一种根据决策树结合信息论的经典算法复杂度可能下界分析

周毅敏 李光耀
(同济大学电信学院 CAD 中心 上海 200080)

摘 要 计算机算法是电子计算机诞生时同时出现的产物。有时甚至认为算法比现代计算机出现得更早。为解决具体问题,出现了各种各样的算法。算法的时间复杂度是算法实现中最关心的问题之一。然而,面对一个问题,是否存在一个算法复杂度不可逾越的界限以及如何确定这个界限却不常作为一个值得研究的问题受到重视。针对这个问题,提出了一个基于决策树和信息论的分析方法来对一些经典算法建模并分析这些算法的时间复杂度可能达到的下界是什么,以及如何计算这个下界等。所提计算方法是真实可行的,对列出的一些经典算法是有效的,并能够应用到其它一些文中未列出的算法中。

关键词 算法,时间复杂度,决策树,信息论,下界
中图法分类号 TP301.5 **文献标识码** A

Analysis of Probable Lower Bound of Time Complexity of Some Classical Algorithms Based on Decision Tree and Information Theory

ZHOU Yi-min LI Guang-yao
(CAD Center of Electronics and Information College, Tongji University, Shanghai 200080, China)

Abstract Algorithms are concurrently born with electronic computers. Sometimes we even believe algorithms' birth is long before that of modern computers. Various algorithms were designed to solve miscellaneous concrete problems. Nevertheless, whether the time complexity of an algorithm's lower bound exists and if so how to determine that lower bound do not usually come to be a problem worth researching or being valued. A method of modeling and analyzing the time complexity's lower bound of some classical algorithms wase proposed based on decision model and information theory and how to calculate it was presented to solve the problem. The method provided is feasible and effective for the classical algorithms listed and is also applicable to those not listed we believe.

Keywords Algorithm, Time complexity, Decision tree, Information theory, Lower bound

1 引言

计算机最初是为了解决简单的数值计算问题,随着科学的发展,现在的计算机已被用于解决各种实际问题。和简单的数值计算不同,解决实际问题需要具体的步骤,即算法。用运行所需的时间以及所需的资源来衡量一个算法的优劣是再自然不过的事情,也是根本目的所在。针对一个问题,研究者们的精力大多集中在针对目前存在的算法,如何提出一个更优越的算法,以占用更少的资源或使用更短的时间。但对它是否存在一个不可比之更少的界限或者对于当今存在的算法是否确实还有提高的余地研究不多。本文希望通过运用决策树和信息论的方法,对某些算法进行具体的分析并最终计算出该算法可能的下界。于是,可能存在更有效的算法优于高于这个界的算法。

2 相关工作

数据挖掘领域中的决策树是一种树状结构,除叶子结点外,每个结点均执行一个决策操作,每个叶子结点代表一个类或分布。从根结点到叶子结点,经过的每条路径经历一系列决策操作(分类操作)。决策树是一种自上而下的分类过程。在数据挖掘领域,提出了各种基于决策树的算法,但其主要都集中在如何对样本集进行划分,即如何选择叶子结点以及如何高效地构建决策树的内部结点,以尽可能地减少数据挖掘的分类过程的决策次数[2,3]。然而,我们认为决策树并不是数据挖掘领域所特有的概念,事实上,即使在一般的程序逻辑中,决策也普遍存在。因此,可以将决策树算法中的概念推广到其它算法中。决策操作又通常是算法中的重要部分,占据了大量的运行时间。我们可以根据这些操作的数量对算法的时间复杂度作出估计。

3 相关理论

3.1 排序

排序算法显然是所有算法中最经典最普遍的算法之一。对 n 个元素来说,目前认为排序算法的最小时间界是 $O(n\log n)$。合并排序、堆排序在最坏情况下达到此界,快速排序在平均情况下达到此界。这些排序算法称为基于比较的排

周毅敏(1980—),男,博士生,主要研究方向为图形图像、虚拟现实、城市仿真,E-mail:1010080053@tongji.edu.cn;李光耀(1953—),男,博士生导师,主要研究方向为图形图像、虚拟现实、城市仿真。

序。基于比较的排序算法的时间复杂度下界是 $\Omega(n\log n)$[1]。这是根据决策树模型得到的。

为了简化问题，先假定有 n 个数，且这 n 个数是各不相同的。这个问题的输入是以某一顺序排列的 n 个数。经过计算机处理后得到这 n 个数的唯一的一个排列，即从小到大或从大到小的一个排列。由于 n 个数的任意顺序排列共有 $n!$ 种，因此，最初可能的情况数为 $n!$ 种。而在算法运行之后，只有原顺序的另一个唯一的排列，仅为唯一一种情况。由于在输入时各种情况出现的概率是相等的，根据香农的信息学理论，算法运行前后概率的变化量为 $n!$。算法所确定的信息量为 $-\log_2(n!)$ 比特。而根据 Stirling 公式，有 $\ln(n!)\approx n\ln(n)-n$，则有 $O(\log n!)=O(n\log n)$。

3.2 理论分析

对高级计算机语言来说，算法中能够对情况进行决策的操作就只有条件判断语句，如 if、while、for 的第三个表达式，switch-case 语句等。其它的任何操作，如算术操作等并没有起到对情况进行决策的作用。因此，我们可以认为真正对情况进行决策的操作就只有这些操作，如果能够计算出这些操作的数量范围，就可以知道这个问题的时间复杂度不会低于进行这些操作的时间复杂度，因为算法至少需要执行这么多的决策操作。我们知道从问题的根源出发，直到到达一个最终情况的叶结点，是一系列的判断操作（即作决策的操作）和其它操作，每个判断操作都是一个二叉分支（注意到 switch-case 语句虽然是多个分支，但经过编译以后仍然是二叉分支的结构），而进行其它操作则不会产生任何分支。这种情形就是但不完全是决策树模型，事实上是另一种二叉树，如图 1 中的情形所示。这样的一棵二叉树虽然不一定能够计算清楚从根结点到叶子结点进行了多少算数及其它操作（图中的蛇形线部分），但是可以实际计算出至少要经历多少个二叉分支的决策操作。

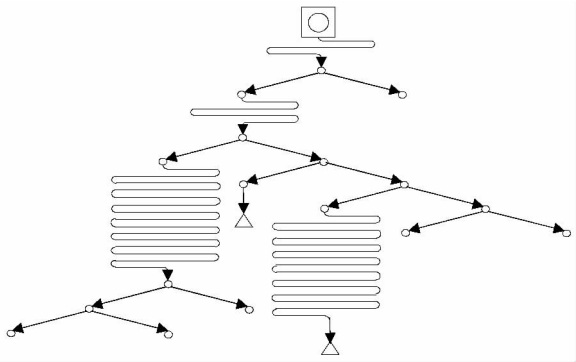

图 1 另一种二叉树的例子

如何计算出至少要执行多少这样的操作其实也不难。决策树相关的文献中已经有很多对此进行了估计[4]。如果我们知道问题最终分成的情况总数，即上述二叉树中的叶结点个数，根据二叉树的有关性质很容易推知，如果情况分割的集合数为 n，即叶结点的个数，则二叉树的高度应为

$$h_{\max}=O(\log n) \tag{1}$$

我们假定最终的情况分布为 $\Pi=(S_1,S_2,\cdots,S_n)$，并假定总的情况全集为 S，即 $\bigcup_{i=1}^{n}S_i=S$，$S_i\cap S_j=\emptyset(i\neq j)$，则 S_i 占总的情况数 S 的比率为 $p_i=|S_i|/|S|$，显然有 $\sum_{i=1}^{n}p_i=1$。

如果存在正数 $p_{\min}$ 和 $p_{\max}$，$\forall i\ p_{\min}\leqslant p_i\leqslant p_{\max}$，则有

$$np_{\min}\leqslant\sum_{i=1}^{n}p_i\leqslant np_{\max} \tag{2}$$

必有

$$1/p_{\max}\leqslant n\leqslant 1/p_{\min} \tag{3}$$

如果这里不是直接统计 n，而是关于 n 的函数 $f(n)$，同时 $p_{\min}$ 和 $p_{\max}$ 也都可以表示成 n 的函数即 $p_{\min}(n)$ 和 $p_{\max}(n)$，且都是递增的，则有

$$O(1/p_{\max})\leqslant O(f(n))\leqslant O(1/p_{\min}) \tag{4}$$

代入式(1)有

$$O(-\log\ p_{\max})\leqslant h_{\max}\leqslant O(-\log\ p_{\min}) \tag{5}$$

根据信息学理论[Elements of Information Theory]中的依概率确定的信息量公式为

$$I=-\log\ P \tag{6}$$

式中，I 为信息量，P 为事件发生的概率。其与这里的算法最大决策数所需的时间复杂度依事件概率的公式在形式上是一致的，与数据挖掘中决策树算法，如果的理论也是一致的。

由此可见，对于某个算法知道确定最终属于某种情况所需确定的信息量的范围就可以知道算法中所需要经历的决策数的范围，算法的时间复杂度就不会低于它。同样道理，如果算法所需确定的信息量已知，但是现有的算法的时间复杂度远高于此，则我们很有理由怀疑是否还存在比当前算法更为有效的算法。

4 对几个经典算法的分析

4.1 基于比较的排序

这个例子已在 3.1 节中给出，不再赘述。

4.2 n 个数的中位数

我们假定计算机所能确定的精度范围为 $[1,M]$，为计算方便，假定共输入 $n+1$ 个数，且 n 为偶数。如果当程序运行结束后，最终确定了中位数为其中的某一个，则可能的情况数 N 满足

$$1^{\frac{n}{2}}(M-2)^{\frac{n}{2}}+2^{\frac{n}{2}}(M-3)^{\frac{n}{2}}+\cdots+(M-2)^{\frac{n}{2}}1^{\frac{n}{2}}\leqslant N\leqslant 1^{\frac{n}{2}}M^{\frac{n}{2}}+2^{\frac{n}{2}}(M-1)^{\frac{n}{2}}+\cdots+M^{\frac{n}{2}}1^{\frac{n}{2}}$$

进行放缩后得到

$$(M-2)(\frac{M-2}{2})^n\leqslant N\leqslant(\frac{M}{2})^nM \tag{7}$$

则概率 P_N 满足

$$O((\frac{2M}{M-2})^{-n})=\frac{(M-2)^n}{2^nM^n}\leqslant P_N=\frac{N}{M^n}\leqslant\frac{M^{n+1}}{2^nM^n}=O(2^{-n})$$

因此，算法所需确定的信息量为

$$O(n\log(\frac{2M}{M-2}))=-\log(\frac{1}{O((\frac{2M}{M-2})^{-n})})\geqslant I_{P_N}\geqslant-\log(\frac{1}{O(2^{-n})})=O(n\log 2)$$

即 I_{P_N} 为 $O(n)$ 比特。根据理论分析知这个算法的时间复杂度比大于等于 $O(n)$。所以，如果使用排序的方法用 $O(n\log n)$ 的时间复杂度进行排序再取出中间数未免有些费时了。事实上，已经存在使用分治思想的 $O(n)$ 时间复杂度的中

位数算法[1]。同时，也可以知道，从渐进复杂度意义上来说，不会存在比它更有效的算法了。

4.3 平面上多点确定距离最近的两点

大平面内给定 n 个点确定其中相距最近的是哪两个点及这个最短距离。和 4.2 节中一样，假定计算机所能确定的精度为[1，M]，假定输入共有 n 个点且位置各不相同。首先在这个 $M\times M$ 的网格平面内放置 n 个点，两点间的最短距离的最大值可以事先算得。如图 2 所示，则最短距离的最大值为

$$D\leqslant\left\lceil\frac{M}{\sqrt{n}}\right\rceil \tag{8}$$

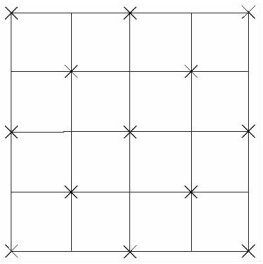

图 2 最短距离的最大值

用 N_d 表示当确定最短距离为 d 后可能的点的分布情况数，则在确定其中两点距离最短之后可能的情况数 N 满足

$$\begin{aligned}N&=\sum_{i=d}^{D}N_i\\&=\sum_{i=d}^{D}[(M^2)(M^2-p_{i,1})(M^2-p_{i,1})\cdots(M^2-p_{i,n-1})]\end{aligned} \tag{9}$$

式中，$p_{i,k}$ 表示在前 k 个点放置后，放置第 $k+1$ 个点时不可放置的位置数。显然有

$$p_{i,k}\leqslant 2(d_i+1)^2\leqslant(2D+1)^2=K \tag{10}$$

和

$$p_{i,k}\geqslant k \tag{11}$$

以及

$$N_d\leqslant 2+3+\cdots+D\leqslant D^2 \tag{12}$$

将式(8)代入式(9)得到

$$\begin{aligned}N&\geqslant\frac{M}{\sqrt{n}}(M^2)(M^2-K)(M^2-2K)\cdots(M^2-(n-1)K)\\&=\frac{M}{\sqrt{n}}(\frac{M^2}{K})(\frac{M^2}{K}-1)(\frac{M^2}{K}-2)\cdots(\frac{M^2}{K}-(n-1))\cdot K^n\\&\geqslant\frac{M}{\sqrt{n}}K^n=\frac{M}{\sqrt{n}}(\frac{4M^2}{n})^n\end{aligned}$$

同时有

$$N\leqslant D^2(M^2*\frac{n}{2})^n\leqslant\frac{M^2}{n}\cdot(M^2-V)^n \tag{13}$$

当 n 较大时，可以认为 V 为一较大固定值。

则概率 P_N 满足

$$O(\frac{1}{n}(\frac{M^2}{M^2-V})^{-n})\geqslant P_N\geqslant\frac{M}{\sqrt{n}}(\frac{4}{n})^n=O(n^{-\frac{1}{2}}(\frac{n}{4})^{-n})$$

因此，算法所需确定的信息量满足

$$\begin{aligned}O(n)&=-\log(\frac{1}{n}(\frac{M^2}{M^2-V})^{-n})\leqslant I_{P_N}\leqslant-\log(\frac{M}{\sqrt{n}}(\frac{4}{n})^n)\\&==O(n\log n)\end{aligned}$$

由此可见，$O(n^2)$ 时间复杂度的算法求出这两个距离最近的点未免有些费时了，事实上存在使用分治法思想的 $O(n)$ 时间复杂度的算法[6]。同时，也可以知道，从渐进复杂度意义上来说，不会存在比它更有效的算法了。

4.4 区间调度问题

给定一个时间区间及一组任务的执行时间，假定同一时刻只能执行一个任务，问这段时间区间内最多可以执行多少个任务，并确定任务的执行时间或者说选中执行的任务的开始时间和结束时间在所有任务中排在的位置。为了简化问题，我们假定所有任务的开始时间和结束时间各不相同。这时，简化的问题可以用图 3 表示(图中只列出了任务数分别为 1，2，3 的情况)，用 N_k^l 表示当任务总数为 k 时，最多可容纳的任务数为 l。

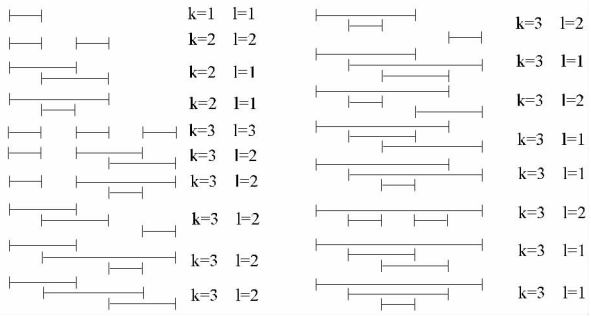

图 3 任务总数为 k 时最多可容纳的任务数

文中虽然没有给出任意给定 k，l 的 N_k^l 的求法，但给出了 N_k^1，N_k^k，N_k^{k-1} 的求法。首先给出当为 k 个任务时可能的情况数。根据组合数的原理，情况数为

$$\frac{\binom{2k}{2}\binom{2k-2}{2}\cdots\binom{2}{2}}{k!}=\frac{(2n)!}{2^n n!} \tag{14}$$

显然有 $N_k^k=1$。下面给出 N_k^1 的求法。为了构造可容纳任务数为 1 的情况，假定已经构造了 k 个任务，它们形成的最多可容纳任务数为 1。若添加 1 个新任务且形成的最多可容纳任务数仍为 1，则必有：它的开始时间必在之前 k 个任务的所有结束时间之前，它的结束时间必在之前 k 个任务的所有开始时间之后。由此可知，第 $k+1$ 至 $k+2$ 的时间区间即为这 $k+1$ 个任务的重叠时间。前 1 至 $k+1$ 个任务的时间点即为这些任务的开始时间，后 $k+2$ 至 $2(k+1)$ 个时间点即为这些任务的结束时间，总的组合数为 $(k+1)!$。对于 k 个任务是 $k!$，即 $N1_k=k!$。另外，根据从 N_{k-1}^{k-1} 加入一个新任务变为的 N_k^{k-1} 各种情况(加入的新任务长度为 1，则共 $k-1$ 种情况；加入新任务长度为 2，则共 $k-1$ 种情况；加入的新任务长度为 $i(4\leqslant i\leqslant k-1)$，有 $2k-i$ 种情况)可以得到 $N_k^{k-1}=2(k-1)^2$。

下面给出当根据问题的要求确定的最终结果时，所确定的情况数。

假定最终确定的结果为从给出的 n 个任务中确定了 k 个要执行的任务，它们的开始时间和结束时间排在所有任务中的相应位置为 i_1，j_1、i_2，j_2，$\cdots$，i_k，j_k。则开始时间位于结果中的第 1 个任务开始时间之前的所有任务的结束时间必须位于结果中的第 1 个任务的开始时间之后，这样的情况总数为 $(n-i_1)^{i_1-1}$。同样道理，开始时间位于第 1 个任务开始时间之后和第 2 个任务开始时间之前的所有任务的结束时间必须位于第 2 个任务的开始时间之后，这样的情况总数为 $(n-i_2)^{i_2-i_1-1}$。所有的可能情况的总数应该为

$$(n-i_1)^{i_1-1}(n-i_2)^{i_2-i_1-1}\cdots(n-i_k)^{i_k-i_{k-1}-1}\leqslant(n-1)^{n-1} \tag{15}$$

对于每一个最终结果的概率 P 满足

$$\frac{1}{\frac{(2n)!}{2^n n!}} \leqslant P \leqslant \frac{(n-1)^{n-1}}{\frac{(2n)!}{2^n n!}} \tag{16}$$

所确定的信息量 I 满足

$$O(n) = -\log\left(\frac{(n-1)^{n-1}}{\frac{(2n)!}{2^n n!}}\right) \leqslant I_P \leqslant -\log\frac{1}{\frac{(2n)!}{2^n n!}} = O(n\log n) \tag{17}$$

现有解决这个问题最好的算法是使用贪心思想的 $O(n\log n)$ 时间复杂度的算法。

4.5 单源最短路径

给定一个带权边有向图，求出从一个源结点到其它每一个结点的最短路径。为了简化问题，假定所有的边取值为正整数，边权值的最大上限为 M。同时我们假设除源结点外，任意两结点间均有双向边连接。源结点和其它结点间都有源结点出发的单向边。实际情况中若无某条边存在的情况可以假定该边的权值为 $+\infty$ 或为 M。只要 M 足够大是不影响我们求解问题的。由于在得到输入前，计算机是不能假设输入有任何限制条件的，因此它必须考虑所有的情况，然后从中确定出最终情况。假定输入顶点个数为 V，边个数为 E（显然有 $V-1 \leqslant E \leqslant V(V-1)$），则最终确定的结果必然为一棵这 V 个结点的生成树，且每种生成树的情况均有可能且必须出现在决策树最终的叶子结点上。根据文献[5]给出的 n 个结点共有 n^{n-2} 种生成树的结论以及我们在 3.2 节中的分析可以得到

$$h_{\max} \geqslant \log V^{v-1} = O(V\log V) \tag{18}$$

即算法的时间复杂度不会低于 $O(V\log V)$。目前解决这个问题最快的使用贪心思想的算法的时间复杂度为 $O(V\log E))$（根据我们的分析 $E \leqslant V(V-1)$，所以实际上就是 $O(V\log V)$）。因此，从渐进复杂度意义上来说，不会存在比它更有效的算法了。

4.6 带权区间调度

带权区间调度问题与区间调度问题的唯一区别就是每个任务赋予了一个权值，并且要使最终调度执行的任务总权值最大。

假设单个任务的权值最大值为 W。沿用 4.4 节中的记号，情况总数为

$$\frac{\binom{2k}{2}\binom{2k-2}{2}\cdots\binom{2}{2}W^n}{k!} = \frac{(2n)!W^n}{2^n n!} \tag{19}$$

假定最终确定的结果为从给出的 n 个任务中确定了 k 个要执行的任务，仍然沿用 4.4 节中的记号，所有可能情况的总数应该为

$$(n-i_1)^{i_1-1}(n-i_2)^{i_2-i_1-1}\cdots(n-i_k)^{i_k-i_{k-1}-1}W^{n-k} \leqslant (n-1)^{n-1}W^n \tag{20}$$

每一个最终结果的概率 P 满足

$$\frac{1}{\frac{(2n)!W^n}{2^n n!}} \leqslant P \leqslant \frac{(n-1)^{n-1}W^n}{\frac{(2n)!W^n}{2^n n!}} \tag{21}$$

由于一般情况中单个任务的最大权值 W 总是远小于 n，因此所确定的信息量 I_P 满足

$$O(n) = -\log\frac{(n-1)^{n-1}W^n}{\frac{(2n)!W^n}{2^n n!}} \leqslant I_P \leqslant -\log\frac{1}{\frac{(2n)!W^n}{2^n n!}} = O(n\log n + n\log W) = O(n\log n) \tag{22}$$

现有解决这个问题最好的算法是使用动态规划的 O(nlogn)时间复杂度的算法。

4.7 二分图匹配

给定一个二分图，要求出它的最大匹配。输入的总情况数可以这样计算，输入有 $2n$ 个顶点，共可能有 $\binom{n}{2}$ 条边，则所有可能的输入情况总数为 $2^{\binom{n}{2}}$。

假定 k 条边的最大匹配已经确定，则这 k 条边的两端的每个结点至多可以连接 $n-1$ 个结点。其余的结点每个至多可以连接 k 个结点。则这样的情况数之多为 $(n-1)^{2k}k^{2(n-k)} \leqslant (n-1)^{2n}$。每一个最终的结果的概率 P 满足

$$\frac{1}{22^{\binom{n}{2}}} \leqslant P \leqslant \frac{n-1^{2n}}{2^{\binom{n}{2}}} \tag{23}$$

$$O(n^2) = -\log\left(\frac{n-1^{2n}}{2^{\binom{n}{2}}}\right) \leqslant I_P \leqslant -\log\frac{1}{2^{\binom{n}{2}}} = O(n^2) \tag{24}$$

即算法的时间复杂度不会低于 $O(n^2)$。如果使用 $O(n^3)$ 的时间复杂度未免有些费时了。目前解决这个问题最快的是时间复杂度为 $O(n^{2.5})$ 的 Hopcroft-Karp 算法。

结束语　决策树模型和信息理论已经在数据挖掘领域被广泛应用，本文中试图将决策树模型推广到一般的程序中，并通过推导得出计算一般问题的可能时间复杂度下界的计算方法，对一些问题进行了相应的计算，得出相应决策树模型和信息理论已经在数据挖掘领域广泛应用，本文中试图将决策树模型推广到一般的程序中，并通过推导得出计算一般问题的可能时间复杂度下界的计算方法，对一些问题进行了相应的计算，得出相应的可能下界。从计算结果来看，这些问题的当今最快的算法或者已经达到了这个下界，或者已经非常接近这个下界。希望这种计算方法能应用到其它类似问题中并能给解决这些问题的算法的研究提供一定的参考。

参 考 文 献

[1] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to Algorithms(2nd Edition)[M]. Massachusetts: The MIT Press, 2002

[2] Garey M R. Optimal Binary Identification Procedures[J]. Siam Journal on Applied Mathematics, 1972, 23(7): 173-186

[3] Bellalal G, Bhavnani S K, Scott1 C. Extensions of Generalized Binary Search to Group Identification and Exponential Costs[C]// Advances in Neural Information Processing Systems, 2010: 1-9

[4] Buhrman H, de Wolf R. Complexity measures and decision tree complexity: a survey[J]. Theoretical Computer Science, 2002, 288: 21-43

[5] Prüfer H. Neuer Beweis eines Satzes über Permutationen[Z]. Arch. Math. Phys., 27: 742-744

[6] Kleinberg J, Tardos È. Algorithm Design [Z]. Boston: Pearson Education, 2006

[7] Cover T M, Thomas J A. Elements of Information Theory(2nd Edition)[M]. Hoboken, New Jersey: John Wiley & Sons Inc., 2006