A publication of ISCA*:
International Society for Computers
and Their Applications

INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS

TABLE OF CONTENTS

* "International Journal of Computers and Their Applications is abstracted and indexed in INSPECT and Scopus."

Volume 27, No. 3, Sept. 2020 ISSN 1076-5204

International Journal of Computers and Their Applications

A publication of the International Society for Computers and Their Applications

ISCA Headquarters···•···278 Mankato Ave, #220, Winona, MN 55987···•···Phone: (507) 458-4517
E-mail: isca@ipass.net • URL: http://www.isca-hq.org

Guest Editors' Note: Special Issue from the ISCA CATA 2020

This Special Issue of the IJCA is a collection of four refereed papers, which are extensions of conference papers selected from the *35th International Conference on Computer and Their Applications* (CATA 2020). Each paper was evaluated by at least two reviewers, judging the originality, technical contribution, significance and quality of the manuscript. Following CATA 2020, a number of high-quality papers were recommended by the Program Co-Chairs to be considered for publication in this Special Issue of the IJCA, resulting in the set of papers provided here. The papers in this special issue cover a wide range of research interests in areas of computers and applications. The topics and main contributions of the papers are briefly summarized below.

Ramblin Cherniak, Qiang Zhu and Sakti Pramanik, in their paper entitled *Update Methods for Maintainability of a Multidimensional Index on Non-ordered Discrete Vector Data,* investigate the issue of efficient processing of box queries of large datasets from multidimensional data. Different approaches for BoND-tree operations, including insertion and deletion tasks, are examined; in particular a general update procedure is developed and applied to a Top Down Update Method and a Bottom Up Update Method. Experiments were conducted to evaluate efficiency, measured in terms of the disk I/Os for performing the updates and effectiveness, measured by the box query I/Os on the resulting BoND-tree after the updates. These tests were performed for both synthetic datasets and real genome datasets. The authors show that both methods perform well in terms of effectiveness while the bottom-up update method provides improved efficiency, particularly for small dimensional data vector cases.

Maher Selim, Ryan Zhou, Wenying Feng and Omar Alam focus on the topic of statistical machine learning models in their paper entitled *Studying Error Propagation for Energy Forecasting Using Univariate and Multivariate Machine Learning Algorithms.* In particular, the authors develop a multivariate framework using generated features that impact recurrent forecasts which reduce compounded future errors. These features convert a univariate model into a multivariate model and the strategy is then applied to four machine learning models: Linear Regression, Support Vector Regression, Long Short-Term Memory neural networks, and the XGBoost regression. Then the four multivariate models are investigated and compared in terms of forecasting accuracy for different time windows. Finally, they apply their approach to long-term energy forecasting. Results show that both short- and long-term forecasting performance are improved using their non-recursive multivariate models.

The focus of the paper *A Unity Framework for Multi-Player VR Applications* by Alexander Novotny, Rowan Gudmundsson, and Frederick C Harris, Jr., is on the development of a framework for multi-user virtual reality activities. In particular, the authors present their work on building a peer-to-peer VR network with associated user interfaces, avatars and voice-chat capabilities. They provide network traffic performance analysis such as host network traffic versus number of connected users and host mean traffic network traffic versus update speed, and show some examples of the additional tools developed for the VR environment. Finally, the paper discusses how the user can easily interact with an object using their proposed VR framework, a function which is crucial for all virtual reality games on the market.

In their paper *Predicting Cerebral Aneurysm Rupture using Gradient Boosting Decision Tree based on Clinical, Computational Fluid Dynamics and Geometric Data*, authors Toshiyuki Haruhara, Masaaki Suzuki, Hideto Ohgi, Hiroyuki Takao, Takashi Suzuki, Soichiro Fujimura, Toshihiro Ishibashi, Makoto Yamamoto, Yuichi Murayama, and Hayato Ohwada discuss the problem of predicting strokes which affect a large population across the globe. Two gradient boosting decision tree methods - the Light GBM and the XGBoost, along with the support vector machine (SVM) method were used in a machine learning process that incorporates information from clinical, hemodynamic, and morphological data. From a set of over 6,000 cases collected over several years, the authors down-selected to 303 unruptured cases and 35 ruptured samples to investigate the validity of their approach. Hyperparameter tuning was performed and classification metrics (sensitivity, specificity and F-measure) were applied to analyze their models. Results show that these learning algorithms can accurately predict cerebral aneurysm, an indication of potential strokes, with the Light BMM model providing the best performance.

We hope you enjoy this special issue of the IJCA and we look forward to seeing you at a future ISCA conference. More information about the ISCA society can be found at http://www.isca-hq.org.

Guest Editors: *Gordon Lee*, San Diego State University, CATA 2020 Program Co-Chair
Ying Jin, California State University at Sacramento, CATA 2020 Program Co-Chair

September 2020

Update Methods for Maintainability of a Multidimensional Index on Non-ordered Discrete Vector Data

Ramblin Cherniak* Qiang Zhu*
University of Michigan - Dearborn, Dearborn, Michigan, USA
Sakti Pramanik†
Michigan State University, East Lansing, Michigan, USA

Abstract

There are numerous applications nowadays such as bioinformatics, cybersecurity, and social media that demand to efficiently process various types of queries on multidimensional (vector) data with values coming from a non-ordered discrete (categorical) domain for each dimension. The BoND-tree index scheme was recently developed to efficiently process so-called box queries on a large dataset in disk from such a vector data space. The index construction (insertion) and query algorithms were introduced in the original work. To maintain such an efficient index structure for a large dynamic dataset, one has to develop efficient and effective methods to support other operations including deletions, updates, and bulk loading. Although studies on deletions and bulk loading for the BoND-tree have been reported in early work, how to efficiently and effectively update the BoND-tree remains an open problem. In this paper, we first present a general update procedure which covers all scenarios including special cases for insertions and deletions. We then examine two approaches to updating the BoND-tree. The relevant algorithms and experimental evaluations are presented. Our study shows that using the bottom-up update method can provide improved efficiency, comparing to the traditional top-down update method, especially when the number of dimensions for a vector that need to be updated is small. On the other hand, our study also shows that the two update methods have a comparable effectiveness, which indicates that the bottom-up update method is generally more advantageous.

Key Words: Non-ordered discrete data; bioinformatics; mutlidimensional index; index maintenance; update method.

1 Introduction

There is an increasing demand to efficiently process various types of queries on non-ordered discrete vector data in contemporary applications such as genome sequence analysis, internet intruder detection, social network analysis, and business intelligence [25, 29, 31, 34, 40, 44, 45]. The vectors with non-ordered discrete values from the domain of each dimension constitute a vector space, called the Non-ordered Discrete Data Space (NDDS). For example, many genome sequence analysis techniques (e.g., DNA sequencing error correction [16] and back-translated protein query on DNA sequences [20]) rely on processing fixed-length subsequences, so-called k-mers, of one or more target genome sequences. $gacct$, $aatga$, and $tagga$ are examples of k-mers of length 5, which can be considered vectors (e.g., $< g,a,c,c,t >$ or simply "$gacct$") in a 5-dimensional NDDS with a domain consisting of non-ordered discrete values (i.e., nucleotide bases: a, g, t and c) for each dimension. Other applications [22, 23, 45] may deal with non-ordered discrete data from domains such as color, gender, season, IP address, social media symbols, user ids, and text descriptions.

One type of query used in many applications for an NDDS are called box queries. A box query retrieves vectors from a dataset in an NDDS that have values from a specified subset of the domain for each dimension. The BoND-tree was recently introduced as a new disk-based indexing structure specifically designed to support efficient processing of box queries for large datasets in an NDDS [7]. The algorithms for the insertion (build) and query operations of the BoND-tree were presented in the original work. However, to maintain an index structure for a dynamic dataset, efficient and effective algorithms for the deletion, update, and bulk loading operations are also needed.

Efficient and effective deletion strategies to remove vectors from the BoND-tree for a dynamically shrinking dataset in an NDDS were studied in [8], while an efficient and effective bulk loading method to build the BoND-tree for a very large input dataset in an NDDS was presented in [12]. To maintain the BoND-tree for a dynamically changing dataset in an NDDS (e.g., to capture changing variants in the genome sequence for a sick person developing a disease), we also need efficient and effective update methods for the index structure. A straightforward method for performing an update operation is to execute a deletion of the outdated vector followed by an insertion of the updated form of the vector. However, more efficient and effective approaches for performing updates in the BoND-tree are yet to be explored. In particular, alternative strategies for updates may be beneficial when taking into account considerations such as whether a particular update is independent from a subsequent update or whether an outdated vector targeted for an update is similar to its new representative form. This paper focuses on studying efficient and effective strategies to support updates for the BoND-tree.

*{rchernia, qzhu}@umich.edu
†sakti.pramanik@gmail.com

Studies on updates for index schemes in a multidimensional Continuous Data Space (CDS) such as the R-tree [17], the R*-tree [1] and their variants have been reported in the literature [3, 24, 38, 39, 48]. Updates with emphasis on moving objects for several R-tree based index trees [4, 26, 36, 41] have also been suggested. The problem of frequent updates for the hB-tree based trees [13, 27, 28] has been examined in [47]. Many CDS index structures including the X-tree [2] adopt the straightforward update approach through a deletion followed by an insertion. Note that the CDS indexing schemes rely on the natural ordering of underlying data and as such cannot directly be applied to an NDDS that is what we are interested in here.

The update issue has also been studied for some index trees that may be applicable to an NDDS. For example, index trees for a metric space [6] (such as the vantage-point tree [18, 42, 46] and the MVP tree [5]) and string indexing techniques based on the Trie structures [11] (such as the suffix tree [43]) have their update techniques reported in the literature [14, 15]. However, these are mainly memory-based structures, while we are interested in performing updates on a dynamic indexing scheme for a large dataset in disk. The M-tree [10] is a disk-based dynamic indexing structure developed for a metric space, which could be applied to an NDDS although its performance is not optimized for an NDDS due to its generality [30, 31]. Another disk-based dynamic indexing structure developed for a metric space is the MB+tree [19] that supports dynamic updates for similarity searches. However, an index scheme supporting similarity queries, such as range queries or k-NN queries, may not be effective for an index scheme that supports box queries. For example, this is evident in the contrasting splitting strategies for the insertion operations of the ND-tree [30], which is an index structure supporting similarity queries in NDDS, to those of the BoND-tree [7]. The BoND-tree was also found to prefer a different deletion strategy [8] from the traditional deletion strategies adopted for the ND-tree[37].

Effective and efficient update strategies are needed to support the maintenance of the BoND-tree. An update strategy yielding the BoND-tree that can support efficient box query processing after updates is said to be effective. An update strategy yielding minimal I/O overhead during the update procedure is said to be efficient.

In this paper, we will present a general procedure for the update operation of the BoND-tree. We will then examine two update strategies for the BoND-tree to support efficient box queries and present the experimental results to evaluate the efficiency and effectiveness of the proposed update methods. In particular, we present a new bottom-up update strategy for the BoND-tree that is efficient and effective for both general random updates and increasingly efficient for updates where the new updated vector is similar on many dimensions to the outdated vector. This is useful for applications where an outdated vector targeted for an update shares many dimensions in common with the updated representation of that vector. For example, a vector representing a DNA gene profile in a bioinformatics database may require such an update when a small percentage of dimensions have changed in the vector due to mutation or cancer. The preliminary results of this work were presented in [9].

The rest of the paper is organized as follows. Section 2 presents preliminary concepts that are useful in our discussions. Section 3 discusses our proposed update methods for the BoND-tree. Section 4 reports the experimental evaluation results. Section 5 concludes the paper.

2 Preliminaries

2.1 Terminology and Concepts

In this section, we present some geometric concepts for an NDDS [7, 21, 30] that are essential to our discussion on update strategies for the BoND-tree.

In general, a *d-dimensional* Non-ordered Discrete Data Space (NDDS) Ω_d is defined as the Cartesian product of d *alphabets* (*domains*): $\Omega_d = A_1 \times A_2 \times ... \times A_d$, where an alphabet $A_i (1 \leq i \leq d)$ consists of a finite number of non-ordered discrete values (letters). A *discrete rectangle* R in Ω_d is defined as $R = \sqcap_{i=1}^{d} S_i = S_1 \times S_2 \times ... \times S_d$, where $S_i \subseteq A_i (1 \leq i \leq d)$ is called the *i-th component set* of R. The *area* of rectangle R is defined as $|S_1| * |S_2| * ... * |S_d|$. We use a *span* to refer to the edge length of a particular dimension for a rectangle, which is normalized by the alphabet size of the corresponding dimension. The *discrete minimum bounding rectangle (DMBR)* of a set SV of vectors is defined as the discrete rectangle whose i-th component set $(1 \leq i \leq d)$ consists of all the letters appearing on the i-th dimension for the vectors in SV.

A box query q on a dataset in an NDDS is a query that specifies a set of values/letters for each dimension. Let $qc_i \subseteq A_i$ be the set of values allowed by box query q along the i-th dimension, where A_i is the alphabet for the i-th dimension $(1 \leq i \leq d)$ of Ω_d. The box query q with box/window $w = \sqcap_{i=1}^{d} qc_i$ will return every vector α in the dataset that falls within this box/window.

A *random-span box query* has the span of its i-component set on each dimension i $(1 \leq i \leq d)$ to be randomly chosen between 1 to $C \leq |A_i|$. For example, with an alphabet being $\{a,g,c,t\}$ across all dimensions where vectors have length $d = 3$, a random box query might have a window/box $w = \{c,t,g\} \times \{g\} \times \{t,g\}$. The measured spans for the three dimensions are 3, 1, and 2, respectively. This query will retrieve vectors having any value from $\{c,t,g\}$ on the 1st dimension, g on the 2nd dimension, and t or g on the 3rd dimension. 6 is the maximum possible number of unique vectors retrieved by such a query.

A *uniform-span box query* has the same span of its i-component set on each dimension i $(1 \leq i \leq d)$. For example, with an alphabet being $\{a,g,c,t\}$ across all dimensions where vectors have length $d = 3$, a uniform box query of span $= 2$ might have a window $w = \{c,g\} \times \{t,c\} \times \{a,g\}$. The edge lengths are uniform with a measured span of 2 for $d = 1,2,$ and 3. 8 is the maximum possible number of unique vectors retrieved by such a query.

We refer to an update with a certain percentage of *fixed dimensions* when we set static the given percentage of all the dimensions for an updating vector. We control this parameter to influence how similar an outdated vector and an updated vector can be. For example, if we set 60% of dimensions to be fixed when performing an update on a vector α (e.g., $tcacg$) of 5 dimensions, then 3 dimensions (e.g., $d = 1, 4$, and 5) are randomly selected to remain unchanged when generating a sample updated vector β (e.g., $tagcg$).

2.2 The BoND-Tree

The BoND-tree is a disk-based balanced index tree that grows upwards as vectors are inserted. The BoND-tree is made up of two types of nodes: non-leaf nodes and leaf nodes. Each non-root node N in the BoND-tree is represented by a corresponding entry in its parent node, which consists of a pointer to N and a DMBR covering all the vectors in the subtree rooted at N. Each entry in a leaf node consists of the indexed vector and a pointer pointing to an associated object in the underlying database, which may provide further information about the indexed vector. All the leaf nodes appear at the same level of the index tree.

Each node has a maximum number M of entries that can be contained in it. M is typically determined by the disk block size. If another entry is added into a node with M entries, this node is said to be *overflow*. Each node also has a minimum number m of entries that have to be contained in it. m is typically determined by a minimum space utilization criterion. If one entry is removed from a node with m entries, this node is said to be *underflow*. M (m) for a non-leaf node may be different from that for a leaf node.

Figure 1 illustrates an example of the BoND-tree with sample nodes and entries for a genome sequence dataset with alphabet $\{a, c, g, t\}$. Note that a general BoND-tree may have more than three levels of nodes. Vectors contained in a leaf node (e.g., node D) determine the DMBR of the corresponding entry in its parent node (e.g., node B) whose entries in turn determine the DMBR of the corresponding entry in its further up parent node (e.g., root node A in this case). For example, we can see that the root node A has an entry with component sets $\{a, c, t\}$ and $\{a, g, t\}$ of its DMBR on the first two dimensions, respectively. The first (resp. the second) component set is made up of the letters appearing on the first (resp. the second) dimension of all the entries in its child node B at the next lower level. As shown in the figure, node B has visible entries whose DMBRs' first component sets are $\{a, t\}$ and $\{c\}$, respectively. Node B has visible entries whose DMBRs' second component sets are $\{a, t\}$ and $\{g, t\}$, respectively. The first two component sets of the DMBR of the corresponding entry in node A indicate that no other letters are contained in the first two component sets of other entries in node B. The vectors in node D determine the DMBR $\{c\} \times \{g, t\} \times ...$ of the corresponding entry in the parent node B, assuming only c has appeared on the first dimension and only g or t has appeared in the second dimension for all vectors in node D in this example.

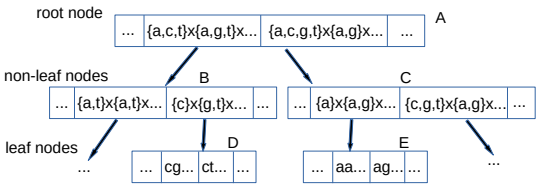


Figure 1: An example of the BoND-tree

When processing a box query using the BoND-tree, at each non-leaf node (starting from the root), we only need to follow its child node(s) whose DMBR(s) has an overlap with the query box/window. Those nodes whose DMBRs do not overlap with the query box/window are pruned during the query processing. Using Figure 1 as an example, let a query box/window $w = \{a, c\} \times \{c, t\} \times ....$ Such a query with w could potentially return, if present, vectors in the result set being "$ac...$", "$at...$", "$cc...$", and "$ct...$". Starting from the root node level, since w may overlap with the DMBR of the node B's entry in node A and the DMBR of node D's entry in node B, the processing of this query may follow the path node A $\rightarrow$ node B $\rightarrow$ node D. On the other hand, since w clearly does not overlap with the DMBR of node C's entry in node A, the subtree rooted at node C is pruned.

More details of the BoND-tree can be found in [7].

3 Update Methods for the BoND-Tree

An update operation is motivated by the need to modify an existing (outdated) vector in a given database/dataset in an NDDS. There are multitudinous reasons that may prompt an update operation in real-world applications. For example, a vector is found to have been inserted with an erroneous value(s) on some dimension(s); a vector is believed to have undergone a transformation on some dimensions since it was inserted or last updated; the alphabet for a particular dimension has been changed so that the vectors with obsolete values on that dimension must be updated.

3.1 General Update Procedure

In general, an update operation can be defined as follows: given an outdated vector α and an updated vector β, the update operation $Update(\alpha, \beta, S)$ on a database/dataset S is to ensure that S has β but not α after the update operation, i.e., $Update(\alpha, \beta, S) = (S - \{\alpha\}) \cup \{\beta\}$. Usually, α and β share many common values and differ only in a few dimensions.

An update procedure needs to account for the following four scenarios in regards to the existence of outdated vector α and updated vector β in the given dataset S.

Scenario 1: *Outdated α does not exist in S, and updated β does not exist in S either.* In this case, the desired β needs to be added into S. The outdated α is intended to be

removed, but it does not exist. Hence, nothing needs to be done for α. The update operation is actually degenerated to an insertion operation $Update(\alpha,\beta,S) = S \cup \{\beta\}$ in this case.

Scenario 2: *Outdated α does not exist in S, and updated β exists in S.* In this case, nothing needs to be done for α or β, i.e., $Update(\alpha,\beta,S) = S$.

Scenario 3: *Outdated α exists in S, and updated β exists in S.* In this case, the outdated α needs to be removed from S. Nothing needs to be done for β. The update operation is actually degenerated to a deletion operation $Update(\alpha,\beta,S) = S - \{\alpha\}$ in this case.

Scenario 4: *Outdated α exists in S, and updated β does not exist in S.* This is the most expected case for an update. In this case, the outdated α needs to be removed from S, and the desired β needs to be added into S, i.e., $Update(\alpha,\beta,S) = (S - \{\alpha\}) \cup \{\beta\}$.

Since the existence of α and β in S is typically unknown in advance, in general, an update procedure must address the above four scenarios to ensure that the database accurately reflects the intent of the update. Scenario 4 is the typical and most interesting update scenario that is considered in evaluating the efficiency and effectiveness of different update strategies in this paper.

For the BoND-tree T built for vectors from a given database S, the update procedure takes as input an outdated vector α that needs to be updated and an updated vector β that represents the desired one after the update. First, the procedure issues a query for vector β on the BoND-tree T to determine if β already exists in T (i.e., S) to avoid any attempt to add a duplicate vector. If vector β exists in T (Scenario 2 or 3), then all that is left is to remove vector α from T if it exists. Specifically, the update procedure tries to locate the leaf node N_α containing vector α in the BoND-tree T. It follows a path P_α from root node *RN* to leaf node N_α. If it is not found, a 'not present' flag is returned (Scenario 2). If such a leaf node N_α is found, the procedure removes vector α from N_α (Scenario 3).

In the event that vector β is not found in T (Scenario 1 or 4), the procedure can involve one of the update methods (to be discussed below) that applies its specific update strategy to decide how the update is performed. Essentially, a suitable leaf node N_β to accommodate vector β must be located. Different strategies may choose a different N_β, which may affect the efficiency and effectiveness of the update. Note that N_β may or may not be the same as N_α if α exists in T (Scenario 4).

Additional update overhead (I/Os) may also occur if either the removal of vector α from leaf node N_α triggers an underflow handling process or the addition of vector β to N_β causes an overflow splitting process.

For the underflow handling process, we adopt the BoND-tree Inspired Node Reinsertion (BNDINR) strategy suggested in [8]. This process is done by invoking function *UnderflowHandling()* in the following discussion. The key idea is to recursively remove each underflow node along the path from the underflow leaf node to the root node in a bottom-up fashion until reaching a parent node (or the root) on the path that is no longer underflow after the removal of its underflow child node. These underflow nodes are put into a reinsertion buffer. The entries in the underflow nodes represent either vectors or sub-trees, depending on whether the underflow node is a leaf node or non-leaf node. The entries in each underflow node in the reinsertion buffer will be directly merged into a sibling node. A good sibling node is chosen according to the following three heuristics in the given priority order:

Least overlap enlargement. Choose a sibling node such that its overlap enlargement with other sibling nodes is minimized after accommodating the entries in the underflow node.

Steady minimum dimensions. Choose a sibling node such that the number of its DMBR's unchanged smallest dimensions is maximized after accommodating the entries in the underflow node.

Least area enlargement. Choose a sibling node such that the area enlargement is minimized after accommodating the entries of the underflow node.

It is possible that a chosen sibling node is overflow after accommodating the entries from the underflow node. In this case, its parent node becomes no longer underflow if it is also in the reinsertion buffer after the chosen sibling node is split into two nodes to handle the overflow. The node reinsertion process to handle the underflow is finished.

The overflow situation is handled by splitting the overflow node into two according to a set of special heuristics recommended in [7]. This process is done by invoking function *OverflowHandling()* in the following discussion. The key idea is to recursively split each overflow node along the path from the overflow leaf node to the root node in a bottom-up fashion until reaching a parent node (or the root) that is no longer overflow after splitting its overflow child node into two new nodes. A good splitting is determined according to the following three heuristics in the given priority order:

Minimum overlap. Choose a split that minimizes the overlap between the DMBRs of the newly created nodes.

Minimum span. Choose a split that splits a dimension that has the smallest span.

Minimum balance. Choose a split that unbalances the distribution of letters between the DMBRs of the newly created nodes the most, while satisfying the minimum space utilization criterion.

During the underflow and overflow handling processes, the relevant DMBRs are adjusted when needed. Even if no underflow or overflow has occurred, the update procedure may still need to adjust the DMBRs in the parent nodes along the path P_α from N_α to root *RN* and/or the parent nodes along a path P_β from N_β to root *RN* when necessary. This is done by invoking function *ComputeDMBR()*, which takes as input a node and its path to the root node and recursively moves up the BoND-tree until no more DMBR changes are detected.

In the following discussion, we present two update strategies to determine a suitable node N_β for vector β, which result in two update algorithms/methods.

3.2 Top-Down Update (TDU) Method

A straightforward strategy for updating vectors in the BoND-tree is the Top-Down Update (TDU) method. This is accomplished by executing a deletion operation followed by an insertion operation. First, the outdated vector α is targeted for deletion. Any underflow scenarios are handled by function *UnderflowHandling()*, and the DMBRs in the BoND-tree are adjusted by function *ComputeDMBR()* when needed for the case having no underflow. If α exists in the BoND-tree (i.e., Scenario 3 or 4), α has to be removed from the tree. Otherwise (i.e., Scenario 1 or 2), nothing needs to be done for α in the tree. Whether or not α exists in the tree, the next step is the same. A query for vector β is performed on the index tree. If β does not exist in the BoND-tree (i.e., Scenario 1 or 4), β has to be inserted into the BoND-tree via the root *RN*. Otherwise (i.e., Scenario 2 or 3), the update is already finished. Any overflow cases are handled by function *OverflowHandling()*, and the DMBRs in the BoND-tree are adjusted by function *ComputeDMBR()* when needed for the case having no overflow. The details of this method are described in Algorithm TDU.

Algorithm 1: Top-down Update (TDU)

Input: (1) the BoND-tree with root *RN*; (2) the outdated vector α; (3) the updated vector β
Output: the root of the modified BoND-tree with α being removed and β being inserted

1 locate the leaf node N_α containing vector α by following a path P_α from root *RN*;
2 **if** *vector* α *exists* **then**
3 remove vector α from leaf node N_α;
4 **if** N_α *is underflow* **then**
5 *UnderflowHandling*(N_α, P_α);
6 **else**
7 *ComputeDMBR*(N_α, P_α);
8 **end if**
9 **end if**
10 query vector β;
11 **if** *vector* β *does not exist* **then**
12 insert vector β via root *RN*;
13 **if** N_β *is overflow* **then**
14 *OverflowHandling*(N_β, P_β);
15 **else**
16 *ComputeDMBR*(N_β, P_β);
17 **end if**
18 **end if**
19 **return** *RN*;

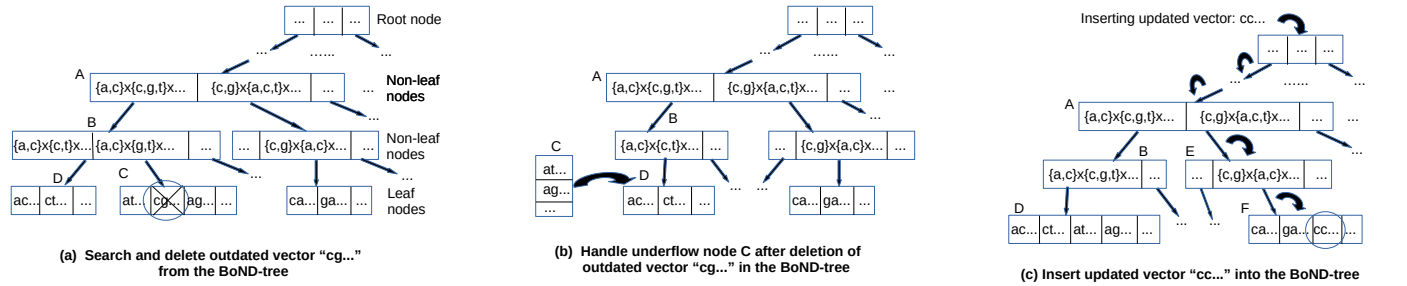
In Algorithm TDU, steps 1 through 9 perform the deletion of α from the given BoND-tree. Steps 10 through 18 perform the insertion of β into the given BoND-tree. Step 12 realizes the actual insertion into the BoND-tree using the insertion heuristics and procedure of [7]. A path P_β from root *RN* to a suitable leaf node N_β is taken to insert vector β into the BoND-tree.

Figure 2 shows an example of a typical top-down update process. Assume that we want to perform an update to change an outdated vector "*cg*..." to an updated vector "*cc*..." in a BoND-tree T built for vectors in a given database/dataset. Note that only the first two dimensions are explicitly displayed in this example. The TDU method first searches for vector "*cg*..." in T by following a path from the root to leaf node C. Vector "*cg*..." is then deleted from node C in T. This process is illustrated in Figure 2(a). The removal of vector "*cg*..." causes node C to be underflow. Node C is then removed from node B. Assume node B is not underflow after removing node C. The vectors in node C are then merged/inserted into a sibling node D. Assume the augmented node D is not overflow – otherwise, node D has to be split into two nodes to replace the original nodes C and D in parent node B. The underflow handling process for node C is illustrated in Figure 2(b). The TDU method then starts an insertion process for updated vector "*cc*..." via the root. Assume the heuristics for insertion [7] selects the path from the root to leaf node F. The updated vector "*cc*..." is then placed in node F, as shown in Figure 2(c). If node F is not overflow, the update process ends. Otherwise, node F has to be split, which may cause its parent node E to be overflow and split. The overflow and split may be propagated to the root, which may make T grow one level taller.

Figure 3 gives an example to illustrate what may occur in an underflow propagation scenario. Let us make the 3rd dimension of vectors/DMBRs also visible in this example. The outdated vector "*cg*..." in Figure 2(a) is now "*cga*..." in Figure 3(a). Like the example in Figure 2, deleting "*cga*..." from node C causes the node to become underflow. The entry for node C that resides in node B must be then removed, and the vectors in node C must be merged into a sibling node D chosen according to the heuristics in Section 3.1. Let node D′ be the augmented node D after the merging as shown in Figure 3(b), which is not underflow. Assume that node B becomes underflow after removing the entry of node C, i.e., the underflow of node C is propagated to node B. As a result, the entry of node B must be removed from its parent node A. Assume node A is not underflow after the removal. The entries in node B for all its child nodes, i.e., nodes D′, H, I, ..., must be merged into a sibling node G chosen according to the heuristics. Figure 3(c) shows that new node G′ is obtained from merging the entries in node B into sibling node G, assuming node G′ is not overflow. If revised node A′ is not underflow after removing the entry of node B, i.e., no further underflow propagation, the underflow handling ends.

3.3 Bottom-Up Update (BUU) Method

An alternative strategy we examine for updating vectors in the BoND-tree is the Bottom-Up Update (BUU) method. The BUU employs a strategy that caches the node DMBRs along the path P_α from the root *RN* down to the leaf node N_α containing the outdated vector α in the typical update situation when α exists in the BoND-tree. Utilizing the cached DMBRs along P_α, the algorithm will compare the vector β against the cached DMBRs from the leaf up to the root until a cached DMBR (if any) is found to contain vector β. At the level this occurs, or the root level if no containing DMBR is found, a local insertion

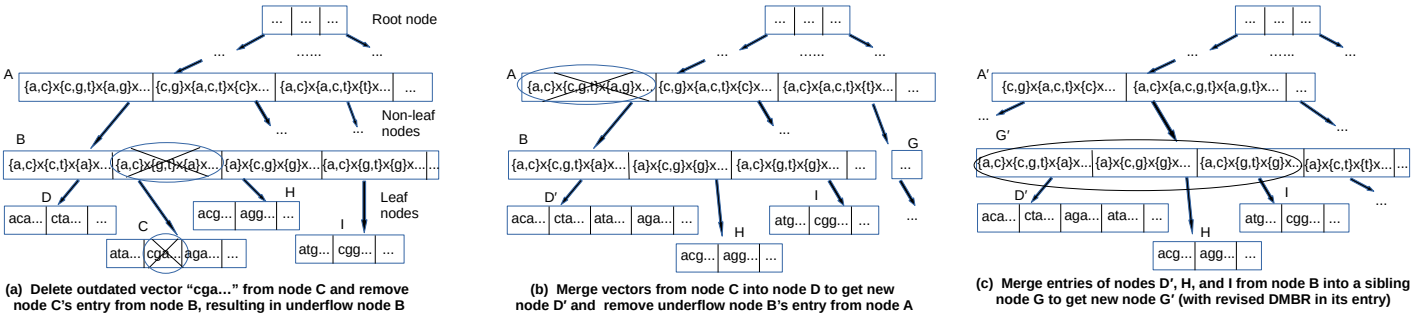


(a) Search and delete outdated vector "cg..." from the BoND-tree

(b) Handle underflow node C after deletion of outdated vector "cg..." in the BoND-tree

(c) Insert updated vector "cc..." into the BoND-tree

Figure 2: Example of top-down update



(a) Delete outdated vector "cga..." from node C and remove node C's entry from node B, resulting in underflow node B

(b) Merge vectors from node C into node D to get new node D' and remove underflow node B's entry from node A

(c) Merge entries of nodes D', H, and I from node B into a sibling node G to get new node G' (with revised DMBR in its entry)

Figure 3: Example of handling underflow propagation during update

is performed via the node at this level. The normal insertion heuristics and procedure of the BoND-tree in [7] are applied to transform path P_α into a path P_β leading down to a leaf node N_β that accommodates vector β. Any overflow scenarios are handled by function *OverflowHandling()*, and the DMBRs for the new path P_β from leaf node N_β up to the root *RN* are adjusted by function *ComputeDMBR()* when needed for the case having no overflow.

For the BUU, the deletion and insertion operations are integrated into one update operation. We find a node with a suitable cached DMBR along the path P_α from which a potential new path P_β down the tree is formed and a leaf node N_β for the vector β is located. A suitable DMBR is the first one from the bottom up which contains vector β. The I/O cost of adding vector β into the BoND-tree is bound in the worst case by the height of the tree with root *RN* when no suitable cached containing DMBR exists.

The best case occurs when vector β is contained in the leaf node N_α's DMBR. In this case, vector β can directly replace vector α in N_α. Effectively, leaf node N_α is leaf node N_β, and path P_α is path P_β. Advantageously no underflow or overflow situations occur that demand additional I/O cost when vector β directly replaces vector α in N_α. Also, the bottom-up update strategy usually avoids the I/O cost incurred by the top-down update strategy when traversing the entire path from root *RN* to the leaf level to find a suitable home for vector β. The details of this method are described in Algorithm BUU.

In Algorithm BUU, steps 1 through 6 determine the update scenario based on whether an insertion of vector β would be needed. Steps 8 through 20 handle scenarios where outdated vector α does not exist in the BoND-tree (i.e., Scenario 1 or 2). A standard insertion process via the root for vector β is performed if β does not exist in the BoND-tree (i.e., Scenario 1). Otherwise, the update is already finished (i.e., Scenario 2). Steps 22 through 29 handle the scenario in which we know vector α exists in the BoND-tree, which needs to be removed, and vector β is also present (i.e., Scenario 3). If the algorithm reaches step 30, we are in the typical update scenario in which we have to remove outdated vector α and add desired vector β (i.e., Scenario 4). Steps 30 through 33 handle the case in which the BoND-tree consists of only one root node which is also a leaf node at the same time. Since α is directly replaced by β, no underflow or overflow processing is needed. Steps 35 through 40 handle the best case in which vector β becomes a direct replacement for vector α and guarantees no underflow or overflow. Steps 41 through 50 handle the underflow situation. In this case, the update process defaults to a standard insertion process of vector β via the root *RN* since the underflow handling may have altered the tree structure and the path of cached nodes may be no longer valid. Steps 51 through 54 climb up the tree until the level where a suitable cached node is found to insert vector β. In the worst case, this node would be in fact the root *RN*. Step 55 through 61 perform a local insertion of vector β via the node PN_i at this particular level so that a path P_β to leaf node N_β is found.

Figure 4 shows two examples of the bottom-up update process. Figure 4(a) illustrates the best scenario in which the outdated vector "*cg...*" is directly replaced by the updated vector "*cc...*" in leaf node C since the DMBR for node C contains both vectors. No underflow or overflow would occur in such a case. The cost of locating the home leaf node for the updated vector is also the minimum. Figure 4(b) illustrates a typical scenario,

Algorithm 2: Bottom-Up Update (BUU)

Input: (1) the BoND-tree with root *RN*; (2) the outdated vector α; (3) the update vector β
Output: the root of the modified BoND-tree with α being removed and β being present

query vector β;
if *vector* β *exists* **then**
  set VectorβAlreadyExist = true;
else
  set VectorβAlreadyExist = false;
end if
locate leaf node N_α containing vector α by following path P_α from root *RN*;
if *vector* α *does not exist* **then**
  if *Vector*β*AlreadyExist* **then**
    return *RN*;
  else
    insert vector β via root *RN*;
    if N_β *is overflow* **then**
      OverflowHandling(N_β, P_β);
    else
      ComputeDMBR(N_β, P_β);
    end if
    return *RN*;
  end if
end if
remove vector α from leaf node N_α;
if *Vector*β*AlreadyExist* **then**
  if N_α *is underflow* **then**
    UnderflowHandling(N_α, P_α);
  else
    ComputeDMBR(N_α, P_α);
  end if
  return *RN*;
end if
if *leaf node* N_α *is the root node RN* **then** `// 0 height tree`
  insert vector β into leaf node N_α;
  return *RN*;
end if
set path P_β = path P_α ; `// finding path for vector` β
if *vector* β *is contained in leaf node* N_α*'s DMBR* **then** `//` β `can directly replace` α
  set leaf node N_β = leaf node N_α;
  insert vector β into leaf node N_β;
  ComputeDMBR(N_β, P_β);
  return *RN*;
end if
if *leaf node* N_α *underflow* **then** `// tree structure changes`
  UnderflowHandling(N_α, P_α);
  insert vector β via root *RN* ; `// default to insert`
  if N_β *is overflow* **then**
    OverflowHandling(N_β, P_β);
  else
    ComputeDMBR(N_β, P_β);
  end if
  return *RN*;
end if
set node PN_i = parent node of leaf node N_α;
while PN_i *is not root* && *vector* β *is not contained in* PN_i*'s DMBR* **do**
  set node PN_i = parent node of PN_i;
end while
insert vector β via node PN_i ; `// new path` P_β `taken to leaf` N_β
if N_β *is overflow* **then**
  OverflowHandling(N_β, P_β);
else
  ComputeDMBR(N_β, P_β);
end if
return *RN*;

in which the BUU method recursively checks the DMBR of the parent node of a current node to see if the updated vector is contained in the DMBR. Once such a DMBR is found (i.e., the DMBR of node B in node A in this example), the updated vector is then inserted into the BoND-tree via the local subtree rooted at the found parent node (i.e., node B in this example) rather than via the root node for the entire tree. The updated vector "*cc*..." is eventually inserted into node D in this example since its DBMR covers the vector.

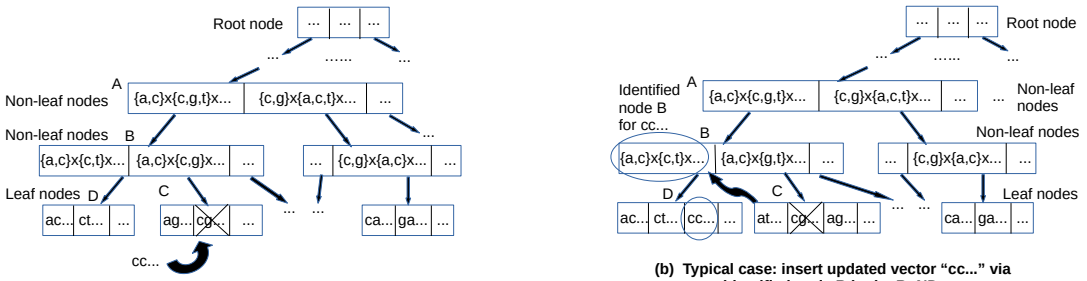
Figure 5 gives an example to illustrate what may occur in an overflow propagation scenario. Let us make the 3rd dimension of vectors/DMBRs also visible in this example. The outdated vector "*cg*..." in Figure 4(b) is now "*cgg*..." in Figure 5(a). Like the case in Figure 4 (b), "*cgg*..." is deleted from node C (assuming no underflow occurs), and node B is identified to be the root of a local subtree whose DMBR covers the updated vector "*ccc*...". The updated vector is then inserted into leaf node D of the tree via node B. Assume node D is overflow after the insertion. It is split into two new nodes D′ and D″, which makes node B become node B′. If node B′ is overflow, it is split into two new nodes B″ and B‴, which makes node A to become node A′. If node A′ is not overflow, the overflow handling ends.

4 Experiments

Experiments were conducted to evaluate the efficiency and effectiveness of the two presented update methods for the BoND-tree. The efficiency is measured in terms of the disk I/Os for performing the updates. The effectiveness is measured by the box query I/Os (average) on the resulting BoND-tree after the updates. The update methods were implemented in C++ on a Dell PC with a 3.6 GHz Intel Core i7-4790 CPU, 12 GB RAM, 2 TB Hard Drive, and Linux 3.16.0 OS.

Two sets of 1,000 randomly-generated box queries were performed on the resulting index tree. One set consists of random-span box queries with a random span (edge length) ranging from 1 to half of the alphabet size for each dimension of the query box. The other set consists of uniform-span box queries with a uniform span of 2 for each dimension of the query box. The disk block size (i.e., the tree node size) was set at 4 KB. In the experiments, we also introduced a "fixed dimension percentage" parameter concerning the updates such that the desired updated vector was guaranteed to have certain values in common with the outdated vector on at least 0%, 25%, 50%, or 75% of its dimensions.

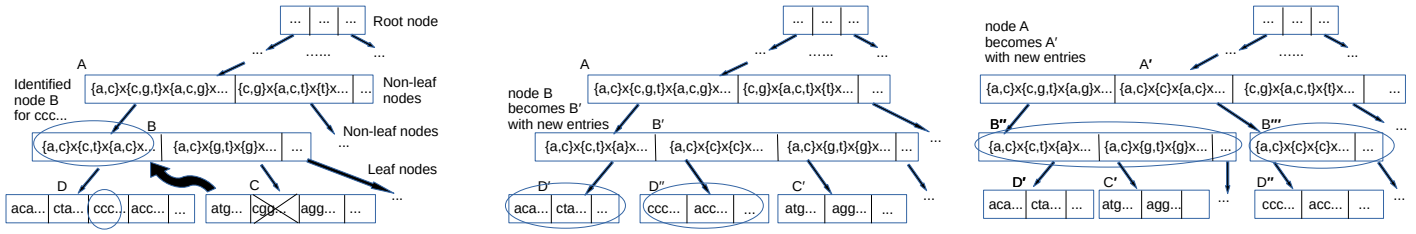
Both synthetic datasets and real genome datasets were used in the experiments. A synthetic data generator was used to generate random data with the uniform distribution. The real genome dataset used is derived from the bacteria.105.1.genomic.fna. A BoND-tree was built to index each dataset. Some representative results from our experiments are reported as follows.



(a) Best case: in-place replace outdated vector "cg..." by updated vector "cc..." in the BoND-tree

(b) Typical case: insert updated vector "cc..." via identified node B in the BoND-tree

Figure 4: Examples of bottom-up update



(a) Node D overflow after inserting updated vector "ccc..." via identified node B in the BoND-tree

(b) Node D is split into new nodes D′ and D″ and Node C becomes node C′ after outdated vector is deleted

(c) Node B′ is split into nodes B″ and B‴

Figure 5: Example of handling overflow propagation during update

4.1 Update Efficiency

In the first set of experiments, we applied each of the two update methods to update 50%, 70%, and 90% of the vectors from each BoND-tree. Tables 1 $\sim$ 4 show the I/O cost incurred from the update process when updating the dataset of synthetic data with 16 dimensions and an alphabet of size 10.

Table 1 shows that, when an updated vector is free to change along all dimensions and become completely independent from an outdated vector, the bottom-up update method (BUU) is comparable to top-down update (TDU) method. However, the bottom-up update method is consistently marginally better because it is bounded in the worst case by the performance of the top-down update method.

Table 1: Number of I/Os for updates on BoND-trees for synthetic datasets with dimensionality = 16, alphabet size = 10, 0% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Update I/Os)	BUU (Update I/Os)
2 M	50%	19151639	18888607
	70%	26860623	26491901
	90%	34573355	34099559
6 M	50%	57034309	56728483
	70%	79856600	79428802
	90%	102684411	102134677
10 M	50%	95012649	94507540
	70%	133016214	132308510
	90%	171019290	170109021

Tables 1 $\sim$ 4 show that increasing the similarity (0% to 75% of fixed dimensions) between an outdated vector and the updated vector clearly yields increasingly better performance for the bottom-up update method over the top-down update method. A similar efficiency benefit with the bottom-up update method was observed on real genome data (see Table 5).

Table 2: Number of I/Os for updates on BoND-trees for synthetic datasets with dimensionality = 16, alphabet size = 10, 25% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Update I/Os)	BUU (Update I/Os)
2 M	50%	19152093	18642310
	70%	26863947	26150218
	90%	34575678	33659498
6 M	50%	57033823	55939969
	70%	79856697	78324221
	90%	102684192	100713707
10 M	50%	95012889	93233586
	70%	133016356	130523685
	90%	171019154	167816000

These tables also show that the top-down update method has negligible differences in I/O cost for performing updates regardless of whether an updated vector is at all related to the outdated vector it is replacing. This is consistent with one's intuition because the top-down update method issues a removal for the outdated vector, and then always issues an insertion via the root node for the updated vector in all cases. In contrast, the bottom-up update method does try to capitalize on any relationship between the updated vector and the outdated vector.

Table 3: Number of I/Os for updates on BoND-trees for synthetic datasets with dimensionality = 16, alphabet size = 10, 50% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Update I/Os)	BUU (Update I/Os)
2 M	50%	19151237	18083357
	70%	26855383	25356375
	90%	34579733	32649765
6 M	50%	57033695	54406359
	70%	79855557	76175575
	90%	102682787	97965034
10 M	50%	95012813	90806427
	70%	133016223	127120088
	90%	171019053	163437333

Table 4: Number of I/Os for updates on BoND-trees for synthetic datasets with dimensionality = 16, alphabet size = 10, 75% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Update I/Os)	BUU (Update I/Os)
2 M	50%	19125252	16412311
	70%	26818328	23022498
	90%	34524669	29642270
6 M	50%	57027989	49738937
	70%	79845601	69644132
	90%	102667407	89556492
10 M	50%	95012452	83190257
	70%	133015786	116444361
	90%	171019251	149730814

Less I/O is incurred as an updated vector traverses less levels in the BoND-tree to find a suitable node location to perform a local insertion. The tendency across a range of fixed dimension percentages shows that the the I/O cost of the bottom-up update method goes down as the percentage of fixed dimensions goes up.

4.2 Update Effectiveness

To evaluate the effectiveness of the proposed update methods for the BoND-tree, we examine the number of I/Os (average) for performing a set of randomly-generated box queries on the resulting BoND-trees after updates for each experiment. Tables 6 and 7 show the observed performance for 1,000 uniform-span box queries run on the resulting BoND-trees after updates for the synthetic datasets. The experimental results show that the query performance obtained by BUU is comparable to that obtained by TDU, irregardless of the percentage of fixed dimensions. Table 8 shows the observed performance for 1,000 random-span box queries run on the resulting BoND-trees after updates for the synthetic datasets. From the results in the table, we can see that the query performance obtained by BUU is comparable to that obtained by TDU for random-span box queries as well. Comparable query performance between TDU and BUU on real genome sequence data was also observed (see Table 9). It is important to obtain the resulting BoND-trees with

Table 5: Number of I/Os for updates on BoND-trees for real genome datasets with dimensionality = 20, alphabet size = 4, 75% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Update I/Os)	BUU (Update I/Os)
2 M	50%	19016031	16851815
	70%	26629245	23604711
	90%	34246081	30362353
6 M	50%	57051316	51512964
	70%	79895890	72180793
	90%	102738854	92848747
10 M	50%	95099141	86886452
	70%	133167594	121698210
	90%	171252292	156576717

Table 6: Number of I/Os for box queries with uniform-span = 2 on BoND-trees after updates for synthetic datasets with dimensionality = 16, alphabet size = 10, 25% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Query I/Os)	BUU (Query I/Os)
2 M	50%	34.878	34.876
	70%	35.466	35.468
	90%	35.628	35.637
6 M	50%	41.031	41.031
	70%	41.088	41.088
	90%	41.246	41.246
10 M	50%	42.997	42.997
	70%	43.000	43.000
	90%	42.999	42.999

comparable query performance after the updates performed by the two methods because it demonstrates that bottom-up update method does not suffer significantly in terms of effectiveness by performing local insertions into a subtree of the BoND-tree. It is not unusual to see different strategies that offer benefits in efficiency weighed against a trade-off in effectiveness and vice versa. However, our empirical study shows that the BoND-tree does not have a significant negative trade-off in terms of effectiveness when using the bottom-up update method over the top-down update method.

Table 7: Number of I/Os for box queries with uniform-span = 2 on BoND-trees after updates for synthetic datasets with dimensionality = 16, alphabet size = 10, 75% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Query I/Os)	BUU (Query I/Os)
2 M	50%	34.336	34.286
	70%	35.067	35.088
	90%	35.445	35.515
6 M	50%	40.916	40.912
	70%	41.026	41.026
	90%	41.124	41.122
10 M	50%	42.989	42.989
	70%	42.999	42.999
	90%	42.998	42.998

Table 8: Number of I/Os for box queries with random-span on BoND-trees after updates for synthetic datasets with dimensionality = 16, alphabet size = 10, 75% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Query I/Os)	BUU (Query I/Os)
2 M	50%	152.626	152.471
	70%	156.590	156.762
	90%	158.553	158.944
6 M	50%	240.444	240.411
	70%	235.808	235.794
	90%	247.632	247.628
10 M	50%	264.862	264.862
	70%	274.353	274.353
	90%	275.445	275.445

Table 9: Number of I/Os for box queries with random-span on BoND-trees after updates for real genome datasets with dimensionality = 20, alphabet size = 4, 75% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Query I/Os)	BUU (Query I/Os)
2 M	50%	24.515	24.519
	70%	24.994	24.992
	90%	24.745	24.742
6 M	50%	33.791	33.795
	70%	34.530	34.514
	90%	35.474	35.454
10 M	50%	40.713	40.728
	70%	43.438	43.466
	90%	41.819	41.848

Table 10: Space utilization for BoND-trees after updates for synthetic datasets with dimensionality = 16, alphabet size = 10, 25% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Space Util.)	BUU (Space Util.)
2 M	50%	0.583970	0.583970
	70%	0.584254	0.584254
	90%	0.585108	0.585080
6 M	50%	0.650328	0.650328
	70%	0.642878	0.642889
	90%	0.638624	0.638624
10 M	50%	0.590993	0.590993
	70%	0.590417	0.590417
	90%	0.590272	0.590272

Table 11: Space utilization for BoND-trees after updates for synthetic datasets with dimensionality = 16, alphabet size = 10, 75% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Space Util.)	BUU (Space Util.)
2 M	50%	0.584311	0.584201
	70%	0.584396	0.583958
	90%	0.585223	0.584993
6 M	50%	0.655938	0.655842
	70%	0.648190	0.648130
	90%	0.643987	0.643862
10 M	50%	0.591704	0.591675
	70%	0.590900	0.590859
	90%	0.590539	0.590486

4.3 Space Utilization

When evaluating an index tree, people usually also examine the space utilization which indicates how efficient the space is utilized for the index tree. We examined the space utilization of the BoND-trees after the updates. The representative space utilization statistics are given in Tables 10 $\sim$ 12 for different parameter configurations. From the data in the tables, we can see that the space utilizations of the BoND-trees after updates performed by the two methods are comparable, regardless of the database size, the percentage of fixed dimensions, and the synthetic/real dataset. This is quite promising since it demonstrates that the bottom-up update method can produce quality BoND-trees not only in terms of query performance but also the space utilization.

4.4 Statistics on Direct Replacement

The best case for the bottom-up update method occurs when an updated vector can directly replace an outdated vector in the leaf node that the outdated vector resides in. In this case, no extra I/O cost is incurred from traversing different branches of the BoND-tree to locate an appropriate home for the updated vector. Overflow and underflow handling situations can also be avoided because the updated vector directly replaces the outdated vector.

We show sample statistics about the best case for the bottom-up update method with a varying percentage of fixed dimensions for updates in Table 13 for synthetic datasets and in Table 14 for real genome datasets. From the tables, we can see that the number of times the best case (direct replacement) has occurred versus the number of times the worst case (update via root) has occurred.

Our results indicate that the likelihood of an updated vector directly replacing an outdated vector tends to increase as the number of dimensions upon which they share the same values increases. If the updated vector is contained by the existing DMBR of the leaf node from which the outdated vector is

Table 12: Space utilization for BoND-trees after updates for real genome datasets with dimensionality = 20, alphabet size = 4, 75% fixed dimensions

DB Size (vectors)	Update % of DB	TDU (Space Util.)	BUU (Space Util.)
2 M	50%	0.618304	0.617815
	70%	0.614028	0.613519
	90%	0.617357	0.616540
6 M	50%	0.615432	0.615114
	70%	0.610998	0.610684
	90%	0.615000	0.614578
10 M	50%	0.614037	0.613729
	70%	0.606661	0.606328
	90%	0.600961	0.600524

Table 13: Number of times direct replacement occurs for synthetic datasets with dimensionality = 16, alphabet size = 10, DB size = 10M vectors, update percentage 90%, BUU method

Percentage of Fixed Dimensions	Direct Replacement	Update via Root
0%	97	8099296
25%	11040	6077723
50%	266895	4049565
75%	2067574	2025833

Table 14: Number times direct replacement occurs for real genome datasets with dimensionality = 20, alphabet size = 4, DB size = 10M vectors, update percentage 90%, BUU method

Percentage of Fixed Dimensions	Direct Replacement	Update via Root
0%	77	7970830
25%	3601	6699825
50%	83585	4944486
75%	1089256	2679402

removed, the direct replacement can take place and the best case is realized.

5 Conclusions

Box queries on non-ordered discrete vector (multidimensional) data are demanded in contemporary applications. To efficiently process box queries, the BoND-tree was recently developed. Although efficient techniques for query, insertion, deletion, and bulk loading for the BoND-tree were studied in earlier work, how to efficiently and effectively perform update operations needs to be explored.

In this paper, we have presented a general update procedure and studied two update strategies for the BoND-tree, i.e., the traditional top-down update method and the promising bottom-up update method. The bottom-up update method is bounded by the worst-case of the top-down update method, in the sense that it resorts to an insertion of the updated vector via the root if no suitable local insertion node closer to the leaf level is found. Furthermore, the bottom-up update method promises better efficiency for applications where an updated vector may be related on some dimensions to the corresponding outdated vector. This is because the I/O cost is reduced when a local insertion closer to the leaf level is realized. This strategy does not impact the effectiveness of subsequent box queries in a significant negative manner when compared to the top-down update method. Given that the bottom-up update method can provide significant performance boost in terms of efficiency without a significant trade-off in effectiveness as well as space utilization, it becomes our general recommendation for processing updates on the BoND-tree in an NDDS.

Our future work includes studying techniques for buffering update operations for applications where a bulk set of updates must be done in which one update is not necessarily independent from the next, integrating bulk loading and updating techniques, and exploring applications utilizing the tree maintenance techniques. We also plan to explore the feasiblity of using advanced Bloom filter structures [33, 34] to efficiently support queries on non-ordered discrete vector data and the maintainability of such structures [32] as well as the evoluationary update strategies [35] for indexing structures.

Acknowledgments

Research was partially supported by the US National Science Foundation (NSF) (under Grants #IIS-1320078 and #IIS-1319909) and The University of Michigan. The authors would like to thank Yarong Gu and Steven Liu for their assistance in implementing programs and obtaining datasets for this work.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In *Proc. of SIGMOD*, pp. 322-331, May 1990.

[2] S. Berchtold, D. A. Keim, and H.-P. Kriegel. "The X-Tree: An Index Structure for High-Dimensional Data". In *Proc. of VLDB*, pp. 28-29, Sept. 1996.

[3] L. Biveinis, S. Šaltenis, and C. S. Jensen. "Main-Memory Operation Buffering for Efficient R-Tree Update". In *Proc. of VLDB*, pp. 591-602, Sept. 2007.

[4] L. Biveinis, and S. Šaltenis. "Towards Efficient Main-Memory Use for Optimum Tree Index Update". In *Proc. of VLDB Endowment*, 1(2):1617-1622, Aug. 2008

[5] T. Bozkaya and M. Ozsoyoglu. "Indexing Large Metric Spaces for Similarity Search Queries". *ACM Trans. on Database Syst.*, 24(3):361-404, Sept. 1999.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. "Searching in Metric Spaces". *ACM Comput. Surv.*, 33(3):273-321, Sept. 2001.

[7] C. Chen, A. Watve, S. Pramanik, and Q. Zhu. "The BoND-Tree: An Efficient Indexing Method for Box Queries in Nonordered Discrete Data Spaces". *IEEE Trans. on Knowl. and Data Eng.*, 25(11):2629-2643, Nov. 2013.

[8] R. Cherniak, Q. Zhu, Y. Gu, and S. Pramanik. "Exploring Deletion Strategies for the BoND-Tree in Multidimensional Non-Ordered Discrete Data Spaces". In *Proc. of IDEAS*, pp. 153-160, July 2017.

[9] R. Cherniak, Q. Zhu, and S. Pramanik. "A Study of Update Methods for BoND-Tree Index on Non-Ordered Discrete Vector Data". In *Proc. of ISCA 35th Int'l Conf. on Computers and Their Applications (CATA)*, pp. 122-133, March 2020

[10] P. Ciaccia, M. Patella, and P. Zezula. "M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces". In *Proc. of VLDB*, pp. 426-435, Aug. 1997.

[11] J. Clément, P. Flajolet, and B. Vallée. "Dynamical Sources in Information Theory: A General Analysis of Trie Structures". *ALGORITHMICA*, 29(1-2):307-369, Feb. 1999.

[12] D.-Y. Choi, AKM T. Islam, S. Pramanik and Q. Zhu, "A Bulk-Loading Algorithm for the BoND-Tree Index Scheme for Non-Ordered Discrete Data Spaces". In *Proc. of 25th Int'l Conf. on Software Eng. and Data Eng. (SEDE)*, Denver, pp. 123-128, September 2016.

[13] G. Evangelidis, D. Lomet and B. Salzberg. "The hB$^{\Pi}$-Tree: A Multi-Attribute Index Supporting Concurrency, Recovery and Node Consolidation". *VLDB J.*, 6(1):1-25, Feb. 1997.

[14] P. Ferragina, R. Grossi, and M. Montagero. "A Note on Updating Suffix Tree Labels". In *Proc. of 3rd Italian Conf. on Algo. and Comp.*, pp. 181-192, March 1997.

[15] A. W.-c. Fu, P. M.-s. Chan, Y.-L. Cheung, and Y. S. Moon. "Dynamic Vp-Tree Indexing for N-Nearest Neighbor Search Given Pair-Wise Distances". *VLDB J.*, 9(2):154-173, July 2000.

[16] Y. Gu, Q. Zhu, X. Liu, Y. Dong, C. Brown, and S. Pramanik. "Using Disk Based Index and Box Queries for Genome Sequencing Error Correction". In *Proc. of 8th Intl Conf. on Bioinfo. and Comp.Biology*, pp. 69-76, 2016.

[17] A. Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching". In *Proc. of SIGMOD*, pp. 47-57, June 1984.

[18] G. R. Hjaltason and H. Samet. "Index-Driven Similarity Search in Metric Spaces (Survey Article)". *ACM Trans. on Database Syst.*, 28(4):517-580, Dec. 2003.

[19] M. Ishikawa, H. Chen, K. Furuse, J. X. Yu, and N. Ohbo. "MB+Tree: A Dynamically Updatable Metric Index for Similarity Search". In *Proc. of WAIM*, pp. 356-373, 2000.

[20] AKM. T. Islam, S. Pramanik, X. Ji, J. R. Cole, and Q. Zhu. "Back Translated Peptide K-Mer Search and Local Alignment in Large DNA Sequence Databases Using BoND-SD-Tree Indexing". In *Proc. of BIBE*, pp. 1-6, 2015.

[21] AKM. T. Islam, S. Pramanik, and Q. Zhu. "The BINDS-Tree: A Space-Partitioning Based Indexing Scheme for Box Queries in Non-Ordered Discrete Data Spaces". *IEICE Trans. on Info. and Sys.*, E102.D(4):745-758, 2019.

[22] D. Kolbe, Q. Zhu, and S. Pramanik. "Efficient K-Nearest Neighbor Searching in Non-Ordered Discrete Data Spaces". *ACM Trans. on Inf. Syst.*, 28(2):7:1-7:33, May 2010.

[23] D. Kolbe, Q. Zhu, and S. Pramanik. "On K-Nearest Neighbor Searching in Non-Ordered Discrete Data Spaces". In *Proc. of ICDE*, pp. 426-435, April 2007.

[24] M.-L. Lee, W. Hsu, C. S. Jensen, B. Cui, and K. L. Teo. "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach". In *Proc. of VLDB*, pp. 608-619, Sept. 2003.

[25] X. Liu, Q. Zhu, S. Pramanik, C. T. Brown and G. Qian, "VA-Store: A Virtual Approximate Store Approach to Supporting Repetitive Big Data in Genome Sequence Analyses". *IEEE Trans. on Knowl. and Data Eng.*, 32(3):602-616, 2020.

[26] B. Lin and J. Su. "Handling Frequent Updates of Moving Objects". In *Proc. of CIKM*, pp. 493-500, Oct. 2005.

[27] D. B. Lomet and B. Salzberg. "The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance". *ACM Trans. on Database Syst.*, 15(4):625-658, Dec. 1990.

[28] D. B. Lomet, and B. Salzberg. "Access Method Concurrency with Recovery". In *Proc. of SIGMOD*, pp. 351-360, June 1992.

[29] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik. "The ND-Tree: A Dynamic Indexing Technique for Multidimensional Non-Ordered Discrete Data Spaces". In *Proc. of VLDB*, pp. 620-631, Sept. 2003.

[30] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik. "Dynamic Indexing for Multidimensional Non-Ordered Discrete Data Spaces Using a Data-Partitioning Approach". *ACM Trans. on Database Sys.*, 31(2):439-484, June 2006.

[31] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik. "A Space-Partitioning-Based Indexing Method for Multidimensional Non-Ordered Discrete Data Spaces". *ACM Trans. on Inf. Syst.*, 24(1):79-110, Jan. 2006.

[32] J. Qian, Q. Zhu, and Y. Wang, "Bloom Filter Based Associative Deletion". *IEEE Trans. on Paral. and Distr. Syst.*, 25(8):1986-1998, 2014.

[33] J. Qian, Q. Zhu, and H. Chen, "Multi-Granularity Locality-Sensitive Bloom Filter". *IEEE Trans. on Comp.*, 64(12):3500-3514, 2015.

[34] J. Qian, Z. Huang, Q. Zhu and H. Chen, "Hamming Metric Multi-Granularity Locality-Sensitive Bloom Filter", *IEEE/ACM Trans. on Netw.*, 26(4):1660-1673, 2018.

[35] A. Rahal, Q. Zhu, and P. A. Larson, "Evolutionary Techniques for Updating Query Cost Models in a Dynamic Multidatabase Environment", *The VLDB Journal*, 13(2):162-176, 2004.

[36] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. "Indexing the Positions of Continuously Moving Objects". In *Proc. of SIGMOD*, pp. 331-342, May 2000

[37] H.-J. Seok, Q. Zhu, G. Qian, S. Pramanik, and W.-C. Hou. "Deletion Techniques for the ND-Tree in Non-Ordered Discrete Data Spaces". In *Proc. of Intl Conf. on Soft. Eng. and Data Eng. (SEDE)*, pp. 1-6, Jan. 2009.

[38] Y. N. Silva, X. Xiong, and W. G. Aref. "The RUM-Tree: Supporting Frequent Updates in R-Trees Using Memos". *VLDB J.*, 18(3):719-738, June 2009.

[39] M. Song, H. Choo, and W. Kim. "Spatial Indexing for Massively Update Intensive Applications". *Inf. Sci.*, 203:1-23, Oct. 2012.

[40] G. Tang, J. Pei, J. Bailey, G. Dong, "Mining Multidimensional Contextual Outliers from Categorical Relational Data". *Intell. Data Anal.*, 19(5):1171-1192,

2015.

[41] Y. Tao, D. Papadias, and J. Sun. "The TPR*-Tree: an Optimized Spatio-Temporal Access Method for Predictive Queries". In *Proc. of VLDB*, pp. 790-801, Sept. 2003.

[42] J. K. Uhlmann. "Satisfying General Proximity/Similarity Queries with Metric Trees". *Inf. Process. Lett.* 40(4):175-179. Nov 1991.

[43] P. Weiner. "Linear Pattern Matching Algorithms". In *Proc. of Ann. Symp. on Switching and Automata Theory*, pp. 1-11, Oct. 1973.

[44] J. Yang, W. Zhang, Y. Zhang, X. Wang, and X. Lin, "Categorical Top-K Spatial Influence Query". *World Wide Web*, 20:175-203, 2017.

[45] J. Yang, C. Zhao, C. Li, and C. Xing, "An Efficient Indexing Structure for Multidimensional Categorical Range Aggregation Query". *KSII Trans. on Intern. and Inf. Syst.*, 13(2):597-618, 2019.

[46] P. N. Yianilos. "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces". In *Proc. of ACM-SIAM Symp. on Discr. Algo.*, pp. 311-321, Jan. 1993.

[47] P. Zhou and B. Salzberg. "The hB-Pi* Tree: an Optimized Comprehensive Access Method for Frequent-Update Multi-Dimensional Point Data". *SSDBM*, 5069:331-347, July 2008.

[48] Y. Zhu, S. Wang, X. Zhou, and Y. Zhang, "RUM+-Tree: A New Multidimensional Index Supporting Frequent Updates". In *Proc. of WAIM, LNCS 7923*, pp. 235-240, 2013.

Ramblin Cherniak received his B.S. in Computer and Information Science at the University of Michigan - Dearborn in 2018. He was a Research Assistant, under the supervision of Dr. Qiang Zhu, working on a research project sponsored by the US National Science Foundation. Some of his research work was reported in several venues including IDEAS'17. His areas of interest include multidimensional indexing, query optimization, web services, and data science. He was an active member for an ACM programming competition team as well as involved in several student organizations including ACM and UPE. He currently works as a software engineer in a software company in Michigan.

Qiang Zhu received his Ph.D. degree in Computer Science from the University of Waterloo, Canada, in 1995. He is currently the William E. Stirton Professor and the Chair of the Department of Computer and Information Science at the University of Michigan - Dearborn. He is also an ACM distinguished scientist and a senior member of the IEEE. He received numerous distinguished research awards. His research interests include query processing and optimization for emerging database systems, big data processing, multidimensional indexing, streaming data processing, autonomous database systems, data mining, and database applications in bioinformatics and the automotive industry.

Sakti Pramanik received the BE degree in electrical engineering from the Calcutta University, the MS degree in electrical engineering from the University of Alberta, Edmonton, Canada, and the PhD degree in computer science from the Yale University. He was awarded the University's gold medal for securing the highest grade among all branches of engineering from Calcutta University. He is currently a Professor Emeritus with the Department of Computer Science and Engineering at the Michigan State University. His research interests include high-dimensional indexing, genome sequence analysis, and multimedia databases.

Studying Error Propagation for Energy Forecasting Using Univariate and Multivariate Machine Learning Algorithms

Maher Selim*, Ryan Zhou*, Wenying Feng * and Omar Alam*
Trent University, Peterborough, Ontario, CANADA K9L 0G2

Abstract

Statistical machine learning models are widely used in time series forecasting. These models often use historical data recursively to make predictions, i.e. future timesteps. This leads to compounding of errors, which may negatively impact the prediction accuracy for long-term prediction tasks. In this paper, we address this problem by using features that can have "anchoring" effect on recurrent forecasts, thus, limiting the impact of compounding errors. The approach is tested with four machine learning models applied to a benchmark energy dataset. It is observed that the addition of generated features improves performance for both short and long time horizons.

Key Words: Linear regression; LSTM; energy forecasting; machine learning; support vector regression; time series forecasting; XGBoost regression.

1 Introduction

Machine learning models are widely used in the energy industry for forecasting future energy prices and demands [1, 23]. Advances in sensor and smart meter technologies have made large quantities of energy data available [13]. This, combined with increasingly accurate predictions produced by machine learning models has made it possible for technologies such as smart grid to flourish.

In the domain of energy forecasting, most machine learning models, such as Long-Term Short Memory (LSTM) [16], use historical values of the electricity load as an input feature. This works well for single timestep predictions, e.g. forecasting the power consumption for the next hour. However, when forecasting multiple timestamps into the future, these models recursively feed back in past predictions. In addition, if the model uses external features, such as the hourly weather reading, forecasts of these features must be generated as well. All these predictions introduce error, which is compounded when fed back into the model as inputs. Without external inputs, models generally become inaccurate or even unstable after several timesteps. This makes multiple timestep forecasting challenging even for models with high single timestep accuracy. As further extension to our previous work [20], in this paper, we continue the study on reducing error propagation for energy forecasting using generated features, i.e. input features that can be calculated from known variables with perfect accuracy even far into the future. These features limit the impact of the accumulated error, as the model is trained on these features along with recursive inputs. We demonstrate the efficacy of this approach using a benchmark energy dataset. Four machine learning models are trained to perform single timestep predictions: Linear Regression (LR) [19], Support Vector Regression (SVR) [8], LSTM neural network [16], and a gradient boosted tree model (XGBoost) [7]. Predictions are then made over a period of one month by recursively feeding in the model outputs from earlier timesteps as inputs for later timesteps. We show that without any generated features, error accumulates rapidly over time while including generated features leads to smaller accumulated errors. We also demonstrate the accuracy of predictions made entirely using generated features, i.e. without recursive inputs. This version of the model allows forecasting for arbitrary timesteps in the future, without the need to predict all values in between.

The remainder of this paper is organized as follows. Section 2 introduces the four machine learning algorithms used in our study. Section 3 describes time series forecasting using univariate and multivariate approaches. Section 4 describes the development of computational models, the experimental set-up and the results. Lastly, Section 5 concludes the paper.

2 Prediction Using Machine Learning Algorithms

Prediction using machine learning has been shown to be efficient in many applications. There are numerous learning algorithms mostly based on statistical and mathematical approaches. For our study, four popular algorithms representing different categories are selected including Linear Regression (LR), Support Vector Regression (SVR), Long Short-Term

*Department of Computer Science. Corresponding author: wfeng@trentu.ca

Memory (LSTM) neural networks, and XGBoost regression.

As a basic method in statistics, LR predicts a future value using a linear function that was obtained by minimizing the discrepancies between predicted and actual output values. Widely applied in industry, linear regression can be easily performed in many platforms such as Excel, R, MatLab, Python and others [22].

SVR is a typical kernel based learning method since it relies on the kernel functions. Different from the linear regression, it provides some flexibility to define how much error is acceptable in the model. The problem is equivalent to finding the equation of a separating hyperplane in a high dimensional space. For example, if we have N observations with y_n is the observed response for the input data x_n, the training data set can be represented as $D = \{(x_i, y_i) \mid i = 1,2,3,...N\}$. The objective of a linear SVR is to find the linear function $f(x) = x'\beta + b$ such that

$$\text{MIN } J(\beta) = \frac{1}{2}\beta'\beta + C\sum_{n=1}^{N}(\xi_n + \xi_n^*) \tag{1}$$

subject to

$$y_n - (x_n'\beta + b) \leq \varepsilon + \xi_n,\ n = 1,2,\cdots,N, \tag{2}$$

$$(x_n'\beta + b) - y_n \leq \varepsilon + \xi_n^*,\ n = 1,2,\cdots,N, \tag{3}$$

$$\xi_n \geq 0,\ \xi_n^* \geq 0,\ n = 1,2,\cdots,N, \tag{4}$$

where the constant C, slack variables ξ_n and ξ_n^* are for the Lagrangian formulation. $\varepsilon > 0$ controls the loss function that ignores the errors within ε distance. $\beta'\beta$ is the $l_2 - norm$ of the coefficient vector. This is a convex quadratic programming problem, since the objective function is itself convex, and those points which satisfy the constraints also form a convex set. For more details on SVR, we refer to [4] and the references within.

LSTM is a type of recurrent neural network architecture designed to extract long-term dependencies out of sequential data and avoid the vanishing gradient problem present in ordinary recurrent networks [12, 16]. These properties make it the method of choice for longer time series and sequence prediction problems [11, 24]. Several variations of the LSTM unit have been successfully applied to energy forecasting and other areas [3, 15]. The standard LSTM architecture [12] described below is applied in our study. Each LSTM cell contains a cell state (h_{t-1}), the long-term memory, and a recurrent input (y_{t-1}) - the short-term memory. It also contains three "gates": neurons which output values between 0 and 1 and are multiplied with the information flowing into and out of the cell. The forget gate σ_f controls the amount of information discarded from the previous cell state. The input gate σ_u operates on the previous state $h[t-1]$, after having been modified by the forget gate, and decides how much of a new candidate state $\tilde{h}[t]$ to add to the cell state $h[t]$. The output $y[t]$ is produced by squashing the cell state with a nonlinear function $g_2(\cdot)$, usually tanh. Then, the output gate σ_o selects the overall fraction of the state to be returned as output.

Gradient boosting is an ensemble technique which creates a prediction model by aggregating the predictions of weak prediction models, typically decision trees. With boosting methods, weak predictors are added to the collection sequentially with each one attempting to improve upon the entire ensemble's performance.

In the XGBoost implementation [7], given a dataset with n training examples consisting of an input $\mathbf{x_i}$ and expected output y_i, a tree ensemble model $\phi(\mathbf{x_i})$ is defined as the sum of K regression trees $f_k(\mathbf{x_i})$:

$$\hat{y}_i = \phi(\mathbf{x_i}) = \sum_{k=1}^{K} f_k(\mathbf{x_i}). \tag{5}$$

To evaluate the performance of a given model, we choose a loss function $l(\hat{y}_i, y_i)$ to measure the error between the predicted value and the target value, and optionally add a regularization term $\Omega(f_k)$ to penalize overly complex trees:

$$L(\phi) = \sum_{i}^{n} l(\hat{y}_i, y_i) + \sum_{k}^{K}(\Omega(f_k)). \tag{6}$$

The algorithm minimizes $L(\phi)$ by iteratively introducing each f_k. Assume that the ensemble currently contains K trees. We add a new tree f_{K+1} that minimizes

$$\sum_{i}^{n} l(\hat{y}_i, y_i + f_{K+1}(\mathbf{x_i})) + \Omega(f_k). \tag{7}$$

In other words, the tree that most improves the current model as determined by L are greedily added. We train the new tree using the objective function (6); this is done in practice by approximating the objective function using the first and second order gradients of the loss function $l(\hat{y}_i, y_i)$ [10].

3 Univariate and Multivariate Input Features

Time series prediction is a problem which aims to predict future values using past values [2]. These are generally past values of the target variable, but this is not necessarily the case. Forecasting models can be broadly classified into univariate and multivariate models based on the number of features used. When forecasting multiple timesteps into the future, models can also be classified into direct, recursive and MIMO approaches [21].

A recursive approach trains a single model to predict a single step in the future, known as a one-step ahead forecast:

$$\hat{x}_t = F(x_{t-1}, x_{t-2}, \ldots)$$

where $x_{(i)}$ represents the value of the variable at timestamp i. This forecasted value is then fed back in as an input and the next timestep is forecasted using the same model:

$$\hat{x}_{t+1} = F(x_t, x_{t-1}, \ldots)$$

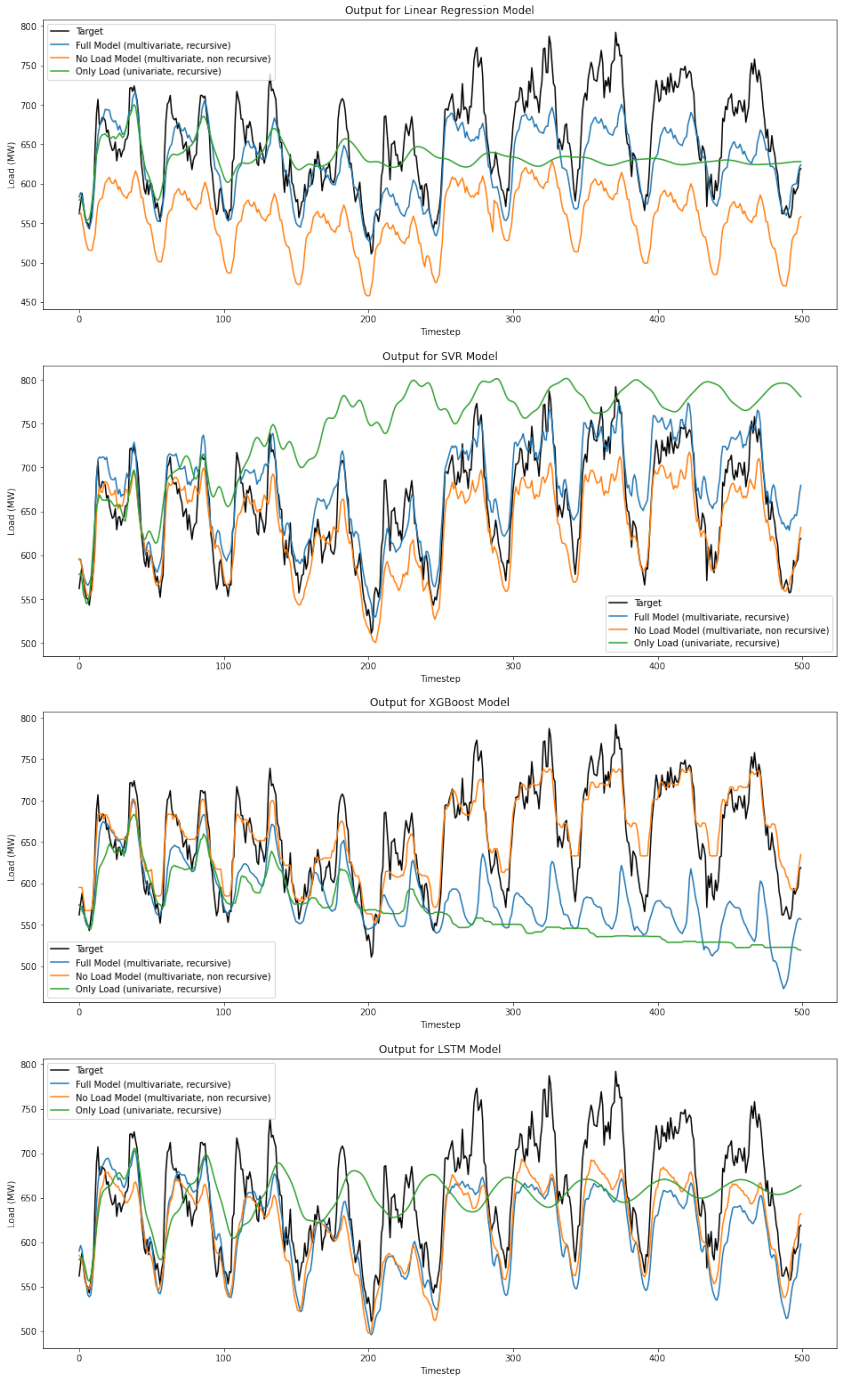


Figure 1: Forecasts for January 1999 using (a) linear regression (b) support vector regression (c) XGBoost regression and (d) LSTM. Full model (blue) uses recursively calculated load and all external features. No load model (orange) uses only external features and no recursion. Only load (green) uses no external features and only recursively calculated load

This process is repeated until the desired time horizon has been reached. This approach is sensitive to accumulated errors, as any error present in the initial prediction will subsequently be carried forward to later predictions when the predicted value is used as input. However, as only one model is used for all predictions, this allows more resources to be invested in the single model. In addition, this approach is flexible in that it allows forecasting for any time horizon, whether or not the model has been trained on that time horizon. A direct approach aims to avoid error accumulation by creating a separate model for each potential time horizon. Thus, a collection of models is trained as expressed by system (8). This avoids propagated

errors as no predicted values are used as input. However, as each model is trained independently, the models may not learn complex dependencies between the values $\hat{x}_t$, $\hat{x}_{t+1}$, $\hat{x}_{t+2}\ldots$. This approach is also computationally much more expensive as multiple models must be trained and stored.

$$\begin{aligned} \hat{x}_t &= F(x_{t-1}, x_{t-2}, \ldots) \\ \hat{x}_{t+1} &= G(x_{t-1}, x_{t-2}, \ldots) \\ \hat{x}_{t+2} &= H(x_{t-1}, x_{t-2}, \ldots) \\ \ldots &= \ldots \end{aligned} \tag{8}$$

The multi-input multi-output (MIMO) strategy attempts to combine the advantages of these approaches by training a single model with multiple outputs to predict all timesteps up to the time horizon simultaneously:

$$[\hat{x}_{t+H}, \hat{x}_{t+H-1}, \ldots, \hat{x}_t] = F(x_{t-1}, x_{t-2}, \ldots)$$

This avoids accumulated error by performing all predictions in one step, as well as modeling any interdependencies between future timesteps. However, this comes at the cost of less flexibility, as all horizons are forecasted using the same model and possible time horizons are limited to those built into the model.

Based on the input features, time series prediction models can be categorized as univariate or multivariate. Univariate models use a single feature, generally the target variable, to predict a future value:

$$\hat{x}_t = F(x_{t-1}, x_{t-2}, \ldots).$$

This has the advantage of allowing smaller and computationally lighter models. Univariate models do not require extra external data and require no feature engineering. However, as they are tied to a single variable, they exhibit more sensitivity to noise and reduced stability for recursive models.

Multivariate time series models use observations of multiple variables or features, often taken simultaneously, and attempt to also describe the interrelationships among the features [5]:

$$\hat{x}_t = F(x_{t-1}, x_{t-2}, \ldots, a^{(1)}_{t-1}, a^{(1)}_{t-2}, \ldots, a^{(2)}_{t-1}, a^{(2)}_{t-2} \ldots)$$

where each $a^{(i)}$ represents the time series of an external feature. This has the obvious advantage of modeling relationships between the target and external variables, but at the cost of a bulkier model and higher computational costs. Building such a model generally also requires obtaining measurements of external features; the difficulty of this is highly dependent on data availability.

It is also possible for a multivariate model to employ no past information about the target variable:

$$\hat{x}_t = F(a^{(1)}_{t-1}, a^{(1)}_{t-2}, a^{(1)}_{t-3}, \ldots, a^{(2)}_{t-1}, a^{(2)}_{t-2}, a^{(2)}_{t-3}, \ldots).$$

In this case, predictions must be made solely based on the relationships of external features to the target variable. Such a model is rarely used in practice as training the model in the first place requires knowledge of past values of the target variable, but may see use if obtaining a full time series of the target value is difficult due to missing or unusable values. In addition, as the output of the model is never used as an input, error accumulation is limited. If future values for the external features can be obtained, this approach allows prediction based on those values without first predicting earlier time horizons.

4 Empirical Study on Energy Forecasting

We study energy forecasting using the four machine learning algorithms described in Section 2. Effects of external features on error propagation are compared for the recursive univariate, multivariate and the modified multivariate techniques.

The linear regression and support vector regression models are implemented using scikit-learn [18]. We use the radial basis function (RBF) kernel for SVR. The gradient boosting model was built using the XGBoost Python library [7] with a maximum tree depth of 12. All other parameters are set to scikit-learn defaults.

The LSTM model is implemented using PyTorch [17] running on Python 3.8. The model consists of four layers: the input layer, two hidden LSTM layers with 16 nodes each, and a linear fully connected aggregation layer as the output. To improve stability, we use a residual connection on the LSTM layers. The model is trained on MAE loss using the Adam optimizer for 30 epochs.

In order to ensure reproducibility of the experiment, the 2001 EUNITE competition dataset [9] is used in our study. This benchmark dataset is well-studied in energy forecasting research [6, 14]. It spans over two years from January 1997 until January 1999. It contains the following fields: the half-hourly electricity load, the daily average temperature, and a flag signifying whether the day is a holiday. In the statistical analysis of the dataset [6, 14], it was found that the electricity load generally decreases during holidays and weekends. This phenomenon depends on the type of the holiday, e.g., Christmas or New Year.

To ensure no outside forecasts are required, we disregard all temperature measurements as these require separate weather forecasts. This ensures model performance is based only on features which can be calculated with perfect accuracy. In addition, we generate the following features based on the prediction timestamp: weekday, ranging from 0 to 6, day of year, ranging from 1 to 365, and hour of day, ranging from 0 to 23. These two features allow the model to pinpoint the day and time within the year and capture daily, weekly and yearly dependencies.

Both datasets were converted into input-output pairs for supervised learning using a sliding window method, whereby timesteps within the window were used as input to predict the next timestep after the window. A window size of 48 timesteps was chosen, corresponding to the previous 24 hours

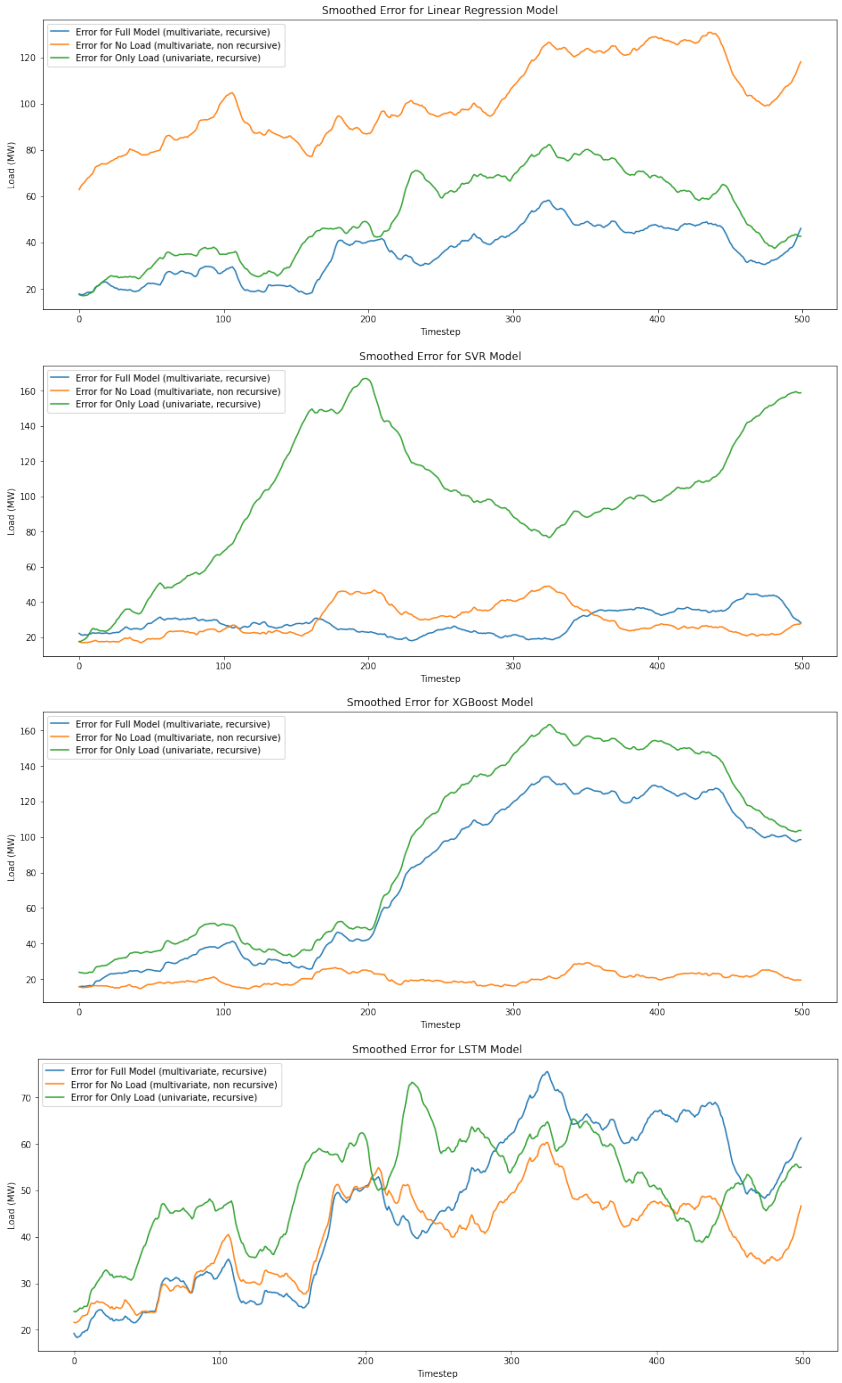


Figure 2: Absolute error of January 1999 forecast, smoothed with a moving average of 50 timesteps for (a) linear regression (b) support vector regression (c) XGBoost regression and (d) LSTM. Full model (blue) uses recursively calculated load and all external features. No load model (orange) uses only external features and no recursion. Only load (green) uses no external features and only recursively calculated load

of activity. As the generated time features were uniformly rather than normally distributed, features were normalized to lie in range $[-1, 1]$. The last month of data was used to evaluate the models. This was done in order to limit potential data leakage by ensuring all evaluation data was drawn from points temporally after the training data. Ten percent of the remaining data was used to validate the models during training, while the remainder was used for training itself. Each model was trained to forecast only one step ahead. We compared three methods: first, a univariate model using only past values of the load to forecast

Table 1: Correlation between forecast error and input feature value for linear regression, LSTM, XGBoost and SVR with rbf kernel. Shown are recursive full models, non recursive (no load) models and recursive univariate (only load) models

Model Name	Load	Weekday	Holiday	Hour	Day Of Year
Linear (Full)	0.5388	-0.0296	0.1878	0.2124	0.2919
Linear (No Load)	*	-0.0284	0.1659	0.2107	0.2414
Linear (Only Load)	0.6260	*	*	*	*
LSTM (Full)	-0.4827	0.2664	-0.0042	-0.0479	0.2982
LSTM (No Load)	*	0.2385	0.5833	0.1944	-0.1310
LSTM (Only Load)	-0.9232	*	*	*	*
XGBoost (Full)	0.6223	-0.3040	-0.0926	0.1341	0.6788
XGBoost (No Load)	*	-0.0505	-0.0083	-0.0560	0.1149
XGBoost (Only Load)	0.8324	*	*	*	*
SVR (Full)	0.1618	-0.1451	-0.0727	-0.0265	0.0239
SVR (No Load)	*	-0.1662	0.0312	0.2091	-0.1016
SVR (Only Load)	-0.7639	*	*	*	*

future values. Each prediction was recursively added to the input for the next timestep. The second was a multivariate model which made use of generated external features in addition to past values of load. The load was updated recursively as in the first model, while external features were calculated directly based on the timestamp of the prediction. The third model removes all recurrent dependency by ignoring previous loads altogether and using only the calculated external features.

For each variant, we compare the performance using four learning models: linear regression, XGBoost, LSTM, and support vector regression. For evaluation, we calculate the absolute error of each model for each timestep, after outputs are scaled back to the original range.

Figure 1 shows the forecasts obtained from the four models. From top to bottom, these are: linear regression, support vector regression, XGBoost, and LSTM.

Green lines represent recursive predictions using the original univariate models, while blue lines represent recursive predictions from the same models with generated features introduced. Orange lines show the non-recursive version which decouples predictions from past values of the output variable and forecasts based only on generated features.

We note that all models are capable of learning short term trends in the data. This is reflected in the high forecast accuracy for short time horizons. We also observe that daily patterns are successfully captured using all methods. The full models generally prove to be the most accurate over short time horizons (less than 1 day), but recursive error begins appear to by as early as the second day, in the case of the LSTM model.

Figure 2 shows the magnitude of the forecasting error for the testing set of January 1999 for all models. To showcase the trend, these are averaged using a moving window of 50 timesteps.

We note that the univariate recursive models generally accumulate significant error by 250 timesteps. This is mitigated in the multivariate recursive models, but due to the recursive nature of the predictions error still rises over time. Nonrecursive models exhibit higher initial error for linear regression and LSTM models while being comparable for SVR and LSTM, but this error remains relatively constant over time. For the linear regression model, nonrecursive error is significantly higher. We believe there are two main reasons for this: first, there is a nonlinear relationship between the features and the load, making prediction difficult for a linear model. Second, the winter of 1999 (our testing set) was unusually cold and resulted in a higher power consumption than previous years. This led to consistent underestimates which were also observed in the SVR and LSTM models. However, use of actual load values in the recursive models anchored these models to higher initial values.

Table 1 shows the Pearson correlation coefficients between error magnitude and feature values. We see that the error correlation decreases from the univariate to the full multivariate model.

5 Conclusion

Forecasting time series with machine learning models has wide applications to our daily life. Reducing errors in the predictions is a paramount concern in the design of these algorithms.

In this paper, we have demonstrated an approach using generated features to convert a univariate model into a multivariate model to mitigate long-term error accumulation. This method can be applied to a variety of machine learning time series models, a selection of which we have studied in this paper. Our experiments show that the addition of generated features improves performance of all univariate models tested over most time horizons, and that it is possible to rely on these added features alone to avoid recursive error accumulation by creating a nonrecursive model. Our results also show that for the majority of models tested, the nonrecursive model can

achieve comparable performance on short time horizons while outperforming recursive models over long time horizons.

This principle of using generated features to create a multivariate model can be used for a wide variety of applications and algorithms. Our method preserves the flexibility of recursive forecasting and allows use of the same model for any forecast length, and can be extended to models which forecast multiple timesteps at once. For future work, performance will be evaluated on other applications such as stock market price forecasting. We will also consider other types of non time-based or composite features which can be generated.

Acknowledgement

Support from the Natural Sciences and Engineering Research Council of Canada (NSERC) is greatly acknowledged.

References

[1] Kadir Amasyali and Nora M. El-Gohary. “Review of Data-Driven Building Energy Consumption Prediction Studies”. Renewable and Sustainable Energy Reviews. Elsevier, 81:1192–1205, 2018.

[2] Gianluca Bontempi, Souhaib B. Taieb and Yann-Ael L. Borgne. “Machine Learning Strategies for Time Series Forecasting”. Lecture Notes in Business Information Processing, Springer-Verlag, 138. DOI: 10.1007/978-3-642-36318-4_3. 2013.

[3] Filippo M. Bianchi, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, and Robert Jenssen. “An Overview and Comparative Analysis of Recurrent Neural Networks for Short Term Load Forecasting”. arXiv preprint arXiv:1705.04378. 2017.

[4] Andreas Christmann and Ingo Steinwart. “Support Vector Machines”. Springer-Verlag, New York, 2008. DOI https://doi.org/10.1007/978-0-387-77242-4.

[5] Chris Chatfield. “Time-Series Forecasting”. CRC Press, ISBN 9781420036206, 2000.

[6] En Chen, Ming-Wei Chang and Chih-Jen Lin. “Load Forecasting using Support Vector Machines: A Study on EUNITE competition 2001”. IEEE, 19:1821–1830, 2004.

[7] Tianqi Chen and Carlos Guestrin. “Xgboost: A Scalable Tree Boosting System”. Proceedings of the 22nd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining. ACM, pp. 785–794, 2016.

[8] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola and Vladimir Vapnik. “Support Vector Regression Machines”. Advances in Neural Information Processing Systems. pp. 155–161, 1997.

[9] EUNITE. “EUNITE Electricity Load Forecast 2001 Competition”. Proceedings of EUNITE. http://neuron-ai.tuke.sk/competition/. 2001.

[10] Jerome Friedman, Trevor Hastie and Robert Tibshirani. “Additive Logistic Regression: A Statistical View of Boosting (with Discussion and a Rejoinder by the Authors)”. Annals of Statistics. 28(2): 337–407, 2000.

[11] John Gamboa and Borges Cristian. “Deep Learning for Time-Series Analysis”. arXiv preprint arXiv:1701.01887. 2017.

[12] Alex Graves and Jurgen Schmidhuber. “Framewise Phoneme Classification with Bidirectional LSTM and other Neural Network Architectures”. Neural Networks. Elsevier. 18 (5-6): 602–610, 2005.

[13] Katarina Grolinger, Alexandra L’Heureux, Miriam A.M. Capretz and Luke Seewald. “Energy Forecasting for Event Venues: Big Data and Prediction Accuracy”. Energy and Buildings. Elsevier, 112: 222–233. 2016.

[14] Jawad Nagi, Keem S. Yap, Farrukh Nagi, Sieh K. Tiong and Syed K. Ahmed. “A Computational Intelligence Scheme for the Prediction of the Daily Peak Load”. Applied Soft Computing. Elsevier, 11 (8): 4773–4788. 2011.

[15] Apurva Narayan and Keith W. Hipel. “Long Short Term Memory Networks for Short-Term Electric Load Forecasting”. 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Banff, Canada. pp. 1050–1059, 2017.

[16] Christopher Olah. “Understanding LSTM Networks”. GITHUB blog. 27. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. 2015.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala. “Pytorch: An Imperative Style, High-Performance Deep Learning Library”. Advances in Neural Information Processing Systems 32 (NIPS 2019). pp. 8026–8037. https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library, 2019.

[18] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincen Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, and Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos and David Cournapeau, Matthieu Brucher, Matthieu Perrot and Édouard Duchesnay. “Scikit-learn: Machine Learning

in Python". Journal of Machine Learning Research. 12: 2825–2830, 2011.

[19] George A. F. Seber and Alan J. Lee. "Linear Regression Analysis". John Wiley & Sons, 2012.

[20] Maher Selim, Ryan Zhou, Wenying Feng and Omar Alam. "Reducing Error Propagation for Long Term Energy Forecasting Using Multivariate Prediction". Proceedings of 35th International Conference on Computers and Their Applications. EPiC Series in Computing, 1: 1–9, 2020.

[21] Souhaib B. Taieb, Gianluca Bontempi, Amir Atiya and Antti Sorjamaa. "A Review and Comparison of Strategies for Multi-step Ahead Time Series Forecasting Based on the NN5 Forecasting Competition". arXiv:1108.3259 [stat.ML]. https://arxiv.org/abs/1108.3259. 2011.

[22] Sanford Weisberg. "Applied Linear Regression". Wiley Series in Probability and Statistics. ISBN 978-1-118-38608-8, 2013.

[23] Kaile Zhou, Chao Fu and Shanlin Yang. "Big Data Driven Smart Energy Management: From Big Data to Big Insights". Renewable and Sustainable Energy Reviews. Elsevier. 56: 215–225, 2016.

[24] Lingxue Zhu and Nikolay Laptev. "Deep and Confident Prediction for Time Series at Uber". arXiv preprint arXiv:1709.01907, 2017.

Maher Selim is a postdoctoral fellow for AI and Machine Learning at Trent University. He obtained his PhD in Physics from the University of Western Ontario, Canada. He also has a M.Sc.in Physics from Helwan University, Egypt. Maher obtained his B.Sc.in Physics from Ain Shams University, Egypt. He is interested in Quantum AI and Quantum Machine learning applications to real world problem.

Ryan Zhou is a Master's student at Trent University. He obtained his B. Eng from Cornell University in Ithaca, New York. He is interested in convolutional and graph neural networks, AI interpretability and machine learning algorithms for regression and time series prediction.

Wenying Feng is a Full Professor at the Department of Computer Science and the Department of Mathematics, Trent University, Canada. She is also an adjunct professor at the School of Computing, Queen's University. Dr. Feng specializes in nonlinear differential equations, nonlinear analysis, machine learning algorithms, mathematical and computational modelling. She has published more than 100 research papers at refereed journals and conference proceedings. She has presented as a keynote speaker, served as program chairs and organized special sessions for international conferences.

Omar Alam is an Assistant Professor at the Department of Computer Science at Trent University. His broad area of interest is in software engineering. In particular, he is interested in Model-Driven Engineering, Aspect-Oriented Modelling, Empirical Software Engineering, and Software Reuse. Dr. Alam has published in premier venues in software engineering, such as MODELS, JSS, SPE, SLE, ICSR, SAM, ICSM. He served as a reviewer and program committee member for various journals and conferences in the field of Model-Driven Engineering and Software Engineering.

A Unity Framework for Multi-Player VR Applications

Alexander Novotny*, Rowan Gudmundsson*, Frederick C. Harris, Jr.*
University of Nevada, Reno, Reno, NV USA 89557

Abstract

We have developed a framework for multi-user Virtual Reality experiences aimed at video games played over a network. Features include tracked avatars, interactable physics objects, peer-to-peer with a user matching system, and voice chat, as well as options to customize these modules for a wide range of support. We go into detail on how several implementation details, such as networking, voice chat, and interaction work. In addition, networking performance details of the framework are included. We also go into detail on how to use the library in Unity for your own projects. We also talk about avatar representation in VR, and how this tool can be used to facilitate many different types of avatar representation.

Key Words: Graphics, user-interface, virtual reality, multiplayer, networking, performance.

1 Introduction

This paper is an extended version of our work published in the Proceedings of CATA 2020 [11].

Multi-Headset Virtual Reality experiences are few and far between which brings about exciting new opportunities when looking at solidifying standards for interacting in this environment. Current multi-player frameworks aren't built to handle the intricacies of Virtual Reality support, and current Virtual Reality frameworks aren't built with the intention of having multiple headsets in the same virtual environment at the same time. As the popularity of virtual reality increases, the need for more diverse experiences will increase as well, and this will lead to a need for multi-player experiences, which have been previously under-explored. In this paper, we introduce a framework and several techniques for multi-user Virtual Reality experiences. This framework builds a foundation for a multitude of multi-headset experiences to be built on top of it. It provides developers with a networking connection layer, voice chat system, and networked physics system.

The rest of this paper is structured as follows: In Section 2 we cover target platforms, other tools, avatars, and social interaction. In Section 3, we discuss the design and implementation of our framework, which includes how we set up our networking stack and multi-user matchmaking as well as performance in Section 3.1, multi-user interaction and object ownership in Section 3.2, avatar representation in Section 3.3, and multi-user voice chat in Section 3.4. We finish the paper with conclusions, successful uses of the framework, and future work in Section 4.

*Department of Computer Science and Engineering, MS 171. Email: anovotny@nevada.unr.edu, rgudmundsson@nevada.unr.edu, fred.harris@cse.unr.edu

2 Background Review

2.1 Target Platforms

Our framework is built for the Unity[14] game engine, but the techniques discussed can easily be extended to any other game engine. Unity was chosen due to its current popularity in individual game development as well as the availability of already established Virtual Reality and networking frameworks, such as Mirror[17], which was chosen for our framework. Mirror allows easy setup for the simple interactions that commonly occur in multiplayer virtual environments while allowing for the possibility of more complex networking interactions. Mirror also allows for simple peer-to-peer communication needed in simple 2-4 player games and server-client communication needed in massively multiplayer experiences. As well, our framework uses OpenVR [15] due to its hardware-agnosticism, however it can easily be extended to other Virtual Reality frameworks. OpenVR also requires Steam[16] to be running, so we targeted Steam users and took advantage of several features of the Steamworks SDK. However none of the methods discussed in this paper require the use of any of these pre-existing frameworks.

For this iteration of the framework we decided to use the HTC Vive as shown in Figure 1 as it has many nice features that we used including a microphone and the option to add more trackers, see Figure 8. Again, this choice is not reflective of the framework but rather an implementation of it. This framework can be extended to any hardware.

2.2 Other Tools

Networking libraries for Unity, such as Mirror [17], Photon [4], and SteamWorks, are common, but come with many downsides to the developer, especially when concerning VR. Photon, for instance, uses a client-server model, where players are matched on Photon's own servers, but Photon expects developers to pay for this service. Developers can also use their own servers or set up server on clients for a peer-to-peer experience, but advertising servers so that users can find each other is also expected to be paid for. Steamworks, meanwhile, provides a peer-to-peer service and a way to match users together for free, but doesn't provide any networking layer for syncing objects in Unity itself. Mirror provides a good peer-to-peer system which syncs objects in Unity, but isn't built to accommodate VR headsets, and by default will not work with

Figure 1: The HTC Vive HMD with controllers

them at all. As well, as a peer-to-peer tool, Mirror does not provide a system of matching users, nor a voice chat system, which are necessary in many modern multi-user experiences.

VR libraries are also common - Unity has a VR library built in, for instance. OpenVR, one of the most popular cross-platform libraries that supports many headsets, also has support for Unity. However, none of these libraries are built to work with multiple headsets, let alone multiple headsets over a network.

2.3 Avatar Representation

When a user is immersed in a virtual environment, there are many choices when it comes to how to represent that user's self/body in the environment. There is good research about the implications/advantages of using different levels of representations of a user's own body in the environment when it comes to immersion, virtual awareness, and computation cost. In singleplayer experiences, increased complexity of player avatars doesn't gain any significant advantage in terms of immersion, while coming at the cost of framerate - something very important in virtual immersion [5, 8, 9, 13]. But in multiplayer experiences, there is an unexplored question of how other users' bodies should be represented and what needs to be tracked/networked to make that level of detail possible.

2.4 Social Interactions

With multiple users in a virtual experience, it becomes important for them to be able to interact in an expected way. This includes having perfect replication of environment, similar interaction schemes between users, and feedback to let players know that they are interacting with another player [12, 18]. A multiplayer Virtual Reality framework must seek to efficiently implement these goals, and allow for other types of social interactions easily.

3 Framework Design

Our framework seeks to fill in the "holes" left by other libraries made for multi-user and VR experiences by providing a peer-to-peer networking stack with user matching which is free to the developer. Our framework is made for Unity, supports a wide variety of user interactions and avatar representations, and has voice chat built in.

3.1 Networking

3.1.1 Networking Setup. The networking API that our framework is built on top of is Mirror. Mirror uses a type of client-server communication where the server can also be a user of the software as well. To make connecting to other users easy, Steamworks was used. Opening of the software requires Steam to be open, and will load a list of friends who are currently running the game, as can be seen in Figure 2. Selecting one of these friends will invite them to join a lobby, thereby starting a server on the user's local machine and marking the two player as a Steam "lobby". Further users will be able to see this lobby and instead of starting another server when inviting those friends, will instead join the already made server as an observer or other user. Steam allows connecting between users with their Steamworks API, allowing easy connections through firewalls without having to know the other users' IP address(es).

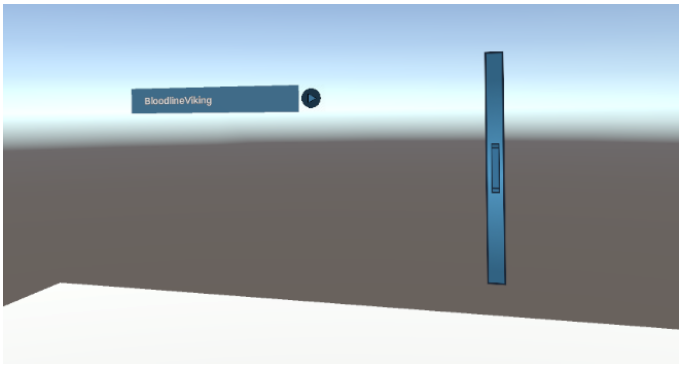


Figure 2: A menu displaying active Steam friends with which to join into a lobby

Starting a server opens a scene normally, but when other users connect to the server, the scene is then cloned to the connecting user. From then on, the server pushes updates to all connected clients syncing the scenes on their computers to the one on the server, so only changes made on the server will be represented to other users. Users can attempt to modify objects in their scene, but these changes will eventually be overwritten by the connection. As well, they can modify objects which are not synced over the network, however these changes will not be represented to other clients.

To override this behaviour, a mechanic called "authority" is used to determine which client has the authority to modify certain objects at any time. Each networked object has a single "authority figure" at any given time, and if this authority figure

is not the server, the syncing behaviour changes to one where the authority figure will push its changes on an object to the server, and the server will re-distribute these changes to all other clients. Using this "authority" method keeps networking costs down to a minimum, and ensures smooth physics if needed.

3.1.2 Networking Performance. Of course, networking performance would be of concern to any potential user of the framework, since poor performance could make it prohibitive for any clients of the end product. The most important performance is that of the host (who is both operating the server and participating in the experience), as they are the central "hub" for all traffic in the experience and will therefore experience the most traffic. The host's download and upload traffic as a function of number of users connected to the experience can be found in Figure 3. As can be seen, the host's download traffic is linear in the number of users, but its upload traffic is quadratic in the number of users. This is due to the increase in number of users which must receive updates and also an increase in the amount of data in each update (from more tracked points). Since the traffic is still within 2-15 KB/s, this should be acceptable for most people's networks, but the number of players within each experience probably shouldn't exceed 10, due to the quadratic nature of the host's upload traffic.

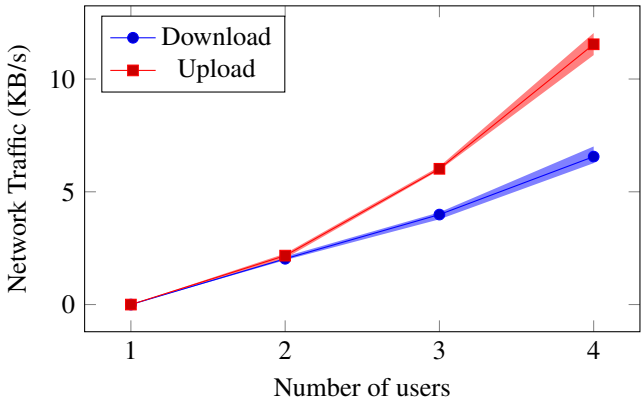


Figure 3: Host network traffic as a function of connected users. Number of tracked points per user is 3. Points taken as mean over 10 second samples from NetLimiter 4 [7] with shaded regions as minimum and maximum observations

Also of interest is the comparison between the host's download traffic and each client's download traffic, which can be found in Figure 4. The client's download speed actually get more efficient as the number of users increase, as more data is sent in each update, whereas the host receives multiple updates from each of the clients.

Finally, the number of updates per second can be adjusted to fit the capabilities of the network and the needs of the program. The network traffic as a function of the updates per second can be found in Figure 5, and is mostly linear, however there is a certain minimum amount of data that needs to be transmitted each second just to maintain a connection and synchronization.

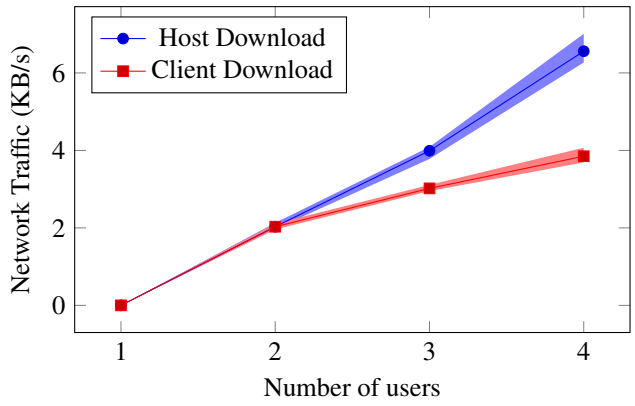


Figure 4: A comparison of the host's download traffic and a client's download traffic as a function of number of connected users. Points taken as mean over 10 second samples from NetLimiter 4 [7] with shaded regions as minimum and maximum observations

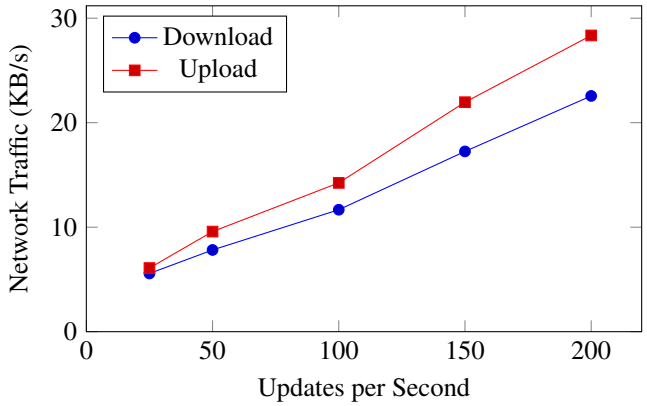


Figure 5: The host's mean network traffic as a function of update speed. Points taken as mean over 10 second samples from NetLimiter 4 [7]

3.2 Multi-User Interaction

3.2.1 Tracking Users. The first step to creating a multi-user environment is tracking those users throughout the environment. This is done natively by many Virtual Reality frameworks, but typically not in a multi-user fashion. Our first attempt to track users in the environment was to simply network the objects tied to the tracked pieces of the user. However, this didn't work as using multiple player objects in a single scene caused each connected player to influence the motions of each player in the scene. OpenVR picks up all player objects in the scene as a controllable entity and so would change the position of the tracked points in each model simultaneously. In order to rectify this, we disabled all the components in the scene tracked by OpenVR which were not controlled by the local player (the

player the current client is supposed to control) (Figure 6). This method seemed like the most reasonable solution to the problem without delving too far into Steam's OpenVR implementation.

Additionally, in order to create a more immersive environment, we decreased the sync rate between the server and the clients to allow for more smooth movement in the players and interactable objects in the scene. We changed the sync rate from it's default of 100 milliseconds to 10 milliseconds. This gave the players almost seamless movement and made interacting with objects with multiple players very fluid and life-like.

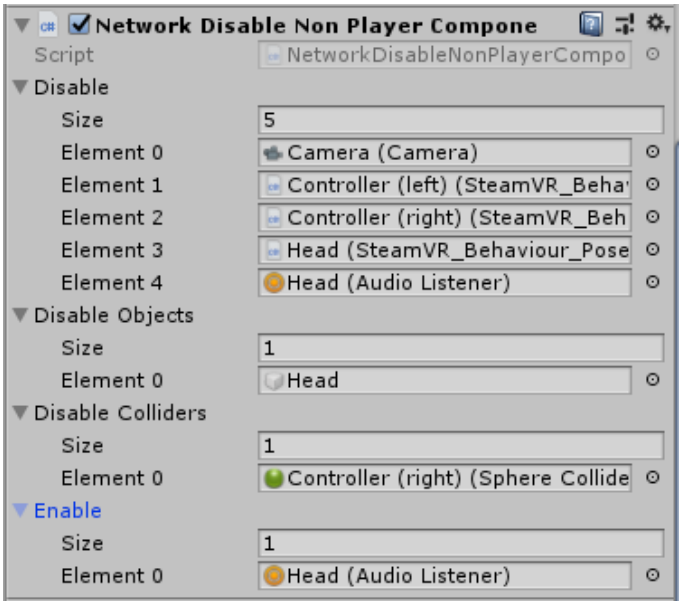


Figure 6: This figure shows the components which are being disabled for non-local players in the scene

3.2.2 Tracking Objects. The next step in creating a multi-user environment is to allow players to interact with objects in the scene together. Doing this on a single machine in Virtual Reality is trivial since we only need to worry about keeping the physics updated on the local machine. In a multi-user setting however, we need to worry about how the physics of a given object is tracked across all clients. Some challenges we faced were figuring out which client should have "authority" over an object at a given time and how physics should be tracked over the network. We settled on only keeping track of physics on the machine which has authority over the tracked object and then just updating the position over all clients (Figure 7). This method seemed to allow for the best performance since no information about the physics is transferred over the network only information about position, rotation, etc.

3.3 Avatar Representation

In single-player virtual experiences, it has been shown that there is no notable increase in immersion or self-presence with

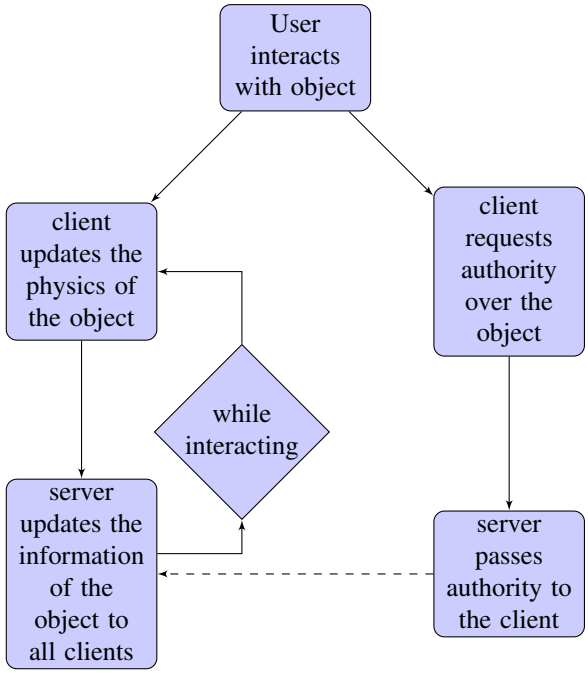


Figure 7: The flow of operation for interacting with an object on a client

the addition of more-realistic player avatars. However, in a multi-player environment, this can change. Not only does the user have to keep track of their own avatar, but they now also must be able to keep track of other avatars as well. There is also now potentially a need for a user to be able to see the same avatar that everyone else is seeing.

To keep networking costs low, only three positions are sent over the network: head position, and hand positions. However, this can be expanded using the trackers shown in Figure 8 to track additional joints in the avatar. After these positions are sent over the network, each client then separately updates the avatars of each player with respect to these positions. This allows for a large amount of freedom with player avatars, including more complex player avatars through the use of techniques such as inverse kinematics. Players can choose which avatars they would like to represent themselves and others without impacting the other players. Some examples of dynamic avatar representations can be seen in Figures 9 to 11.

3.4 Multi-User Voice Chat

Another form of interaction one might wish to have in a multi-user virtual environment is speech. Indeed, every modern Head-Mounted Display made for Virtual Reality (including, in our case, the HTC Vive) has a microphone array built-in with this purpose in mind, meaning voice chat is accessible to everyone.

Steamworks makes using these microphones easy - the library will automatically pick up on and compress any audio from the microphone on the headset. This is stored in a buffer until the appropriate retrieval function is called, upon which time

Figure 8: Trackers which can sync with the Vive to add additional tracking points to the avatar [1]

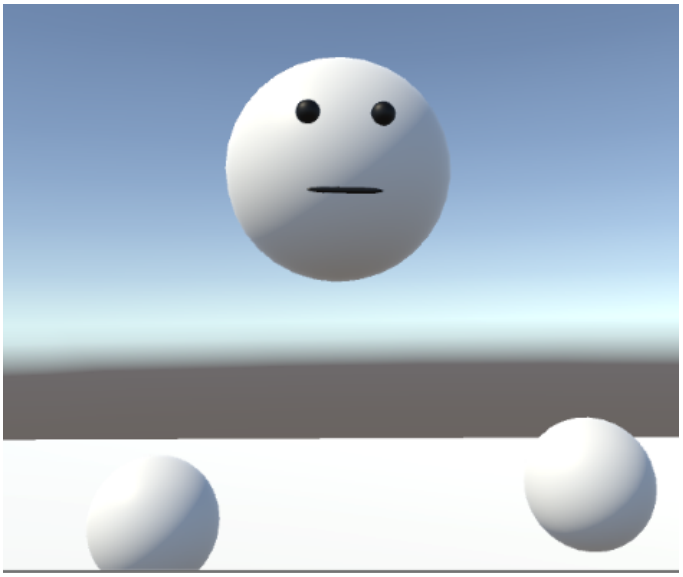


Figure 9: A simple player model which allows for good social interactions

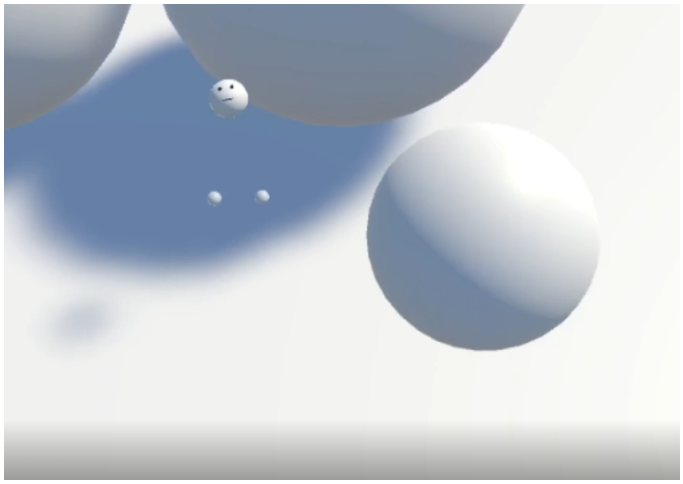


Figure 10: A much smaller player avatar - player avatars can be as flexible as you want!

Figure 11: A complex player avatar rigged over the network [3]

it gives access to 16-bit compressed Pulse-Code Modulation (PCM) audio. This is ideal for sending over the network, and we implemented a custom network package to deliver the audio containing a buffer for the audio, a player ID to keep track of the origin of the audio, and a channel ID for special purposes. This is sent to the server, which then re-sends it to every other client. When a client receives this package, it finds the audio source associated to the player ID, decodes the audio, and stores it in a buffer waiting to play. This is a bit tricky, as the C# version of Steamworks returns a buffer of 8-bit integer values to represent 16-bit audio and Unity requires 32-bit floating point values between -1 and 1. As well, C# is little-endian (i.e. high order bytes are stored after low order bytes), so there is a bit of finesse required to turn this audio into something useable. Once this is done, however, the audio source is set to stream from the buffer of incoming voice audio, making the audio sound like it is coming from that player.

As well, one can use the aforementioned channel ID to change how this works. Channels can be used to filter out certain players from hearing other players, joining certain voice chat channels, and changing which audio source to play the incoming audio from. For instance, a mining evacuation simulator used voice channels to make player audio come out of walkie-talkies instead of being played directly from the other player, as players were often on other sides of the mine [2].

By Using SteamVR actions, this allows players and developers flexibility in how they want to be able to talk to other players. By default, the framework is set up to use a push-to-talk schema where as user will push a button to start talking and then push it again to stop while an indicator lets them know that they are broadcasting Figure 12. This can be easily configured by both developers and users to become a hold-to-talk scheme or an always-on scheme where users are always chatting. The framework doesn't send voice packages over the network unless noticeable audio is detected, so silent users of an always-on scheme won't cause network stress. As well, action sets can

be configured to make a more dynamic voice chat experience. For instance, the aforementioned mining simulator uses walkie-talkies whose push-to-talk button doesn't become available until the player picks up the walkie-talkie into their hand thereby switching action sets [2].

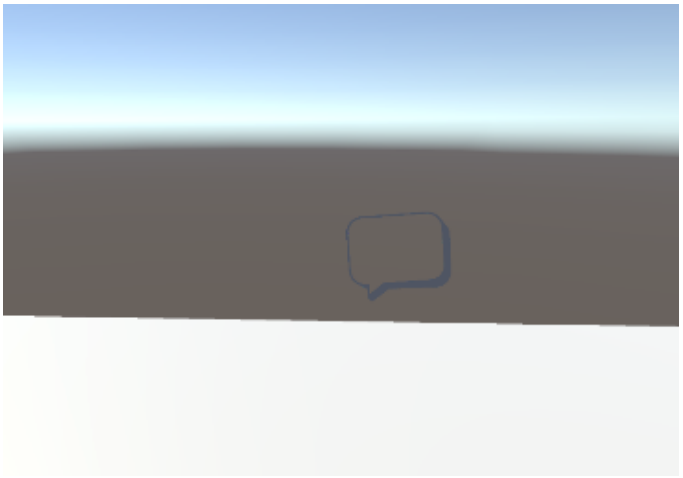


Figure 12: An indicator lets a user know that they are broadcasting to other players

4 Conclusions and Future Work

We believe this library will provide to be a useful framework for other multi-user virtual reality experiences, which are becoming more and more common. Integration with Steam, the world's largest game distribution and VR platform, and Unity, one of the largest game engines in use today, together with being free to the developer, we believe this framework will be a great boon to developers looking to get into this new market.

We are planning to include IBM Watson support with the voice-chat feature, which would allow for written transcripts of audio sessions during the experience. Planned use cases include 'replays' of scenarios in the experience, as well as voice-to-text chat in the experience. We also plan to use the feature in our own virtual reality game to trigger certain features of the game off of certain key phrases.

Various projects and papers have already successfully made use of the framework for multi-player games and multi-user experiences. METS VR [2], a mining evacuation training simulator, requires multiple users to be able to train simultaneously and an "operator" who operates the simulation from within the digital space. VFireVI [6] simulates wildfires in a virtual space and allows multiple users to experience and cause the virtual wildfires together using a centralised fire simulation server. Several video games have also been developed which make use of the framework, such as a cooperative puzzle game [3] and a competitive "clan-like" tower defense game [10]. Our own virtual reality game is also in the works, which makes use of asymmetric player avatars to have players solve puzzles using specific roles.

Acknowledgment

This material is based in part upon work supported by the National Science Foundation under grant numbers IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Amazon. "HTC Vive Tracker (2018) - European Version", 2018. https://www.amazon.com/HTC-Vive-Tracker-European-Version/dp/B07BYVB3RW/ref=olp_product_details?_encoding=UTF8&me=&qid=1570916982&sr=8-6, Last Accessed: Aug. 17, 2020.

[2] Kurt Andersen, Simone José Gaab, Javad Sattarvand, and Frederick C. Harris. "METS VR: Mining Evacuation Training Simulator in Virtual Reality for Underground Mines". In Shahram Latifi, Editor, *17th International Conference on Information Technology–New Generations (ITNG 2020)*. Springer International Publishing, Cham, pp. 325-332, 2020.

[3] Lucas Calabrese, Andrew Flangas, and Frederick C. Harris. "Multi-User VR Cooperative Puzzle Game". In Shahram Latifi, Editor, *17th International Conference on Information Technology–New Generations (ITNG 2020)*. Springer International Publishing, Cham, pp. 293-299, 2020.

[4] Exit Games. "Photon". https://www.photonengine.com/en/pun, Last Accessed: Aug. 17, 2020.

[5] L. Kruse, E. Langbehn, and F. Stelnlcke. "I Can See on My Feet While Walking: Sensitivity to Translation Gains with Visible Feet". In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 305-312, March 2018.

[6] Christopher Lewis, Ronn Siedrik Quijada, and Frederick C. Harris. "vFireVI: 3D Virtual Interface for vFire". In Shahram Latifi, Editor, *17th International Conference on Information Technology–New Generations (ITNG 2020)*. Springer International Publishing, Cham, pp. 309-315, 2020.

[7] Locktime Software. "NetLimiter 4". https://www.netlimiter.com/, Last Accessed: Aug. 17, 2020.

[8] J. Lugrin, M. Ertl, P. Krop, R. Klüpfel, S. Stierstorfer, B. Weisz, M. Rück, J. Schmitt, N. Schmidt, and M. E. Latoschik. "Any "Body" There? Avatar Visibility Effects in a Virtual Reality Game". In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 17-24, March 2018.

[9] J. Lugrin, M. Wiedemann, D. Bieberstein, and M. E. Latoschik. "Influence of Avatar Realism on Stressful Situation in VR". In *2015 IEEE Virtual Reality (VR)*, pp. 227-228, March 2015.

[10] Andrew E. Munoz, Zach Young, Sergiu Dascalu, and Frederick C. Harris. "TDVR: Tower Defense in Virtual Reality: A Multiplayer Strategy Simulation". In Shahram Latifi, Editor, *17th International Conference on Information Technology–New Generations (ITNG 2020)*. Springer International Publishing, Cham, pp. 301-307, 2020.

[11] Alexander Novotny, Rowan Gudmundsson, and Frederick C. Harris, Jr. "A Unity Framework for Multi-User VR Experiences". In Gordon Lee and Ying Jin, Editors, *Proceedings of 35th International Conference on Computers and Their Applications*, EPiC Series in Computing, EasyChair, 69: 13-21, 2020.

[12] D. Roth, C. Klelnbeck, T. Feigl, C. Mutschler, and M. E. Latoschik. "Beyond Replication: Augmenting Social Behaviors in Multi-User Virtual Realities". In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 215-222, March 2018.

[13] D. Roth, J. Lugrin, D. Galakhov, A. Hofmann, G. Bente, M. E. Latoschik, and A. Fuhrmann. "Avatar Realism and Social Interaction Quality in Virtual Reality". In *2016 IEEE Virtual Reality (VR)*, pp. 277-278, March 2016.

[14] Unity Technologies ApS. Unity. `https://unity.com/` Last Accessed: Aug. 17, 2020.

[15] Valve Corporation. OpenVR. `https://github.com/ValveSoftware/openvr`, Last Accessed: Aug. 17, 2020.

[16] Valve Corporation. Steam. `https://steampowered.com`, Last Accessed: Aug. 17, 2020.

[17] vis2k. Mirror. `https://github.com/vis2k/Mirror`, Last Accessed: Aug. 17, 2020.

[18] C. Wienrich, K. Schindler, N. Döllinqer, S. Kock, and O. Traupe. "Social Presence and Cooperation in Large-Scale Multi-User Virtual Reality - The Relevance of Social Interdependence for Location-Based Environments". In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 207-214, March 2018.

Alexander Novotny is currently a student at the University of Nevada, Reno. He is pursuing a BS with majors in Pure Mathematics and Computer Science and Engineering. He is currently working as a Software Engineer at Scientific Games, developing slot machine games and in-house development tools. He is currently taking graduate-level courses in preparation for an accelerated MS program in Computer Science and Engineering at the University of Nevada, Reno.

Rowan Gudmundsson completed his BS in Computer Science and Engineering at the University of Nevada, Reno in 2020. He is currently working as a Software Engineer on the design and implementation of computer games in Unity. His research interests include tool design, and tool construction, and alternative applications for virtual reality. His current goals are to gain experience in the field and then pursue research opportunities while receiving his MS in Computer Science.

Frederick C. Harris Jr. received his BS and MS degrees in Mathematics and Educational Administration from Bob Jones University, Greenville, SC, USA in 1986 and 1988 respectively. He then went on and received his MS and Ph.D. degrees in Computer Science from Clemson University, Clemson, SC, USA in 1991 and 1994 respectively.

He is currently a Professor in the Department of Computer Science and Engineering and the Director of the High Performance Computation and Visualization Lab at the University of Nevada, Reno, USA. He is also the Nevada State EPSCoR Director and the Project Director for Nevada NSF EPSCoR. He has published more than 250 peer-reviewed journal and conference papers along with several book chapters. He has had 14 PhD students and 78 MS Thesis students finish under his supervision. His research interests are in parallel computation, simulation, computer graphics, and virtual reality. He is also a Senior Member of the ACM, and a Senior Member of the International Society for Computers and their Applications (ISCA).

Predicting Cerebral Aneurysm Rupture using Gradient Boosting Decision Tree based on Clinical, Computational Fluid Dynamics and Geometric Data

Toshiyuki Haruhara[1], Masaaki Suzuki*[,1], Hideto Ogi[1], Hiroyuki Takao[2,3], Takashi Suzuki[2], Soichiro Fujimura[2,3], Toshihiro Ishibashi[2], Makoto Yamamoto[3], Yuichi Murayama[2], Hayato Ohwada[1]

[1] Tokyo University of Science, Noda-shi, Chiba 278-8510, JAPAN
[2] Jikei University School of Medicine, Minato-ku, Tokyo 105-8461, JAPAN
[3] Tokyo University of Science, Katsushika-ku, Tokyo 125-8585, JAPAN

Abstract

Advances in computing continue to improve the human condition in many ways, including in healthcare. This paper focuses on the issue of accurately predicting a stroke, particularly a subarachnoid hemorrhage. A stroke is a serious cerebrovascular condition that causes brain cells to die, due to an abrupt blockage of the arteries that supply blood and oxygen to the brain. A stroke could also be caused when blood vessels burst, causing bleeding in the brain. Since the onset of a stroke may be very sudden, its prevention is often difficult. In Japan, a stroke ranks fourth as the major cause of death. It is associated with high medical costs and these problems are exacerbated by the aging population. Thus, stroke prediction and treatment are important issues to address. Based upon a patient's risk of stroke, preventive treatments are usually provided. However, since characterizing the risks of stroke typically depends upon an individual's health history as well as the skillset of the doctor, a highly accurate prediction method for strokes that is independent of the doctor's experience and skills would be highly desirable. This study focuses on a predictive method for a particular type of stroke—the subarachnoid hemorrhage. Two gradient boosting decision tree methods—the LightGBM and the XGBoost, along with the support vector machine (SVM) method were used to predict the rupture of cerebral aneurysms using a machine learning process that incorporates clinical, hemodynamic, and morphological information. These classification models were used to analyze samples from 338 cerebral aneurysm cases (35 ruptured, 303 unruptured). Simulations of the cerebral blood flow was used to calculate the hemodynamic features, while the surface curvature was extracted from 3D blood-vessel-shape data as morphological features. Comparing the machine learning methods using the performance metrics of sensitivity and specificity, it was found that the LightGBM method yielded the best performance, with a sensitivity of 0.77 and a specificity of 0.83 in accurately predicting potential rupture of cerebral aneurysm.

Key Words: Artificial intelligence; machine learning; gradient boosting decision tree; computational fluid dynamics simulation; stroke; subarachnoid hemorrhage; cerebral aneurysm rupture.

1 Introduction

Advances in computing continue to improve the human condition in many ways, including in healthcare. This paper focuses on the issue of accurately predicting a stroke, particularly a subarachnoid hemorrhage. A stroke is a generic term that corresponds to cerebral infarction, cerebral hemorrhage, and subarachnoid hemorrhage. It occurs as a result of a severe cerebrovascular injury and causes the destruction of brain cells due to an abrupt blockage of arteries transferring blood and oxygen to the brain or due to bleeding in brain tissue provoked by a burst of blood vessels. In many cases, stroke may occur suddenly and without warning signs; therefore, it is difficult to prevent. In 2018 in Japan, stroke took fourth place among the leading causes of death due to illness and the first place among the causes of being bedridden. Therefore, it is crucial to introduce effective methods of early prediction and treatment for stroke patients. Reducing the incidence of stroke requires a preventive strategy that lowers the associated risks. However, the outcome of stroke risk evaluation significantly depends on the individual judgments and expertise of doctors. Therefore, a highly accurate method for predicting stroke risk regardless of the experience and judgment of medical personnel is required.

Some of the existing stroke-prediction models [15], [16] incorporated the features that were known as clinically verified or were manually selected by medical experts. For example, [11], [14], and [26] used the medical history data of patients

*Email: m-suzuki@rs.tus.ac.jp

as input features in their research. Amini et al. [2] applied the k-nearest neighbor algorithm [1] and the C4.5 decision tree method [22] to predict a stroke onset based on the medical history data of patients. Moreover, several studies suggested to analyze vascular imaging with the purpose of predicting disease onsets. For example, Nogueira et al. [20] employed vascular imaging to predict clinical outcomes through investigating the risk of symptomatic intracerebral hemorrhage among the patients undergoing the intravenous thrombolytic treatment. Bentley et al. [4] utilized computerized tomography brain images as the inputs into a support vector machine (SVM) algorithm [9] to predict strokes. More recently, Kim et al. [13] applied a convolutional neural network (CNN) to aneurysm images to predict the rupture of small-sized intracranial aneurysms and demonstrated the applicability of deep learning algorithms to the rupture risk assessment.

In several other reports, the state of cerebral blood flow was considered in addition to the medical information of patients to predict a stroke onset [7], [19]. Morino et al. [18] applied particle image velocimetry and laser doppler velocimetry to evaluate the velocity profiles of ruptured and unruptured intra-aneurysmal hemodynamics. Xiang et al. [28] examined how an inlet waveform affected the predicted hemodynamics in the patient-specific aneurysm geometries. Furthermore, several research groups acknowledged the significance of the wall shear stress (WSS), energy loss (EL), and pressure loss coefficient (PLC) in predicting the rupture of cerebral aneurysms [21], [23], [25].

Among these studies, only a limited number of them considered combining the data from various technological sources to successfully predict a stroke onset. Aranda and Valencia [3] relatively recently employed machine learning (ML) using combined geometric data and intracranial blood-flow-simulation data to analyze 60 saccular aneurysm samples. In this regard, in our previous study, we suggested combining the clinical, hemodynamic, and morphological information within a classification model to realize an enhanced prediction of stroke [24]. Moreover, we aimed to develop a highly accurate stroke-onset prediction method based on ML techniques. Specifically, we constructed an ML-based model predicting whether a cerebral aneurysm would rupture and cause subsequent subarachnoid hemorrhage. The proposed model relied on logistic regression and SVM incorporating the clinical information, and hemodynamic information derived from the computational fluid dynamics (CFD) simulation data of cerebral blood flow, as well as the morphological information obtained from the 3D blood-vessel-shape data. Analyzing the performance of logistic regression as a classification model for a total of 338 cerebral aneurysm data samples, we found that it yielded a sensitivity of 0.64 and a specificity of 0.85.

In [10], Haruhara et al. applied LightGBM [12], a type of gradient boosting decision tree method actively used in the recent data analysis competitions, as a classifier. Here, the obtained time-series data from the CFD simulation of the cerebral blood flow were considered as hemodynamic features. Additionally, the surface curvature data representing a cerebral aneurysm obtained from the 3D blood-vessel-shape data were considered as morphological features. In the present study, the main purpose is to find an effective classification model for predicting the rupture of cerebral aneurysms. To achieve this, we conduct the experiments to evaluate and compare the prediction performance of two gradient boosting decision tree methods, LightGBM and XGBoost [6], along with a conventional ML method, SVM.

In this present paper, which is an extension of [10], we describe the data that are required to construct the proposed classification model (Section 2), the process of developing the classifiers (Section 3), the results of the numerical experiments, and the implications of these results (Section 4). The conclusions are presented in Section 5.

2 Dataset

In the present study, we considered the total of 6,470 cases previously registered in the Jikei University database. Among these cases, we first extracted the cases for analysis based on the occurrence location of an aneurysm. If a case was unruptured, we extracted the cases that are being observed and have not been treated in the past. If a case was ruptured, we extracted the cases that ruptured during the follow-up visits. In addition, we applied a morphological classification to restrict the cases to those in which the length, width, and the neck of a bulge were each < 10 mm but at least one of them was > 3 mm. Furthermore, we restricted the set of the considered unruptured cases to those in which the follow-up period[1] was over two years and then analyzed all the consecutive cases. Finally, 338 cases were selected for the analysis. We extracted the clinical, hemodynamic, and morphological information from 338 cases, including 303 unruptured and 35 ruptured samples.

2.1 Clinical Information

The following clinical information attributes were considered in each case: patient age; sex; aneurysm location; the history of subarachnoid hemorrhage (SAH); the history of smoking (SH); diabetes mellitus (DM); hypertension (HT); hyperlipidemia (HL); alcohol consumption (AC); polycystic kidneys (PK); cerebral hemorrhage (CH); hormone replacement (HR); the date of the last consultation (discretized in units of three months and ten days.); the family history of SAH (FH_SAH); the family history of unruptured aneurysms (FH_UA); and the family history of PK (FH_PK). A total of 32 features were extracted from the patient medical history. Comparisons of main clinical features between the rupture and unruptured groups are shown in Table 1.

[1] The follow-up period is defined as the time between the initial consultation and the final consultation.

Table 1: Statistical comparison of clinical features between the rupture and unruptured groups (The data represent the number of samples for categorical features and the mean value for continuous features)

		Rupture (n=35)	Unruptured (n=303)
Age [years]		62.7	69.9
Sex	Male : Female	24 : 11	188 : 115
SAH	No : Yes	35 : 0	303 : 0
SH	None : Past : Current	24 : 8 : 3	183 : 68 : 52
DM	No : Yes	35 : 0	283 : 20
HT	No : Yes	17 : 18	135 : 168
HL	No : Yes	31 : 4	245 : 58
AC	No : Yes	29 : 6	258 : 45
PK	No : Yes	35 : 0	303 : 0
CH	No : Yes	35 : 0	303 : 0
FH_SAH	No : Yes	31 : 4	261 : 42
FH_UA	No : Yes	34 : 1	300 : 3
FH_PK	No : Yes	34 : 1	302 : 1

2.2 Hemodynamic Information

The hemodynamic information was obtained through the cerebral blood flow CFD simulation. CFD is regarded as a branch of the fluid mechanics that employs numerical analyses to solve the problems associated with fluid dynamics. The simulation results exhibited the physical blood-flow characteristics, including PLC, EL, energy loss per unit volume (ELV), inflow concentration index (ICI), WSS, oscillatory shear index (OSI), low shear-stress area percentage (LSA), low shear index (LSI), and shear concentration index (SCI). In our previous study [24], we considered only the maximum, minimum, amplitude, and the average values of these quantities. In the present research, we additionally computed the ratios between the maximum and minimum values for PLC, EL, ELV, ICI, LSA, LSI, and SCI. Among these characteristics, PLC, EL, and WSS were reported as representative in predicting whether a cerebral aneurysm could rupture [21], [23], [25]. In addition, we extracted the time-series features of the cerebral blood flow velocity, pressure, shear force, and WSS based on the CFD simulation data. The duration of registering these time-series data was 0.8 seconds while the sampling interval was 0.05 seconds. Concerning a cerebral aneurysm, we identified the positions where the value of each physical parameter was at maximum during 0.8 seconds and the positions where the variance of each physical parameter took the maximum value during 0.8 seconds (eight positions in total). Thereafter, the rates of change during the time window of 0.05 seconds for each physical parameter at those positions were employed as the time-series features in the ML model. Overall, a total of 181 features were extracted from the blood-flow-simulation data. Comparisons of main hemodynamic features between the rupture and unruptured groups are shown in Table 2.

The calculation conditions for the CFD simulation are summarized below. We utilized a prototype CFD solver (Siemens Healthcare GmbH, Forchheim, Germany, "Not to be used for Diagnosis and/or Therapy") based on the Lattice Boltzmann method [5]. Concerning the physical properties of blood, the fixed density and viscosity values were set, and the non-Newtonian fluids were disregarded. After considering the laminar flow field, the two pulses were calculated based on the pulse conditions, and only the results corresponding to the second pulse were used. The outlet boundary condition was set to the average static pressure of 0 Pa. The calculations were established in a structured computational grid with the maximum size of 0.1 mm. More detailed description of the process was provided in the previously published studies [21],[25].

2.3 Morphological Information

In the present study, the considered morphological information about a cerebral aneurysm included the maximum aneurysm height, maximum neck diameter, neck area, volume, aspect ratio, sidewall or bifurcation type, and the presence or absence of a bleb. To extract additional features from the 3D blood-vessel-shape data stored in the stereolithography (STL) format, we estimated the curvatures on the vessel surface and employed these characteristics as features. This method allowed extracting the following four characteristics related to the surface curvature: mean curvature, Gaussian curvature, root mean square curvature, and absolute curvature. MeshLab [8] was utilized to read and analyze the STL files representing the blood-vessel-shape data and to derive the surface curvatures. We considered the histograms of each of the four surface curvatures as morphological features. As a result, a total of 257 features were extracted from the morphological data. Comparisons of main morphological features between the rupture and unruptured groups are shown in Table 3.

Table 2: Statistical comparison of main hemodynamic features between the rupture and unruptured groups (The data represent the mean value)

	Rupture (n=35)	Unruptured (n=303)
Time-averaged PLC	1.17	1.49
Time-averaged EL [mW]	2.6×10^{-4}	2.6×10^{-4}
Time-averaged ICI	0.72	0.76
Time-and-space-averaged WSS [Pa]	2.57	3.26
Space-averaged OSI	0.02	0.02
Time-averaged LSA	0.45	0.37
Time-averaged LSI	0.17	0.14
Time-averaged SCI	4.18	2.98

Table 3: Statistical comparison of main morphological features between the rupture and unruptured groups (The data represent the number of samples for categorical features and the mean value for continuous features)

		Rupture (n=35)	Unruptured (n=303)
Max. height [mm]		5.02	3.84
Max. neck diameter [mm]		4.97	4.89
Neck area [mm^2]		17.0	15.7
Volume [mm^3]		83.2	46.4
Aspect ratio		0.83	0.69
Type	Sidewall : Bifurcation	9 : 26	115 : 188
Bleb	No : Yes	23 : 12	279 : 24

3 Classification Model for the Cerebral Aneurysm Rupture Prediction

In the present study, the LightGBM, XGBoost, and SVM methods were employed as the classifiers to predict whether a cerebral aneurysm would rupture or not.

3.1 Machine Learning Algorithms

Available as open-source software libraries, LightGBM and XGBoost provide gradient-boosting decision tree frameworks. Gradient boosting is an algorithm that uses a technique called boosting, which is a type of ensemble learning. In ensemble learning, multiple models ("weak learners") are trained to solve the same problem and then are combined to obtain better predictive performance. In particular, boosting is applied to sequentially train weak learners based on the errors of a previous weak learner. Gradient-boosting is one of the most widely used methods in recent data analysis competitions due to its high efficiency and predictive power. LightGBM and XGBoost are both gradient-boosting algorithms that use decision trees as weak learners, but one of the differences between them is the way of training decision trees. When training each decision tree, two strategies can be employed: level (depth)-wise tree growth and leaf-wise tree growth. Level-wise tree growth splits all leaves at a given depth before adding more depth. Most decision tree learning algorithms, including XGBoost, grow trees level-wise. Leaf-wise tree growth, on the other hand, determines on splits leaf-by-leaf basis, i.e., selectively splits the leaf node that reduces the loss the most. LightGBM grows trees leaf-wise. Compared with traditional level-wise tree growth, the leaf-wise tree growth adopted by LightGBM is more efficient and tends to require less time and memory for training. However, it is concluded that LightGBM's performance is not necessarily better than XGBoost's and should be used on a case-by-case basis.

SVM can be used to map the input data onto a high-dimensional feature space using a kernel function and then to construct an optimal classification hyperplane maximizing the distance between the hyperplane and the nearest data points of each class. In this study, a SVM with the radial basis function (RBF) kernel, a nonlinear kernel, is employed as it is not possible to separate the data linearly.

3.2 Hyperparameters

All three classifiers can internally produce the predicted probability values that range between 0.0 and 1.0 rather than the predicted label values such as "rupture" or "unruptured". Therefore, we had to set a probability threshold to label the outcome as rupture or unruptured. To determine the threshold value, the harmonic mean of the sensitivity and specificity was

computed. Sensitivity and specificity can be calculated from the confusion matrix. The confusion matrix is a table that classifies the prediction results of the classification model for the test data into four categories, True positive, True negative, False positive, and False negative, and summarizes the number of each (Table 4).

Table 4: Schematic of the confusion matrix

		Actual class	
		Positive	Negative
Predicted class	Positive	True positive	False positive
	Negative	False negative	True negative

The meaning of each category is as follows:

- True positives (TP): Number of positive samples that have been correctly predicted as positive.
- True negatives (TN): Number of negative samples that have been correctly predicted as negative.
- False positives (FP): Number of negative samples that have been mispredicted as positive.
- False negatives (FN): Number of positive samples that have been mispredicted as negative.

In this study, the rupture of a cerebral aneurysm is defined as positive, and unruptured is defined as negative. The sensitivity was calculated using Eq. (1) and represented the fraction of the ruptured samples that were correctly predicted out of the total number of ruptured samples.

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{1}$$

The specificity was computed by Eq. (2) and was regarded to the fraction of the correctly predicted unruptured samples out of the total number of unruptured samples.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN}+\text{FP}} \tag{2}$$

A threshold value maximizing the harmonic mean of the sensitivity and specificity was regarded as the optimal threshold value (T_{opt}). We calculated the harmonic mean H using the following equation:

$$H = \frac{2 \cdot \text{Sensitivity} \cdot \text{Specificity}}{\text{Sensitivity} + \text{Specificity}} \tag{3}$$

To improve the sensitivity of a classifier, fine-tuning was performed through multiplying the obtained optimum threshold T_{opt} by 0.9 as per the equation below:

$$T^*_{\text{opt}} = 0.9 \cdot T_{\text{opt}} \tag{4}$$

The other hyperparameters were manually tuned to maximize the sum of the sensitivity and specificity.

4 Results and Discussion

We used the 338 cerebral aneurysm data samples (35 ruptured, 303 unruptured) extracted in Section 2 to construct the three classifiers and test their performance by stratified 10-fold cross-validation (CV). CV is a statistical technique to evaluate predictive models by splitting the original data into two segments: one used to train the model and the other used to evaluate it. In k-fold CV, the original data is first randomly divided into k equally sized folds. Of the k folds, a single fold is retained as the test data for evaluating the model, and the remaining $k-1$ folds are used as training data. Subsequently, k iterations of training and test are performed such that within each iteration, a different fold of the data is held-out for the test. The k results from the folds can then be combined (or otherwise averaged) to produce a single estimation. Figure 1 represents a schematic of a CV with $k = 10$. Stratified k-fold CV, in which the folds are selected so that each fold contains roughly the same proportions of class labels, is typically used for classification problems.

4.1 Hyperparameter Tuning

Tables 5 represent the probability threshold values, T^*_{opt}, resulting from the learning of the training data.

Table 5: Probability threshold values

LightGBM	XGBoost	SVM
0.113	0.407	0.095

The results of hyperparameter tuning for LightGBM and XGBoost are provided in Table 6. Here, "n_estimator" is the number of decision trees to build, "learning_rate" is a parameter that adjusts the weighting of new trees, and "max_depth" is the maximum depth of the decision trees. The other hyperparameters were assigned with default values, and their details are available on [17], [27]. Table 7 provides the results of hyperparameter tuning for SVM. Here, "C" is a regularization parameter that adjusts how much misclassification is allowed, and "gamma" is a kernel coefficient that adjusts the range of influence of a single training data.

Table 6: Hyperparameters selected for the LightGBM and XGBoost

Hyperparameter	Candidate values	Selected value
n_estimators	10, 50, 100	10
learning_rate	0.01, 0.02, 0.05, 0.1	0.02
max_depth	3, 5, 6, 8, 10	6

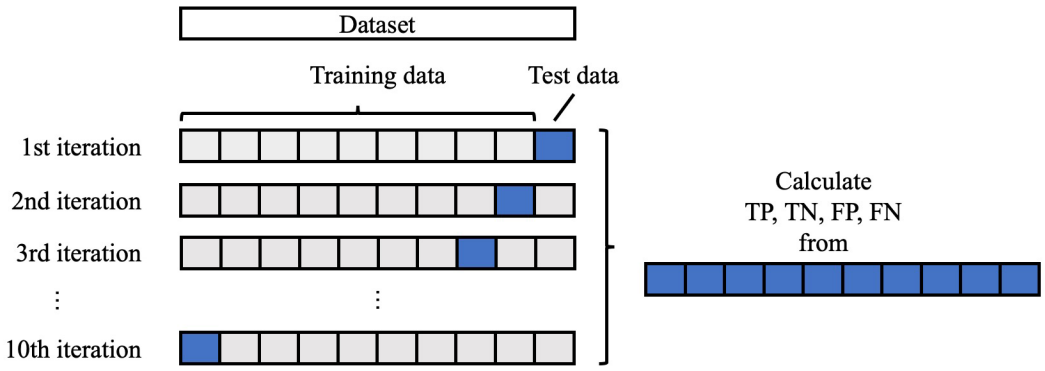


Figure 1: Schematic of a 10-fold CV

Table 7: Hyperparameters selected for SVM with the RBF kernel

Hyperparameter	Candidate values	Selected value
C	0.01, 0.1, 1, 10, 100	1.0
gamma	*'scale', 'auto'*	*'auto'* *

* if 'auto', uses the reciprocal of the number of features.

4.2 Predicting Cerebral Aneurysm Rupture

The performance of the considered classification models were evaluated in terms of the sensitivity, specificity, and F-measure that was defined as a harmonic mean of the precision and sensitivity and was computed according to Eq. (5).

$$\text{F-measure} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}} \tag{5}$$

Tables 8–10 represent the confusion matrix, and Table 11 summarizes the performance measures resulting from the classification of the test data. Using the newly added features, namely, the time-series data obtained through CFD simulation and the surface curvature data corresponding to a cerebral aneurysm, the sensitivity of the considered models was greatly improved compared to our previous study in which the achieved sensitivity was equal to 0.64 [24]. However, the specificity values were rather low for XGBoost (0.34) and SVM (0.46). In general, high sensitivity is often considered important in constructing a model to classify the presence/absence of disease. However, in predicting the rupture of cerebral aneurysms, a high number of false positives become problematic because of the risk of complications during the treatment of cerebral aneurysms that are expected to rupture. Therefore, in addition to sensitivity, specificity needs to be also high, and the balance between these two parameters is crucial. Specifically, one of the considered criteria while introducing the proposed classifier into a clinical site is that both sensitivity and specificity have to be 0.8 or higher. Here, the classification based on the LightGBM method was found to be appropriate as it provided the best balance of sensitivity and specificity compared with the XGBoost and SVM methods.

Table 8: Confusion matrix obtained by LightGBM

N=338		Actual class	
		Rupture	Unruptured
Predicted class	Rupture	27	53
	Unruptured	8	250

Table 9: Confusion matrix obtained by XGBoost

N=338		Actual class	
		Rupture	Unruptured
Predicted class	Rupture	27	201
	Unruptured	8	102

Table 10: Confusion matrix obtained by SVM

N=338		Actual class	
		Rupture	Unruptured
Predicted class	Rupture	25	165
	Unruptured	10	138

5 Conclusions

The main purpose of this research was to identify an effective classification model for predicting the rupture of cerebral aneurysms. To achieve this, we implemented two gradient boosting decision tree methods, LightGBM and XGBoost,

Table 11: Performance measures resulting from the classification

	LightGBM	XGBoost	SVM
Sensitivity	0.77	0.77	0.71
Specificity	0.83	0.34	0.46
F-measure	0.47	0.21	0.22

along with the conventional ML approach, SVM, and then constructed the classifiers incorporating the clinical, CFD, and geometric data. We applied the developed models to the prediction of cerebral aneurysm ruptures based on 338 cerebral aneurysm data samples (including 35 ruptured and 303 unruptured). We compared the performance of the considered models in terms of sensitivity and specificity and found that the LightGBM achieved the best performance in predicting the potential rupture of a cerebral aneurysm, reaching the sensitivity of 0.77 and the specificity of 0.83. The future related research will be focused on evaluating the contribution of each data source and feature into prediction performance, interpreting and understanding the importance of each feature from a medical perspective, and conducting the statistical evaluation of predictive performance through repeated runs of CV.

Acknowledgments

This paper is based on the results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO). CFD calculations were performed in collaboration with Siemens Healthcare based on a collaboration agreement with Jikei University.

References

[1] N.S. Altman, "An Introduction to Kernel and Nearest-neighbor Nonparametric Regression," *The American Statistician*, 46(3): 175–185, 1992.

[2] L. Amini, R. Azarpazhouh, M.T. Farzadfar, S.A. Mousavi, F. Jazaieri, F. Khorvash, R. Norouzi, and N. Toghianfar, "Prediction and Control of Stroke by Data Mining," *Int. J. Prev. Med.*, 4(Suppl 2): S245–249, 2013.

[3] A. Aranda and A. Valencia, "Computational Study on the Rupture Risk in Real Cerebral Aneurysms with Geometrical and Fluid-mechanical Parameters using FSI Simulations and Machine Learning Algorithms," *J. Mech. Med. Biol.*, 19(3): 1950014, 2019.

[4] P. Bentley, J. Ganesalingam, A.L.C. Jones, K. Mahady, S. Epton, P. Rinne, P. Sharma, O. Halse, A. Mehta, and D. Rueckert, "Prediction of Stroke Thrombolysis Outcome using CT Brain Machine Learning," *Neuroimage Clin.*, 30(4): 635–640, 2014.

[5] S. Chen and G.D. Doolen, "Lattice Boltzmann Method for Fluid Flows," *Annual Review of Fluid Mechanics*, 30(1): 329–364, 1998.

[6] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.785–794, 2016.

[7] B. Chung and J.R. Cebral, "CFD for Evaluation and Treatment Planning of Aneurysms: Review of Proposed Clinical Uses and Their Challenges," *Ann. Biomed. Eng.*, 43(1): 122–138, 2015.

[8] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: An Open-Source Mesh Processing Tool," *Proc. of the 6th Eurographics Italian Chapter Conference*, pp.129–136, 2008.

[9] C. Cortes and V.N. Vapnik, "Support-vector Networks," *Machine Learning*, 20(3): 273–297, 1995.

[10] T. Haruhara, H. Ohgi, M. Suzuki, H. Takao, T. Suzuki, S. Fujimura, T. Ishibashi, M. Yamamoto, Y. Murayama, and H. Ohwada, "Predicting Cerebral Aneurysm Rupture by Gradient Boosting Decision Tree using Clinical, Hemodynamic, and Morphological Information," *EPiC Series in Computing: Proc. of 35th International Conference on Computers and Their Applications*, 69: 180–186, 2020.

[11] G.A. Hitman, H. Colhoun, C. Newman, M. Szarek, D.J. Betteridge, P.N. Durrington, J. Fuller, S. Livingstone, H.A. Neil, and CARDS Investigators, "Stroke Prediction and Stroke Prevention with Atorvastatin in the Collaborative Atorvastatin Diabetes Study (CARDS)," *Diabet Med.*, 24(12): 1313–1321, 2007.

[12] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems*, 30: 3149-3157, 2017.

[13] H.C. Kim, J.K. Rhim, J.H. Ahn, J.J. Park, J.U. Moon, E.P. Hong, M.R. Kim, S.G. Kim, S.H. Lee, J.H. Jeong, S.W. Choi, and J.P. Jeon, "Machine Learning Application for Rupture Risk Assessment in Small-Sized Intracranial Aneurysm," *J. Clin. Med.*, 8(5): 683, 2019.

[14] B. Letham, C. Rudin, T.H. McCormick, and D. Madigan, "Stroke Prediction and Stroke Prevention with Atorvastatin in the Collaborative Atorvastatin Diabetes Study (CARDS)," *Annals of Applied Statistics*, 9(3): 1350–1371, 2015.

[15] T. Lumley, R.A. Kronmal, M. Cushman, T.A. Manolio, and S. Goldstein, "A Stroke Prediction Score in the Elderly: Validation and Web-based Application," *J. Clin. Epidemiol.*, 55(2): 129–136, 2002.

[16] T.A. Manolio, R.A. Kronmal, G.L. Burke, D.H. O'Leary, and T.R. Price, "Short-term Predictors of Incident Stroke in Older Adults," *Stroke*, 27(9): 1479–1486, 1996.

[17] Microsoft Corporation, "LightGBM Documentation," https://lightgbm.readthedocs.io, Accessed: July 21, 2020.

[18] T. Morino, T. Tanoue, S. Tateshima, F. Vinuela, and K. Tanishita, "Intra-aneurysmal Blood Flow Based on Patient-specific CT Angiogram," *Experiments in Fluids*, 49(2): 485–496, 2010.

[19] Y. Murayama, S. Fujimura, T. Suzuki, and H. Takao, "Computational Fluid Dynamics as a Risk Assessment Tool for Aneurysm Rupture," *Neurosurg. Focus*, 47(1): E12, 2019.

[20] R.C. Nogueira, E. Bor-Seng-Shu, N.P. Saeed, M.J. Teixeira, R.B. Panerai, and T.G. Robinson, "Meta-analysis of Vascular Imaging Features to Predict Outcome Following Intravenous rtPA for Acute Ischemic Stroke," *Frontiers in Neurology*, 7(77): 1–8, 2016.

[21] Y. Qian, H. Takao, M. Umezu, and Y. Murayama, "Risk Analysis of Unruptured Aneurysms using Computational Fluid Dynamics Technology: Preliminary Results," *AJNR Am. J. Neuroradiol.*, 32(10): 1948–1955, 2011.

[22] J.R. Quinlan, "C4.5: Programs for Machine Learning," *Morgan Kaufmann Publishers*, 1993.

[23] M. Shojima, M. Oshima, K. Takagi, R. Torii, M. Hayakawa, K. Katada, A. Morita, and T. Kirino, "Magnitude and Role of Wall Shear Stress on Cerebral Aneurysm: Computational Fluid Dynamic Study of 20 Middle Cerebral Artery Aneurysms," *Stroke*, 35(11): 2500–2505, 2004.

[24] M. Suzuki, T. Haruhara, H. Takao, T. Suzuki, S. Fujimura, T. Ishibashi, M. Yamamoto, Y. Murayama, and H. Ohwada, "Classification Model for Cerebral Aneurysm Rupture Prediction using Medical and Blood-flow-simulation Data," *Proc. of the 11th International Conference on Agents and Artificial Intelligence*, 2: 895–899, 2019.

[25] H. Takao, Y. Murayama, S. Otsuka, Y. Qian, A. Mohamed, S. Masuda, M. Yamamoto, and T. Abe, "Hemodynamic Differences Between Unruptured and Ruptured Intracranial Aneurysms During Observation," *Stroke*, 43(5): 1436–1439, 2012.

[26] T.J. Wang, J.M. Massaro, D. Levy, R.S. Vasan, P.A. Wolf, R.B. D'Agostino, M.G. Larson, W.B. Kannel, and E.J. Benjamin, "A Risk Score for Predicting Stroke or Death in Individuals with New-onset Atrial Fibrillation in the Community: The Framingham Heart Study," *JAMA*, 290(8): 1049–1056, 2003.

[27] xgboost developers, "XGBoost Documentation," https://xgboost.readthedocs.io, Accessed: July 21, 2020.

[28] J. Xiang, A.H. Siddiqui, and H. Meng, "The Effect of Inlet Waveforms on Computational Hemodynamics of Patient-specific Intracranial Aneurysms," *J. Biomech.*, 47(16): 3882–3890, 2014.

Toshiyuki Haruhara is a graduate student in the Department of Industrial Engineering, Graduate School of Science and Technology, Tokyo University of Science (Noda city). His research interests include predicting the rupture of cerebral aneurysms using CFD and medical data. He is responsible for developing AI in a joint team with the Department of Neurosurgery, Jikei University School of Medicine.

Masaaki Suzuki is a Junior Associate Professor in the Department of Industrial Administration, Faculty of Science and Engineering, Tokyo University of Science, Japan. Dr. Suzuki works in the area of computational engineering. Dr. Suzuki earned a Ph.D. at the University of Tokyo in the Department of Quantum Engineering and Systems Science.

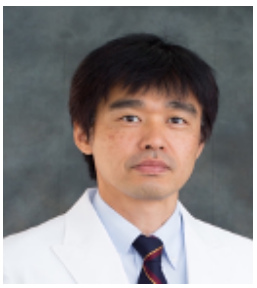
Hideto Ogi is in his first year for his master's in Industrial Administration, Faculty of Science and Engineering, Tokyo University of Science, Japan. He graduated from the Department of Industrial Administration, Faculty of Science and Engineering, Tokyo University of Science in Mar. 2020. His research focus is on the prediction of cerebral aneurysm rupture by ML using 3D shape data.

Hiroyuki Takao received his master's degree from the Department of Neurosurgery, Jikei University School of Medicine (JUSM), Minato-ku, Japan, 2003. He graduated from JUSM's neurosurgery doctoral course in 2010. He was a researcher (JUSM) from 2003 to 2007, an assistant professor (JUSM) from 2007 to 2012, an assistant researcher (University of California) from 2012 to 2013, and an associate professor (JUSM) for one month in 2013. He has been an associate professor at the Department of Neurosurgery and the Department of Innovation for Medical Informaion Techology JUSM since 2015. His recent research interests are developing IoT applications in ICT

medicine research and creating software for analyzing cerebral aneurysm simulations.

Takashi Suzuki graduated from the Department of Mechanical Engineering, Tokyo University of Science, Katsushika-ku, Japan, 2012. Then he received a Ph.D. of Engineering from the Graduate School of Engineering, Tokyo University of Science, 2017. He is a member of the collaborative research team between the Graduate School of Mechanical Engineering, Tokyo University of Science and the Department of Neurosurgery, Jikei University School of Medicine.

Soichiro Fujimura is a Research Fellowship for Young Scientists of Japan Society for the Promotion of Science. He graduated the Department of Mechanical Engineering, Tokyo University of Science, Katsushika-ku, Japan in Mar. 2015 with the highest honor. He also completed a doctoral course of engineering from the Tokyo University of Science graduating in Mar. 2020. In 2014, he joined the research collaboration project with the Department of Neurosurgery, the Tokyo Jikei University School of Medicine, Minato-ku, Japan, and has been working as a visiting researcher. His research focus is the analysis of cerebral-blood-flow dynamics using CFD (Computational Fluid Dynamics) and FEM (Finite Element Method)-based structural analysis.



Toshihiro Ishibashi graduated from the Jikei University School of Medicine (JUSM), Minato-ku, Japan, 1994. Then he finished residency program of JUSM as a medical doctor. He had a research fellowship in Medical Engineering, JUSM from 1999 to 2001. He was a trainee of Department of Neurosurgery, National cerebral and cardiovascular centre from 2001 to 2003. He is an associate professor at the Department of Neurosurgery JUSM since 2013. His field is Endovascular neurosurgery. Board certification; Diplomate, Japanese Board of Neurosurgery. Diplomate, Japanese Board of Society of Neuroendovascular Therapy.

Makoto Yamamoto has been a professor in the Department of Mechanical Engineering, Faculty of Engineering, Tokyo University of Science, Tokyo, Japan since 2004. He received the Bachelor, Master and Doctor degrees from the University of Tokyo, Tokyo, Japan, in 1983, 1985 and 1989, respectively. After finishing his doctor course, he joined Ishikawajima-Harima Heavy Industry Co. Ltd. and engaged in the research and development of a jet engine compressor. In 1991, he moved to the current university as a lecturer. His research field is Computational Fluid Dynamics (CFD) for industrial and medical problems.

Yuichi Murayama is a professor and chairman of Department of Neurosurgery at Jikei University School of Medicine, Tokyo, Japan. He was also a Professor of the Division of Interventional Neuroradiology, at UCLA School of Medicine. His academic interest is Stroke related innovation, including endovascular devices, such as Matrix coils or Onyx liquid embolic system, IT stroke network system, and Hybrid OR system. Currently he serves as a World Federation of Neurosurgical societies (WFNS) advisory panel of cerebrovascular diseases and therapy committee. He trains hybrid Neurosurgeons who can perform both open vascular technique and endovascular treatment.

Hayato Ohwada graduated from the Department of Industrial Administration, Faculty of Science and Technology, Tokyo University of Science, Noda City, Japan, in 1983. He also completed a Doctoral course from the Tokyo University of Science Graduate School, Division of Science and Engineering Industrial Administration, in Noda City, Japan, graduating in 1988. He was a research associate (Tokyo University of Science) from 1988 to 1998, a lecturer (Tokyo University of Science) from 1999 to 2000, and an associate professor (Tokyo University of Science) from 2001 to 2004. He has been a professor in the Department of Industrial Administration, Faculty of Science and Engineering, Tokyo University of Science since 2005. His research interests are in the fields of inductive logic programming and bioinformatics.

Journal Submission

The International Journal of Computers and Their Applications is published four times a year with the purpose of providing a forum for state-of-the-art developments and research in the theory and design of computers, as well as current innovative activities in the applications of computers. In contrast to other journals, this journal focuses on emerging computer technologies with emphasis on the applicability to real world problems. Current areas of particular interest include, but are not limited to: architecture, networks, intelligent systems, parallel and distributed computing, software and information engineering, and computer applications (e.g., engineering, medicine, business, education, etc.). All papers are subject to peer review before selection.

A. Procedure for Submission of a Technical Paper for Consideration

1. Email your manuscript to the Editor-in-Chief, Dr. Ziping Liu at: zliu@semo.edu.
2. Illustrations should be high quality (originals unnecessary).
3. Enclose a separate page (or include in the email message) the preferred author and address for correspondence. Also, please include email, telephone, and fax information should further contact be needed.
4. **Note**: Papers shorter than 10 pages long will be returned.

B. Manuscript Style:

1. **Word document**: The text should be **double-spaced** (12 point), **single column** and **single-sided** on 8.5 X 11 inch page.
   LaTex Document: Use IJCA template, which is double column and put in pdf format on 8.5 X 11 inch page.
2. An informative abstract of 100-250 words should be provided.
3. At least 5 keywords following the abstract describing the paper topics.
4. References (alphabetized by first author) should appear at the end of the paper, as follows: author(s), first initials followed by last name, title in quotation marks, periodical, volume, inclusive page numbers, month and year.
5. The figures are to be integrated in the text after referenced in the text.

C. Submission of Accepted Manuscripts

1. The final complete paper (with abstract, figures, tables, and keywords) satisfying Section B above in **MS Word format** should be submitted to the Editor-in-Chief. If one wished to use LaTex, please see the corresponding LaTex template.
2. The submission can be through an email attachment(s). **The following electronic files should be included:**
   - Paper text (required).
   - Bios (required for each author).
   - Author Photos (jpeg files are required) or photos can be integrated into the text.
   - Figures, Tables, and Illustrations. These should be integrated into the paper text file.
3. Reminder: The authors photos and short bios should be integrated into the text at the end of the paper. All figures, tables, and illustrations should be integrated into the text.
4. The final paper should be submitted in (a) a word format or (b) a pdf LaTex format. For those authors using LaTex, please follow the guidelines and template.
5. Authors are asked to sign an ISCA copyright form (http://www.isca-hq.org/j-copyright.htm), indicating that they are transferring the copyright to ISCA or declaring the work to be government-sponsored work in the public domain. Also, letters of permission for inclusion of non-original materials are required.

Publication Charges

After a manuscript has been accepted for publication, the contact author will be invoiced a publication charge of **$500.00 USD** to cover part of the cost of publication. For ISCA members, publication charges are **$400.00 USD** publication charges are required.