An Embedded Analysis Method for Complexities of Problems

Xiaodong Wang, Weibin Wu, and Daxin Zhu

***Abstract*—This paper suggests an embedded analysis method to estimate the complexities of problems. It is generally difficult to estimate tight lower bounds for many problems and algorithms. Traditionally, lower bounds are obtained either by reduction or by a direct analysis. In this paper, a new idea is presented for estimating the lower bounds of problems and algorithms. In conjunction with two algorithm design paradigms divide and conquer and incremental construction, we can derive good lower bounds from the lower bounds of the corresponding sub-problems.**

***Index Terms*—algorithm, computational complexity, lower bounds**

I. Introduction

The lower bounds of problems are generally obtained by the following two ways:

(1) By means of a direct argument on the height of the computation/decision tree[1];

(2) By means of transformations of the problem [1, 5].

In this paper, a new train of thought is introduced.

Suppose we want to estimate the lower bound of problem Q. We may first find another problem P such that problem Q is contained in problem P as its sub-problem when we solve problem P by a certain algorithm. If problem P exists, the lower bound of problem Q may be obtained by taking advantage of the relation between Q and P if the lower bound of problem P is known so far or it is easier to estimate.

We shall show the power of the technique to analyze the lower bound of a problem in this manner later. We are interested in the asymptotic analysis, i.e., the behavior of the algorithm as the input size approaches infinity. Since expected case analysis is usually harder to tackle, and moreover the probabilistic assumption is sometimes difficult to justify, our emphasis will only be placed on the worst case analysis.

Manuscript received January 10, 2011. This work was supported in part by the Natural Science Foundation of Fujian under Grant No.2009J01295 and the Haixi Project of Fujian under Grant No.A099.

Xiaodong Wang is with *College of Mathematics & Computer Science, Quanzhou Normal University, Quanzhou 362000, China* (phone: +86-595-22916878; fax: +86-595-22916879; e-mail: wangxiaodong@qztc.edu.cn).

Weibin Wu and Daxin Zhu are with *College of Mathematics & Computer Science, Quanzhou Normal University, Quanzhou 362000, China* (e-mail: dex@qztc.edu.cn).

In the following 4 sections we describe our new ideas on obtaining the lower bounds of problems.

In section 2 we describe the notation and definitions necessary for following discussions. In section 3 we provide a novel idea for embedding analysis with divide-and-conquer algorithm design paradigm. In section 4 another idea for estimating the lower bounds of problems in conjunction with algorithm design paradigm incremental construction is presented. Finally some concluding remarks are in section 5.

II. Notation and Definitions

Let $A(P)$ represent the set of all the algorithms for solving the problem P. If $A\in A(P)$ and the size of the problem P is n, the corresponding *ADT* (Algebraic Decision Tree) or *ACT* (Algebraic Computation Tree) is denoted by $PAT(n)$ and the height of $PAT(n)$ denoted by $H(PAT(n))$.

Definition 2.1 For any $A\in A(P)$, if there exists a real function $f_P(n)$ such that $H(PAT(n))=\Omega(f_P(n))$, then $f_P(n)$ is said to be an asymptotic lower bound of the problem P [1].

Definition 2.2 If $f_P(n)$ is an asymptotic lower bound of the problem P and there exists $A\in A(P)$ such that $H(PAT(n))=O(f_P(n))$, then A is said to be an asymptotically optimal algorithm to solve the problem P and $f_P(n)$ is called the asymptotic infimum of P [1].

Definition 2.3 If there exist a real function $f_P(n)$ and positive constants c and n_0 independent of $A\in A(P)$ such that for any $A\in A(P)$, $H(PAT(n))\ge cf_P(n)$, when $n\ge n_0$, then $f_P(n)$ is called a strong asymptotic lower bound of P [1].

Proposition 2.4 If problem P has a strong asymptotic infimum then it has an asymptotic infimum too, and the strong asymptotic infimum of P is the asymptotic infimum of P. Any strong asymptotically optimal algorithm to solve problem P is also asymptotically optimal[1].

Definition 2.5 For a problem of size n, we can partition the problem into a sub-problems, each of size n/b ($a\ge b>1$), and recursively solve each sub-problem and then combine the solutions of the a sub-problems to obtain the solution of the original problem. This type of problem solving techniques is called a (a,b) divide-and-conquer algorithm or an (a,b) *-DAC* for short.

In divide-and-conquer algorithms, the proto typical recurrence has the form

$$T(n) = aT(n/b) + d(n) \qquad (1.1)$$

where $a > 0$ and $b > 1$ are arbitrary real numbers, and $d(n) \geq 1$ is the driving function. Examples include the case $a = b = 2$, $d(n) = n$ (Mergesort [7]), and $a = 7$, $b = 2$, $d(n) = n^2$ (Strassen's matrix multiplication [1]). In the spirit of improving Strassen's algorithm, many similar recurrences have appeared in matrix multiplication, e.g., $a = 143640$, $b = 70$, $d(n) = n^2$ in an algorithm of Pan [8].

Recurrences arise naturally in computer science from the analysis of algorithms, combinatorial analysis. Techniques for solving recurrences are among the standard repertoire of algorithmic textbooks (e.g., [4, 6, 8]).

In an influential note, Bentley, Haken and Saxe proved a master theorem for the divide-and-conquer algorithms [3] under a fairly general hypothesis on $d(n)$.

The master recurrence can be generalized to

$$T(n) = \sum_{i=1}^{k} a_i T(n/b_i) + d(n) \qquad (1.2)$$

Wang and Fu [11, Theorem 3.5] gave an integral bound on solutions to a parametric form of (1.1) in which the constants a, b are now functions of n . Roura [9] and Verma [10] provide the general theorems for the multi-term recurrences (1.2).

The ideas of the above techniques for solving master recurrences will be useful in the following discussion.

III. Embedded Analysis with Divide-and-Conquer

Let problem Q be the merge step problem of the problem P if (a,b) -DAC algorithm is employed to solve problem P . For convenience, suppose a unit time is needed for solving problem P of size 1. If $T_P(n)$ denotes the total time spent by the (a,b) -DAC algorithm in solving problem P of size n and $M_Q(n)$ denotes the time spent in solving problem Q of size n , then $T_P(n)$ satisfies the following recursive equation:

$$T_P(n) = \begin{cases} 1 & n = 1 \\ aT_P(n/b) + M_Q(n) & n > 1 \end{cases} \qquad (3.1)$$

Without loss of generality, let $n = b^k$, then $k = \log_b n$.

Since $T_P(n/b^k) = T_P(1) = 1$, we can easily derive from recursive equation (3.1) that $T_P(n)$, the total time spent by the (a,b) -DAC algorithm is

$$T_P(n) = a^k \left(1 + \sum_{i=1}^{k} M_Q(b^i)/a^i \right) \qquad (3.2)$$

Let $f_P(n)$ be an asymptotic lower bound of P . Without loss of generality, assume that

$$f_P(n) \uparrow \infty \text{ when } n \uparrow \infty \qquad (3.3)$$

Set $k = \log_b n$, $g_P(k) = f_P(b^k)/a^k = f_P(n)/n^{\log_b a}$ and $\limsup g_P(k) = \alpha$, then α may be one of the following three cases:
(1) $\alpha = 0$, (2) $0 < \alpha < +\infty$, (3) $\alpha = +\infty$.

Now let us examine the efficiency of the (a,b) *-DAC* algorithm for solving problem P in the three cases respectively.

A. The results with $\alpha = 0$

When $\limsup_{k \to +\infty} g_P(k) = 0$, $\lim_{k \to +\infty} g_P(k) = 0$ due to $g_P(k) \geq 0$.

Theorem 3.1 Let $\lim_{n\to\infty} f_P(n)/n^{\log_b a} = 0$,then $f_P(n) = o(T_P(n))$.

Proof. It follows from (3.2) that

$$0 < \frac{f_P(n)}{T_P(n)} = \frac{f_P(n)}{a^k \sum_{i=0}^{k} M_Q(a^i)/b^i} \leq \frac{f_P(n)}{n^{\log_b a}} \to 0$$

When $n \to \infty$.

Thus $\lim_{n\to\infty} f_P(n)/T_P(n) = 0$.

That is $f_P(n) = o(T_P(n))$. ■

Theorem 3.1 indicates that if $\alpha = 0$ then any (a,b) *-DAC* algorithm to solve problem P cannot reach the order of $f_P(n)$. Therefore, if $f_P(n)$ is the asymptotic infimum of the problem P , then any (a,b) *-DAC* algorithm to solve the problem P is not optimal.

B. The results with $0 < \alpha < +\infty$

Theorem 3.2 Let $\limsup_{n\to\infty} f_P(n)/n^{\log_b a} = \alpha < +\infty$, then $T_P(n) = \Omega(f_P(n))$.

Proof. From $\limsup_{n\to\infty} f_P(n)/n^{\log_b a} = \alpha < +\infty$, we know that for a given e , $0 < e < \alpha$,

$f_P(n)/n^{\log_b a} \leq \alpha + e < 2\alpha$, when n is sufficiently large. It follows from (3.2) that

$$\frac{T_P(n)}{f_P(n)} = \frac{a^k \sum_{i=0}^{k} M_Q(a^i)/b^i}{f_P(n)} \geq \frac{n^{\log_b a}}{f_P(n)}$$

Therefore, $\frac{T_P(n)}{f_P(n)} = \frac{1}{2a} > 0$ when n is sufficiently large.

This means $T_P(n) = \Omega(f_P(n))$. ■

Theorem 3.2 indicates that $f_P(n)$ is an asymptotic lower bound of the (a,b) -*DAC* algorithm if $0 < a < +\infty$.

In other words, we can know nothing about the lower bound of the problem Q from the lower bound of the problem P if $f_P(n)$ is an asymptotic lower bound of P.

C. The results with $a = +\infty$

In order to obtain meaningful results we make a further assumption herein that

for $a \geq b > 1$, there exists an integer n_0 such that

$$d_P(n) = f_P(n) - af_P(n/b) > 0 \qquad (3.4)$$

where $n \geq n_0$.

In this case we have

$$g_P(k) - g_P(k-1) = \frac{f_P(b^k) - af_P(b^{k-1})}{a^k} = \frac{f_P(n) - af_P(n/b)}{n^{\log_b a}}$$

Therefore $+\infty = a = \lim_{k\to+\infty} \sup g_P(k) = \lim_{k\to+\infty} g_P(k)$.

Theorem 3.3 Let $f_P(n)$ be an asymptotic lower bound of the problem P and

$$d_P(n) > 0, \ \lim_{n\to\infty} f_P(n)/n^{\log_b a} = +\infty,$$

then $d_P(n)$ is an asymptotic lower bound of the problem Q, i.e., $M_Q(n) = \Omega(d_P(n))$.

Proof. Suppose there is an algorithm B to solve problem Q such that $M_Q(n) = o(d_P(n))$.

$T_P(n)$, the computing time required by the (a,b) -*DAC* algorithm employing algorithm B, is given by (3.2). Thus,

$$0 < \frac{T_P(n)}{f_P(n)} = \frac{a^k \sum_{i=0}^{k} M_Q(a^i)/b^i}{f_P(n)} = \frac{a^k \sum_{i=0}^{k} M_Q(a^i)/b^i}{f_P(b^k)/a^k} = \frac{X_k}{Y_k}$$

Since $\Delta X_k = X_k - X_{k-1} = M_Q(b^k)/a^k$

$$\Delta Y_k = Y_k - Y_{k-1} = \frac{f_P(b^k) - af_P(b^{k-1})}{a^k} = \frac{f_P(n) - af_P(n/b)}{n^{\log_b a}}$$

it follows from $d_P(n) > 0$ that $\Delta Y_k > 0$ when k is sufficiently large.

It follows from the assumption $\lim_{n\to\infty} f_P(n)/n^{\log_b a} = +\infty$ that $Y_k \to +\infty$ when $k \to +\infty$.

By Stolz's theorem [2] we conclude that

$$\lim_{k\to\infty} X_k / Y_k = \lim_{k\to\infty} \Delta X_k / \Delta Y_k$$

$$= \lim_{k\to\infty} \frac{M_Q(b^k)}{f_P(b^k) - af_P(b^{k-1})} = \lim_{n\to\infty} \frac{M_Q(n)}{d_P(n)} = 0.$$

Therefore, $\lim_{n\to\infty} \frac{T_P(n)}{f_P(n)} = 0$.

That means $T_P(n) = o(f_P(n))$.

This contradicts the fact that $f_P(n)$ is an asymptotic lower bound of the problem P.

Thus, for any algorithm to solve the problem Q we always have $M_Q(n) = \Omega(d_P(n))$, i.e., $d_P(n)$ is an asymptotic lower bound of Q. ■

Theorem 3.4 Under the assumption of Theorem 3.3, for a given (a,b) -*DAC* algorithm to solve the problem P, if $M_Q(n) = O(d_P(n))$, then

(1) in the (a,b) -*DAC* algorithm, the algorithm to solve the problem Q is asymptotically optimal and $d_P(n)$ is the asymptotic infimum of the problem Q.

(2) the (a,b) -*DAC* algorithm is asymptotically optimal and $f_P(n)$ is the asymptotic infimum of the problem P.

Proof. Conclusion (1) follows from Theorem 3.3 immediately. So, only conclusion (2) remains to be proved.

From $M_Q(n) = O(d_P(n))$ we know that there exist positive constants $c > 0$ and $n_0 > 0$ such that $M_Q(n) \leq cd_P(n)$ when $n \geq n_0$.

Set $i_0 = \lfloor \log_b n_0 \rfloor + 1$, then $b^i \geq n_0$ when $i \geq i_0$.

Thus $M_Q(b^i) \leq cd_P(b^i)$ when $i \geq i_0$.

Consequently, when $k \geq i_0$,

$$\sum_{i=0}^{k} M_Q(b^i)/a^i = \sum_{i=0}^{i_0-1} M_Q(b^i)/a^i + \sum_{i=i_0}^{k} M_Q(b^i)/a^i$$

$$\leq c_1 + c\sum_{i=i_0}^{k} d_P(b^i)/a^i$$

$$= c_1 + c\sum_{i=i_0}^{k}\left(\frac{f_P(b^i)}{a^i} - \frac{f_P(b^{i-1})}{a^{i-1}}\right)$$

$$= c_1 + c\left(\frac{f_P(b^k)}{a^k} - \frac{f_P(b^{i_0-1})}{a^{i_0-1}}\right)$$

$$\le c_2 + cf_P(b^k)/a^k$$

It follows from (3.2) that when $n \ge n_0$,

$$T_P(n) \le a^k(c_2 + cf_P(b^k)/a^k) = c_2 n^{\log_b a} + cf_P(n).$$

Noticing $n^{\log_b a} = o(f_P(n))$,

we have $T_P(n) = O(f_P(n))$.

Therefore, the (a,b) -*DAC* algorithm is asymptotically optimal and $f_P(n)$ is the asymptotic infimum of the problem P. ■

Especially, given $a \ge b > 1, b \ge \log_b a$ and $g \ge 1$, we have the following corollary.

Corollary 3.5 Let $n^b \log^g n$ be an asymptotic lower bound of the problem P and the problem Q be the merge step problem of the (a,b) -*DAC* algorithm to solve the problem P, then

(1) $h(n)$ is an asymptotic lower bound of problem Q;

(2) the algorithm to solve problem Q in $O(h(n))$ time is asymptotically optimal.

In this situation, $h(n)$ is the asymptotic infimum of Q; $n^b \log^g n$ is the asymptotic infimum of P and the corresponding (a,b) -*DAC* algorithm to solve the problem P is asymptotically optimal, where

$$h(n) = \begin{cases} n^b \log^g n & a < b^b \\ n^b \log^{g-1} n & a = b^b \end{cases}$$

Proof. Taking $f_P(n) = n^b \log^g n$ in Theorem 3.3 and Theorem 3.4 one can see immediately that $f_P(n) \uparrow +\infty$ and $\lim\limits_{n\to\infty} f_P(n)/n^{\log_b a} = \lim\limits_{n\to\infty} n^{b-\log_b a} \log^g n = +\infty$.

$$d_P(n) = f_P(n) - af_P(n/b)$$

$$= n^b \log^g n - a(n/b)^b \log^g (n/b)$$

$$= \left(1 - \frac{a}{b^b}\right) n^b \log^g n - \frac{a^g \log b}{b^b} n^b \log^{g-1} n + o(n^b \log^{g-1} n)$$

Thus, $d_P(n) > 0$ when n is sufficiently large. This yields that all of the conditions of Theorem 3.4 and Theorem 3.5 are satisfied. Noticing $d_P(n) = \Theta(h(n))$, we finish the proof from Theorem 3.3 and Theorem 3.4. ■

It is a more special case when $b = 1$ and $g = 1$ in Corollary 3.5.

In this case $a = b$.

Corollary 3.6 Let $n \log n$ be an asymptotic lower bound of problem P and the problem Q be the merge step problem of the (a,b) -*DAC* algorithm to solve the problem P, then

(1) n is an asymptotic lower bound of problem Q;

(2) the algorithm to solve the problem Q in $O(n)$ time is asymptotically optimal.

In this situation, the corresponding (a,b) -*DAC* algorithm to solve the problem P is asymptotically optimal; n is the asymptotic infimum of Q and $n \log n$ is the asymptotic infimum of P.

IV. The Incremental Construction

Now let us consider another type of algorithm, the so-called incremental construction. The main idea of the algorithm of this type is that the solution is constructed or computed in an iterative manner. For a problem $P(n)$ with size n, the solution of $P(1)$ is constructed first, and then the solution of $P(i+1)$ is recursively constructed from the solution of $P(i)$, $i = 1,2,\mathbf{L}, n-1$.

Let the problem $Q(n)$ connected with the problem $P(n)$ be the iterative computation of $P(n)$. Let $R_Q(n)$ be the time spent in solving the problem $Q(n)$ and $T_P(n)$ be the time spent in solving the problem $P(n)$ by the corresponding incremental algorithm, then $T_P(n)$ satisfies the following equation:

$$T_P(n) = \sum_{i=1}^{n-1} R_Q(i)$$

For this type of algorithm construction we have the following results.

Theorem 4.1 Let $f_P(n)$ be an asymptotic lower bound of $P(n)$ and $f_P(x)$ be a continued differentiable function on $(0,+\infty)$, and $f'_P(x) \uparrow +\infty$ when $x \uparrow +\infty$, then $f'_P(n)$ is an asymptotic lower bound of $Q(n)$.

Proof. It is easily seen from Lagrange's mid-value theorem and the properties of $f_P(x)$ that

$$f'_P(x+1) > f_P(x+1) - f_P(x) > f'_P(x) \qquad (4.1)$$

$$f_P(x+1) - f_P(x) > f'_P(x) > f_P(x) - f_P(x-1) \qquad (4.2)$$

if $x \in (1,+\infty)$.

If there is an algorithm to solve the problem $Q(n)$ such that $R_Q(n) = o(f'_P(n))$, the corresponding incremental construction algorithm to solve the problem $P(n)$ will spend a computing time of $T_P(n) = \sum_{i=1}^{n-1} R_Q(i)$.

Thus, $0 \le \frac{T_P(n)}{f_P(n)} = \frac{\sum_{i=1}^{n-1} R_Q(i)}{f_P(n)} = \frac{X_n}{Y_n}$

$$\Delta X_n = X_n - X_{n-1} = R_Q(n-1)$$

$$\Delta Y_n = f_P(n) - f_P(n-1) > f'_P(n-1)$$

Therefore, when n is sufficiently large, $\Delta Y_n > 0$ and $\lim_{n\to+\infty} Y_n = +\infty$.

It follows from Stolz's theorem [2] that

$$0 \le \lim_{n\to+\infty} X_n / Y_n = \lim_{n\to+\infty} \Delta X_n / \Delta Y_n$$

$$= \lim_{n\to+\infty} \frac{R_Q(n-1)}{f_P(n) - f_P(n-1)} \le \lim_{n\to+\infty} \frac{R_Q(n-1)}{f'_P(n-1)} = 0$$

Thus $\lim_{n\to+\infty} T_P(n) / f_P(n) = 0$, i.e., $T_P(n) = o(f_P(n))$.

This is a contradiction. Therefore $R_Q(n) = \Omega(f'_P(n))$.

This means that $f'_P(n)$ is an asymptotic lower bound of $Q(n)$. ■

Theorem 4.2 Under the assumption of Theorem 4.1, if there is an algorithm to solve the problem $Q(n)$ such that $R_Q(n) = O(f'_P(n))$, then

(1) the algorithm to solve the problem $Q(n)$ is asymptotically optimal and $f'_P(n)$ is the infimum of $Q(n)$;

(2) the corresponding incremental algorithm to solve the problem $P(n)$ is asymptotically optimal and $f_P(n)$ is the asymptotic infimum of $P(n)$.

Proof. The conclusion (1) follows from Theorem 4.1 immediately. So only the conclusion (2) remains to be proved. From $R_Q(n) = O(f'_P(n))$ we know that there exist positive constants $c > 0$ and $n_0 > 0$ such that $R_Q(n) \le cf'_P(n)$ when $n \ge n_0$. According to (4.2),

$$R_Q(n) \le c(f_P(n) - f_P(n-1)) \text{ when } n \ge n_0.$$

Thus, when n is sufficiently large,

$$T_P(n) = \sum_{i=1}^{n-1} R_Q(i) = \sum_{i=1}^{n_0-1} R_Q(i) + \sum_{i=n_0}^{n-1} R_Q(i)$$

$$\le c_1 + c\sum_{i=n_0}^{n-1} \left(f_P(i+1) - f_P(i)\right)$$

$$= c_1 + c\left(f_P(n) - f_P(n_0)\right) = O(f_P(n))$$

Therefore, the corresponding incremental algorithm is asymptotically optimal and $f_P(n)$ is the asymptotic infimum of $P(n)$. ■

Corollary 4.3 Let $b \ge 1$ and $g > 0$ and $n^b \log^g n$ be an asymptotic lower bound of $P(n)$, then $n^{b-1} \log^g n$ is an asymptotic lower bound of $Q(n)$.

V. Concluding Remarks

Lower bound problems are generally concerned in the field of design and analysis of algorithms. It is difficult however to estimate their tight lower bounds for many algorithms. Traditionally, lower bounds are obtained either by reduction or by a direct analysis. In this paper, a new idea is presented for estimating the lower bounds of problems.

In conjunction with two algorithm design paradigms divide and conquer and incremental construction, we can derive good lower bounds from the lower bounds of the corresponding sub-problems. This is a powerful tool for design and analysis of algorithms.

References

[1] A. V. Aho, Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts, 1974.

[2] T. M. Apostol, *Mathematical Analysis*, Addison-Wesley, Reading, Massachusetts, Second Edition, 1974.

[3] J. L. Bentley, D. Haken, and J. B. Saxe. A general method for solving divide-and-conquer recurrences. *ACM SIGACT News*, 12(1): 36–44, 1980.

[4] G. Brassard and P. Bratley. *Fundamentals of Algorithms*. Prentice-Hall, Inc, Englewood Cliffs, 1996.

[5] T. H. Corman, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, Cambridge, Massachusetts and New York, second edition, 2001.

[6] J. Kleinberg and ´ E. Tardos. *Algorithm Design*. Addison Wesley, Boston, 2005.

[7] G. S. Lueker. Some techniques for solving recurrences. *Computing Surveys*, 12(4): 419–436,1980.

[8] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, Reading, Mass., 1989.

[9] S. Roura. Improved master theorems for divide-and-conquer recurrences. *J. of the ACM*, 48(1): 170–205,2001.

[10] R. M. Verma. A general method and a master theorem for divide-and-conquer recurrences with applications. *J. of Algorithms*, 16(1): 67–79,1994.

[11] X. Wang and Q. Fu. A frame for general divide-and-conquer recurrences. *Infomation Processing Letters*, 59(1): 45–51, 1996.