维普资讯 http://www.cqvip.com

子集和问题的改进算法*)

李肯立　李庆华　张红君

(华中科技大学计算机科学与技术学院　武汉430074)

An Improved Algorithm for the Subset Sum Problem

LI Ken-Li　LI Qing-Hua　ZHANG Hong-Jun

(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract Due to the importance of the subset sum problem in cryptosystem, in the past two decade, much effort has been done in order to find techniques that could lead to practical algorithms with reasonable running time. Based on the two-list four-table algorithm, this paper proposes an improved algorithm for the solution of subset sum problem. To find a solution for the n-element subset problem, the proposed algorithm needs $O(2^{n/2-\varepsilon})$ memory, $1\leqslant\varepsilon\leqslant n/4$, and in $O(\varepsilon(2^{n/2}))$ time. The theoretic analysis and the computational experiments show that our proposed algorithm can solve the subset problem instances that can't be solved before, thus it is an improved result over the past researches.

Keywords Subset sum problem, NP-complete, Two-list four-table algorithm, Merkle-Helman cryptosystem

1. 导言

子集和问题可描述如下：给定 n 个正整数 $W=(w_1,w_2,\cdots,w_n)$和正整数 M，要求寻找这样一个子集 $I\subseteq\{1,2,\cdots,n\}$，使得 $\sum w_i=M, i\in I$。子集和问题属于 NP 完全问题[1]，直接的枚举搜索可能遍历问题的所有2^n 个解空间，即直接搜索最坏情况下的时间复杂性为 $O(2^n)$。由于求解该问题的指数时间复杂性，子集和问题在信息密码学领域和数论研究中具有极重要的应用[2,3]。因此，采用算法设计策略降低求解子集和问题过于庞大的计算量，具有重要的理论和实际意义。

分枝限界算法对于某些实例表现了较好的时间性能[5]，但其最坏情形的时间复杂性仍为 $O(2^n)$。1974年，Horowitz 和 Sahni[4]利用分治方法，提出了著名的二表算法(two-list algorithm)，算法的时间和空间复杂性被分别降至 $O(n2^{n/2})$和 $O(2^{n/2})$。Schroeppel 和 Shanir[5]于1981年将二表算法加以改进，利用4个子表动态产生两个排序表中子集和元素的方式(two-list four-table algorithm)，使算法的空间需求降为 $O(2^{n/4})$，时间需求则仍然保持为 $O(n2^{n/2})$不变。同时，Schroeppel 和 Shanir 在该文中提出了如下著名的猜想："在最多可忽略一个线性因子 n 的情形下，$T=O(2^{n/2})$是顺序求解该问题的时间下界"。因此，在这之后，随着并行处理技术的日臻成熟，人们将更多的注意转向子集和问题并行算法的研究[6~8]，而对该问题串行算法的进一步研究，能见诸文献的成果较少。最近，Cornuejols 和 Dawande 在文[9]中提出了一些由市场共享问题产生的子集和问题，对这些实例，尽管使用了包括二表算法在内的6种不同处理方法，仍然无法对其有效求解。文[10]则从理论上深入分析了这些子集和实例出现难解性的原因。

本文利用归并原理，基于 Schroeppel 和 Shanir[5]的算法的设计思想，从时间和空间性能上对二表算法作进一步的改进，提出一种 $T=O(\varepsilon(2^{n/2}))$、$S=O(2^{n/2-\varepsilon})$的改进算法，其中$1\leqslant\varepsilon\leqslant n/4$。这样，当 $\varepsilon=1$时，本文算法的时间复杂性为 $O(2^{n/2})$，只及二表算法和动态二表算法的$1/n$，而且 Schroeppel 和 Shanir[5]动态二表算法可成为本文算法当 $\varepsilon=n/4$时的一种特例。将本文算法和动态二表算法用于求解 Cornuejols-Dawande 难解实例的计算实验也表明，本文算法可以显著提高文[9]中使用二表算法求解子集和问题的实际性能，从而将可求解的子集和实例的规模扩大数倍。由于应用本文算法可以将破译的子集和密钥系统的维数 n 扩大到70以上，本文结论可望在信息密码学领域产生较大影响。

本文介绍作为本文算法基础的动态二表算法；提出改进算法；将本文算法用于求解 Cornuejols-Dawande 难解实例的数值实验；最后对本文进行简单总结和提出今后工作的研究方向。

2. 动态二表算法

由于二表算法对表 A、B 的搜索是单方向顺序进行的，因此只要能够得到每次搜索时的对应元素，存储器中没有必要保存表 A、B 的全部元素，Schroeppel 和 Shanir 正是基于这一思想，利用大小为 $O(2^{n/4})$的4个平衡子表动态生成表 A、B 子集和元素的方法，提出了下述的动态二表算法[5](two-list four-table algorithm)：

动态二表算法

Ⅰ、生成阶段：

1. 将向量 W 平均分成 W_{11}、W_{12}、W_{21}、W_{22}等4部分。
2. 依次产生4部分的全部子集和元素，得到4个平衡表 T_1、T_2、T_3、T_4。
3. 将表 T_2、T_4分别按升序和降序排序。
4. 将所有可能的序偶$(P_i,\text{first}(T_2))$，$P_i\in T_1$插入优先队列 Q_1中；将所有可能的序偶$(P_j,\text{first}(T_4))$，$P_j\in T_3$插入另一优先队列 Q_2中。

Ⅱ、搜索阶段：

*)本文得到国家自然科学基金(60273075)、国家"八六三"高技术研究发展计划(863-306 ZD-11-01-06)资助。李肯立　博士生，研究领域为并行处理、组合优化。李庆华　教授，博士生导师，研究领域为并行处理、组合算法。

维普资讯 http://www.cqvip.com

Repeat the following steps until either Q_1 or Q_2 becomes empty in which case print "unsolvable" and halt。

1. $(P_1,P_2)\leftarrow$ pair with smallest P_1+P_2 sum in Q_1; $(P_3,P_4)\leftarrow$ pair with largest P_3+P_4 sum in Q_2。
2. $S=(P_1+P_2)+(P_3+P_4)$。
3. if $S=M$ print "solvable" and stop。
4. if $S<P$ do
   delete $(P_1,P_2)$from Q_1
   if the successor P_2^1 of P_2 in T_2 is defined,
   insert (P_1,P_2^1)into Q_1。
5. if $S>P$ do
   delete $(P_3,P_4)$from Q_2。
   if the successor P_4^1 of P_4 in T_4 is defined,
   insert (P_3,P_4^1)into Q_2。

算法中2个优先队列的数据结构通过堆实现，其中队列Q_1用 min-堆实现，Q_2用 max-堆实现。由于在结点数为$2^{n/4}$的堆中做插入或删除操作需要$O(n/4)$的时间，而取出最小或最大元素可在$O(1)$的时间内完成，因此，Schroeppel 和 Shanir 的动态二表算法的时间复杂性为：$4O(2^{n/4})+2O(n/4\times 2^{n/4})+2O(n/4\times 2^{n/2})=O(n2^{n/2})$。虽然这一时间界和二表算法的时间界一样，但算法的实际运行时间应比二表算法快一倍以上，而空间需求则由二表算法的$O(2^{n/2})$降为保存4个子表所需的$O(2^{n/4})$。

3. 改进算法

从以下两个方面对上述算法做进一步改进。

首先，算法中表生成阶段步骤2和步骤3中对子表T_2、T_4中子集和元素的生成与排序可以采用下述归并方式一并完成。开始时得到自然排序表$[0,w_1]$，将w_2依次与表中两个元素相加得表$[w_2,w_1+w_2]$，然后用与两个表长度之和相同的时间对它们归并，得到一个含有4个元素的排序表，再将w_3分别与归并后排序表中各元素相加，再归并，…，最后便可得到所需的各含$2^{n/4}$个子集和元素的两个排序表T_2、T_4。按照这种方式，由k个分量$w_1,w_2,\cdots,w_k(1\leqslant k\leqslant n)$生成的含$2^k$个有序子集和元素的表所需要的时间为：

$$\sum_{0\leqslant i\leqslant k-1}O(3\times 2^i)=O(2^k) \tag{1}$$

显然，利用这种归并方式执行 Horowitz 和 Sahni 的二表算法，其时间复杂性可由原来的$O(n2^{n/2})$降为$O(2^{n/2})$，但空间复杂性则保持$O(2^{n/2})$不变。而以此方式执行 Schroeppel 和 Shanir 动态二表算法时虽然能使算法运行时间有所减少，但不至改变其时间和空间复杂性。

其次，为了降低算法对存储空间的需求，采用以时间换空间的算法设计策略来降低算法的空间复杂性。Schroeppel 和 Shanir 动态二表算法中使用4个平衡子表T_1、T_2、T_3、T_4生成2个排序表A、B中的元素，这4个子表中，表T_2、T_4必须是分别按升序和降序排序的有序表。可以将这样的4个子表设置成非平衡表，即子表$T_i(i=1,2,3,4)$中所含的子集和元素数不完全相同。具体来说，子表T_1和T_3分别包含由分量$w_1,w_2,\cdots,w_\varepsilon$和$w_{n/2+1},w_{n/2+2},\cdots,W_{n/2+\varepsilon}(1\leqslant\varepsilon\leqslant n/4)$所生成的$2^\varepsilon$个子集和元素，而另外两个子表$T_2$、$T_4$则分别包含剩下的$n/2-\varepsilon$个分量所生成的$2^{n/2-\varepsilon}$个有序子集和元素。这样，改进的子集和算法可描述如下。

改进算法

Ⅰ. 生成阶段：

1. 将向量W分成$W_{11}=(w_1,w_2,\cdots,w_\varepsilon)$、$W_{12}=(w_{\varepsilon+1},w_{\varepsilon+2},\cdots,w_{n/2})$、$W_{21}=(w_{n/2+1},w_{n/2+2},\cdots,w_{n/2+\varepsilon})$、$W_{22}=(w_{n/2+\varepsilon+1},\cdots,w_n)$等4部分，其中$1\leqslant\varepsilon\leqslant n/4$。
2. 生成表T_1、T_3的全部子集和元素。
3. 以归并方式分别生成按升序和降序排序的表T_2、T_4。
4. 将所有序偶$(P_i,\text{first}(T_2))$，$P_i\in T_1$插入优先队列Q_1中；将所有序偶$(P_j,\text{first}(T_4))$，$P_j\in T_3$插入另一优先队列Q_2中。

Ⅱ. 搜索阶段：

同动态二表算法。

算法实现过程中，优先队列Q_1、Q_2仍然使用 min-堆和 max-堆实现。显然，算法的空间需求为：$2O(2^\varepsilon)+2O(2^{n/2-\varepsilon})$，由于$1\leqslant\varepsilon\leqslant n/4$，因此改进算法的空间复杂性为$O(2^{n/2-\varepsilon})$。执行改进算法各主要步骤所需的时间分别为：表生成阶段中步骤2为$2\times 2^\varepsilon$；步骤3为$6\times 2^{n/2-\varepsilon}$；步骤4为$2\times 2^\varepsilon$；在表搜索阶段，每搜索一元素对需$\log(2^\varepsilon)=\varepsilon$的时间，这样搜索阶段的执行时间为$2\varepsilon\times 2^\varepsilon$。因此，执行改进算法所需的总时间为：$2+2^\varepsilon+6\times 2^{n/2-\varepsilon}+2\times 2^\varepsilon+2\varepsilon\times 2^{n/2}=O(\varepsilon 2^{n/2})$。当$\varepsilon$从1变化到$n/4$时，算法的时间和空间将相应地从$O(2^n)$和$O(2^n)$变化到$O(n2^n)$和$O(2^{n/4})$。因此，Schroeppel 和 Shanir 动态二表算法可成为改进算法在$\varepsilon=n/4$时的一种特例。

4. 改进算法的应用

在文[11]中，Cornuejols 和 Dawande 提出了著名的市场共享问题的可行性问题，这一可行性问题就是通常的多子集和问题，该问题的描述如下：给定一$m\times n$矩阵A和一m维列向量b，要求判断是否存在一n维二值向量$X=(x_1,x_2,\cdots,x_n)$，使得下式

$$\sum_{j=1}^{n}a_{ij}x_j=b_i,i=1,2,\cdots,m \tag{2}$$

成立。用$(na_{\max})^{i-1}$同时乘以上述约束中的第i个约束的两边，其中$a_{\max}=\max_j\{a_{ij}\}$，再将所得等式加到一起便成为前面所述的子集合问题的约束

$$\sum_{i=1}^{n}w_ix_i=M \tag{3}$$

已经证明：多子集和问题(2)有解的充分必要条件是子集和问题(3)有解，因此，求解多子集和问题实质上就变成求解子集和问题。文[9]所提出实例的参数值分别为：$m=\{3,4,5,6\}$，$n=10(m-1)$，a_{ij}是介于0到99的随机独立整数，$b_i=\lfloor\frac{1}{2}\sum_{j=1}^{n}a_{ij}\rfloor$，$i=1,2,\cdots,m$。文中使用了包括 Horowitz 和 Sahni 的二表算法[5]在内的6种方法，结果表明：由于运行这些算法的计算机在运行时间和空间上的限制，对这些实例的求解是十分困难的。

为了验证改进算法对原有二表算法在时间和空间性能上的改善程度，我们将改进算法和原来的二表算法进行了数值实验。实验中改进算法的参数ε首先取$\varepsilon=1$，这样，改进算法的执行时间将达到最小值$O(2^{n/2})$，但当待求解问题的实例规模较大，计算机对问题的求解受到存储空间限制时，ε则取成能使该实例顺利求解的值。我们不知文[9]中为何不使用比二表算法空间复杂性($O(2^n)$)更低的动态二表算法，因此，对ε

(下转第76页)

维普资讯 http://www.cqvip.com

容。

参考文献

1 Jia Weijia, Xuan D, Zhao W. Integrated Routing Algorithms for Anycast Messages. Appear in IEEE Communication Magazine

2 TAN Guo-Zhen, GAO Wen. Shortest Path Algorithm in Time-Dependent Networks. CHINESE J. COMPUTERS, 2002, 25(2)

3 FENG Jing, MA Xiao-Jun, GU Guan-Qun. Research of Network Model Adapt to QoS Routing Mechanism. CHINESE J. COMPUTERS, 2000, 123(8)

4 黄晓斌,李琦. Agent 技术在地理信息领域中的作用. 计算机科学, 2002, 29(9)

5 HE Xiao-Yan, FEI Xiang, LUO Jun-Zhou, WU Jie-Yi. A Scheme for QoS-Based Routing Using Genetic Algorithm in Internet. CHINESE J. COMPUTERS, 2000, 23(11)

6 WANG Ming-Zhong, Xie Jian-Ying, ZHANG Jing-Yuan. Strategy of Constructing Minimum Cost Multicast Routing Tree with Delay and Delay Variation Bounds. CHINESE J. COMPUTERS, 2002, 25(5)

7 顾尚杰,薛质编著. 计算机通信网基础. 电子工业出版社, 2000

(上接第17页)

$=n/4$时的改进算法(即动态二表算法)也进行了实验。实验结果如表1示。本实验中使用512M 内存、1.0G Intel Celeron CPU 的计算机,程序代码使用 Turbo C 编写,实验中规定程序可以使用的存储空间不超过256M,如果运行时间超过10000秒,则认为该实例无法求解,在表1中该实例下用符号"----"表示。从表1可见,无论 ε 取1还是 $n/4$,改进算法的综合性能都明显优于文[9]中二表算法的性能。应该指出的是,改进算法中 ε 可以根据实际计算环境进行调整这一特点对于求解该问题是十分重要的,因为当待求解实例的变量数较多时,囿于求解时间和存储空间的限制,在计算机上便无法对这些实例求解,如对表1中变量数为60的实例,二表算法和动态二表算法都不能对其求解,但改进算法则只需将 ε 置为7,便可在规定的时间界内求解此类实例。

表1 改进算法与原文算法的性能比较

问题		原文算法		改进算法(ε 可变)		改进算法($\varepsilon=n/4$)	
变量数	是否有解	空间	时间(秒)	空间	时间(秒)	空间	时间(秒)
30	否		0.21		0.02		0.10
30	否	512kB	0.22	512kB	0.03	6.40kB	0.11
30	否		0.23		0.03		0.12
40	否		10.86		0.82		5.41
40	否	24MB	10.94	24MB	0.87	60.1kB	5.64
40	是		8.08		0.79		4.76
50	否		—		78.43		398.7
50	是	768MB	—	256MB	37.46	322kB	104.9
50	是		—		52.52		217.4
60	否				4117		—
60	是	16GB	—	256MB	3465	2.42MB	—
60	是		—		2173		—

注:当变量数 $n=30$、40时 $\varepsilon=1$,$n=50$、60时,ε 分别等于4和7。

结论 本文基于动态二表算法,使用4个非平衡的子表动态产生二表算法中2个排序表中子集和元素的方法,提出一种求解子集合问题的改进算法。算法允许使用 $O(2^{n/2-\varepsilon})$ 的存储空间($1\leqslant\varepsilon\leqslant n/4$),在 $O(\varepsilon(2^{n/2}))$ 的时间内求解子集和问题。算法可视计算环境和问题实例的规模,通过调整 ε 在1到 $n/4$ 内的取值,有效求解此前不能求解的子集合实例。特别地,当 $\varepsilon=n/4$ 时,Schroeppel 和 Shanir 的动态二表算法[6]可成为本文算法的一种特例。若如 Schroeppel 和 Shanir[6]所断言:"$T=O(2^{n/2})$ 是顺序求解子集和问题的时间下界",那么本文算法将是求解此问题的最优算法。将本文算法与现有求解算法在理论和数值实验上进行的对比分析表明,本文算法明显改进了此前该领域的研究结果。

参考文献

1 Garey M R, Johnson D S. Computers and intractability: A guide to the theory of NP-Completeness. San Francisco: W. H. Freeman and Co, 1979

2 Chor B, Rivest R L. A knapsack-type public key cryptosystem based on arithmetic in finite fields. IEEE Trans. Inform. Theory, 1988, 34 (5): 901～909

3 Laih C-S, Lee, J-Y, Harn L, Su Y-K. Linearly shift knapsack public-key cryptosystem. IEEE J. Selected Areas Commun, 1989, 7 (4): 534～539

4 Horowitz E, Sahni S. Computing partitions with applications to the knapsack problem. J. ACM, 1974, 21(2): 272～292

5 Schroeppel R, Shamir A. A $T=O(2^{n/2})$, $S=O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM J. Comput, 1981, 10 (3): 456～464

6 Ferreira A G. A parallel time/hardware tradeoff $T\cdot H=O(2^{n/2})$ for the knapsack problem. IEEE Trans. Comput, 1991, 40(2): 221～225

7 Lou D C, Chang C C. A parallel two-list algorithm for the knapsack problem. Parallel Computing, 1997, 22: 1985～1996

8 Chang H K-C, Chen J J-R, Shyu S-J. A parallel algorithm for the knapsack problem using a generation and searching technique. Parallel Computing, 1994, 20(2): 233～243

9 Cornuejols G, Dawande M. A class of hard small 0-1 programs. 6th International IPCO Conference. Lectures Notes in Computer Science 1998, 1412: 284～293

10 Aardal K, Bixby R E, Hurkens C A J, Lenstra A K, et al. Market split and basis reduction: towards a solution of the Cornuejils-Dawande instances. In: 7th Intl. IPCO Conf. Lecture Notes in Computer Science, 1999, 1610: 1～16