arXiv:2301.00316v1 [cs.DS] 1 Jan 2023

希尔排序的优化视角

Oscar Skean
肯塔基大学计算机科学系
oscar.skean@uky.edu

Richard Ehrenborg
肯塔基大学数学系

Jerzy W. Jaromczyk肯塔基大学计算机科学系

摘要

希尔排序是一种因简单而颇具吸引力的排序方法，但分析其效率却需要付出不少努力。该算法的核心是预先选定并在排序过程中使用的增量序列。增量序列的选择会影响希尔排序的效率，因此也推动了对其进行理论和实验分析。我们通过基于新的参数化函数确定高效增量序列，为希尔排序研究作出贡献。具体而言，我们采用参数网格搜索，为不同输入规模的排序任务寻找最优参数，并通过观察三个指标的最小开销进行评估：比较次数、交换次数和运行时间。结果表明，对于选定的小数组大小，我们的方法找到的序列在比较次数方面优于最先进的增量序列。此外，对于选定的大数组大小，我们基于函数的序列在运行时间方面优于 Tokuda增量序列。不过，在以交换次数最少为目标时，我们未观察到显著改进。

1 引言

希尔排序算法是最早被发现的排序方法之一。该算法发表于 1959 [1], ，早期受到关注，是因为它内存需求低且实现简单。尽管如此，对它的分析仍然很困难，而且尚未完成。如今，该算法已实际应用于内存受限环境、嵌入式系统和 bzip2压缩器。近年来，它还应用于数据无关排序 [2] 以及全同态加密 [3]。

希尔排序是一种原地比较排序，也可视为插入排序的推广。对于大小为 N 的数据数组A，希尔排序使用预先确定的增量序列 $1 = k_1 < \cdots < k_m < N$进行操作。该算法对 A执行 m 趟：从最大的 k_m 开始，直到最小的 k_1。在第 j 趟中，对于给定的增量 k_{m-j}，对由数据元素 $A(i)$ $A(i + k_{m-j})$ $A(i + 2 \cdot k_{m-j})$ .. .for $i = 0, \ldots,$ $k_{m-j} - 1$组成的 k_{m-j} 个子数组执行插入排序。我们称满足不等式 $A(i) > A(i + k)$ 的一对 $(i, i + k)$ 为一个 k-逆序对 。第 j 趟结束后，原先存在于 A 中的所有 k_{m-j}-逆序对都已消除。没有 k-逆序对的数组称为 k-有序。请注意，最后一趟使用 k_1 ，这等同于插入排序，是保证有序性所必需的。因此，增量序列的作用是在代价高昂的最后一趟插入排序之前，尽可能地对 A 进行预排序。

本文的主要成果如下：

1. 通过对序列生成函数进行实验优化，推导出新的高效希尔排序序列。对于给定的数组大小，这些序列在比较次数方面优于众所周知的高效序列（例如 Tokuda、Ciura）。在测试的数组大小范围内，这些序列的运行时间也优于 Tokuda序列，使其成为最快的基于函数的序列。2. 我们通过测量对随机排列进行排序所需的比较次数、交换操作和运行时间，展示了实验分析结果，并将我们提出的方法与众所周知的序列进行了比较。

传统上，希尔排序的改进来自寻找具有理论性质的增量序列。我们将在第2节讨论一些尤为重要的序列。接着在第3节，我们将介绍参数化序列——

生成函数，这些函数可以生成希尔排序序列。随后，我们通过网格搜索优化参数，为选定的数组大小找出该函数所能生成的最佳序列。第4节将讨论我们的实验方法，用以比较优化后的模板序列与第2节提到的基准序列的性能。

2 背景

选择合适的增量序列对希尔排序的性能至关重要。大量研究致力于寻找优秀的序列 [4, 5, 6]。最早提出的一些序列基于2 [7, 1]的幂。随后，Pratt 证明，序列 2^p3^q在最坏情况下得到的逆序对数量为 $\Theta(Nlog^2N)$ [8]。我们在表1中将该序列称为 Pratt-23。对于任何希尔排序序列，该序列目前仍具有已知的最佳渐近时间复杂度。不过，它的常数因子非常大，因此推动了新序列的开发。

用于证明 Pratt-23 时间复杂度的技术，是对已进行 2-排序和 3-排序的序列中的逆序对进行计数。对此进行自然推广的一种方法，是应用 Frobenius问题，界定前几趟排序已经排好序的部分。Frobenius问题的典型表述如下：假设你有 k枚面额分别为 u_1、 u_2、..... u_k的硬币。用这些硬币的非负整数线性组合无法凑出的最大金额是多少？这个最大值称为 Frobenius数 [9]。在希尔排序的语境中，硬币可对应增量大小，而 Frobenius数可对应使用这些增量排序后剩余的最大逆序对。利用 Frobenius问题，人们提出了若干序列；尽管它们的时间复杂度更差，但其常数因子低于 Pratt-23 [10, 11]。

在这些序列之后，增量序列的选择重点从寻找理论上表现良好的序列转向寻找实验中表现良好的序列。例如，人们观察到，增长率为 2.25 的几何序列在实践中往往表现良好。这一观察构成了 Tokuda序列 [6] 的基础。其函数形式见表1。据我们所知，它至今仍是文献中最具竞争力的函数。

下一项改进来自 Ciura 在 2002 [4] 中提出的方法。Ciura序列摒弃了基于函数的序列这一思路，转而直接寻找最佳的增量元素集合。Ciura 找到了适用于数组大小为 128 和 1000 的最佳序列，以及一种被推测在数组大小大得多时表现更佳的序列。我们在表1中将它们分别称为 Ciura-128、Ciura-1000 和 Ciura-Large。

Ciura序列也标志着希尔排序性能衡量方式的转变。此前，大多数研究统计算法执行的交换次数 [8]，部分原因是某些证明技术依赖于统计逆序对的数量。Ciura 则转而着重优化比较次数，因为研究发现，比较次数与希尔排序的计算时间关系更为直接。比较是指检查一对数组元素是否构成逆序。交换通常指用于修正一个逆序的变量交换操作。当比较成本较高时，最小化比较次数尤其有益，例如在对大型卫星数据进行排序时。同样，在内存受限系统中，最小化交换次数也有益。在本研究中，我们将在第4节明确区分所优化的衡量指标。关于增量序列历史的全面介绍，请参阅 [5]。

3 参数化模板函数

直接优化增量序列的方法（如 [4], ）随着 N 增大，计算成本会迅速攀升。这种增长是因为随着 N 增大，序列元素的预期数量及其可能取值范围都会增加。为缓解这一问题， [4] 的作者找到了合适的序列前缀，并通过少量取值对其进行扩展优化。然而，当 N 非常大时，仅仅找到这个前缀也会耗费极高的成本。

在这里，我们将寻找最优序列的问题表述为优化预定义序列生成函数的参数。根据我们在表现良好的序列中观察到的规律，我们定义了以下两个函数。

$$k_A(i) = \lfloor (a^{\lfloor \frac{i}{b} \rfloor} \cdot c^{\lfloor \frac{i}{d} \rfloor})^f + e \rfloor \tag{1}$$

$$k_B(i) = \lfloor (a \cdot b^{\lfloor \frac{i}{c} \rfloor}) + d \rfloor \tag{2}$$

我们在表1中将（1）称为 Ours-A，将（2）称为 Ours-B。这两种格式的指数中都包含下取整函数。我们发现，这使函数能够表示更加“混沌”的序列，从而有助于提升性能。Ours-A 的独特之处在于参数 f，它是一个有助于调节增长的指数。Ours-B 的参数 a 则

序列名称	函数	针对 N 优化	参数	初始项
Ciura [4]	-	128	-	1 4 9 24 85 126
	-	1000	-	1 4 10 23 57 156 409 995
	-	大型	-	1 4 10 23 57 132 301 701 1750
Tokuda [6]	$\left\lceil \frac{(9/4)^k-1}{-1}\right\rceil_{(9/4)}$	-	-	1 4 9 20 46 103 233 525 ...
我们的方法 A	$\lfloor (a^{\lfloor \frac{i}{b} \rfloor} \cdot c^{\lfloor \frac{i}{d} \rfloor})^f + e \rfloor$	128（计算）	表 2	1 4 9 24 85 150 ...
		1000（计算）	表 2	1 4 10 23 57 153 400 ...
		1000（时间）	表 2	1 3 7 16 33 85 179 472 ...
我们的方法B	$\lfloor (a \cdot b^{\lfloor \frac{i}{c} \rfloor}) + d \rfloor$	10000（计算）	表 2	1 4 10 27 72 187 488 ...
Pratt-23 [8]	有序 $2^p \cdot 3^q$	-	-	1 2 3 4 6 8 9 ...
Pratt-25	有序 $2^p \cdot 5^q$	-	-	1 2 4 5 8 10 15 16 ...
Pratt-34	有序 $3^p \cdot 4^q$		-	1 3 4 9 12 16 24 ...

表1：实验中进行比较的增量序列

模板	a	b	c	d	e	f
Ours-A128-Comp	2.6321	1.6841	2.1570	0.7360	3	0.7630
Ours-A1000-Comp	3.5789	2.6316	3.8158	2.1579	3	0.7632
Ours-A1000-Time	2.75	2.75	3.7142	2.4286	2	0.7429
Ours-B10000-Comp	4.0816	8.5714	2.2449	0	-	-

表 2：模板函数的优化参数

旨在实现类似的目的，不过是通过乘法来实现。Ours-B 的参数较少，因此优化速度更快。为简洁起见，我们仅针对大小为 128 和 1000 的数组优化 Ours-A，并针对大小为 10000 的数组优化 Ours-B。此外，如果针对大小为 1000 的数组优化格式 A 以减少比较次数，我们将其记为 Ours-A1000-Comp。在下一节中，我们将介绍优化（1）和（2）的实验流程，以及将它们与其他基准序列进行比较的方法。

4 实验流程

由于该函数高度非凸，因此难以使用梯度下降等高效技术。相反，我们采用网格搜索方法。与直接进行序列优化相比，使用函数网格搜索的一个优点是，搜索空间的大小与 N 无关。它仅取决于搜索的粒度和边界。这与 Ciura 使用的方法形成对比，在该方法中，经过测试的序列数量会随 N [4]增长。

对于 Ours-A，我们将参数 a、b、c、d、f的网格定义为 0.5 到 5 之间等间距的 20 个值。我们还允许 e 取包含两个端点在内的 0 到 10 之间的整数值。由于 Ours-B 的参数更少，我们可以采用更细粒度的方法。对于参数 a、b、c，我们测试了从 0 到 10 等间距的 50 个值。对于参数 d，我们将其限制为与 Ours-A 中的 e 相同。

我们测试的数据数组包含 N 个互不相同的值，范围为 1 到 N，并使用 Fischer-Yates 洗牌算法对其进行打乱。对于网格空间中的每组参数，我们计算 1000 次迭代的平均代价。随后，我们将产生最低平均代价的参数组视为最优。此代价可以定义为比较次数、交换次数或时间。

由于 Ciura序列针对特定数组大小进行了优化，且没有可用于扩展它们的方法，因此不适合在大型数组大小下将它们与任何基于函数的序列直接比较。扩展 Ciura序列的一种方法是以其最后一项为起点，按比率 2.25 构造几何级数。我们在测量 Ciura序列的性能时采用这种扩展方法。

序列	N=20		N=128		N=200	
	μ_{CO}	μ_{EX}	μ_{CO}	μ_{EX}	μ_{CO}	μ_{EX}
Ours-A128-Comp	76 ± 6	38 ± 6	**998 ± 33**	531 ± 33	1786 ± 46	948 ± 48
Ours-A1000-Comp	76 ± 6	39 ± 7	1004 ± 32	516 ± 31	1787 ± 44	919 ± 45
Ours-A1000-Time	79 ± 5	39 ± 7	1035 ± 26	468 ± 27	1832 ± 38	846 ± 39
Ours-B10000-Comp	76 ± 7	33 ± 5	1096 ± 52	535 ± 36	**1775 ± 49**	960 ± 49
Ciura-128	76 ± 6	37 ± 6	**998 ± 32**	531 ± 33	1800 ± 46	970 ± 49
Ciura-1000	76 ± 7	39 ± 7	1006 ± 31	519 ± 34	1787 ± 45	920 ± 44
Ciura-Long	76 ± 7	39 ± 7	1004 ± 32	516 ± 32	1794 ± 44	907 ± 42
Tokuda	76 ± 6	37 ± 6	1020 ± 28	490 ± 28	1808 ± 42	891 ± 43
Pratt-25	111 ± 4	27 ± 4	1732 ± 16	345 ± 17	3207 ± 21	610 ± 24
Pratt-23	136 ± 3	25 ± 4	2209 ± 13	**333 ± 15**	4095 ± 19	**589 ± 21**
Pratt-34	95 ± 4	29 ± 4	1424 ± 16	374 ± 19	2593 ± 25	660 ± 26

表3：对小型数组排序所需的操作次数，为1000种随机数组排列的平均值。 μ_{CO} 表示比较次数， μ_{EX} 表示交换次数。

序列	N=1000		N=2000		N=5000	
	μ_{CO}	μ_{EX}	μ_{CO}	μ_{EX}	μ_{CO}	μ_{EX}
Ours-A128-Comp	13250 ± 203	7847 ± 199	30530 ± 378	18611 ± 384	91122 ± 973	57728 ± 904
Ours-A1000-Comp	12941 ± 167	7004 ± 155	29596 ± 293	16234 ± 282	86821 ± 768	50349 ± 770
Ours-A1000-Time	13193 ± 144	6461 ± 146	30120 ± 263	14913 ± 257	87455 ± 548	44305 ± 552
Ours-B10000-Comp	12980 ± 186	7245 ± 177	29643 ± 305	17241 ± 325	86514 ± 617	57388 ± 817
Ciura-128	13300 ± 166	7003 ± 168	30359 ± 318	15987 ± 310	88193 ± 629	46689 ± 627
Ciura-1000	**12918 ± 161**	7002 ± 155	**29534 ± 282**	16138 ± 274	86641 ± 757	47852 ± 751
Ciura-Long	13035 ± 142	6701 ± 149	29567 ± 246	15427 ± 261	**86232 ± 502**	45347 ± 496
Tokuda	13116 ± 143	6556 ± 142	29888 ± 241	14952 ± 228	86838 ± 454	44116 ± 472
Pratt-25	26211 ± 68	4318 ± 72	62722 ± 122	9755 ± 131	194196 ± 263	**28195 ± 278**
Pratt-23	34380 ± 64	**4253 ± 69**	82785 ± 106	**9669 ± 116**	259088 ± 242	28354 ± 257
Pratt-34	20974 ± 89	4671 ± 87	50038 ± 153	10543 ± 160	154298 ± 372	30448 ± 372

表4：对 1000 种随机数组排列进行平均后，排序中等大小数组所需的操作次数

4.1 筛选

我们采用两种技巧来缩小网格搜索空间。

首先，我们注意到，不同的参数集合可能会产生相同的序列。例如，对于模板函数 Ours-A，(a, b) 和 (c, d) 的顺序并不重要。我们预先计算网格中产生的所有序列，并仅对不重复的序列进行实验。例如，对于 $N = 10000$，这会将网格搜索空间中的参数集合数量从 150 多万个减少到约 100 万个。

其次，我们使用序贯分析作为低通滤波器，筛除明显较差的序列。这种统计方法最初于 [4] 被应用于希尔排序。给定均值的界限和方差的上界，序贯分析只需进行几次重复，就能以一定的置信度判断样本均值是否低于均值界限。序贯分析使我们能够快速接受均值比较次数较少的优良增量序列。任何通过筛选器的序列，随后都会进行完整的 1000 次迭代，以获得更准确的均值估计。我们采用与 [4] 相同的设置。

Ciura序列是针对比较次数进行优化的；由于它们被公认为最实用的序列之一，我们也针对比较次数优化了模板函数。

4.2 硬件

实验在一台配备 8 核 Intel Xeon W-3225 的 Ubuntu 机器上进行。统计比较次数和交换次数的实验采用多线程方式进行。涉及时间测量的实验则

序列	N=10000	
	μ_{CO}	μ_{EX}
Ours-A128-Comp	206356 ± 1796	132351 ± 1797
Ours-A1000-Comp	196336 ± 1707	119012 ± 1710
Ours-A1000-Time	194052 ± 879	98952 ± 883
Ours-B10000-Comp	192029 ± 992	209292 ± 1293
Ciura-128	195256 ± 1106	105544 ± 1109
Ciura-1000	193778 ± 1895	111338 ± 1897
Ciura-Long	**191435 ± 892**	101680 ± 897
Tokuda	192574 ± 795	98071 ± 796
Pratt-25	450131 ± 516	**62191 ± 526**
Pratt-23	604502 ± 451	66923 ± 725
Pratt-34	355382 ± 723	63272 ± 462

表5：对大小为 10000 的数组进行排序所需的操作次数，取 1000 次随机数组排列的平均值

序列	运行时间（毫秒）
Ours-A128-Comp	3.15 ± 0.08
Ours-A1000-Comp	3.02 ± 0.06
Ours-A1000-Time	**3.01 ± 0.06**
Ours-B10000-Comp	3.04 ± 0.07
Ciura-128	3.07 ± 0.06
Ciura-1000	**3.01 ± 0.06**
Ciura-Long	3.04 ± 0.07
Tokuda	3.06 ± 0.08
Pratt-25	5.00 ± 0.09
Pratt-23	6.35 ± 0.11
Pratt-34	4.17 ± 0.08

表6：对 1000 个随机数组排列进行排序的时间，取均值；数组大小为 1000

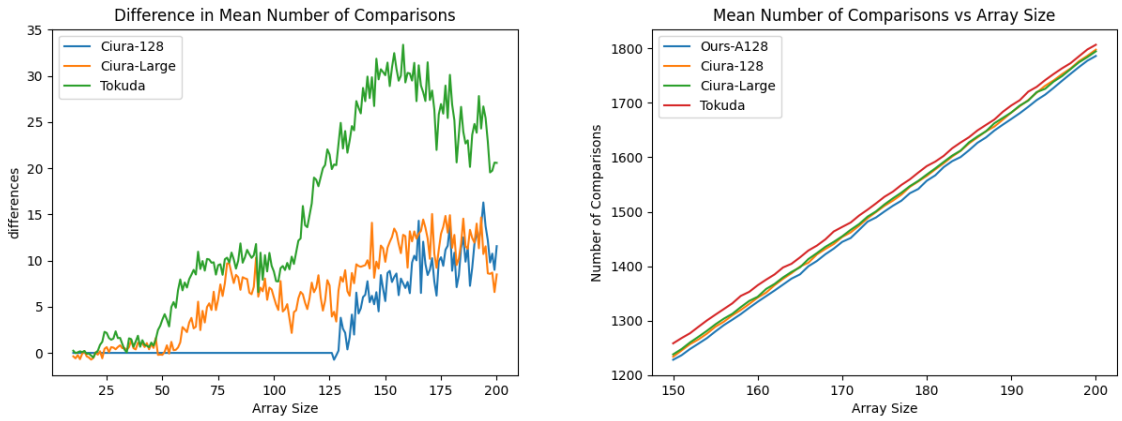


图 1：（左）展示不同数组大小下，基准序列与 Ours-A128 的比较次数差异。正值表示 Ours-A128 使用的比较次数更少。（右）展示数组大小超过 Ours-A128 优化目标时的比较次数。

采用单线程，因为任何多线程应用程序都可能导致时间测量出现偏差。所有代码均使用 Python 编写。

4.3 结果

我们为 Ours-A 和 Ours-B 找到的最佳参数见表 2。此外，序列的前几项见表1。

对于大小为 200 的数组，我们发现，Ours-B10000-Comp 在比较次数方面优于所有其他受测序列，如表3所示。图 1 以图形形式呈现了该表的数据。这些图表显示，由模板函数生成的序列对于比其优化目标更大的数组大小仍能表现良好，但提升并不显著。此外，我们发现，Ours-A128-Comp 和 Ours-A1000-Comp 的比较次数均达到 Ciura 和 Tokuda序列的水平，但未能超越它们。值得注意的是，Ciura-128 和 Ours-A128-Comp 有若干初始项相同。
我们还测试了中大型数组，结果如表4和表5所示。表5的图示请参见附录中的图 1。我们提出的新序列性能接近 Ciura序列，但未能超越它们。不过，对于此处测试的所有数组大小，Ours-B10000仍优于 Tokuda序列。请注意，Tokuda序列目前是已知的最佳函数生成序列。因此，我们展示了一种新的基于函数的序列，在比较次数方面优于其他基于函数的序列。如前所述，当比较是主要操作时（例如对大型卫星数据进行排序），这一点尤其有用。

另一方面，我们在优化序列以尽量缩短运行时间方面的实验结果如表6所示。相关序列 Ours-A1000-Time 的运行时间与 Ciura-1000序列相近。两者都比其他所有受测序列更快。Ours-A1000-Time序列尤其值得关注，因为它的前几个元素（1 3 7）不同于大多数其他快速序列（通常以 1 4 9 开头）。这种差异或许说明，以（1 4 9）开头的序列虽然可能非常有利于减少比较次数，但并不能保证整体运行时间也表现良好。

5 结论

希尔排序的改进通常源于寻找具有更优理论性质的增量序列。本文沿着 [4] 的研究路径，引入了一个用于寻找改进增量序列的实验框架。对于指定的数组大小，我们生成的增量序列在比较次数方面优于所有已知的增量序列。此外，据我们所知，Ours-A1000-Time 是运行时间最快的基于函数的序列。不过，它的性能达到了 Ciura序列的水平，但并未超越该序列。通过采用不同的序列生成函数或实验设置，仍有可能进一步改进；这将留待未来工作研究。尽管本文所介绍的序列针对选定的数组大小进行了优化，但也可以针对任何感兴趣的数组大小重复进行优化。

参考文献

[1] D. L. Shell，《高速排序过程》，《*ACM* 通讯》，第 2 卷，第 7 期，第 30–32 页，1959年。[2] M. T. Goodrich，《随机化希尔排序：一种简单的数据无关排序算法》，《*ACM* 杂志》（*JACM*），第 58 卷，第 6 期，第 1–26 页，2011 年。[3] J.-W. Lee、Y.-S. Kim 和 J.-S. No，《全同态加密的改进的希尔排序分析》，《*IEEE Access*》，第 9 卷，第 126198–126215 页，2021 年。[4] M. Ciura，《希尔排序平均情况下的最佳增量》，载于 计算理论基础国际研讨会，第 106–117 页，施普林格，2001 年。[5] R. Sedgewick，《希尔排序及相关算法的分析》，载于欧洲算法研讨会，第 1–11 页，施普林格，1996 年。[6] N. Tokuda，《一种改进的希尔排序》，载于 *IFIP* 第十二届世界计算机大会：算法、软件、体系结构——信息处理*'92*，第 *1* 卷，第 449–457 页，1992 年。[7] T. N. Hibbard，《最小存储排序的实证研究》，《*ACM* 通讯》，第 6 卷，213，1963 5，206第。页 – 。[8] V. R. Pratt，《希尔排序与排序网络》，技术报告，斯坦福大学计算机科学系，1972 年。[9] E. S. Selmer，《关于弗罗贝尼乌斯线性丢番图问题》，1977 年。

[10] J. Incerpi 和 R. Sedgewick，“希尔排序的改进上界”，《计算机与系统科学杂志》，第 31 卷第 2 期，第 210–224 页，1985 年。[11] E. S. Selmer，“关于希尔排序和 Frobenius问题”，1987 年。