A Nearly Fastest and Asymptotically Optimal General Parallel Branch-and-Bound Algorithm*

WU Ji-gang[1,2], JI Yong-chang[2], CHEN Guo-liang[2]

[1](Department of Computer Science, Yantai University, Yantai 264005, China);

[2](Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

E-mail: jgwu@mail.ustc.edu.cn; gangjiwu@263.net

Received July 13, 1999; accepted October 22, 1999

Abstract: Branch-and-Bound (B&B) is a problem-solving technique which is widely used for various problems encountered in operations research and combinatorial mathematics. In this paper, the lower bound of running time $\Omega(m/p+h\log p)$ (the base of all logarithms in this paper is 2) is presented for general parallel best-first B&B algorithms on shared memory systems, where p is the number of processors available, h is the number of expanded nodes, and m is the total number of active nodes in state-space tree. In addition, a new general parallel best-first B&B algorithm on PRAM-EREW is proposed by devising the shared memory into p cubeheaps. Theoretical analysis shows that it is the fastest algorithm for $h<p\cdot 2^p$, and it is asymptotically optimal in this type of general parallel B&B algorithms. Computational experiments are conducted to solve 0—r knapsack problem.

Key words: branch-and-bound; cubeheap; PRAM-EREW; parallel algorithm; combinatorial search

Branch-and-Bound (B&B) algorithm is one of the fundamental schemes for the combinatorial search problems. It has been widely applied to such NP-hard optimization problems as intelligent system design, integer programming, SAT problems, and theorem proving. Many researches concerning the theory and application of B&B have been reported in literature[1~11]. The situation in this area can be generalized as: research efforts for serial algorithms are primarily problem-oriented, and are still a hot spot[2~5]. On the other hand, in the 1980's, much work, devoted to the evaluation of the theoretical performance of parallel B&B algorithms, showed the speedup abnormality and formulated the relation between the number of expanded nodes due to parallel B&B algorithms and that due to serial B&B algorithms[6,7]. Currently, many new fruits have sprung up one after anoth-

* This project is supported by the Ph. D. Fund of the Ministry of Education of China under Grant No. 9703825 (国家教育部博士点基金). WU Ji-gang was born in 1963. He received the BS degree in computational mathematics from Lanzhou University in 1983, and studied some M. S. degree courses in Southeast University in 1984. He was assistant professor, lecturer successively in Lanzhou University from 1983 to 1993. He is an associate professor in Yantai University and a doctoral candidate in computer science and technology at University of Science and Technology of China. His current research interests include parallel computing, theorem of algorithm, computational geometry, and artificial intelligence. JI Yong-chang was born in 1971. He received the BS degree in computer science from Anhui University in 1993, the MS degree in parallel computing from University of Science and Technology of China in 1996, and the Ph. D. degree in 1998. His research interests include parallel computational models, design and analysis of algorithm. CHEN Guo-liang was born in 1938. He is a professor in high performance parallel computing at University of Science and Technology of China. His interests include parallel computing, computer architecture, and computational geometry.

er, for example, self-stabilizing distributed B&B algorithm[8], parameterized B&B strategy[9], central and distributed control schemes in a distributed environment[5,6], and load balancing schemes of multiprocessor[10,11].

Under the condition of the same problem and the same strategy, data structure is the crux of implementing B&B algorithm. Hash is an efficient technique to implement selection rule on average, but B&B algorithm is usually used to solve NP-hard problems. The number of the expanded subproblems increases so fast that there is not enough memory to implement Hash list in practice. Heap is an effective data structure for B&B algorithm, and more efficient data structure called cubeheap is presented to improved it. Moreover, the lower bound of running time $\Omega(m+h\log h)$ is presented for general serial B&B algorithm[12]. Based on these previous researches, general parallel B&B algorithm on PRAM-EREW is discussed in this paper. The algorithm discussed in this paper belongs to the first type of general parallel B&B algorithms according to Gendron & Crainic's classification, in which the best subproblem is selected and expanded further in parallel in each iteration. We present the lower bound of parallel running time for this type of general parallel B&B algorithm, and then devise a nearly fastest general parallel B&B algorithm based on the cubeheap structure. It is proved that the algorithm is also asymptotically optimal in cost on PRAM-EREW.

We use the following notations in this paper.

h: the number of expanded subproblems when the first optimal solution is found.

m: the total number of created subproblems when the first optimal solution is found.

r: the subproblem number of a (sub)problem.

p: the number of processors available.

PE_i: the ith processor.

$|\mathfrak{B}|$: the size of the data structure $\mathfrak{B}$.

In the following, Section 1 shows serial B&B algorithm and cubeheap structure, and describes parallel operations on cubeheap as the preparing knowledge of the paper. Section 2 designs and analyzes our parallel B&B algorithm, and gives the proof of its optimality by presenting the lower bound of parallel running time. Section 3 shows our experimental results for $0-r$ knapsack problem. The last section summarizes the research efforts.

1 Preliminaries

In essence, a general B&B algorithm can be viewed as an enumeration method for the optimization problem: $Z(\mathfrak{p})=\min_{x\in S}F(x)$, in which F is a real function, S is a subset of real vector space. Assuming $\mathfrak{p}$ could be solved by finitely enumerating the elements of $\mathfrak{p}$, B&B algorithm uses the four basic rules below for given problems. Branching Rule divides the feasible solution set S into subsets $S_1, S_2, \ldots, S_n$, in which $S=\bigcup_{i=1}^{n}S_i$ and $S_i\cap S_j=\varnothing$ for $i\neq j$. Let $\mathfrak{p}_i$ denote the optimization subproblem corresponding to S_i, $Z(\mathfrak{p}_i)$ be its optimal value, then, $Z(\mathfrak{p})=\min_{1\leqslant i\leqslant n}Z(\mathfrak{p}_i)$. Subsequent division proceeds recursively until every subproblem can be tackled easily. Selection Rule chooses the most promising subproblem for further branching, judging by the problem-specific lower bounds of subproblems under discussion. Elimination Rule recognizes and eliminates subproblems that cannot yield an optimal solution to the original problem. A subproblem Q can be eliminated, either when it has been solved, or there exists another subproblem R which dominates Q, i.e., $Z(R)\leqslant Z(Q)$. Notice that for the latter case, early recognition is attainable by comparing the lower bound of Q's and the upper bound of R's optimal value. Termination Rule determines whether a feasible solution has been optimal. Among the above, branching rule depends on specific problems, while selection rule and elimination rule rely on specific search strategies and data structure.

We model a B&B algorithm as a rooted tree T with a cost function $f(\cdot)$ over its leaves. The goal is to find

the least-cost leaf in T. The input of the algorithm is the root of the tree T. The other nodes with the evaluation function $g(\cdot)$ of the lower bound in the tree are generated on-line by the expansion procedure. Expanding a node results in generating its children. If w is a child of v then $g(v)<g(w)$. v is an active node if and only if $g(v)<z$. For describing and analyzing our algorithm briefly, we assume without loss of generality that expanding one node produces (at most) r active subnodes in the tree T, i.e., $m=r\times h$, and the time required to produce a subnode is upper-bounded by a known constant. The running time unit is the comparison between the $g(\cdot)$s of the subnodes in tree T. The general serial B&B procedure can be stated in the following algorithm[13,14] in which *root* indicates the original problem, $f(\cdot)$ is the cost function, $g(\cdot)$ is the evaluation function of lower bound as input and z is the optimal solution as output.

Algorithm GSB&B(T,f,g,z)
BEGIN
1. *liveset*:={*root*}; /* initial */
2. *root* is a solution node THEN $z:=f(root)$ ELSE $z:=\infty$ ENDIF;
3. WHILE *liveset* contains a node x with $g(x)<z$ DO
3.1 $x:=$ node in *liveset* with best $g(\cdot)$; /* select the best subproblem to expand */
3.2 Delete x from *liveset*;
3.3 FOR each child y of x DO
IF y is a solution and $f(y)<z$ THEN $z:=f(y)$ ENDIF;
IF y is not a leaf &$(g(y)<z)$ THEN add y to *liveset*; ENDIF
ENDFOR;
ENDWHILE;
END.

On the computational complexity of GSB&B, the following Theorem 1 and the more sufficient data structure cubeheap were proposed in our previous research[12]. We quote them here for readability of this paper.

Theorem 1. Every general serial B&B algorithm takes at least $\Omega(m-h\log h)$ comparisons to find the first optimal solution.

Definition 1. A min-heap is a binary tree with heap-property: it has the heap shape, and the minimum element is at the root in the first level. The size of a heap is the number of elements in it.

In this paper, a min-heap is called a heap in short. The problem of heap construction and heap operations has received considerable attention in many articles. In the parallel models of computation, optimal heap construction algorithms and optimal heap operation algorithms have also been developed[15~17]. In a heap of size n, the algorithms[16] with running time $O\left(\frac{n}{p}-\log n\right)$ for heap construction and $O\left(\frac{\log n}{p}+\log\log n\right)$ for insertion and deletion operations will be quoted in this paper. These algorithms achieve the best possible PRAM-EREW running time for any number of processors p.

Definition 2. Let $\Delta_1,\Delta_2,\ldots,\Delta_j$ be j heaps, the smallest elements of them be $\Delta_1(1),\Delta_2(1),\ldots,\Delta_j(1)$, respectively. $A[1:j]$ be an address array satisfying that $\Delta_{A[1]}(1),\Delta_{A[2]}(1),\ldots,\Delta_{A[j]}(1)$ also construct a heap called logic-heap. A cubeheap consists of A and $\{\Delta_1,\Delta_2,\ldots,\Delta_j\}$, denoted by $CH=(A,\{\Delta_1,\Delta_2,\ldots,\Delta_j\})$. Its size is $|CH|=\sum_{i=1}^{j}|\Delta_i|$, and each Δ_i is called atom-heap.

Cubeheap is a variant of, but different from, traditional heap. The unit of the insertion operation on cubeheap is an atom heap Δ. When $|\Delta|$ and $|\Delta_i|$ equal 1 for each i, the cubeheap degenerates to traditional heap. The serial insertion and deletion operations on cubeheap are discussed in detail in Ref. [12], but the similar discussions are omitted here. The parallel insertion and deletion operations on cubeheap are discussed as follows.

Parallel insertion operation on cubeheap CH is easily performed by using the following procedure PCUBINSERT(CH) only on logic-heap $A[1:j]$ in running time $O\left(\frac{\log j}{p}+\log\log j\right)$[16].

Procedure PCUBINSERT(CH)

/* insert an atom-heap Δ into cubeheap CH with p processors */

BEGIN

Insert $\Delta(1)$ into logic-heap $A[1:j]$ by using Pinotti's parallel insertion algorithm[16].

END.

Parallel deletion operation on cubeheap CH is performed by the following procedure PCUBDELETE(CH). In the procedure, the statement PDELETE(Δ) denotes the invocation of an optimal parallel heap deletion scheme[16], where Δ is an atom-heap with size r.

Procedure PCUBDELETE(CH)

/* delete the top element on cubeheap CH with p processors */

BEGIN

1. PDELETE($\Delta_{A[1]}$); /* deletion in the first atom-heap in parallel */

2. Heapify the logic-heap $A[1:j]$ with new $\Delta_{A[1]}(1)$ in parallel by using Pinotti's algorithm[16].

END.

The running time of the PCUBDELETE(CH) is the sum of that of two steps, i.e.,

$$O\left(\frac{\log r}{p}+\log\log r\right)+O\left(\frac{\log j}{p}+\log\log j\right)=O\left(\frac{\log j}{p}+\log\log j+\frac{\log r}{p}+\log\log r\right)$$

where j is the size of logic-heap.

2 Parallel Best-First B&B Algorithm and Its Running Time

The general parallel B&B algorithm discussed in this paper belongs to the first type of parallel B&B algorithm according to Gendron & Crainic's classification, in which the best subproblem is selected and expanded into r subproblems in parallel with p processors in each iteration. In this section, we call this type of parallel B&B algorithm the parallel best-first B&B algorithm.

2.1 Design of Parallel Best-first B&B Algorithm

Supposing there are p processors available $PE_1, PE_2, \ldots, PE_p$, we devise the shared memory into p cubeheaps $CH_1, CH_2, \ldots, CH_p$. This is different from the traditional ideas, which regard the shared memory as only one heap[1,3,8]. The best node up to the present is the node of minimal $g(\cdot)$ in the tree T. The parallel algorithm executes the following steps iteratively on the computing model PRAM-EREW.

· Each cubeheap CH_i contributes its local best b_i, and p processors select the global best in parallel from the p local bests $b_1, b_2, \ldots, b_p$.

· Assume the global best is b_j, $1\leqslant j\leqslant p$. p processors $PE_1, PE_2, \ldots, PE_p$ delete the global best b_j in the cubeheap CH_j in parallel.

· The p processors expand the global best b_j and produce (at most) r subnodes in parallel.

· The p processors construct the r subnodes into an atom-heap Δ in parallel.

· The p processors select the cubeheap with the smallest logic-heap, denoted as CH_s, from $CH_1, CH_2, \ldots, CH_p$ in parallel.

· The p processors insert the atom-heap Δ into the smallest cubeheap CH_s in parallel.

Suppose that the cubeheap $CH_i=\langle A_i, \{\Delta_1^i, \Delta_2^i, \ldots\}\rangle$ with the logic-heap of size l_i, i.e., $|A_i|=l_i$, $i=1, 2, \ldots, p$, $SL=\{l_1, l_2, \ldots, l_p\}$, and $SB=\{b_1, b_2, \ldots, b_p\}$, where $b_i=\Delta_{A_i[1]}^i(1)$ is the local best in CH_i for $1\leqslant i\leqslant$

p. Based on the idea above, we give the following formal description of the parallel best-first B&B algorithm, in which the PSELECT(S,a) is the invocation of the parallel selection scheme[19], selecting the smallest element a from the set S of size p in running time $O(\log p)$ with p processors.

INPUT: The tree T, cost function $f(\cdot)$ and evaluation function $g(\cdot)$.

OUTPUT: The optimal solution z.

Algorithm GPB&B(T,f,g,z)

BEGIN

1. $A_1[1]$:=the address of $g(root)$; $\Delta_1^1(1)$:=$g(root)$;
   /* $root$ indicates the original problem and the CH_1 is initialized */
   FOR j=2 TO p PARA-DO $A_j[1]$:=$\varnothing$;
   /* cubeheaps CH_2,CH_3,...,CH_p are initialized */
2. IF $root$ is a solution node THEN z:=$f(root)$ ELSE z:=∞;
3. WHILE cubeheaps contain a node x with $g(x)<z$ DO
   3.1 PSELECT(SB,j); /* p processors select the global best b_j from $\{b_1,b_2,\ldots,b_p\}$ in parallel */
   3.2 PCUBDELETE(CH_j); /* p processors delete the global best b_j from CH_j in parallel */
   3.3 FOR i=1 TO p PARA-DO
      /* p processors expand the global best b_j into (at most) r subnodes $y_1,y_2,\ldots,y_r$ in parallel */
      Create sets $S=\{y_1,y_2,\ldots,y_r\}$; $SF=\{f(y_1),f(y_2),\ldots,f(y_r)\}$;
      $SG=\{g(y_1),g(y_2),\ldots,g(y_r)\}$;
      ENDFOR;
   3.4 PSELECT(SF,j); /* p processors select good upper bound of z */
   3.5 IF $f(y_j)<z$ THEN z:=$f(y_j)$;
   3.6 Construct the r subnodes into an atom-heap Δ in parallel;
   3.7 PSELECT(SL,k);
      /* p processors select a cubeheap CH_k in parallel, whose logic-heap is the smallest */
      PCUBINSERT (CH_k,Δ);
      /* p processors insert atom-heap Δ into the smallest cubeheap CH_k */
   ENDWHILE

END.

2.2 Theoretical analysis of the GPB&B

First of all, we give the lower bound of running time stated in the following theorem for the parallel best-first B&B algorithm.

Theorem 2. Every parallel best-first B&B algorithm runs at least $\Omega(m/p+h\log p)$ time with p processors on shared memory system.

Proof. For this type of parallel algorithms, in the algorithm GSB&B (in Section 1), because the steps 3.1, 3.2 and 3.3 cannot be executed synchronously in single iteration, selecting the best node x from m active nodes ($m\geq p$)with p processors takes at least $\Omega(\log p)$ comparisons[19]. Since the WHILE-loop stops after being executed h times iteratively, $\Omega(h\log p)$ comparisons are necessary. On the other hand, the best case is that m active nodes are distributed evenly to the p processors, i.e., each processor processes about m/p active nodes in parallel. The $\Omega(m/p)$ computational time is necessary. Hence, $\Omega(m/p)+\Omega(h\log p)$, i.e., $\Omega(m/p+h\log p)$ running time is necessary for this type of parallel algorithms. The theorem is proved. □

Let's give the following analysis on the parallel running time of algorithm GPB&B. In the jth iteration, the PSELECT(SL,k) and the PINSERT(CH_k,Δ) in step 3.7 give assurance that the size of each logic-heap is no

more than $\lceil j/p \rceil$, i.e., the maximum in the set SL is less than $\lceil j/p \rceil$. Step 3.1 runs in $O(\log p)$ time. Step 3.2 runs in $O((1/p)\cdot\log(j/p)+\log\log(j/p)\mid(1/p)\cdot\log r+\log\log r)$ time according to the running time of parallel procedures PCUBDELETE in Section 1. In step 3.3, each processor can process at most $\lceil r/p \rceil$ subnodes, hence, step 3.3 runs in $O(r/p)$. Step 3.4 runs in $O(\log r)$ time. Step 3.5 runs in $O(1)$. Step 3.6 runs in $O(r/p+\log r)$ with p processors by using Pinotti's algorithm[16], and step 3.7 runs in $O(\log p+1/p\cdot\log(j/p)+\log\log(j/p))$. Summing up the analysis above, we obtain the following running time in jth iteration,

$$O\left(\frac{1}{p}\cdot\log\frac{j}{p}+\log\log\frac{j}{p}+\frac{r}{p}+\log r+\log p\right) \tag{1}$$

Let $T(m,h,r,p)$ and $C(m,h,r,p)$ be the running time and the cost of algorithm GPB&B, respectively. Because

$$\begin{aligned}&\sum_{j=1}^{h}\left(\frac{1}{p}\cdot\log\frac{j}{p}+\log\log\frac{j}{p}+\frac{r}{p}+\log r+\log p\right)\\&\leqslant\frac{h}{p}\cdot\log\frac{h}{p}+h\cdot\log\log\frac{h}{p}+\frac{r\cdot h}{p}+h\cdot\log p+h\cdot\log r\\&=\left(\frac{m}{p}+h\cdot\log p\right)+\left(\frac{h}{p}\cdot\log\frac{h}{p}+h\cdot\log\log\frac{h}{p}+h\cdot\log r\right),\end{aligned}$$

hence, the total running time of the PB&B algorithm is

$$T(m,h,r,p)=O\left(\frac{m}{p}+h\cdot\log p\right)+O\left(\frac{h}{p}\cdot\log\frac{h}{p}+h\cdot\log\log\frac{h}{p}+h\cdot\log r\right)$$

And

$$\frac{\frac{h}{p}\cdot\log\frac{h}{p}+h\cdot\log\log\frac{h}{p}+h\cdot\log r}{h\cdot\log p}=\frac{\log\frac{h}{p}}{p\log p}+\frac{\log\log\frac{h}{p}}{\log}+\frac{\log r}{\log p} \tag{2}$$

The first item of Eq. (2) is no more than 1 for $h<p^{p+1}$, the second item is no more than 1 for $h<p\cdot 2^p$, and the third item is a constant for given problem and given branching strategy. Conclusively, the value of Eq. (2) is no more than a constant $\frac{\log r}{\log p}+2$ for $h<\min\{p^{p+1},p\cdot 2^p\}$, i.e. $h<p\cdot 2^p$. Hence we obtain $T(m,h,r,p)=O\left(\frac{m}{p}+h\log p\right)$ for $h<p\cdot 2^p$. In other words, the parallel algorithm GPB&B is the fastest one for $h<p\cdot 2^p$ according to Theorem 2. For example, when we use $p=2^4$ processors, the GPB&B is the fastest algorithm if the number of expanded nodes is less than 2^{20}. This condition is easily satisfiable in practice.

On the other hand, the cost of algorithm GPB&B is

$$C(m,h,r,p)=p\times T(m,h,r,p)=O\left(m+h\cdot\log\frac{h}{p}+p\cdot h\cdot\log\left(r\cdot p\cdot\log\frac{h}{p}\right)\right)$$

Because the computational complexity of serial algorithm GSB&B is $O(m+h\log h)$, and $m+h\cdot\log\frac{h}{p}<m+h\log h$, we obtain

$$\begin{aligned}\frac{m+h\cdot\log\frac{h}{p}+p\cdot h\cdot\log\left(r\cdot p\cdot\log\frac{h}{p}\right)}{m+h\log h}&\leqslant 1+\frac{p\cdot h\cdot\log\left(r\cdot p\cdot\log\frac{h}{p}\right)}{m+h\log h}\\&=1+\frac{p\cdot\log\left(r\cdot p\cdot\log\frac{h}{p}\right)}{r+\log h}\end{aligned}$$

Since

$$\lim_{h\to\infty}\frac{p\cdot\log\left(r\cdot p\cdot\log\frac{h}{p}\right)}{r+\log h}=0$$

hence, the cost $C(m,h,r,p)$ is asymptotically optimal.

Summarizing the analysis above, we obtain the following theorem.

Theorem 3. The GPB&B is the fastest general parallel best-first B&B algorithm for $h<p\cdot 2^p$, and it is

asymptotically optimal.

3 Experimental Results

The computational experiments are conducted on Dawning-1000 Massively Parallel Processing system with 32 computing nodes in a 2D wormhole mesh. We use our algorithm GPB&B to solve the following combinatorial optimization problem called $0-r$ knapsack problem.

The $0-r$ knapsack problem can be formulated as:

$$\text{maximize} \sum_{i=1}^{n} c_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} a_i x_i \leqslant B$$

$$x_i \in \{0,1,\ldots,r\},\ i=1,2,\ldots,n$$

where c_i and a_i respectively denote the value and the weight of item i, and B denotes the capacity of the knapsack.

In our experiments, r is fixed to 7, a_i is an integer randomly generated in $[1,800]$, c_i is randomly generated and independent of a_i. The goal is to test the constant coefficient in theoretical running time $T(m,h,r,p)$ for different instances with variables p,n, and B. The following tables show the number of comparisons between $g(\cdot)$s that the GPB&B needed, and the ratios between the experimental results and the theoretical running time for randomly generated test problems in each case.

Table 1 The ratios for different p values ($B=7000$, $n=200$)

p	*lowbound*	$T'(m,h,r,p)$	$T'(m,h,r,p)/lowbound$	T_{test}	$T_{test}/T'(m,h,r,p)$
2	28 038	258 546	9.22	533 048	2.06
4	40 468	206 774	5.11	421 974	2.04
6	49 049	194 587	3.97	390 425	2.01
8	55 500	190 725	3.44	376 827	1.98
10	60 656	189 679	3.13	369 820	1.95
12	64 947	189 777	2.92	365 857	1.93
14	68 621	190 818	2.77	363 500	1.91
16	71 832	191 315	2.66	362 072	1.89
18	74 684	192 332	2.58	361 214	1.88
20	77 248	193 397	2.50	360 723	1.87

In the three tables, $lowbound=m/p+h\log p$, T_{test} is the experimental results for the number of comparisons between $g(\cdot)$s that the GPB&B needed.

$$T'(m,h,r,p)=\left\lceil \frac{m}{p}+h\cdot\log p+\frac{h}{p}\cdot\log\frac{h}{p}+h\cdot\log\log\frac{h}{p}+h\cdot\log r \right\rceil$$

which is the running time $T(m,h,r,p)$ with the constant coefficient 1.

Table 1 considers a fixed $0-r$ knapsack problem in which $n=200$, $B=7\ 000$. We employ our algorithm GPB&B to solve it with different numbers of processors from 2 upto 20. The number of expanded nodes h is 17 633 and the total number of active nodes m is 20 811 when the first optimal solution is found. In the fourth column, $\frac{T'(m,h,r,p)}{lowbound}<3$ for $p\geqslant 12$, which shows that the GPB&B algorithm runs in the $O(m/p+h\log p)$ time with constant coefficient 3, i.e. the GPB&B is the fastest algorithm for $h<p\cdot 2^p$ since $h=17\ 633<12\cdot 2^{12}\leqslant p\cdot 2^p$. In the sixth column, $\frac{T_{test}}{T'(m,h,r,p)}$ is no more than 2.1 and decreases with the increasing p, This result shows that the theoretical running time $T(m,h,r,p)$ in section 2 is correct, and with smaller coefficient. This result is also presented in Table 2 and Table 3.

Table 2 The ratios for different n values ($B=6000$, $p=8$)

n	h	m	$T'(m,h,r,p)$	T_{test}	$T_{test}/T'(m,h,r,p)$
20	77	86	621	1 122	1.81
40	72	80	576	1 039	1.80
60	45	87	345	602	1.74
80	483	577	4 473	8 553	1.91
100	744	2 923	7 309	13 591	1.86
120	1 934	3 604	19 369	37 433	1.93
140	3 600	6 705	37 014	71 922	1.94
160	728	2 671	7 121	13 279	1.86
180	28 197	31 947	309 619	613 060	1.98
200	8 057	9 308	84 737	166 894	1.97

Table 3 The ratios for different B values ($n=150$, $p=8$)

B	h	m	$T'(m,h,r,p)$	T_{test}	$T_{test}/T'(m,h,r,p)$
1 000	43	100	330	571	1.73
2 000	50	143	395	681	1.72
3 000	234	473	2 095	3 893	1.86
4 000	81	200	670	1 190	1.78
5 000	144	527	1 273	2 277	1.79
6 000	121	487	1 061	1 874	1.78
7 000	64	330	538	908	1.69
8 000	544	2 605	5 316	9 719	1.83
9 000	182	1 065	1 688	2 953	1.75
10 000	632	3 887	6 333	11 415	1.80

Table 2 considers the random instances for fixed $B=6\ 000$, $p=8$, and different numberss of items n from 20 upto 200, while Table 3 considers the random instances for fixed $n=150$, $p=8$, and different capacities of the knapsack B from 1 000 upto 10 000. In these two tables, different instances lead to different h and m, but the ratios are close to and less than 2. These results clearly show the correctness of our analysis in Section 2.

4 Conclusions

Branch-and-bound algorithm has been widely applied to NP-hard optimization problems. A B&B algorithm runs in different time lengths for different problems and different B&B strategies. But for a given problem and given B&B strategy, the running time of the B&B algorithm heavily depends on the data structure. In this paper, under the condition of given problem and the given strategy, the general parallel best-first B&B algorithm is discussed on shared memory system. The lower bound of parallel running time is presented for this type of parallel B&B algorithm. A new general parallel algorithm is proposed on PRAM-EREW by devising the shared memory into p cubeheaps which are more efficient data structure than traditional heap structure. Theoretical analysis shows that the algorithm GPB&B proposed in this paper is a near.y fastest one, and it is also asymptotically optimal. These conclusions are significant since h grows exponentially for NP-hard optimization problems. Some simulated computations are performed for 0-r knapsack problem which is a famous NP-hard optimization problem. The experimental results show the correctness of our analysis.

Acknowledgment The authors thank Wei Yuan for giving us some references and Wan Ying-yu for giving useful discussions on this paper.

References:

[1] Gendron, B., Crainic, T. G. Parallel branch-and-bound algorithms: survey and synthesis. Operations Research, 1994, 42(6): 1042~1065.

[2] Liao Ching-Jong. New node selection strategy in the branch-and-bound procedure. Computers & Operations Research,

1994,21(10):1095～1101.
[3] Cheng Chun-Hung. B&B clustering algorithm. IEEE Transactions on Systems, Man and Cybernetics, 1995,25(5):895～898.
[4] Wu Ji-gang. General data structure and algorithms for branch-and-bound search. In: Information Intelligence and Systems, the 1996 IEEE International Conference on Systems, Man and Cybernetics. Beijing: International Academic Publisher, 1996,2(4):1586～1588.
[5] Nguyen, V. T. Global optimization techniques for solving the general quadratic integer programming problem. Computational Optimization and Applications, 1998,10(2):149～163.
[6] Wah, B. W., Ma, E. Y. W. MANIP——a multicomputer architecture for solving combinatorial extremum search problems. IEEE Transactions on Computers, 1984,C-33(5):377～390.
[7] Lai, T. H., Sahni, S. Anomalies of parallel branch-and-bound algorithms. Communications of the ACM, 1984,27(6):594～602.
[8] Yahfoufi Nassir, Dowaji Salah. Self-stabilizing distributed branch-and-bound algorithm. In: International Phoenix Conference on Computers and Communications. Piscataway, NJ: IEEE Computer Society, 1996. 246～252.
[9] Jonsson, J., Shin, K. G. Parameterized branch-and-bound strategy for scheduling precedence-constrained jobs on a multiprocessor system. In: Proceedings of the International Conference on Parallel Processing. Piscataway, NJ: IEEE Computer Society, 1997. 158～165.
[10] Shinano, Y., Harada, K., Hirabayashi, R. Control schemes in a generalized utility for parallel branch-and-bound algorithms. In: Proceedings of the International Parallel Processing Symposium, IPPS. Los Alamitos, CA: IEEE Computer Society, 1997. 621～627.
[11] Mahapatra, N. R., Dutt, S. Adaptive quality equalizing: high-performance load balancing for parallel branch-and-bound across applications and computing systems. In: Proceedings of the International Parallel Processing Symposium, IPPS. Los Alamitos, CA: IEEE Computer Society, 1998. 796～800.
[12] Wu Ji-gang, Chen Guo-liang, Wu Ming. Cubeheap and branch-and-bound algorithms. Journal of Software, 2000,11(7):984～989 (in Chinese).
[13] Nau, D. S., Kumar, V., Kanal, L. General branch-and-bound and its relation to a A^* and AO^*. Artificial Intelligence, 1984,23:29～58.
[14] Lai, T. H., Sprague, A. Performance of parallel branch-and-bound algorithms. IEEE Transactions on Computers, 1985,C-34(10):962～964.
[15] Rao, V. N., Zhang, W. Building heaps in parallel. Information Processing Letters, 1991,37:355～358.
[16] Pinotti, M. C., Pucci, G. Parallel algorithms for priority queue operations. Theoretical Computer Science, 1995,148:171～180.
[17] Carlsson, S., Chen, J., Mattsson C. Heaps with bits. Theoretical Computer Science, 1996,164:1～12.
[18] 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.
[19] Chen Guo-liang. Design and Analysis of Parallel Algorithms. Beijing: Higher Education Press, 1994 (in Chinese).

附中文参考文献:

[12] 武继刚,陈国良,灵明.立体堆与分枝界限算法.软件学报,2000,11(7):984～989.
[19] 陈国良.并行算法设计与分析.北京:高等教育出版社,1994.

几乎最快与渐近最优的并行分枝界限算法

武继刚[1,2], 计永昶[2], 陈国良[2]

[1](烟台大学 计算机系,山东 烟台 264005);
[2](中国科学技术大学 计算机科学与工程系,安徽 合肥 230027)

摘要: 分枝界限算法是求解组合优化问题的技术之一,它被广泛地应用在埃运筹学与组合数学中.对共享存储的最优优先一般并行分枝界限算法给出了运行时间复杂度下界 $\Omega(m/p+h\log p)$,其中 p 为可用处理器数,h 为扩展的结点数,m 为状态空间中的活结点数.通过将共享存器设计成 p 个立体堆,提出了 PRAM-EREW 上一个新的一般并行分枝界限算法,理论上证明了对于 $h<p2^p$,该算法为最快且渐近最优的并行分枝界限算法.最后对0 r 背包问题给出了模拟实验结果.

关键词: 分枝界限;立体堆;PRAM-EREW;并行算法;组合搜索
中图法分类号: TP301 **文献标识码:** A