献[14].

关于上述一般分枝界限算法,假定扩展一个结点花费常量时间,我们给出如下的定理来反映它的复杂度下界.

定理. 在状态空间搜索树中,设 m 为分枝界限算法找到一个最优解时所评估的结点(子问题)个数,h 为被扩展的结点个数,则任何串行分枝界限算法在找到第1个最优解时至少花费 $\Omega(m+h\log h)$ 时间.

证明:设 A 为任意的一个有效的串行分枝界限算法,并且在状态空间树中评估了 m 个结点而扩展了 h 个结点后找到了问题的一个最优解,显然,它必须至少花费 $\Omega(m)$ 时间来管理这 m 个结点.除此之外,它还要逐次选出 h 个结点 $n_1,n_2,\ldots,n_h$ 进行相继的扩展,这些结点在 $g(\cdot)$ 值上满足 $g(n_1)\leqslant g(n_2)\leqslant\ldots\leqslant g(n_h)$.这就是说,当算法 A 找到一个最优解时,$n_1,n_2,\ldots,n_h$ 按照 $g(\cdot)$ 已具有增序,由于排序 n 个元素的复杂度下界为 $\Omega(n\log n)$,从而这部分工作至少花费 $\Omega(h\log h)$ 时间,即整个算法 A 至少花费 $\Omega(m)+\Omega(h\log h)=\Omega(m+h\log h)$ 时间,由于 A 的任意性知,所证定理成立. □

3 立体堆在分枝界限算法中的应用

本节的算法复杂度分析依据文献[13,15]中的结果:构造一个大小为 n 的堆花费 $2n$ 次比较是足够的;在大小为 n 的堆上插入一个元素 $\log\log n+O(1)$ 次比较是足够的;在大小为 n 的堆上删除堆顶元 $\log n+\log\log n+O(1)$ 次比较是足够的.

假设一个分枝界限算法在找到第1个最优解时共生成了 m 个结点而扩展了 h 个结点,其中每步扩展生成的结点个数依次为 $r_1,r_2,\ldots,r_h$,则 $\sum_{i=1}^{h}r_i=m$.令 $s_j=\sum_{i=1}^{j}r_i$ 如果简单地使用一个堆来管理这 m 个结点,使用堆上最快的插入、删除算法,第 i 步扩展需在当前大小为 s_{i-1} 的堆上删除最小元(堆顶元素)要花费 $\log s_{i-1}+\log\log s_{i-1}+O(1)$ 次比较,还要将此步生成的 r_i 个结点插入堆中,复杂度为 $\sum_{j=1}^{r_i}\log\log(s_{i-1}+j)$,从而整个过程的复杂度为

$$\sum_{i=1}^{h}\left(\log s_{i-1}+\log\log s_{i-1}+\sum_{j=1}^{r_i}\log\log(s_{i-1}+j)\right)=\sum_{i=1}^{h}(\log s_{i-1}+\log\log s_{i-1})+\sum_{i=1}^{m}\log\log i.$$

而 $\sum_{i=1}^{m}\log\log i>\sum_{i=m/2}^{m}\log\log i>(m/2)\log\log(m/2)$,单就此项对于较好的分枝策略($h=O(\log m)$),与其复杂度下界 $\Omega(m+h\log h)$ 相比,简单的堆结构不是实现分枝界限算法的最优数据结构.

我们将使用立体堆结构来实现分枝界限算法中的liveset.设对具体的某个问题,分枝界限算法每步最多生成 r 个活结点,r 的大小根据具体问题是可以事先确定的,最多为最大的分枝数.用立体堆来实现liveset的具体过程为:

(1) 算法的第 i 步扩展一个结点必生成若干个设为 $r_i(\leqslant r)$ 个子结点,将这 r_i 个子结点构成一个堆;

(2) 将这 r_i 个结点构成的堆插入当前的立体堆中,此时堆顶堆 H_0 的大小为 i;

(3) 在新的立体堆上删除根结点.根结点为当前 $g(\cdot)$ 值最小的结点,应作为下一步的扩展结点,因而在立体堆上删除.

第(1)步将 r_i 个结点建堆,花费的时间为 $1.625r_i+O(\log r_i\log^* r_i)$[15],第(2)步花 $\log\log i-O(1)$ 时间将原子堆插入立体堆,第(3)步最坏情况下花费 $\log r_i+\log\log r_i+\log i+\log\log i+O(1)$ 时间,其中 $\log r_i+\log\log r_i+O(1)$ 为调整最小元所在原子堆的并销,而 $\log i+\log\log i+O(1)$ 为调整当前堆顶堆的并销.这样,每扩展一步并选出下一个扩展结点所需时间为以上3步之和,共计为 $1.625r_i+\log r_i+\log i+\log\log r_i+\log\log i+O(\log r_i\log^* r_i)$.

使用立体堆的一般分枝界限算法可形式化为如下形式:

```
ALGORITHM CUB&B
BEGIN
    (1) Cubeheap:={root}
    (2) IF root is a solution
        THEN Z:=f(root) ELSE Z:=-∞
    (3) WHILE g(x)<Z DO
```

```
/* x is the root of cubeheap */
    (3.1) Delete x from cubeheap
    (3.2) FOR each child y of x DO
          /* 扩展结点 x */
          IF y is a solution and f(y)<Z
          THEN Z:=f(y)
    (3.3) Construct all child y with g(y)<Z
          of x into a Min-heap.
    (3.4) Insert the MIN heap into cubeheap
  ENDwhile
END.
```

若扩展h个结点便可找到一个最优解,则上述算法的while循环将被执行h次,由于$r=\max\{r_i\},1\leqslant i\leqslant n$,从而整个算法的复杂度为$T(m,h,r)$:

$$\begin{aligned}&\sum_{i=1}^{h}(1.625r_i+\log r_j+\log i+\log\log r_j+\log\log i+O(\log r_i\log^* r_i))\\&<2m+h\log r+h\log h+h\log\log h+h\log\log r-O(h)\\&=O(m+h\log r+h\log h)=O(m+h\log\max\{r,h\}).\end{aligned}$$

在具体应用中,不同的问题与不同的分枝界限策略将导出不同的m与h,即将导出不同的计算复杂度,但对众多的组合搜索问题,r是事先可确定的,并且$r\ll h$,即我们使用立体堆结构导出的一般分枝界限算法的复杂度可运行在$O(m-h\log h)$时间内,即达到了时间复杂度的下界,从而这一立体堆结构为实现分枝界限算法的几乎最优的数据结构.

4 实验结果

对于下述作业分派问题,我们实现了使用立体堆结构的分枝界限算法.

设有n个作业$J_1,J_2,\ldots,J_n$和n个机器$M_1,M_2,\ldots,M_n$.已知J_i被M_j加工的代价为$a_{ij},1\leqslant i,j\leqslant n$,并且一台机器只能加工一个作业,一个作业也只能在一台机器上加工.求一个作业分派使得完成n个作业花费的代价最小.设$(x_1,x_2,\ldots,x_n)$表示J_i被M_{x_i}加工,$i=1,2,\ldots,n$的一个作业分派,S为解空间,则该组合优化问题的目标函数为$f(x_1,x_2,\ldots,x_n)=\sum_{i=1}^{n}a_{ix_i}$,求一个作业分派使得$Z=\min\{f(x_1,x_2,\ldots,x_n)\},1\leqslant x_i\leqslant n,i=1,2,\ldots,n$.

对于这一具体问题,最大的分枝数r为n.我们对8×8到20×20的5种矩阵每种用两组随机矩阵(a_{ij})来验证我们的算法.使用的机器型号为Celeron 300A,具体实验结果见表1.其中$lowbound=m+h\log h$,$upbound=m+h\log h+h\log r+h\log\log h$,$expe$为算法花费的数据比较次数.从表1中的数据获知$lowbound<expe<upbound$,它表明本文给出的理论下界与算法的复杂度分析结果是正确的.从$expe/lowbound$栏的数据可以看出,对于该作业分派问题,算法的复杂度为$\beta(m+h\log h)$,其中$1.29<\beta<1.62$,符合我们的理论分析.

Table 1 Comparison, upper bound, and lower bound of algorithm CUB&B with cubeheap

表1 使用立体堆结构的分枝界限算法的数据比较次数与理论上界和下界

$n\times n$	h	m	*Lowbound*	*Expe*	*Upbound*	*Expe/Lowbound*
8×8	11	49	93	120	219	1.29
	129	324	1 356	1 808	2 712	1.33
12×12	737	2 271	9 641	14 815	17 808	1.53
	2 135	5 709	31 329	45 048	54 118	1.43
16×16	251 608	304 623	4 833 567	6 524 637	7 654 279	1.34
	12 818	17 501	196 953	283 635	329 816	1.44
18×18	184 518	296 414	3 617 738	5 060 717	5 759 332	1.39
	19 876	67 530	365 670	594 950	612 084	1.62
20×20	95 948	279 920	1 911 036	2 965 692	3 150 436	1.55
	17 760	23 494	289 894	399 226	473 228	1.37

在算法的具体实现中我们发现,原子堆的大小对某些问题随着算法的执行会逐渐变小,这就导致堆顶堆的不必要增大,也就会带来并行处理的负载不平衡.对此我们采用限制原子堆的大小的方案,即当此步生成的结点数未达到某个界限值时就不单独作为一个原子堆进行建堆与插入,设置缓冲区,等待下一步扩展的结果再决定是否构造原子堆.这一措施将对并行处理的负载不平衡有所改善,它也是我们对并行分枝界限算法进一步的研究工作之一.

5 结 语

在假定扩展一个结点花费常量时间的前提下,我们基于数据结构给出了一般分枝界限算法的复杂度下界.在讨论了立体堆之后,本文实现了一个使用立体堆的具体分枝界限算法.理论和实验结果均表明立体堆结构在复杂度量级上为实现分枝界限算法的一个最优数据结构,同时也揭示了我们给出的下界是紧的.由于本文提出的理论结果针对一般的分枝界限算法,从而它对求解组合优化问题的具体分枝界限算法具有普遍意义.

致谢 谢幸博士与庄泗华同学对本文的完成给予了极大的帮助,在此深表感谢.

参考文献

1 Kumar V, Kanal L. A general branch and bound formulation for under-standing and synthesizing And/Or tree search procedures. Artificial Intelligence, 1983,21(1-2):179~198

2 Nau D S, Kumar V, Kanal L. General branch and bound and it's relation to A^* and AO^*. Artificial Intelligence, 1984,23(1):29~58

3 Nguyen V T. Global optimization techniques for solving the general quadratic integer programming problem. Computational Optimization and Applications, 1998,10(2):149~163

4 Washburn A R. Branch and bound methods for a search problem. Naval Research Logostics, 1998,45(3):243~257

5 Wah B W, Ma E Y W. Manip…… a multicomputer architecture for solving combinatorial extremum search problems. IEEE Transactions on Computer, 1984,C-33(5):377~390

6 Yu C F, Wah B W. Efficient branch-and-bound algorithms on a two-level memory system. IEEE Transactions on Software Engineering, 1988,14(9):1342~1356

7 Quinn M. Analysis and implementation of branch-and-bound algorithms on hypercube multicomputer. IEEE Transactions on Computer, 1990,39(3):384~387

8 Kaklamanis C, Persiano G. Branch-and-Bound and backtrack search on mesh-connected array of processors. Mathematics Systems Theory, 1994,27(5):471~489

9 Yang M K, Das C R. Evaluation of a parallel branch-and-bound algorithm on a class of multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1994,5(1):74~86

10 Gendron B, Crainic T G. Parallel branch-and-bound algorithms: survey and synthesis. Operations Research, 1994,42(6):1042~1066

11 Wu Ji-gang. General data structure and algorithms for branch and bound search. Information Intelligence and Systems, 1996 IEEE International Conference on Systems, Man and Cybernetics. Beijing: International Academic Publishers, 1996. 1586~1588

12 Aho A V, Hopcroft J E, Ullman J D. The Design and Analysis of Computer Algorithm. Reading, MA: Addison-Wesley Publishing Company, 1974. 87~92, 172~176

13 Wu Ji-gang, Zhu Hong. The least basic operations on heap and improved heapsort. Journal of Computer Science and Technology, 1994,9(3):261~266

14 Chen Guo-liang. Design and Analysis of Parallel Algorithms. Beijing: High Educational Press, 1994
(陈国良. 并行算法设计与分析. 北京:高等教育出版社,1994)

15 Gonnet G H, Munro J I. Heaps on heaps. SIAM Journal of Computing, 1986,15(4):964~971

Cubeheap and Branch-and-Bound Algorithms

WU Ji-gang[1,2] CHEN Guo-liang[1] WU Ming[1]

[1](*Department of Computer Science and Technology University of Science and Technology of China Hefei* 230027)

[2](*Department Computer Science and Engineering Yantai University Yantai* 264005)

Abstract Branch-and-Bound (B&B) algorithm is one of the fundamental methods for combinatorial optimization problems. The running time is dominated by the data structure used to implement B&B algorithm for the given problem and the related branching strategy. In this paper, the data structure called Cubeheap and the related algorithms (INSERT and DELETE) are discussed. The lower bound $\Omega(m+h\log h)$ of the running time for general B&B algorithm is proposed by constructing the mapping between the B&B procedure and the sorting procedure, where m is the number of the evaluated nodes and h is the number of the expanded nodes. According to the lower bound, Cubeheap is the near-optimal data structure to implement the general B&B algorithm. The experimental results for a concrete combinatorial optimization problem, Job-assignment, are obtained by running the B&B algorithm with the Cubeheap. The method to improve the balance of the Cubeheap is also proposed.

Key words Branch-and-Bound, combinatorial search, algorithm, computational complexity.