An Efficient Sample-Based Pivot Selection Strategy for Quickselect

Dongmin Lee (이동민) [1]

[1]Dept. of Computer Science and Engineering, Seoul National University (서울대학교 컴퓨터공학부)

Introduction

- The choice of pivots for quickselect [1] affects its performance greatly.
- Common pivot strategies include random selection [1] and median-of-3 [2].
- This project improves on previous work on sampling-based pivot strategies [3] by estimating the order statistics of random samples with a **discrete beta distribution without replacement**.
- This estimate is used to derive a robust pivot strategy that is consistently efficient regardless of input characteristics.

Background

The following notational conventions are used.

A	The input array.
n	Size of the array.
k	Rank of the target element.
A_i	ith element of the array A.
$A_{(i)}$	ith smallest element of the array A.
$r_A(x)$	Rank of the element x in A.

At each iteration of the quickselect algorithm, a pivot p is chosen and the array is partitioned based on each element's relation to p. Then, the algorithm continues in the partition that contains $A_{(k)}$.

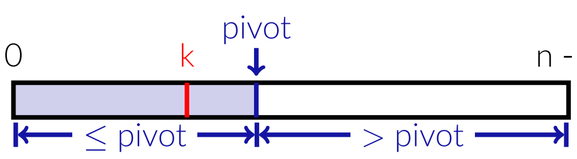

Figure 1. A visualization of the partition operation.

Importantly, the exact order of the chosen pivot can only be determined **after** the partition operation. Thus, choosing a good pivot efficiently is challenging.

If we choose a pivot that is close to $A_{(k)}$ (i.e. $r_A(p) \approx k$), the search space can be reduced very efficiently. Consider the following figure for an example of how two well-chosen pivots close to $A_{(k)}$ can drastically reduce the search space.

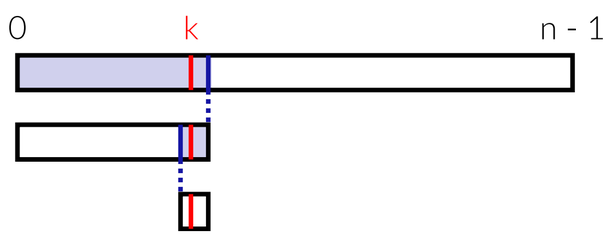

Figure 2. Pivots close to $A_{(k)}$ can reduce the search space quickly.

However, proximity to $A_{(k)}$ by itself is not enough. Consider the following figure, where a pivot is repeatedly selected to be slightly to the left of $A_{(k)}$.

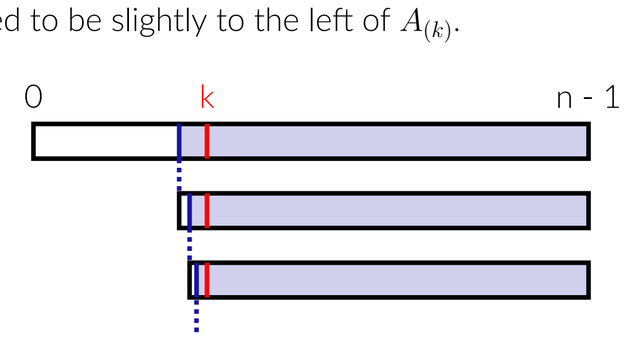

Figure 3. Simply choosing pivots near $A_{(k)}$ may have poor results.

Even though the chosen pivots are close to $A_{(k)}$, the search space is not reduced effectively. Thus, not only is it important for the pivot's rank to be close to k, but it should also be at the **correct** side of k. That is, the pivot's rank should be closer to the middle of the array than k.

Then, how can we choose pivots that are good: pivots that are ranked near k yet closer to the middle than k?

By **randomly sampling** the array and carefully choosing a pivot from the sample, we can efficiently choose a pivot that has a high probability of satisfying both conditions.

Methods

The basic architecture of the sampling pivot strategy is to randomly sample m values from the array, then choose a pivot from the sample S.

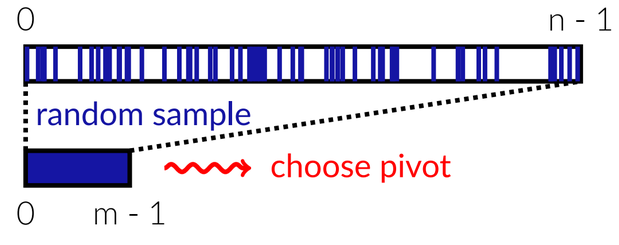

Figure 4. A visualization of the sampling pivot strategy.

The sampling can be done in-place by using a truncated Fisher-Yates shuffle. [4] (This creates a random sample **without replacement**.)

Intuitively, it is reasonable to assume that the $\lfloor \frac{m}{n}k \rfloor$th smallest value in S will estimate $A_{(k)}$.

More formally, the probability distribution of $S_{(i)}$ can be derived as follows.

$$P\left[r_A\left(S_{(i)}\right) = x\right] = \binom{x}{i}\binom{n-x-1}{m-i-1} \Big/ \binom{n}{m}$$

$$E\left[r_A\left(S_{(i)}\right)\right] = \frac{n+1}{m+1}(i+1) - 1$$

$$\sigma\left[r_A\left(S_{(i)}\right)\right] = \sqrt{\frac{(n+1)(n-m)(m-i)(i+1)}{(m+1)^2(m+2)}}$$

Thus, if we select the $\frac{(k+1)(m+1)}{n+1} - 1$'th smallest value from S, its rank in A will estimate k.

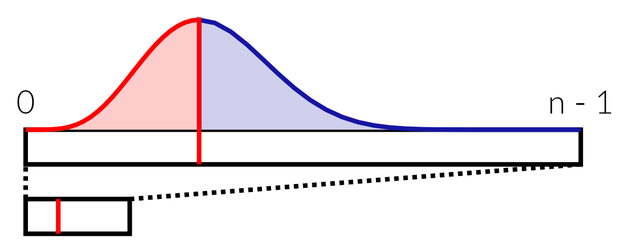

Figure 5. The probability distribution of $r_A\left(S_{(i)}\right)$.

However, simply choosing values close to $A_{(k)}$ is not a good pivot strategy. Note that the red part of the distribution in figure 5, which signifies a poor pivot (values further away from the middle than $A_{(k)}$), is very large.

To minimize the probability of choosing a bad pivot, the rank to select can be **shifted** towards the middle proportional to $\sigma\left[r_A\left(S_{(i)}\right)\right]$. A larger shift reduces the probability of choosing a bad pivot, but also increases the average distance between the pivot and $A_{(k)}$.

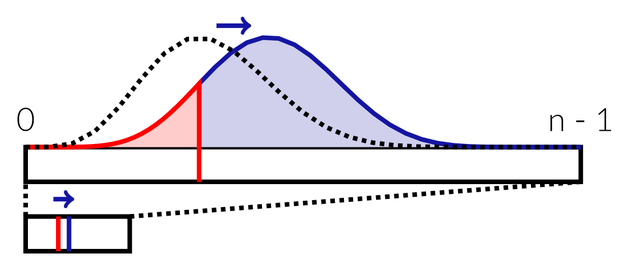

Figure 6. The shifted probability distribution.

Note that the red area in figure 6 is much smaller than the one in figure 5 thanks to the shift.

Finally, using the fact that the search space reduces to $\frac{n}{\sqrt{m}}$ every two iterations, the optimal m can be shown to be $\sim n^{\frac{2}{3}}$. Thus the final algorithm is as follows.

Function ChoosePivot(A, n, k)

if $n <$ *THRESHOLD* **then**
  return *random number in* $[0, n)$;
else
  $m \leftarrow n^{\frac{2}{3}}$;
  $S \leftarrow$ randomly sample m elements from A;
  $i \leftarrow \frac{m+1}{n+1}(k+1) - 1$;
  $\sigma \leftarrow \sqrt{\frac{(n+1)(n-m)(m-i)(i+1)}{(m+1)^2(m+2)}}$;
  Shift i towards $\frac{m-1}{2}$ by $2\frac{m}{n}\sigma$;
  return *QuickSelect*(S, m, *round*(i));
end

Results

The performance of four quickselect algorithms was measured on various datasets. Three of them ('random', 'ninther', and 'sampling') are identical except for the pivot strategy, and 'libstdc++' uses GNU libstdc++'s implementation of std::nth_element().

For each benchmark, the input data was 100 randomly generated arrays of 2×10^6 uniformly distributed integers.

The first experiment measured the performance of each algorithm for different values of k. The performance of each algorithm was measured for $k = 0$ to 2×10^6 with an interval of 10^5 (21 measurements in total).

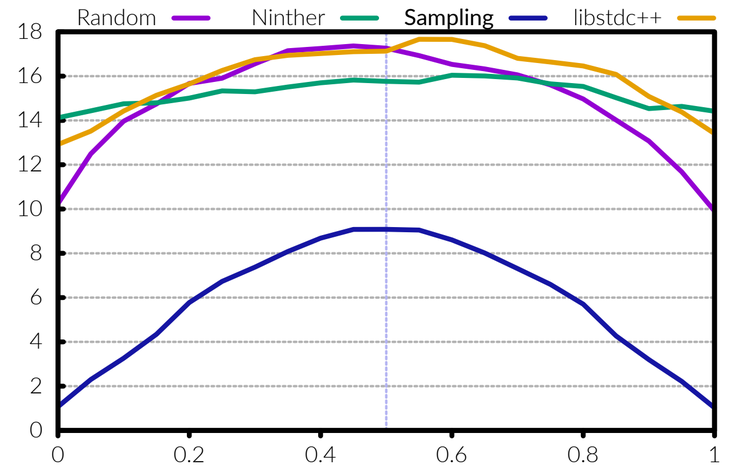

Figure 7. The relation between k/n (the relative position of the target) and the time taken (in ms) for each algorithm.

Thanks to its adaptive nature, the 'sampling' strategy excels for k values farther from the middle. It outperforms every other algorithm significantly for all k.

The next experiment measured the performance of calculating the median of the inputs.

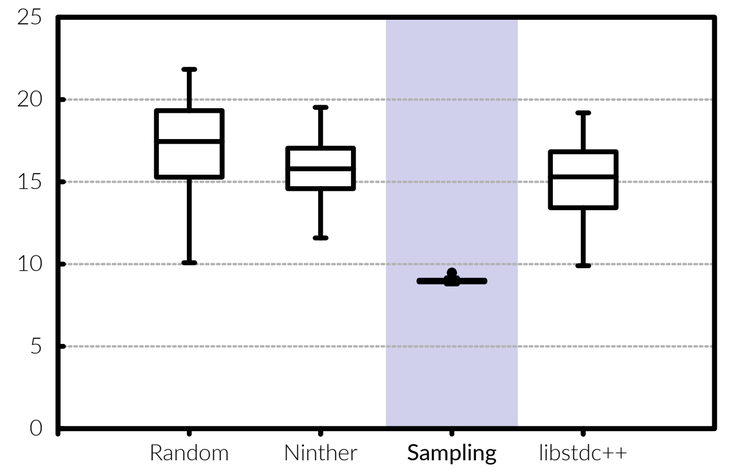

Figure 8. A box plot of the time taken (ms) for each algorithm to calculate the median for 100 datasets.

While the other algorithms vary in performance significantly depending on the input, the 'sampling' strategy is remarkably consistent. Its worst case is faster than the best case of the other algorithms.

Conclusion

The application of statistical analysis to design a pivot strategy for quickselect was a success. The new pivot strategy significantly outperformed existing algorithms, and was extremely consistent over different inputs.

Future work to rigorously verify the theoretical basis of this algorithm may prove to be fruitful.

References

[1] C. A. Hoare, "Algorithm 65: find," *Communications of the ACM*, vol. 4, no. 7, pp. 321–322, 1961.

[2] P. Kirschenhofer, H. Prodinger, and C. Martinez, "Analysis of hoare's find algorithm with median-of-three partition," *Random Structures & Algorithms*, vol. 10, no. 1-2, pp. 143–156, 1997.

[3] C. Martínez, D. Panario, and A. Viola, "Adaptive sampling strategies for quickselects," *ACM Trans. Algorithms*, vol. 6, jul 2010.

[4] R. A. Fisher and F. Yates, *Statistical tables for biological, agricultural and medical research*. Hafner Publishing Company, 1953.

[5] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, R. E. Tarjan, *et al.*, "Time bounds for selection," *J. Comput. Syst. Sci.*, vol. 7, no. 4, pp. 448–461, 1973.

[6] J. W. Tukey, "The ninther, a technique for low-effort robust (resistant) location in large samples," in *Contributions to Survey Sampling and Applied Statistics*, pp. 251–257, Elsevier, 1978.