最短加法链的随机幂树方法

江顺亮 许庆勇 黄 伟 叶发茂 徐少平
(南昌大学计算机系 南昌 330031)

摘 要 幂树法是求解最短加法链的一种简单近似方法,其计算效率高,一次可获得大量结果,但是精度偏低。随机幂树方法在扩展幂树时保持一层一层扩展,同时随机地扩展叶子结点,重复生成随机幂树并更新最优结果,在保持计算效率高的同时极大改善了计算精度。对于所有$n<24924$的数,通过9次重复生成随机幂树,准确率可达95%以上,平均达到97%,而且确保结果是次优结果。该方法在普通计算机上的求解规模可达155691199。

关键词 最短加法链,幂树法,随机化算法,近似算法
中图法分类号 TP301.6 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.3.047

Randomized Power Tree Method for Shortest Addition Chains

JIANG Shun-liang XU Qing-yong HUANG Wei YE Fa-mao XU Shao-ping
(Department of Computer Science, Nanchang University, Nanchang 330031, China)

Abstract The power tree method is a simple addition chain approximation method for shortest addition chains with its high computing efficiency and ability to generate multi-results by single running, and unfortunately its accuracy is poor. Randomized power tree method improves calculation accuracy significantly while maintaining high efficiency. The method extends nodes randomly one layer by one layer, and updates the better results by multi-running. For all numbers of n less than 24924, accuracy rate above 95% with an average of 97% was achieved by nine times running, while guaranteeing that the result is suboptimal. The solved problem is up to 155691199 scale by ordinary desktop PC.

Keywords Shortest addition chain, Power tree method, Randomized algorithms, Approximation algorithms

1 引言

加法链是一种有限数列,其中第一个数是1,后面的数是前面任意两个数之和,前面两个数可以是同一个数。某个数的最短加法链是能达到该数且链长最短的加法链。最短加法链与大指数幂乘的高效算法息息相关,从而在密码学中有着广泛的应用[1,2]。如果计算单个数的最短加法链,那么获得广泛公认的高效算法是基于有向无环图的方法[3],逐步深化回溯法[4]也是一个简单易实现的方法;如果同时计算大量的最短加法链,那么微软的Clift基于图论去除大量冗余,开发了一种高效的方法[5],并已经计算出小于2^{32}的所有数的最短加法链。由于最短加法链属于NP问题[6],求解近似解的方法比较多[6-10]。各种进化或演化的近似算法也开发出来了[8-10],为了减少已知结果的计算,数据库技术也用于最短加法链的具体应用中[2]。Knuth的幂树法是一种简单易实现的近似方法[6],它利用多叉树来存储和扩展加法链,每个数只在树中出现一次,但其精度不高,所以常常用于计算最短加法链的上界[4]。本文的随机幂树法在幂树生成过程中引入随机因素,少数几次运算即可极大改善幂树的精度,同时计算效率依然非常好。

2 幂树法的缺陷

最短加法链的幂树是按层序扩展的,最大特点是一个数只能在树中出现一次,树根到一个结点的结点链所代表的整数链就是叶子结点的加法链。但是扩展一个结点时可以有不同的扩展顺序,如图1所示。Knuth采用小数优先(见图1(a)),这样的扩展可以保证尽量多的结点获得最短加法链。

程序验证表明:对所有小于24924的数,小数优先扩展的结果可以获得85%的正确率,前5个不正确的数为77、154、233、293、308,不正确的结果中次优结果(只比最短加法链长1)比率高达99.9%,只有2个数的结果比最短加法链长2,这2个数是8719和17438。因此,幂树法获得的结果作为其它计算方法的上界还是不错的,但是85%的正确率比较低。如果采用大数优先的扩展方法,计算结果将急剧恶化,正确率只有7%,次优结果19%,因此仅仅改变扩展新结点的顺序,幂树法基本没有改进的空间,这也许是幂树法自Knuth提出后

到稿日期:2014-04-22 返修日期:2014-07-23 本文受2013中国国家自然科学基金(61363046),2012中国国家自然科学基金(41261091),2011中国国家自然科学基金(61163203)资助。

江顺亮(1965—),男,博士后,教授,博士生导师,主要研究方向为计算机模拟与仿真、算法设计与分析、人工智能,E-mail:jiangshunliang@ncu.edu.cn;**许庆勇**(1982—),男,博士生,讲师,主要研究方向为人工智能、机器视觉;**黄 伟**(1982—),博士,讲师,主要研究方向为人工智能、图像处理;**叶发茂**(1978—),男,博士,主要研究方向为数字图像处理、计算机图形学;**徐少平**(1976—),男,博士,副教授,主要研究方向为机器视觉、计算机图形学。

没见论文对它进行改进的重要原因。

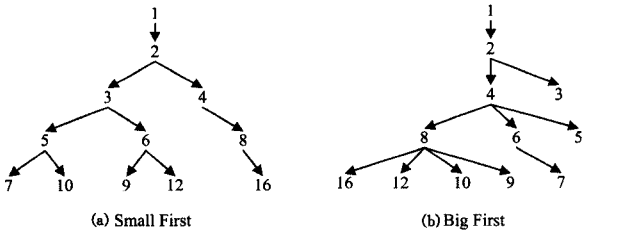

图 1　幂树的不同扩展顺序

3　随机幂树法

幂树法的正确率虽然有点低，但也达到 85%；如果能保持这种正确率并随机生成幂树，则只需重复运行几次，同时更新最优结果，便能快速提高正确率。如果幂树中的一个结点是否属于最短加法链是个纯概率事件，则根据概率推算，2 次运行其正确率可达 98%，3 次运行可达 99.7%。但这样的前提条件是生成的幂树是随机的且保持较高正确率。上节提到，扩展层的结点可按随机顺序扩展，但这样生成的幂树，不能保证正确率，因此要改变随机策略。

本文提出的随机化策略是：每次从扩展层随机选取一个点且尝试生成一个新结点，而不是把该结点能生成的结点全部计算出来；不论是否生成新结点，下一次再从扩展层随机选一个点尝试生成另一个新结点；重复这样的过程直至扩展层的所有结点都不能生成新结点，然后把所有新生成的结点作为新的扩展层的结点。生成的一个具体实例过程如图 2 所示，图中①、②表示的是旁边结点生成的顺序，从这些顺序可以看出扩展层的随机选取情况：结点 7 可以从上一层的结点 8、结点 5 和结点 6 中的任何一个结点生成，图 2 中是从结点 5 生成的，结点 6 可生成结点 12、结点 9、结点 7，图 2 中只生成结点 12 和结点 9。

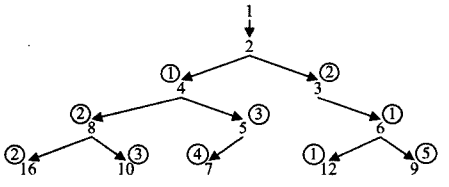

图 2　随机化生成幂树示意图

这样生成的幂树无疑随机性大大增强，同时正确率只有轻微的降低。因此这种随机化策略完全可以用于提高幂树的正确率，本文称之为全随机幂树法。另外一种随机化策略是随机选一个点之后，生成全部可生成的点，这种随机化策略的幂树法叫半随机幂树法。上节提到的小数优先扩展的幂树法，本文中叫“幂树法”或“knuth 的幂树法”。

3.1　全随机幂树法

为了阐述方便，定义两种文献中常用的加法链。一个加法链是一系列正整数：$a_0, a_1, a_2, a_3, \cdots, a_n$，其中 $a_0=1, a_k=a_i+a_j, k>i\geqslant j$。对于固定的 $n=a_m$，$l(n)=\min(m, a_m=n)$ 为 n 的最短加法链链长；如果限定 $a_k=a_{k-1}+a_j$，则 $l^*(n)=\min(m, a_m=n)$ 为 n 的星最短加法链链长。幂树法属于 $l^*(n)$ 星最短加法链，原因是 $i<k-1$ 的结点在以前层中已经扩展完了，所以每次扩展时 $i=k-1$。

为了简化全随机幂树法的生成过程，每个结点只记录幂树中的父结点编号，用一个数组（程序中用的是 STL 的 vector，下同）记录扩展层的结点和用另一个等长的数组记录扩展情况，用一个数组记录新生成的结点，用其它一个数组记录扩展层的扩展情况。扩展时要用到两个结点，一个是正在扩展的结点 a_{k-1}，它是随机选取的，另一个是从它到根结点的加法链上的点 a_j，第二个点的选取只需从扩展结点到根结点顺序选取。因此记录下选取的点，下次扩展时只需取其父结点即可。在选取第二个结点的环节上，没有必要随机选取；实验表明扩展时随机选取第一个点、顺序选取第二个点的效果已经非常理想了；如果第二个点的选取也采用随机选取，算法实施的复杂性大大增加，但这种随机性其实已经隐藏在第一个点的随机选取上，因为只要第一个点是随机选取的，新结点就可随机地出现在下一层可生成位置的任何位置；因此本文并没有探讨随机选取第二个点的方案。

3.2　全随机幂树的生成算法

全随机幂树的生成算法如下：

输入：正整数 n

输出：幂树，采用双亲存储法，用一个数组存储父结点，结点深度就是该结点的加法链链长，通过父结点回溯可以获得加法链。

(1)根结点初始化

a)1→扩展层结点 a_{k-1}

b)1→扩展用到的第二个结点 a_j

c)1→deep，幂树深度，即链长

(2)循环，直至扩展层没有结点

a)随机从扩展层取一个结点 a_{k-1}

b)取出该结点到根结点链上的结点 a_j

c)新结点 $a_{new}=a_{k-1}+a_j$

d)如果 $a_{new}\leqslant n$，且该数是幂树中没有出现的数

i. 记录 deep 作为 a_{new} 的加法链链长

ii. a_{new}→下一层的结点

iii. 记录 a_{k-1} 作为 a_{new} 的父结点

e)如果 a_j 没有父结点，则从扩展层删除 a_{k-1}，否则 a_j 的父结点作为 a_{k-1} 的下一个扩展的第二个结点，即下一个 a_j

f)如果扩展层没有结点

i. deep++，深度加 1

ii. 下一层的结点→扩展层，即扩展下移

g)循环，转至 2

(3)输出 n 的加法链及链长

多次生成随机幂树，当出现更好的结果时，需要更新和保留更好的结果，上面算法只需在步骤 2 d)的 i 中进行处理，由于比较简单，本文就不给出具体方法。

4　结果与分析

为了验证以上算法，用 C++编制了程序，开发工具为 CodeBlocks 13.2，操作系统是 win8.1，CPU 2.60GHz，内存为 4GB。程序的运行是利用 clock 函数获得程序计时部分的 CPU 时钟计时单元数，然后用常数 CLOCKS_PER_SEC 计算出以秒为单位的时间。

幂树的存储采用顺序存储的双亲表示法，具体用一维数组 parent 来存储幂树中父结点的数值。最短加法链的大量已知结果来源于互联网的公开结果（http://wwwhomes.uni-bielefeld.de/achim/addition_chain.html）。将计算性能与微软 Clift 公布的结果[5]进行比较，发现全随机幂树法的计算速

度是有优势的。我们也实现了带剪枝的回溯法和迭代加深回溯法[4]，用它可以计算单个数的最短加法链的最优结果，同时把全随机幂树法应用于它的上界估算，获得了较好的性能提升。为便于讨论，用 $S(n)$ 表示加法链的链长，$l(n)$ 表示最短加法链的链长。

4.1 全随机幂树法的随机化效果

选择 $n=2447$，检验随机化的效果。选择 $l(2447)=14$ 是因为它是比较难求解的一个数。用回溯法求解只能搜索到 4 个最短加法链的解，$S(n)=15$ 的解却有 7209 个。10 次独立运行生成全随机化幂树，获得 2447 的加法链结果如表 1 所列。10次运行结果完全相同的只有一次，即第 2 次和第 7 次求得的加法链完全相同。当然，这两次的幂树还是不同的，只是对于 $n=2447$ 的加法链相同。因此，达到了随机化的效果。

表 1　10 次运行获得 $n=2447$ 的不同加法链

运行次数	加法链：n=2447															
1	1	2	4	5	9	18	23	46	92	110	202	404	808	1616	2414	2447
2	1	2	4	8	16	32	64	65	129	193	386	515	1030	1159	2381	2447
3	1	2	4	8	16	32	64	128	129	193	322	515	837	1159	2381	2447
4	1	2	4	8	16	32	64	128	256	257	385	642	1027	2054	2439	2447
5	1	2	4	8	16	32	64	65	129	193	322	644	1288	1481	2125	2447
6	1	2	4	8	16	32	64	128	129	258	387	515	1030	2060	2447	
7	1	2	4	8	16	32	64	65	129	193	386	515	1030	1159	2318	2447
8	1	2	3	5	7	14	28	56	61	122	244	488	976	983	1471	2447
9	1	2	4	8	16	32	64	128	129	161	322	644	773	1546	2319	2447
10	1	2	4	8	16	32	33	49	82	115	230	460	509	969	1938	2447

4.2 全随机幂树法的正确率

现在的研究已经求出了 $n\leqslant 2^{64}$ 的所有最短加法链的结果，但可以公开获得的，作者只发现 $n\leqslant 24924$ 的全部结果（http:// www homes. uni-bielefeld. de/achim/add31. bits. bz2）。因此，针对 $n\leqslant 24924$ 的具体结果，可以考查全随机幂树所有结点的 $S(n)$ 及其正确率。

对于 $n\leqslant 24924$，进行 10 次独立的运算，全部结点的 $S(n)$ 结果如表 2 所列。计算表明，全随机幂树获得结点的最优结果 $l(n)$ 的比例最高，仅比 knuth 的幂树法略低；结果为 $l(n)+1$，即次优结果的比例比 knuth 的幂树法要略高；结果为 $l(n)+2$ 的比例与 knuth 的幂树法相比，增加较多，但占全部解的比例依然非常低。因此，可以认为随机幂树法只比 knuth 的幂树法性能略低。为了增加可信度，针对 $n\leqslant 24924$，进行了 50 次独立运行，平均结果为：84%为最优结果，16%为次优结果，0.02%比最优结果大 2（knuth 幂树的这 3 个数据是 85%、15%和 0.01%），更坏的结果没有出现。

表 2　$n\leqslant 24924$ 的全部结点的 $S(n)$

Run Case	l(n)	l(n)+1	l(n)+2	l(n)+3
1	21060	3859	5	0
2	20419	4496	9	0
3	20519	4388	17	0
4	20799	4115	10	0
5	20645	4275	4	0
6	21392	3528	4	0
7	21627	3295	2	0
8	21060	3859	5	0
9	21123	3797	4	0
10	20586	4332	6	0

4.3 全随机幂树法的重复运算

上面的计算，每次生成幂树是独立不相关的。如果每次运算时，对更好结果进行更新，多次运算后，可以很好地改善随机幂树的结果。这种对更好结果进行更新的多次运算，本文叫“更新运算”，与之对比，每次计算结果相互独立，叫“独立运算”。更新运算可逐步消去比次优结果更差的结果，完全消去比次优结果更差的结果只需要少数几次计算即可。需要的次数每次运行时都会不同，但变化不大，50 次多次重复生成幂树并更新结果，获得了需要次数的统计分布，结果为：平均 2.9，均方差 1.1，最大为 7。

4.4 全随机幂树法的次优结果

100 次进行更新运算（更新 10 次）的结果表明：更新 10 次的更新运算可以确信地消除比次优结果更差的结果，而且把最优结果的比例提升到 94%以上。由于比次优结果更差的结果可以较快地消去，因此只需考虑最优结果和次优结果。因此，重点考查更新 10 次的更新运算。100 次更新运算的次优结果如图 3 所示。结果显示：100 次运算中最差的是 5.6%的次优结果，而且只出现了一次，其它都低于 4.5%；最可能的结果是 2.0%～2.4%，平均 2.6%，这也意味着更新 10 次的更新运算可以获得平均 97.4%的最短加法链最优结果。

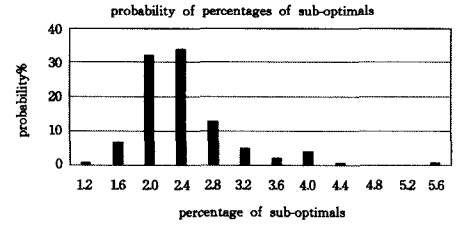

图 3　次优结果的比例（100 次更新运算）

毫无疑问，更新次数愈多，全随机幂树法获得最优结果的比率愈高，而且其它结果也是次优结果，即 $l(n)+1$，因此次优结果的比例愈低意味最优结果的比率愈高，因此主要讨论次优结果的比例。图 4 显示了随着更新次数的增多，次优结果逐步减少的趋势，由于图 4 中的曲线是 20 次运算的平均结果，从而曲线比较平滑。另外，图 4 中的 y 坐标是对数坐标，这种减少的趋势是非常明显的，次优结果的平均比例从独立运算的 15%，降到更新运算的 2.5%、1.3%、0.8%、0.6%和 0.5%（更新次数分别为 10、20、30、40 和 50），即使是 20 次运算中的最大值，这几个数分别为 4.5%、2.0%、1.3%、1.0%和 0.8%。因此可以说，全随机幂树法性能是非常稳定的。与之形成对比的是半随机幂树法，独立运算的次优结果高达

42%，更新运算的次优结果比例下降为 11%、6.8%、5.5%、4.4%和3.8%，因此半随机幂树法的性能远低于全随机幂树法。

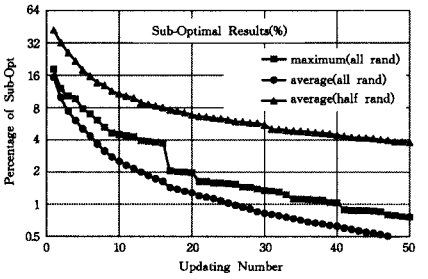

图 4　更新次数与次优结果的比例(其它是最优结果)

当全随机幂树法的更新次数超过 50 时，正确率提高非常缓慢而且有极限。24924 个数据中有 10 个不论更新计算多少次从没有获得过最优解 $l(n)$，其中最小的一个是 12509，其它 9 个为：13207、13705、15473、16537、20753、22955、23219、23447 和 24797。其实，这 10 个数有个共同点，它们满足 $l(n)<l^*(n)$，即最短加法链不是星最短加法链，这意味其最短加法链无法在限定 $a_k=a_{k-1}+a_j$ 时获得。而幂树法生成的加法链是 $l^*(n)$，所以随机幂树法在 $n\leqslant 24924$ 时正确率的理论极限是 99.96%。

除去满足 $l(n)<l^*(n)$ 的数外，还有几个数是非常难获得最优结果的，包括 4297、8594、16785、17188、20761、23251，这些数往往需要几百上千次的更新运算才能获得最优结果；更小的数中，2447 是一个相对难获得的最优结果的数，不过只需要十几次更新运算。大部分数只需少数几次更新运算就获得了最优结果。难获得最优结果的原因是什么，暂时没有梳理清楚，只是发现一个明显的原因是最短加法链解的个数，但相关性不是很高；有的数只有唯一的解，但全随机幂树法很容易生成其最短加法链，比如 1024 和 4096 等。

4.5　全随机幂树法的复杂度

由于要存储幂树，幂树法的空间复杂度是 $O(n)$。全随机幂树法还要存储扩展层结点及其扩展情况，但这些存储空间与存储幂树的存储空间相比小多了，因此其空间复杂度仍是线性复杂度 $O(n)$。幂树法的计算能力主要受存储幂树的影响。在多种 PC 机上运行的结果表明，现在普通的计算机可以求解 $n<155691199$(约 1 亿 5 千万)的规模；在此规模下，任务检测器显示全随机幂树法占用内存 1.567GB，当 $n=298695487$ 时，占用内存超过 2GB，程序无法分配内存而退出。

幂树法每个数值会存储一次，其时间复杂度应该是 $O(n)$。每个数值虽然只存储一次，但会判断多次，加之需要进行随机化处理，全随机幂树法的时间复杂度比 $O(n)$ 略高。通过计算多个 $c(r)$ 可验证其时间复杂度，$c(r)=\min(n,l(n)=r)$ 表示最短加法链链长为 r 的最小 n 值。截止 2014 年 3 月，已知 $r\leqslant 39$ 的所有 $c(r)$ 值，一般认为 $c(r)$ 的最短加法链是比较难计算的[5,9]。我们用全随机幂树法计算了所有 $r\leqslant 34$ 的 $c(r)$ 最短加法链，发现全随机幂树法很容易计算这些值，绝大部分 1 次计算就获得最优结果，偶尔需要重复计算 1 次。1 次随机幂树法计算 $c(r)$ 加法链的详细结果如表 3 所列。依据表 3 的数据获得全随机幂树法的时间复杂度是 $O(n^{1.12})$，比线性略高，与分析吻合。

表 3　全随机幂树法计算 $c(r)$ 的加法链结果

r	n=c(r)	Solved l(n)	运行时间 s
26	1176431	26	3.75
27	2211837	27	7.17
28	4169527	28	14.5
29	7624319	29	27.7
30	14143037	30	55.09
31	25450463	31	111.19
32	46444543	32	227.2
33	89209343	33	436.89
34	155691199	34	815.58

4.6　计算性能的比较

微软的 Clift 给出了他的最短加法链法与其他快速方法的计算时间估计[5]，在 12 个 2.66GHz CPU 上求出所有 $n\leqslant 2^{32}$ 的最短加法链估计需要一个月左右，而当时已知最快的方法[3]求出所有 $n\leqslant 2^{26}$ 的结果需要 1.5 年。根据这些数据和本文表 3 中的数据，我们的方法在双核 2.60GHz CPU 上求出所有 $n\leqslant 2^{32}$ 的最短加法链近似结果，保守估计(50 次重复更新)计算时间为 19 天，与 Clift 方法的计算时间大致相当，而我们的硬件计算能力只有微软 Clift 的十分之一，因此全随机幂树法的计算速度要比 Clift 的方法快大约一个量级。Clift 的方法采用图论中的一些理论大量消去冗余，这与幂树法每个结点只保留一种结果相类似，但 Clift 的方法是计算最短加法链的最优解，它不仅复杂而且有很多判断和计算，因此全随机幂树法比它计算速度快是合理的。我们的硬件条件远逊于微软，由于内存限制，只计算到 2^{27} 的规模，与 Clift 相差较大。

Clift 多次刷新最短加法链的世界计算记录(http://ww-whomes.uni-bielefeld.de/achim/addition_chain.html)，根据 Clift 论文中的数据[5]，全随机幂树法比基于有向无环图的方法[3]快 3 个量级，而 Clift 在论文中宣称该方法[3]是当时(2011 年)已知的最快方法。

与逐步深化回溯法[4]相比，全随机幂树法的计算时间基本可以忽略，因此可以把全随机幂树法应用于逐步深化回溯法的上界计算，这样可以有效地降低计算时间。根据对(2000 to 8000，step:41)共 146 个数据的运算，我们的实验表明，逐步深化回溯法[4]的时间复杂度是 $O(n^{1.96})$，总计运算时间 156s。如果把计算上界的方法从 knuth 的幂树法改为全随机幂树法(重复 10 次)，计算时间在(74s，107s)之间，则计算效率提高了 31% 52%，平均提高了 43%。效率提高的主要原因是更少的数据需要在 $l(n)$ 层进行搜索。我们的实验同时表明，数据越大计算效率提高越明显，对(10000 to 15000，step:41)共 121 个数据的运算，逐步深化回溯法的计算时间为 1106s，应用全随机幂树法计算上界后，计算时间降到(329s，377s)之间，计算时间是应用 knuth 幂树法计算上界的 1/3 左右。

结束语　随机幂树法是按层序扩展的，只是在同一层中随机选一个点生成一个新点，然后再随机选一个点生成一个新点。这样随机生成的幂树与 knuth 的幂树相比，性能略有

下降，但重复运算几次可以很好地改善其性能。10 次重复运算可以消去比次优结果差的解，且把最短加法链的正确率从 85%提高到 97%，加之其计算快速，随机幂树法可以作为其它精确解的上界计算方法，比如把它应用于回溯法可以提升计算效率 45%到 66%，而且对于大数效果更好。对于 $n \leqslant 160000000$ 的数，随机幂树法作为最短加法链的近似计算方法也是一个不错的选择。

参 考 文 献

[1] 周平，寇应展，王韬，等. 一种改进的针对滑动窗口模幂运算实现的密码数据 Cache 计时攻击[J]. 计算机科学，2013，40(3)：201-205

[2] Nedjah N，de Macedo Mourelle L. High-performance SoC-based implementation of modular exponentiation using evolutionary addition chains for efficient cryptography[J]. Applied Soft Computing，2011，11(7)：4302-4311

[3] Bleichenbacher D，Flammenkamp A. An efficient algorithm for computing shortest addition chains[OL]. http://www. homes. uni-bielefeld. de/ achim /addition_chain. html，1997

[4] Zhu Da-xin，Wang Xiao-dong. An Efficient Algorithm for Optimal Addition Chains[J]. TELKOMNIKA，2013，11(11)：6447-6453

[5] Clift N M. Calculating optimal addition chains [J]. Computing，2011，91：265-284

[6] Knuth D E. The art of computer programming：seminumerical algorithms(3rd ed)[M]. Addison-Wesley，Reading，1997：461-485

[7] Thurber E G. Efficient generation of minimal length addition chains[J]. SIAM J Comput，1999，28：1247-1263

[8] Bahig H M. Star reduction among minimal length addition chains [J]. Computing，2011，91：335-352

[9] 瞿云云，包小敏，刘花，等. 大整数模幂的固定基窗口组合算法[J]. 计算机应用研究，2013，30(3)：679-681

[10] Adan J，Hillel R M，Cindy G，et al. A Simulated Annealing Algorithm for the Problem of Minimal Addition Chains[C]//EPIA' 11 Proceeding of the 15th Protugese Conference on Progress in Artificial Intelligence：Lecture Notes in Computer Science. 2011：311-325

[11] Saúl D I，Efrén M M，Luis Guillermo O H. Addition chain length minimization with evolutionary programming[C]//GECCO '11 Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation. ACM New York，NY，USA. 2011：59-60

(上接第 227 页)

文算法和传统算法对入侵信号的频域徙动特征进行挖掘，得到本文改进的基于信息融合度传递的混合蛙跳算法和传统的混合蛙跳算法进行网络入侵信号的频域徙动特征挖掘结果，如图 5 所示。

分析图 5 中结果可知，采用本文算法滤波降噪等预处理，并采用信息融合度传递方案，入侵信号的频域徙动特征能够得到准确挖掘，特征的波脊亮点明显，而传统算法不能有效滤除噪声干扰，特征挖掘的聚类中心矢量向模糊边缘贴近，导致搜索精度不高、挖掘效能受限。上述实验从直观上展示了本文算法当进行入侵信号的频域徙动特征挖掘时的优越性能。以此为基础，通过 1000 次 Monte Carlo 实验，测试基于本文的特征挖掘算法对网络入侵信号的检测性能，测试结果得出，采用本文算法当信噪比为－3dB 时检测概率在 98%以上，而传统方法仅为 45%，证明本文算法在提高入侵信号检测性能上具有同样优势。

结束语 无线传感器网络和 Internet 网络混合组网的网络为一种高效的功率自激网络，功率自激混合组合网络攻击的入侵信号表现为一种具有频域徙动特征的谐振信号，对这种入侵信号的特征挖掘算法的研究决定着功率自激混合组合网络的安全。本文提出一种基于混合蛙跳最优模因组信息融合度传递的频域徙动入侵特征挖掘算法，首先对功率自激组合网络的系统模型和入侵信号数学模型进行构建，并对入侵信号的频域徙动特征进行数据信号采集和分析，设计改进的 IIR 滤波器进行抗干扰滤波处理，提高信号的纯度。基于最优模因组信息融合度传递策略，实现对频域徙动入侵特征挖掘算法的改进。通过实验得出本文算法能准确挖掘入侵信号的频域徙动特征，特征的波脊亮点明显，在低信噪比下提高了入侵信号的检测性能。

参 考 文 献

[1] 郑纪彬，符渭波，苏涛，等. 一种新的高速多目标检测及参数估计方法[J]. 西安电子科技大学学报：自然科学版，2013，40(2)：82-88

[2] 靳晓艳，周希元，张琬琳. 多径衰落信道中基于自适应 MCMC 的调制识别[J]. 北京邮电大学学报，2014，37(1)：31-34

[3] Zhu Q Y，Yang X F，Yang L X，et al. Optimal control of computer virus under a delayed model[J]. Applied Mathematics and Computation，2012，218(23)：11613-11619

[4] 邓兵，陶然，平殿发，等. 基于分数阶傅里叶变换补偿多普勒徙动的动目标检测算法[J]. 兵工学报，2009，30(10)：1034-1039

[5] 叶青，黄炎磊. 非均匀分布入侵检测模型的研究与仿真[J]. 科技通报，2013，29(8)：169-171

[6] 赵鹏军，邵泽军. 一种新的改进的混合蛙跳算法[J]. 计算机工程与应用，2012，48(8)：48- 50

[7] 张伟，师奕兵，周龙甫，等. 基于改进粒子群算法的小波神经网络分类器[J]. 仪器仪表学报，2010，31(10)：2203-2209

[8] 张永铮，肖军，云晓春，等. DDoS 攻击检测和控制[J]. 软件学报，2012，23(8)：2258-2072

[9] 夏秦，王志文，卢柯. 入侵检测系统利用信息熵检测网络攻击的方法[J]. 西安交通大学学报，2013，47(2)：14-19

[10] 吴春琼. 基于特征选择的网络入侵检测模型[J]. 计算机仿真，2012，29(6)：136-139

[11] 王睿. 一种基于回溯的 Web 上应用层 DDOS 检测防范机制[J]. 计算机科学，2013，40(11A)：175-177

[12] 李振刚，甘泉. 改进蚁群算法优化 SVM 参数的网络入侵检测模型研究收[J]. 重庆邮电大学学报：自然科学版，2014，26(6)：785-789