ISSN:0975-9646

# Adaptive Sorting Using Machine Learning

Somshubra Majumdar, Ishaan Jain, Kunal Kukreja, Professor Kiran Bhowmick

*Computer Engineering Department, Mumbai University*
*Dwarkadas J. Sanghvi College of Engineering*
*Plot No. U-15, J.V.P.D. Scheme, Bhaktivedanta Swami Marg, Vile Parle (West), Mumbai-400056, India.*

***Abstract-*** **Sorting algorithms and their implementations in modern computing requires improvements in sorting large data sets effectively, both with respect to time and memory consumed. This paper is aimed at reviewing multiple adaptive sorting algorithms, on the basis of selection of an algorithm based on the characteristics of the data set. Machine Learning allows us to construct an adaptive algorithm based on the analysis of the experimental data. We reviewed algorithms designed using Systems of Algorithmic Algebra and Genetic Algorithms. Both methods are designed to target different use cases. Systems of Algorithmic Algebra is a representation of pseudo code that can be converted to high level code using Integrated toolkit for Design and Synthesis of programs, while the Genetic Algorithm attempts to optimize its fitness function and generate the most successful algorithm.**

***Keywords-*** **Sorting, Machine Learning, Object oriented programming**

## I. Introduction

Sorting is defined as the operation of arranging an unordered collection of elements into monotonically increasing (or decreasing) order. Specifically, S = {a1, a2 ….an} be a sequence of n elements in random order; sorting transforms S into monotonically increasing sequence S'= {a1 ', a2 '…… an '} such that ai '≤ aj' for 1≤ i ≤ j ≤ n, and S' is a permutation of S [1].

There are certain characteristics of a data set that can be preprocessed to obtain some valuable information about the data set itself. A few characteristics obtained from a data set are its size and the degree of pre-sortedness. Different sizes of a data set necessitate utilization of different algorithms to sort them. Pre-sortedness can be described as the degree to which the initialized data set is already sorted. A sequence of integers to be sorted could be characterized by more than its length, but also by its degree of pre-sortedness. Three measures of pre-sortedness are used [2]:

- The number of inversions (INV),
- The number of runs of ascending subsequences (RUN)
- The length of the longest ascending subsequence (LAS)

RUN metric is shown to be the most efficient. RUN is calculated as number of subsets of the data set that are already sorted divided by the number of elements in the data set itself. A data set sorted in the descending order will have a pre-sortedness value of 1 while a data set sorted in the ascending order will have a value equal to 1/n.

Using the aforementioned characteristics of the data set, we can make use of certain strategies to optimize the performance of the algorithm used to sort the set. Two main strategies that can be used are Machine Learning and Genetic algorithm. Machine learning allows us to classify the data for making decisions on which algorithm is optimal, while Genetic Algorithm modifies itself to optimize the performance of the algorithm. We discuss these two in detail below.

Machine learning explores the study and construction of algorithms that can learn from and make predictions on data [3]. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions [4], rather than following strictly static program instructions. We can categorize machine learning techniques based on the desired output of a machine-learned system [4]:

- Classification : Input sets are classified into one or more output classes based on a model
- Regression: the outputs are continuous rather than discrete values.
- Clustering: Inputs are divided into output groups. However, unlike classification, the groups are not known beforehand

Genetic algorithm is a search technique that simulates the process of natural selection. This technique is routinely used to obtain useful solutions to optimization and search problems. Genetic algorithm techniques are inspired by natural evolution - as in inheritance, mutation, selection and crossover. In such an algorithm, a set of possible candidate solutions to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered [5].

In this paper, in section 2, we discuss the features of the data set that we utilize, along with the various approaches to solve this problem. In section 3, we discuss our proposed model as well as methodology in creating the data sets and pre-sorting them. In section 4, we conclude with our future work.

## II. Literature Survey

An adaptive sorting algorithm is developed with the help of some pre-existing well known sorting algorithms, namely - Insertion sort, Shell sort, Heap sort, Merge sort and Quicksort. An adaptive sorting algorithm uses certain characteristics of the input dataset to select one or more sorting algorithms to sort the dataset. Generally pre-sortedness is computed using the RUNS metric, which is described as the number of sorted subsets or the data set, divided by the number of elements in that dataset.

Machine learning is similar to computational statistics in the sense that it also focuses on prediction-making. It has strong ties to mathematical optimization, which delivers implementations, theory and application domains to the field. Machine learning is used in a range of digital tasks where coding and implementing explicit algorithms is infeasible [4]. Example applications include spam filtering, speech and handwriting recognition, sentiment analysis, Natural Language Processing (NLP) and computer vision. Machine learning and pattern recognition can be viewed as two facets of the same field.

Algebraic algorithmics (AA in short) is an important domain of computer science, which was born from a collaboration of algebra, logic and algorithm schemes. It provides a standard for the knowledge about subject directions with the help of algebra and also deals with obstacles like standardization, specification of correctness and modification of algorithms. AA uses high-level abstractions of programs, represented by Systems of Algorithmic Algebras (SAA). One of the fundamental problems of AA is to increase the degree of flexibility of programs to particular use cases. Specifically, the problem can be solved at the disadvantage of argument-motivated creation of algorithm specifications by means of more complex algorithms [6].

Genetic algorithms are widely used in obtaining solutions for optimization and search problems. Evolution of the candidate solutions starts from a random population. This is an iterative process, where the population in each iteration is called as a generation. In every epoch, the fitness of every individual in the pool is evaluated such that the fitness is usually the value of the heuristic function in the optimization problem under consideration. A genetic algorithm requires a representation of the solution space (in the form of an array of bits) and a fitness function to test the solution space [8]. After the genetic representation and the fitness function is defined, a Genetic Algorithm begins by initializing a population of candidate solutions and then attempts to improve it through multiple applications of the mutation, crossover, inversion and selection operations on the population [8].

The authors of the paper “Optimizing Sorting with Machine Learning Algorithms” [7] talks about two methods of generating efficient sorting techniques. The first one uses machine learning algorithms to generate a function for specific target machine that is used to select the best algorithm. Their second approach builds on the first approach and constructs new sorting algorithms from a few fundamental operations. Using the array size, they are able to determine if the input set can fit into the cache memory and using the entropy of the input data, differentiate between the relative performances of radix sort with other comparison based sorting algorithms. Using sorting primitives as operations in genetic algorithms, they are able to create a composite algorithm which outperforms several other algorithms.

Yatsenko proposes the use of Decision Trees to classify the best sorting algorithm for the given data set [11]. They consider five sorting algorithms, mainly Insertion sort, Merge sort, Quick sort, Heap sort and Shell sort to analyses and decide the best algorithm for the given data set. They do this using various Decision Tree learning algorithms such as ID3, C4.5, NewId, ITrule and CN2. They then use SAA to formalize the algorithm used for conversion into C++ and Java code using IDS (Integrated toolkit for Design and Synthesis of programs). The drawbacks of this analysis is that there has been no consideration for multi-threaded algorithms which would drastically improve sorting time and also the fact that the study focuses on only smaller data sets (size < 100).

Before we analyze the adaptive sorting algorithm, we first need to analyze the performance of various standard sorting algorithms and their time and space complexities, expressed using the Big O notation.

TABLE 2.1 Comparison of Sorting Techniques Based on the Parameters of Our Model Based on Previous Studies and Our Experimental Results [10]

| **Sorting Method** | **Best Case Time Complexity** | **Worst Case Time Complexity** | **Space Complexity** |
|---|---|---|---|
| Bubble Sort | O(n) | O($n^2$) | O(1) |
| Insertion Sort | O(1) | O($n^2$) | O(n) |
| Shell Sort | O(n $\log^2$ n) | O($n^2$) | O(n) |
| Heap Sort | O(n) | O(n log n) | O(1) |
| Quicksort | O(n log n) | O($n^2$) | O(log n) |
| Merge Sort | O(n log n) | O(n log n) | O(n) |

Based on a preliminary analysis of the various sorting algorithms widely known, we can eliminate the use of certain algorithms that do not perform well for any case, or are too primitive to produce the output efficiently. Sorting techniques such as Bubble Sort and Selection sort have an average case complexity of O($n^2$) with best case complexity of O(n) and O($n^2$) respectively. Other considered sorting algorithms perform relatively better for all cases and as such negate the usage of Bubble and Selection Sort. The exception here is that while Insertion sort also has a complexity of O($n^2$), it’s best case complexity is only O(1), which means that there are edge cases in which insertion sort produces the output in least time as compared to the other algorithms.

## III. PROPOSED MODEL AND METHODOLOGY

### A. *Construction of Data Set:*

Since we do not have an example data set to utilize for our analysis, we begin by generating a data set consisting of integers generated using a uniform Gaussian random number generator. Data sets of sizes varying sizes from 50 to 1 million uniformly distributed integers is generated. Due to the requirement of preprocessed arrays, we create 7 such replicas of each data set, whose subsets are then sorted according to the given input parameters of “pre-sortedness”. Thus each replicated dataset is pre-sorted according to the vector of 1/n to 1 where n is the size of the data set. Thus we create 42 data sets each having nearly 100 sample arrays. The exception to this is the data set of 1 million integers, of which there are far fewer samples. This is because the sorting time of such large data sets using Insertion Sort and

Shell Sort is far too large to be compared with faster sorting algorithms such as parallel merge sort and quicksort. It can be assumed that either parallel merge sort or quicksort will be the winning algorithm when such large data sets are given to the model.

Figure 3.1 describes the distribution of the generated arrays each having been pre-sorted to various degrees of pre-sortedness. Due to the large number of smaller sized arrays, the number of arrays which are almost completely sorted is higher than any other type of array.

## B. *Feature Selection*

For our model, we will be using two features that are characteristic to each of the data sets. The first attribute will be the size of the array, as larger data sets cannot be sorted in an efficient manner using algorithms such as Insertion Sort and Shell Sort. The second attribute will be the pre-sortedness of the array, computed as the “RUNS” metric.

$$RUNS = \frac{Number\ of\ ascending\ subsequences}{Number\ of\ elements\ in\ the\ array}$$

Thus, for a completely sorted array, the RUNS value will be 1 / size of the array, whereas for an array sorted in descending order, RUNS value will be 1.

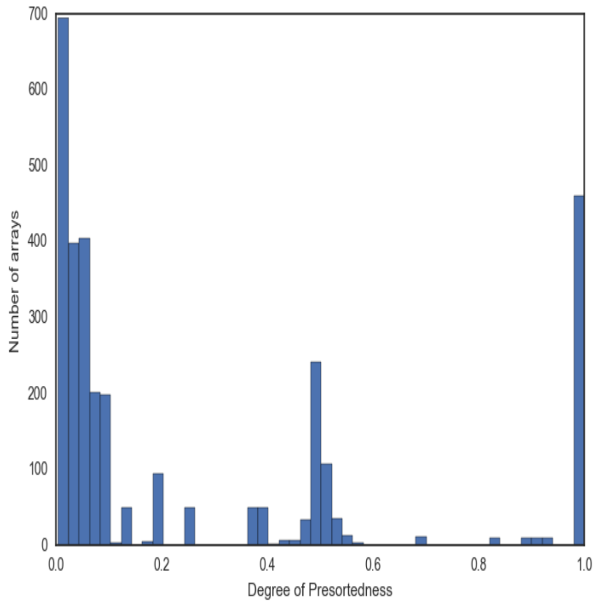

**Figure 3.1 Number of instances of arrays with different degrees of pre-sortedness (RUNS value)**

## C. *Preprocessing*

These pre-sorted data sets are then sorted using Insertion Sort, Shell Sort, Heap Sort, Merge Sort, Quicksort, and Parallel Merge Sort. The sorting time for each algorithm on each array of each data set is computed and stored for later analysis. Alongside these results, we also determine the winning algorithm that is the algorithm which uses the minimum time to sort the given data set. In case of ties, we choose the algorithm which performs the best on similar data sets. Algorithms tie when data set size is small, as multiple algorithms sort the data set in almost exactly the same amount of time. In such cases, we see which algorithm generally performs as the best algorithm in other data sets of the same size, and so assume it is also the current winner.

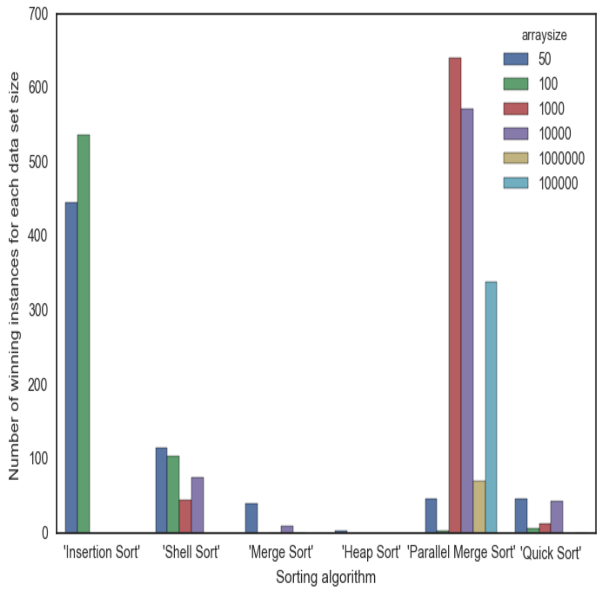

**Figure 3.2 Number of instances in dataset where algorithm is the winner**

In Figure 3.2, we see that depending solely on the data set size, each algorithm is well suited to sort a given dataset of some size in the shortest time possible. It is seen that heap sort performs poorly for any given data set size, at least in comparison to other more efficient algorithms such as Merge sort and Quicksort. Due to this, we will be removing the negligible number of instances where heapsort outperforms other algorithms from the final dataset.

## D. *Proposed Model*

Following Yatsenko’s model [11], we utilize a Decision Tree to analyze and learn the features of the array size and degree of pre-sortedness and the winning algorithm as the target feature. We also use Gaussian Naive Bayes to classify the features, as well as train a multiclass Support Vector Machine with the Linear Dual Coordinate Descent as the optimization algorithm. The decision tree will also attempt to classify the data set with the splitting criterion as the gini score or with the information entropy score. We will also attempt to limit the depth of the tree to prevent overfitting. Naive Bayes and multi class Support Vector Machine are also used to cross validate the classification results.

While Yatsenko’s model [11] focused on smaller data sets of size smaller than 100 elements, our focus is on larger data sets of sizes in the range of 50 elements to 1 million elements. We also focus on sorting large data sets using parallel algorithms and the threshold where thread creation cost is insignificant compared to the gain in sorting speed. Our model also attempt to measure the optimal threshold for dividing a data set into partitions for improving the performance of Merge Sort, Quick Sort and Parallel Merge Sort.

## IV. CONCLUSION

The experiment is aimed at proposing a method to sort large data sets using Machine Learning while reviewing other methods such as Genetic Algorithms. Machine Learning allows us to build an algorithm to sort data sets based on their characteristics. The characteristics that affected the sorting time of the algorithm were then identified based on the results obtained in Figure 3.2. A means of representing these characteristics has been developed to facilitate error free classification by using a Machine Learning algorithm. These characteristics i.e size and pre-sortedness, will be the features of our data set, which acts as the input to our supervised classification learning algorithm.

The prospects of further investigations in this direction are to review different classification algorithms to pick the best one. Based on the results obtained, we can add more features that improve our classification. This classification hence obtained will help us pick the best sorting algorithm for a given data set.

## REFERENCES

[1] M.J. Quinn, "*Parallel Programming in C with MPI and OpenMP*" Tata McGraw Hill Publications, 2003, p. 338

[2] Guo, H.: "Algorithm selection for sorting and probabilistic inference: A machine learning-based approach". Ph.D. dissertation, Kansas State University (2003)

[3] Ron Kohavi; Foster Provost (1998). "Glossary of terms". *Machine Learning* 271–274.

[4] C. M. Bishop (2006). *Pattern Recognition and Machine Learning*. Springer.ISBN 0-387-31073-8.

[5] Mitchell, Melanie (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press. ISBN 9780585030944.

[6] Doroshenko, A., Tseytlin, G., Yatsenko, O., Zachariya, L.: Intensional Aspects of Algebra of Algorithmics. Proceedings of International Workshop "Concurrency, Specification and Programming" (CS&P'2007), 27–29 September 2007, Lagow (Poland) (2007)

[7] Li, Xiaoming, Maria Jesus Garzaran, and David Padua. "Optimizing Sorting With Machine Learning Algorithms." *2007 IEEE International Parallel and Distributed Processing Symposium* (2007): n. pag. Web.

[8] Whitley, Darrell (1994). "A genetic algorithm tutorial". *Statistics and Computing* **4** (2): 65–85. doi:10.1007/BF00175354

[9] "The Analysis of Heapsort". *Journal of Algorithms* **15**: 76–100.doi:10.1006/jagm.1993.1031.

[10] Donald Knuth. *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0.

[11] Olena Yatsenko. (2011). On Application of Machine Learning for Development of Adaptive Sorting Programs in Algebra of Algorithms, *Concurrency, Specification and Programming*, September 28-30, Pułtusk, Poland, pp. 577-588