# 线性时间选择算法的改进算法

孙妍超，陈瑜*

(四川大学 计算机学院，成都市 610225)

(*通信作者电子邮箱 yuchen@scu.edu.cn)

**摘 要:** 线性时间选择算法是分治法算法中的经典算法之一，可在线性时间复杂度内求解“从 $n$ 个元素中找到第 $k$ 小的元素”的问题。但是，当 $k$ 值较小时，经典算法中使用“中位数的中位数”进行基准划分元素的选择方法并没有达到最优化。针对该问题，本文对经典的线性时间选择算法进行了改进，并提出一种基准划分元素的动态选择方法。该方法在每一次迭代中通过对中位数和 $k$ 值进行对比来确定最优的基准划分元素。基于此方法，可增大搜索空间的剪枝范围，进而使得改进算法的性能有较好的提升和改进。充分的实验测试表明，改进后的算法在运行时间上优于经典算法，在一些情况下效率甚至可以有数倍的提升。

**关键词:** 线性时间选择；分治法；算法；剪枝；时间复杂度

**中图分类号:**TP301.6 **文献标志码：A**

# Modified Linear Time Selection Algorithm

SUN Yanchao, CHEN Yu*

(College of Computer Science, Sichuan University, Chengdu 610225, China)

**Abstract:** The linear time selection algorithm is one of the classic divide-and-conquer algorithms. It can resolve the classic problem within linear time complexity of selecting the *k*-th smallest element in the set of *n* elements. However, when the value of *k* is smaller, the strategy of selecting the median of the median as the pivot element in the classic algorithm is inefficient. To tackle this problem, in this study, we improve the canonical linear selection algorithm. A novel method to select the pivot element dynamically is proposed, which determines the best pivot element by comparing the median and the value of k in every iteration. Based on this method, the search space can be pruned dramatically. Extensive experiments show that our new method outperforms the canonical algorithm. Moreover, in some cases the efficiency can be improved by several times.

**Keywords:** the linear selection**;** divide-and-conquer**;** algorithm**;** pruning**;** the time complexity

## 0 引言

线性时间选择问题[1-3]是分治法算法中的经典算法之一。经典的线性时间选择问题是指，给定线性序集中 $n$ 个元素（不含重复元素）和一个整数 $k$, $1\le k\le n$,要求从这 $n$ 个元素中找出第 $k$ 小的元素，即如果将这 $n$ 个元素由小至大排列时，排在第 $k$ 个位置的元素即为要找的元素。这一问题在实际中有着广泛的应用，提升它的时间性能有着重要的意义。

对线性时间选择问题直观的求解方法是先对 $n$ 个元素进行排序，再直接选择第 $k$ 个元素。然而，这种求解算法的时间复杂度最优只可达到 $O（nlogn）$。当 $k$=1 或 $n$ 时，线性时间选择问题退化为求解 $n$ 个元素中的最小值或最大值问题。此时，只需对 $n$ 个元素遍历一遍即可找到第 $k$ 小的元素。显然，其时间复杂度是 $O(n)$。但是，对于更一般的数值 $k$，能否在线性时间有效求解第 $k$ 小的问题？针对此问题，文献[1]进行了研究，并提出了经典的线性时间选择算法，可以在线性时间复杂度 $O(n)$下求解此问题。线性时间选择算法已成为基础性算法中被广泛应用的经典算法之一，在算法理论研究[4-9]、数据库技术[10]、数据挖掘[11,12]和图像处理[13]等诸多领域都有着广泛的应用。

基金项目：国家自然科学基金资助项目（U1333113）。

This work is partially supported by the National Natural Science Foundation of China (U1333113).

作者简介：孙妍超(1995 年-)，女，天津人，四川大学计算机学院本科生；

陈瑜（1974 年-），男，陕西洋县人，博士，四川大学计算机学院讲师，本文通讯作者。

**SUN Yanchao**, born in 1995, undergraduate in Sichuan University

**CHEN Yu**, born in 1974, Ph. D, Lecturer. His research interests include evolutionary computation and data mining.

虽然对于经典的线性时间选择算法已有广泛的研究，但是在进一步的研究中，我们发现经典的线性时间选择算法仍有改进空间。本文针对经典线性时间选择算法中的关键步骤进行了优化，并提出了一种基于动态选择基准划分元素的新的线性时间选择改进算法。理论分析和实验结果均表明，相比经典的线性时间选择算法，本文中提出的改进的线性时间选择算法在性能方面有明显的提升。

## 1 线性时间选择算法

如何在线性时间内求解元素选择问题具有非常广泛的应用价值。文献[1]提出了经典的线性时间选择算法，其主要思想是：借鉴快速排序[14]算法中的“划分”过程，参照基准划分元素 x，将原序列中的 $n$ 个元素划分为低区（由小于 x 的元素组成的区域）、x 和高区（由大于 x 的元素组成的区域）等三个部分。根据各个区域中元素个数与待查询的第 k 小元素的k值进行比对，只对含第$k$小元素的部分进行递归求解，不符合要求的元素部分则进行剪枝。如何选择基准划分元素是整个算法的核心步骤。在经典的线性时间选择方法中采用了“中位数的中位数”方法来选择基准划分元素[1]，其选择策略可描述如下：

（1）将 $n$ 个输入元素划分为$\lceil n/5\rceil$个组，每组 5 个元素，且至多只有一个组由剩下的 $n$ *mod* 5 个元素组成；

（2）用任意一种排序算法将每组的 5 个元素升序排序，并取出每组元素的中位数，共$\lceil n/5\rceil$个；

（3）递归调用 *Select*[1]选择算法找出这$\lceil n/5\rceil$个中位数元素的中位数 $x$。若$\lceil n/5\rceil$是偶数，就找它的两个中位数中较大的一个，将其作为基准划分元素；

（4）利用 *Partition* 函数[14]，按中位数的中位数 $x$ 对输入数组进行划分。令 $j$ 等于划分低区（包括 $x$）的元素数目。若 $j=k$，则返回 $x$；否则，若 $j<k$,则在低区递归调用 *Select* 选择算法以找出第 $k$ 小的元素；若 $j>k$, 则在高区找第$(k-j)$小的元素。

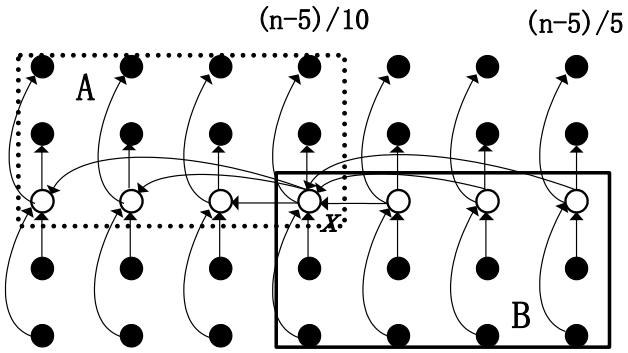


图 1：选择基准划分元素

图 1 是该划分策略的示意图。图中竖直方向上 5 个点为一组。圆点表示待选的 $n$ 个元素，空心圆点代表每组元素的中位数，箭头由较大元素指向较小元素。点 $x$ 为中位数的中位数，即该方法选定的基准划分元素。

为简化该问题，经典线性时间选择问题中假设 $n$ 个元素序列中无重复元素。图 1 中，$x$ 至少比 $3\lfloor(n-5)/10\rfloor-1$ 个元素大（由图 1 可知，区域 B 包含的部分的元素的值（除去 $x$）必然比 $x$ 的值大）。同理，$x$ 也至少比 $3\lfloor(n-5)/10\rfloor-1$ 个元素小（区域 $A$）。当 $n\geqslant 75$ 时，$3\lfloor(n-5)/10\rfloor-1\geqslant n/4$。也就是说，按此基准划分元素所得的两个子序列的长度都至少缩短了 1/4，使得搜索空间可被剪枝掉 1/4 大小，从而使得算法在最坏情况下仍保持良好的性能。表 1 给出了经典线性时间选择算法的描述,该算法返回数组 $a[p:r]$中第 $k$ 小的元素。

表 1：经典线性时间选择算法[1]

**Algorithm 1: *Select***$(a, p, r, k)$

1. **if** $r-p<75$
2. **then** sort $a[p:r]$ ；
3. **return** $a[p+k-1]$;
4. $n\leftarrow r-p+1$;
5. **for** $i\leftarrow 0$ **to** $(n-5)/5$
6. 将 $a[p+5*i]$至 $a[p+5*i+4]$的第三小元素与 $a[p+i]$交换位置；
7. $x\leftarrow$*Select*$(a,p,p+(n-5)/5,(n-5)/10)$;
8. $i\leftarrow$*Partition*$(a,p,r,x)$;
9. $j\leftarrow i-p+1$;

```
10.     if k<=j
11.       then return Select(a,p,i,k);
12.     else return Select(a,i+1,r,k-j);
```

```
    Partition(a,p,r,x)
1.      找到 a[y]=x;
2.      exchange a[p]↔a[y];
3.      i←p-1 , j←r+1;
4.      while TRUE
5.        do repeat j←j-1
6.             until a[j]<=x
7.          repeat i←i+1
8.             until a[i]>=x
9.        if i<j
10.         then exchange a[i]↔a[j];
11.       else return j
```

在表 1 的算法中，设 $n=r-p+1$ 为输入元素数组的长度，算法的递归调用只在 $n\geq 75$ 时才执行。当 $n<75$ 时，算法 *Select* 的计算时间不超过某个常数 $C_1$。设对 $n$ 个元素的数组调用 *Select* 算法需要 $T(n)$时间，则找中位数的中位数所需时间不超过 $T(n/5)$。现已证明，按照算法所选的基准划分元素 $x$ 进行划分所得到的两个子数组分别至多只包含有 $3n/4$ 个元素。因此，无论对哪一个子数组调用 *Select* 算法都至多需要 $T(3n/4)$ 时间。在执行过程中，划分函数 *Partition*（）需要 $O(n)$时间，*for* 循环需要 $O(n)$时间。因此，*Select* 算法所需时间复杂度 $T(n)$ 可由递归式（1）得到[3]：

$$T(n)\leq\begin{cases} C_1 & n<75 \\ C_2 n+T(\frac{n}{5})+T(\frac{3n}{4}) & n\geq 75 \end{cases} \quad (1)$$

求解递归式（1）可知 $T(n)=O(n)$。因此，经典线性时间选择算法实现了用线性时间复杂度求解第 $n$ 小元素的选择问题。

在经典线性时间选择算法中，其核心之处体现在两个方面：（1）借鉴了快速排序的思想。依照基准划分元素将搜索空间分割为三个部分，从而提高了搜索效率；（2）基准划分元素的选择。将中位数的中位数作为基准划分元素，可剪枝掉 1/4 的搜索空间，使得搜索效率有显著改进。然而，在深入研究该算法时，我们发现该算法仍有改进空间。在实践应用中，我们注意到通常在搜索第 k 小个元素的问题中，k 值均是较小的值（若是较大的值，可转化对称的搜索第 k 大个元素问题）。我们发现当 $k$ 值较小的时候，经典线性时间选择算法中采用的这种剪枝策略只能慢慢逼近目标，而不能迅速缩小索搜空间。针对这一点，在本文中，我们提出了一个新的基准划分元素的选择策略来提升算法的效率。

## 2 改进的线性时间选择算法

线性时间选择算法的一个核心之处在于每次迭代时可以确认基准元素 $x$ 右下方的 1/4 区域内的元素都大于 $x$（不考虑重复元素），因此可以将搜索空间剪枝掉 1/4，从而提高搜索效率。如果能提高剪枝掉元素的数量，则算法的性能会得到很好的提升。

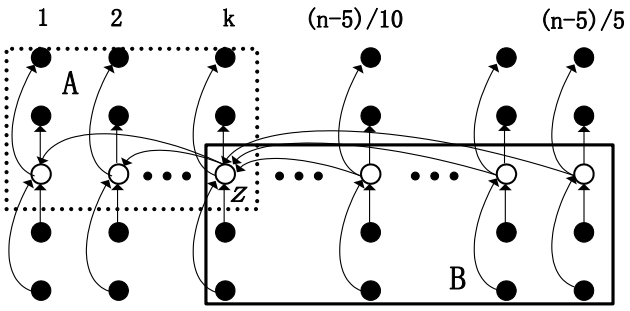


图 2：动态基准划分元素的选择

通过对经典线性时间选择算法的基准划分元素的选择策略进行分析，可以发现正是由于每次选择都无差别地选取“中位数的中位数”，才导致了当 $k$ 比较小时，需要多次递归才能达到目标。实际上，在多数情况下，如果直接选择每列元素的中位数序列中第 $k$ 小的元素，即可快速的缩小搜索范围。如图 2 所示，假设 $z$ 为每列元素的中位数序列中第 $k$ 小的元素。图 2 中左上角虚线框内的 $A$ 区域中的 $3k$-1 个元素必定小于 $z$ 的值（因为根据 $k$ 的含义，$k$ 只能是取大于等于 1 的值，故有 $3k-1>k$）。经过划分后，在低区（比基准划分元素 $z$ 的值小的部分）的元素数目必定大于 $k$ 个，所以会将高区（比

基准划分元素 $z$ 的值大的部分）的部分剪枝。而右下角实线框内的区域 $B$ 部分中的 $3\lfloor(n-5)/5-(k-1)\rfloor-1$ 个元素都必定大于元素 $z$ 的值，故剪枝掉的部分至少包含的元素数目为 $3\lfloor(n-5)/5-(k-1)\rfloor-1$。在 $n>>k$ 的情况下，此方法可以剪枝掉约 1/2 的元素。然而，经典线性时间选择算法中选取基准划分元素 $x$ 的策略仅能剪枝掉 $3\lfloor(n-5)/10\rfloor-1$ 个元素。

$$\begin{aligned}&3\lfloor(\frac{n-5}{5}-(k-1)\rfloor-1-(3\lfloor\frac{n-5}{10}\rfloor-1)\\&=3\lfloor\frac{n-5}{10}-k+1\rfloor\\&\approx(3\lfloor\frac{n-5}{10}\rfloor-1)\end{aligned}\qquad(2)$$

显然，由公式（2）可知，相比经典线性时间选择算法，我们的基准划分元素的选择方法可以将剪枝效率提高大约 1 倍。

但是，仅考虑 $k$ 值是不行的，因为有可能出现两种特殊情况：（1）$(n\text{-}5)/5<k<n$。 这时中位数的数目小于 $k$ 个，此时不能再继续选择第 $k$ 小的中位数；（2）$(n\text{-}5)/10<k<(n\text{-}5)/5$。此时若选择第 $k$ 小的中位数反而会使划分效果变差。因此，我们提出一种基准划分元素的“动态选择”策略。在基准划分元素的选择中，选择中位数中第 $min((n\text{-}5)/10, k)$小的元素作为基准划分元素，即可根据不同的查询问题自动调节基准划分值的大小。下面分析这种方法的可行性和优越性。

在递归的任意一步，$n$ 为当前元素序列中元素的个数，需返回序列中第 $k$ 小的元素值（$n<75$ 时不再递归，直接排序后选择）。

（1）当 $1\le k\le(n\text{-}5)/10$ 时， $min((n\text{-}5)/10, k)=k$，故选择第 $k$ 小的中位数 $z$ 作为基准划分元素。

当各组的元素都按照每组中位数的值从小到大排好序后，第 $k$ 组的后两个元素和第 $k+1$ 至 $(n\text{-}5)/5$ 组后三个元素必定大于元素 z 的值。此时，将元素 z 作为基准划分元素后即可剪枝掉这部分，故可以保证至少减去 $3\lfloor(n-5)/5-(k-1)\rfloor-1$ 个元素。与经典线性时间选择算法中至少剪去 $3\lfloor(n-5)/10\rfloor-1$ 个元素相比，多剪掉了 $3\lfloor(n-5)/10-(k-1)\rfloor$ 个元素。尤其当 $k<<n$ 时，用新方法剪枝掉的元素数目近似为原算法的两倍。

而在剩下的元素中，第 1 至 $k$-1 组的前三个元素和第 $k$ 组的前两个元素都必定比 $z$ 小。因此，在划分后至少有 $3k$-1 个元素位于比 $k$ 小的一侧，可以保证第 $k$ 小的元素包含在留下的元素中，即不会漏解或无解。

（2）当$(n\text{-}5)/10<k\le n$ 时，$min((n\text{-}5)/10, k)= (n\text{-}5)/10$。此时与经典算法相同，依然选择中位数的中位数作为划分的基准。

综上所述，对 $1\le k\le n$ 的任意整数 $k$， 设每一次划分需在当前所剩的 $n'$个元素中选择第 $k'$小的元素 $z$，此时应选择每组中位数中第 $m$ 小的元素，则对 $m$ 有公式（3）：

$$m=\begin{cases}k' & k'\le\dfrac{n'-5}{10}\\[2ex]\dfrac{n'-5}{10} & k'>\dfrac{n'-5}{10}\end{cases}\qquad(3)$$

由于改进后的算法可以在当 $k<(n\text{-}5)/10$ 时，相比经典线性时间算法剪枝掉更多的元素。因此，虽然其时间复杂度和经典算法均为线性时间复杂度 $O(n)$，但是在很多情况下优于经典线性时间选择算法。尤其在实践应用中，当 $k$ 值较小时，由于需要进行多次函数递归，$1\le k\le(n\text{-}5)/10$ 的情况会发生多次。因此，在运行时间上改进的算法在效率上会呈现一个明显的优化。

基于以上策略，我们提出了一个新的改进的线性时间选择算法，表 2 中的算法 2 给出了详细的描述。

表 2：改进的线性时间选择算法

**Algorithm 2:** ***NewSelect***$(a, p, r, k)$

```
1.  if   r-p<75
2.     then sort a[p:r] ;
3.     return a[p+k-1];
4.  n←r-p+1;
5.  for i←0 to (n-5)/5
6.     将 a[p+5*i]至 a[p+5*i+4]的第三小元素与 a[p+i]交换位置;
```

7. $m$←min(($n$-5)/10, $k$)；
8. $z$←*NewSelect*($a$,$p$, $p$+($n$-5)/5,$m$)；
9. $i$←*Partition*($a$,$p$,$r$,$z$)//此处的 Partition 函数与表 1 中相同；
10. $j$←$i$-$p$+1；
11. **if** $k$<=$j$
12. **then return** *NewSelect*($a$,$p$,$i$,$k$)；
13. **else return** *NewSelect*($a$,$i$+1,$r$,$k$-$j$)；

表 2 的算法 2 与经典线性时间选择算法的时间复杂度均为 $O(n)$。但是，当 $k$ 值较小时，由于有更多的元素被剪枝掉，所以在此种情况下，算法的性能较经典线性选择算法有较好的提升。

## 3 实验

我们使用 *C++*语言进行了算法的实现，并在 *Windows7* 环境下进行测试。测试方法为：随机产生 $n$ 个无重复整数，并对 $k$ 进行从 1 到 $n$ 的循环赋值。每一轮循环中建立大小为 $n$ 的数组并随机填充，分别使用经典线性时间选择算法和本文中提出的改进的线性时间选择算法选择出第 $k$ 小的元素，并记录算法所耗费的时间。

由于 $n$ 个无重复整数序列是随机产生的，随机产生的元素序列对查询第 $k$ 小元素的算法中的 $k$ 值会产生随机化的影响，属于典型的舍伍德型概率型算法。因此，为准确的衡量算法的时间性能，我们在实验中采用平均时间作为算法性能的评价标准。我们对每种情况都进行了独立的 100 次测试，取平均值作为测试结果，并根据各种情况的测试结果的平均值，计算出本文算法的平均运行时间占原经典算法的百分比。

### 3.1 不同数据规模下的性能评测

图 3 为 $n$=1000 至 30000 时两种算法的平均运行时间图。这里的平均运行时间是对 $k$ 取 1 至 $n$ 的运行时间的平均值。在每一组实验中，两种算法均在同一组数据元素上进行测试。由图 3 可知，新算法在小规模数据集上的 $k$ 值的各种输入规模下的运行时间都小于经典线性时间选择算法。在该数据段，本文算法的运行时间平均为原经典算法的 80.84%。

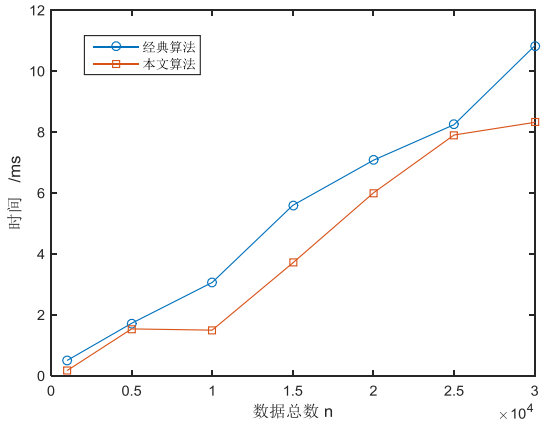


图 3：小规模数据集上的运行时间对比

为测试算法在更大规模数据元素集上的性能差异，图 4 中给出了 $n$ 增大至十万、百万级数据元素个数时的实验结果。

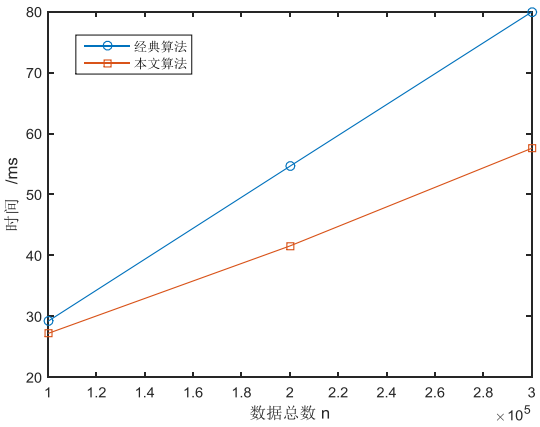


图 4：大规模数据集上的运行时间对比

图 4 显示了当 $n$ 为十万级别的时两种算法平均时间性能的比较。由图 4 可知，随着输入规模 $n$ 的增大，本文的改进的线性时间选择算法的优势随之显著增加。该数据段下本文算法的运行时间平均仅为经典算法的 74.95%。

在该组测试中，当 $n$ 的规模为一百万数量时，本文算法可以比经典线性时间选择算法平均提前 30*ms* 完成任务；当 $n$ 达到一千万数量时，新算法的优势则更加明显。该组实验测试表明，两种算法在平均性能上有着显著的差异。在第 4.2

节，我们进一步测试了两种算法对不同输入 $k$ 值的性能差异影响。

### 3.2 不同 k 值下时间性能的比较

第 3.1 节对两种算法的平均性能进行了测试，该节测试在不同的 $k$ 值输入下，两种算法的性能差异。

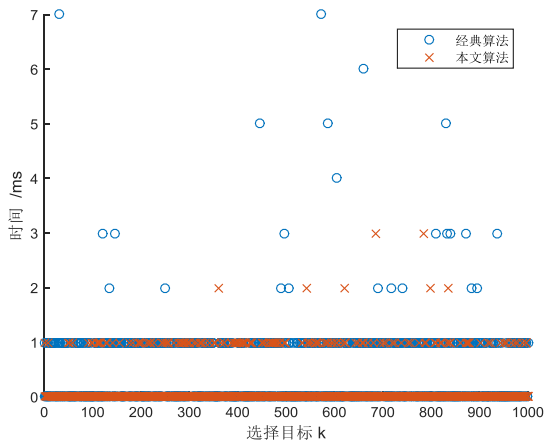


图 5：$n$=1000 时，不同输入 $k$ 值对应的时间

图 5 为 $n$=1000 时两种算法的实验结果。由图 5 可知，直观地看到新算法平均时间十分接近 0$ms$，而经典线性时间选择算法的平均时间更接近 1$ms$；其次，改进后算法的稳定性更强，可以看出经典线性时间选择算法对某些 $k$ 值的运行时间远高于平均时间，最差的情况可以达到 7$ms$，而新算法中很少有极端情况发生，且最差也能在 3$ms$ 内完成任务。这种情况下本文算法的平均运行时间仅为原算法的 32.79%。

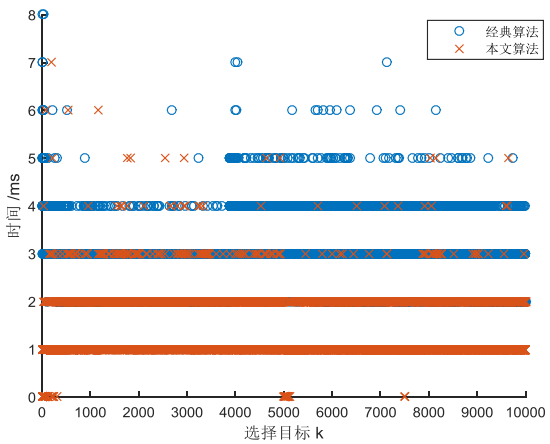


图 6：$n$=10000 时，不同输入 $k$ 对应的时间

图 6 中 $n$=10000 时，两种算法的差异更加明显，经典线性时间选择算法的平均时间点聚集在 4$ms$ 附近，而本文改进算法平均只需要 1~2$ms$，对某些情况甚至 0$ms$ 就可以完成任务。这种情况下本文算法的平均运行时间是经典算法的 48.76%

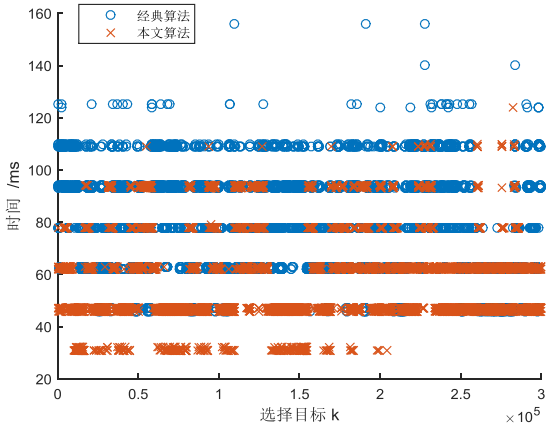


图 7：$n$=300000 时，不同输入 $k$ 对应的时间

如图 7 所示，对于 $n$=300000 的输入规模，本文改进算法几乎只需要经典线性时间算法一半的时间即可完成选择。而且几乎没有超过 100$ms$ 的情况出现。即使基数已经较大，本文算法也能以经典算法 72.01%的时间完成任务。

图 5、6 和 7 的实验中给出了在固定的输入元素序列规模下，不同的 $k$ 值输入对算法时间性能的影响。该组实验结果表明，本文算法在绝大多数的情况下都优于经典线性时间选择算法。

### 3.3 较小 k 值下时间性能的比较

第 3.1 和 3.2 对两种算法的全局时间性能进行了对比。然而，在实际应用中，通常更倾向于从给定的 $n$ 个元素序列中选择“较小的某个元素”，而非经典线性时间选择算法中比较接近中间的某个数（中位数除外）。例如，在田径比赛中，通常只需要选出长跑比赛中用时最短的前三名选手。所以选择 $n$ 个元素中较小的某些元素的情形在实际应用场景中有着更广泛的应用。因此，研究对较小的 $k$ 值处理更快的算法也

尤为迫切。基于实际的应用需求，我们测试了在 $n$ 较大的情况下（以一百万和一千万为例），本文算法和经典线性时间选择算法对较小的 $k$ 值输入的运行时间性能比较。

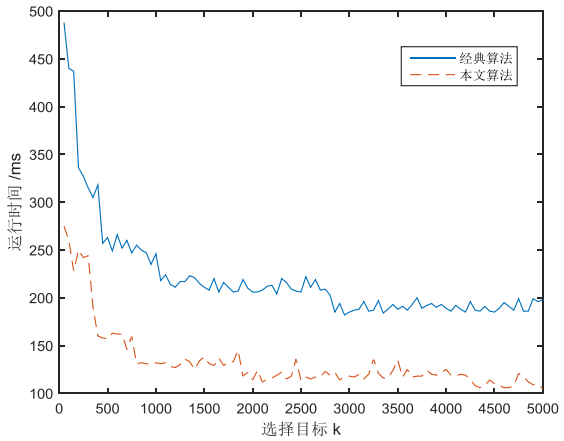


图 8：n=1000000，1≤$k$≤5000 时，运行时间与 $k$ 的关系

图 8 中，当元素数目为一百万时，在 1~5000 的范围中均匀选取一百个值作为选择目标 k。可以看出本文算法在每个点都比经典线性时间选择算法快许多，在这种情况下本文算法的效率是经典算法的 2 倍左右。

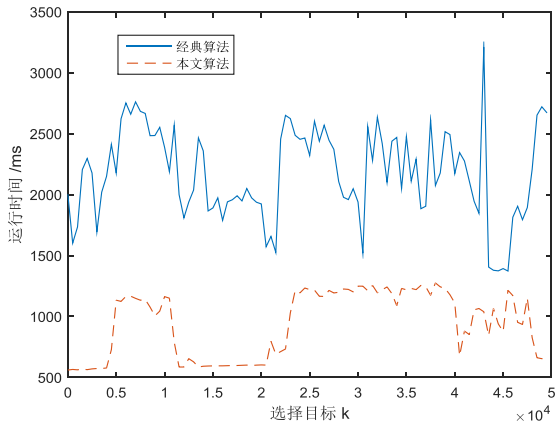


图 9：$n$=10000000，1≤$k$≤50000 时，运行时间与 $k$ 的关系

图 9 中，元素数目增长到了一千万。从 1~50000 的范围中均匀选择 100 个值作为选择目标 k，运行结果是对所有选定的 $k$ 值，本文方法都优于经典线性时间选择方法。经典线性时间选择算法的运行时间在多数情况下是本文算法的两倍甚至更多，而且结果不稳定，会出现较多的极端情况，需要超过 3000$ms$ 才能找出目标元素，本文算法的最长时间却不到 1500$ms$。

以上评测数据表明，改进后的线性时间选择算法不仅在全局上平均时间性能要优于经典线性时间选择算法，而且在 $k$ 较小的情况下，本文算法有着更出色的效率和稳定性。

## 4 延展和改进

### 4.1 关于重复元素的问题

在本文涉及的算法中，为了方便阐述算法，均假设序列中无重复元素，这是为了保证在某个划分基准调用函数 *Partition* 对元素序列进行划分之后所得的两个子序列的长度均不超过限定范围。而当序列中存在重复元素时，应当在划分后（即表 1 算法中第 8 行）添加一条语句，将所有与基准元素相等的元素集中在一起。若这样的元素个数 $m$≥1，并且 j≤$k$≤$j$+$m$-1 时，就不必再递归调用，直接返回 $a[j]$即可；否则，最后一行（即表 1 算法中第 12 行）修改改为调用 *Select*（$i$+$m$+1，$r$, $k$-$j$-$m$）即可。

### 4.2 进一步优化

根据分析及测试结果可知，当 $k$ 较小的时候，本文提出的改进的线性时间选择算法速度更快。但是，当 $k$>($n$-5)/10 的时候却没有 $k$ 较小时效率高，这是由算法“选择第 $k$ 小”的特征所决定的。但这一问题只需要在原基础上增加一步对 $k$ 的判断以及转化即可解决，即“当 $k$>$n$/2”时，转换为从原数组中选择第($n$-$k$+1)大的元素，这样只需要将表 2 中算法的排序顺序修改为由大到小排列即可。

## 5 结语

经典线性时间选择算法虽然可在线性时间复杂度下高效求解第 $k$ 小问题，但由于其选择固定的“中位数的中位数”处的元素作为基准划分元素，导致当 $k$ 值较小时效率较低。针对该问题，本文对经典的线性时间选择算法进行了改进，并提出了一个新的基准划分元素动态选择方法。充分的实验测

试可知，当 $k$ 值较小时，改进的算法在时间效率上具有较好优越性和更好的稳定性。本文提出的对线性时间选择算法的改进可广泛应用于实际应用之中。而且经过进一步的延展和优化，该算法可以在有若干相等元素时仍然适用，并且当 $k$ 值较大时，运行效率也可有明显提升。

## 参考文献


[1] BLUM M, FLOYD R W, PRATT V, et al. Time bounds for selection[J]. Journal of Computer and System Sciences, 1973, 7(4): 448-461.

[2] CORMEN T H, LEISERSON C E, RIVEST R L, et al. 算法导论[M]. 潘金贵，顾铁成，李成法等译. 2 版. 北京:机械工业出版社, 2008：109-115 (CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to algorithm[M]. PAN J G, GU T C, LI C F translated 2nd ed. Beijing: China Machine Press. 2008：109-115 )

[3] 王晓东，计算机算法设计与分析[M]，4 版. 北京：电子工业出版社， 2014：26-29 (WANG X D. The design and analysis of computer algorithm[M]. 4th ed.Beijing: Electronic Industry Press. 2014：26-29)

[4] PATERSON M. Progress in selection[J]. Lecture Notes in Computer Science, 1997, 1097:368--379.

[5] FONG K，LI M M，LIANG H Y ,et al.Average-case complexity of the min-sum matrix product problem[J],Theoretical Computer Science, 2016,609:76-86.

[6] AHTAI M, FELDMAN V, HASSIDIM A, et al. Sorting and selection with imprecise comparisons[M] Automata, Languages and Programming. Springer Berlin Heidelberg, 2015:37-48.

[7] BARBAY J,GUPTA A,SATTI S R,Near-optimal online multiselection in internal and external memory[J],Journal of Discrete Algorithms,2016,36:3-17.

[8] BALOUCH A. PCM-oMaRS algorithm: parallel computation of median- omniscient maximal reduction steps, [J] American Research Journal of Computer Science and Information Technology. 2015,1(1):1-11

[9] BALOUCH A. Optimization of position finding step of PCM- oMaRS Algorithm with statistical information[J]. International Arab Journal of Information Technology, 2016, 13(1A):208-2014.

[10] RIANI M, PERROTTA D, CERIOLI A. The Forward Search for Very Large Datasets[J]. Journal of Statistical Software, 2015, 67(Code Snippet 1)

[11] PAKHIRA M K. A linear time-complexity k-means algorithm using cluster shifting[C],IEEE Sponsored International Conference Iccicn-. 2014:1047-1051.

[12] DAVIDSON S, KHANNA S, MILO T, et.al. Top-k and Clustering with Noisy Comparisons,ACM Transactions on Database Systems,2014, 39(4):35:1-39

[13] KINDLMANN G, CHIW C, SELTZER N, et al. Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis[J]. IEEE Transactions on Visualization & Computer Graphics, 2016, 22(1):1-1.

[14] HOARE C A R. Quicksort[J]. Computer Journal, 1962, 5(1):10-15.



作者信息：

孙妍超(1995-)，女，天津市武清县，四川大学本科生。

陈瑜（1974 年-），男，陕西洋县，博士，讲师，四川大学计算机学院，主要研究方向：进化计算与数据挖掘。

**SUN Yanchao**, born in 1995, undergraduate in Sichuan University

**CHEN Yu**, born in 1974, Ph. D, Lecturer.His research interests include evolutionary computation and data mining.


This work is partially supported by the National Natural Science Foundation of China (U1333113).

本文有任何疑问请联系作者：孙妍超，手机号是：15680801690，邮箱是 souniangao@163.com 。