MULTI-DIMENSIONAL REGULAR EXPRESSIONS FOR OBJECT DETECTION *

TODD C. TORGERSEN †, V. PAÚL PAUCA ‡, ROBERT J. PLEMMONS §, DEJAN NIKIC ¶, JASON WU ∥, AND ROBERT RAND **

Abstract. We present a novel extension of traditional finite-automata-based methods to find multi-dimensional objects in images. Our approach extends regular expressions and deterministic finite automata (DFA) to create multi-dimensional pattern models. We demonstrate the effectiveness and efficiency of our approach for finding target objects in 3D LiDAR image data sets, using an implicit geometry (IG) representation of the data. The work described here relies on geometric information alone, and in its current form is sensitive to scale and orientation. Some variation in scale can be accommodated directly in the multi-dimensional regular expressions. Further, we feel strongly that existing techniques from the computer vision literature can be adapted to yield efficient algorithms for object recognition which are robust with respect to variations in scale and orientation. Further, the techniques presented here can also be extended to integrate non-geometric information, such as material and spectral characteristics from hyperspectral image data.

Key words. Geometric object detection, automata, regular expressions, LiDAR image data.

AMS subject classifications. 85, 68, 65, 51, 48.

1. Introduction. Simply stated, the problem examined in this paper is to efficiently detect objects of interest in 3D image data sets. Specifically, we are motivated to detect target objects in 3-D LiDAR data sets [3, 5]. Our approach is based on an implicit geometry [12] which is imposed on the raw LiDAR data (i.e. cloud of irregularly spaced points) that infers an underlying equally spaced point structure. Thus, we assume here the raw data has been pre-processed into a 3-dimensional data volume of (equal sized) voxels, where each voxel value comes from a finite set \{0, 1, 2, ..., s-1\}.

We introduce a novel approach for target detection in 3-D sets based on multi-dimensional regular expressions and the corresponding deterministic and nondeterministic finite automata [8]. Regular expressions are well-known models that are traditionally used for pattern matching in strings. Several generative models that extend regular expressions to 2-D have been proposed in the literature over the last few decades, such as EOL-regular matrix languages [13], tiling systems [6], tiling rewriting grammars [14], and pure 2D picture grammars [16]. Some of the implications in the extension to 2-D relative to the classical properties of 1D regular versus context-free languages have also been studied, see e.g. [2]. Unlike these generative models, our approach is focused on the detection of 2-D or 3-D objects, whose geometry has been specified using regular expressions, in a given 3-D data set. The grammar models formalism for object detection in computer vision [7] is a related approach that uses the concept of a bag grammar for object representation.

---

*This work was supported in part by the National Geo-Spatial Intelligence Agency.
†Department of Computer Science, Wake Forest University, Winston-Salem, NC 27109 (torgenser@wfu.edu).
‡Department of Computer Science, Wake Forest University, Winston-Salem, NC 27109 (paucavp@wfu.edu).
§Departments of Computer Science and Mathematics, Wake Forest University, Winston-Salem, NC 27109 (plemmons@wfu.edu).
¶The Boeing Company, Seattle WA. (Dejan.Nikic@boeing.com).
∥The Boeing Company, Seattle WA. (Jason.Wu@boeing.com).
**National Geo-Spatial Intelligence Agency, Springfield, VA 22150 (Robert.S.Rand@nga.mil).
While context free and bag grammars provide a powerful model for object generation and detection, grammars present a number of difficulties including choice of parsing algorithm, NP-hardness of parsing for bag grammars [4], ambiguity, and Turing undecidability of grammar equivalence [1, 4, 15]. Regular expressions have several advantages over grammars including:

- Ambiguity is not an issue for regular expressions.
- Choice of parsing algorithm is not an issue for regular expressions. Object detection is accomplished efficiently by simulating a deterministic finite automata (DFA) on an input data set.
- Practical questions such as the equivalence of two regular expressions are decidable in polynomial time.

In addition to the provable properties above, a review of many practical examples suggests that the full expressive power of context free grammars (and their corresponding pushdown automata) is not needed for objects of practical interest. The fundamental difference between DFA and pushdown automata (PDA) is the inclusion of an unbounded stack memory in the PDA model. Target objects are always contained in a relatively small (finite) support region, and therefore a model including unbounded stack memory is not necessary.

We assume the reader is familiar with traditional regular expressions, nondeterministic finite automata (NFA), and DFA in the context of 1-D strings of finite symbols. The unfamiliar reader is referred to [8, 15] for a review of these traditional concepts. In this paper, we treat voxel values as “symbols” and extend traditional regular expressions, NFA, and DFA to higher dimensions.

2. Definitions and Notation. In this section, we introduce the definition of regular expressions and its formal extension to higher dimensions. Notation for specifying regular expressions in higher dimensions is also established.

2.1. 1-D Regular Expressions and Regular Languages. Let \( \Sigma \) be a finite alphabet. We define a language to be a set of finite strings over the alphabet \( \Sigma \).

**Concatenation.** Let \( L_1 \) and \( L_2 \) be languages over \( \Sigma \). We define the concatenation of two languages \( L_1 \cdot L_2 \) as: \( L_1 \cdot L_2 = \{ a \cdot b \mid a \in L_1 \text{ and } b \in L_2 \} \), where \( a \cdot b \) denotes the concatenation of two strings \( a \) and \( b \).

**Kleene Closure.** Let \( L \) be a language over alphabet \( \Sigma \) and \( L^n \) be defined recursively by: \( L^0 = \{ \epsilon \} \), \( L^n = L \cdot L^{n-1} \). We define the Kleene closure of \( L \) denoted by \( L^* \) as: \( L^* = \bigcup_{i=0}^{\infty} L^i \).

Regular expressions over alphabet \( \Sigma \) along with the language they represent are defined as follows:

1. The symbol \( \phi \) is a regular expression denoting the empty language (the empty set).
2. The symbol \( \epsilon \) is a regular expression denoting the language containing the empty string.
3. The symbol \( a \) where \( a \in \Sigma \) is a regular expression denoting the language \( \{ a \} \).

If \( R \) is a regular expression, we use the notation \( L(R) \) to denote the language associated with \( R \).

4. If \( R \) and \( S \) are regular expressions, then \( (R \mid S) \) is a regular expression, denoting the language \( L(R) \cup L(S) \).
5. If \( R \) and \( S \) are regular expressions, then \( (R \cdot S) \) is a regular expression, denoting the language \( L(R) \cdot L(S) \).

\[ ^1 \text{While each string in a regular language is a finite string, a regular language itself may be infinite.} \]
6. If \( R \) is a regular expression, then \((R^*)\) is a regular expression denoting the language \( (L(R))^* \).

In practice, we avoid excessive use of parentheses by assigning a precedence ordering to the operations of union, concatenation and Kleene closure (in that order from lowest to highest), similar to the way in which addition, multiplication and exponents are used in traditional elementary algebra. Additionally, we adopt the notation \( a^n \) to denote \( n \) repetitions of the symbol \( a \). Informally, we will also use the juxtaposition of two symbols \( ab \) to indicate \( a \cdot b \).

**Range notation:** For the purposes of object detection it is useful to have a shorthand notation for expressing a pattern which occurs at least \( m \) times, but no more than \( n \) times. Let \( a \) be a symbol. We propose the notation:

\[
R = a^{m:n}
\]

to denote such a range. For example, \( a^{3:5} \) denotes a string of contiguous \( a \) symbols which is either length 3, or 4, or 5. Notice that this example is still regular, since we can express it as

\[
R = aaa \mid aaaa \mid aaaaa
\]

The symbol \( a \) could be replaced by any regular expression to allow patterns that are repeated a limited number of times.

**2.2. Extending Regular Expressions to Higher Dimensions.** We extend the definition of 1-D regular expressions using a naturally recursive definition. Let \( R_n \) denote an \( n \)-dimensional regular expression. We formally define \( R_n \) recursively in terms of a \((n - 1)\)-dimensional regular expressions as follows:

1. The expression \( R_1 \) is an 1-dimensional regular expression equivalent to a traditional regular expression, as defined in Section 2.1.
2. The expression \( R_n = [R_{n-1}] \) is an \( n \)-dimensional regular expression of length one (along dimension \( n \)). The square brackets are part of the notation to enclose a sub-expression of one lower dimension. The language \( L(R_n) \) is the set of all \( n \)-dimensional objects which consist of a single \((n - 1)\)-dimensional object matching the expression \( R_{n-1} \), and where that \((n - 1)\)-dimensional object is embedded in \( n \)-dimensional space.
3. If \( R_n \) and \( S_n \) are a \( n \)-dimensional regular expressions, then \((R_n \mid S_n)\) is a regular expression, denoting the language \( L(R_n) \cup L(S_n) \).
4. If \( R_n \) and \( S_n \) are a regular expressions, then \((R_n \cdot S_n)\) is a regular expression, denoting the language \( L(R_n) \cdot L(S_n) \).
5. If \( R_n \) is a regular expression, then \((R_n)^*\) is a regular expression denoting the language \((L(R_n))^*\).

We refer to the expressions defined above as the class of Multi-Dimensional Regular Expressions (MDRE).

**2.3. A 2-D Example.** We consider the objects illustrated in Figure 1. We consider only two symbols in this example: 0 for an empty pixel, and 1 for an occupied pixel. The diagram on the left models a tower or pole that consists of a base of five occupied pixels (horizontally) and six occupied pixels (vertically). The object on the right models a structured object.

We impose an ordering on the dimensions: the horizontal direction is designated the top-level dimension, and the vertical direction is designated as the low-level dimension. The origin is the lower left corner of the image. Our regular expression for
the pole object is then:

\begin{equation}
R_{(2)} = [1][1][1 
\end{equation}

Notice that $R_{(2)}$ is a concatenation of 2-D regular expressions along the horizontal direction, each of which specifies 1-D regular expressions within [ ] along the vertical direction.

The regular expression (2.1) does not specify that the area surrounding the pole is not occupied. In this example, the regular expression (2.1) would also match the structured object illustrated in Figure 1 (right). If we wish to be more specific regarding the surrounding area, we must build that specification into the regular expression. For example:

\begin{equation}
R_{(2)} = [1 \cdot 0][1 \cdot 0^5][1^6][1 \cdot 0^5][1 \cdot 0]
\end{equation}

Regular expression (2.2) will match the pole object in Figure 1 (left), but will not match the structured object in Figure 1 (right).

We can accommodate some variation in the object using our notation regarding ranges. Suppose the base is either one or two pixels on either side of the center tower, and suppose the tower height is somewhere between four and six pixels high. Our regular expression is then:

\begin{equation}
R_{(2)} = [1]^{1:2} \cdot [1^{4:6}] \cdot [1]^{1:2}
\end{equation}

Intuitively, the concept of “symbol” in an $n$-dimensional regular expression is replaced by an $(n-1)$-dimensional regular expression. To match an $n$-dimensional regular expression with an $n$-dimensional input data set, we impose an ordering $d_1, d_2, \ldots d_n$ on the dimensions. As the input data is scanned in the top-level dimension, we encounter a sequence of starting points where an $(n-1)$-dimensional expression may be matched. Next, we discuss this $n$-dimensional regular expression matching approach in detail.

**3. Expression Matching for $n$-Dimensional Objects.** In two dimensions, a recursive algorithm scans along the top level dimension (e.g., horizontal). As each regular 1-dimensional expression is encountered, it is processed (vertically) using 1-dimensional techniques. This recursive approach can be generalized to an arbitrary number of dimensions. Each data dimension corresponds to a recursive level in the
computation performed by an \( n \)-dimensional DFA. If the pattern extent implied by the regular expression is bounded by a constant, then the time complexity of a target search is bounded by a linear function of the number of voxels contained in the data set.

The first step in processing an \( n \)-dimensional regular expression is to construct an \( n \)-dimensional deterministic finite automata (DFA) equivalent to the given \( n \)-dimensional regular expression. We use the notation \( M^{(n)} \) to denote an \( n \)-dimensional DFA. We use the notation \( L\left(M^{(n)}\right) \) to denote the language recognized by \( M^{(n)} \). We now formally define an \( n \)-dimensional DFA as follows:

**Base Case:** A 1-dimensional finite automata is a traditional DFA.  

**General Case:** An \( n \)-dimensional deterministic finite automata is a 5-tuple:

\[
M^{(n)} = \left( Q^{(n)}, \Sigma^{(n)}, \delta^{(n)}, q_0^{(n)}, F^{(n)} \right)
\]

where
- \( Q^{(n)} \) is a finite set of states.
- \( \Sigma^{(n)} \) is a finite set of \((n-1)\)-dimensional deterministic finite automata.
- \( \delta \) is a function: \( \delta : Q^{(n)} \times L\left(M^{(n-1)}\right) \rightarrow Q^{(n)} \) where \( M^{(n-1)} \) is an \((n-1)\)-dimensional DFA.
- \( q_0^{(n)} \subseteq Q^{(n)} \) is a starting state.
- \( F^{(n)} \subseteq Q^{(n)} \) is a set of accepting states.

Fortunately, well-known 1-D construction techniques extend naturally to the higher dimensional cases. Given an \( n \)-dimensional regular expression, the construction of an equivalent \( n \)-dimensional DFA proceeds by a dimensionally recursive application of standard algorithms for converting regular expressions to equivalent DFA. A regular expression is first converted to a nondeterministic finite automata (NFA) using a construction based on the recursive definition of regular expressions. The concept of \( \epsilon \)-closure \([8]\) is used to construct a DFA which is equivalent to the intermediate NFA form. The states in the newly constructed DFA consist of sets of states from the NFA. Further details regarding 1-D construction techniques can be found in \([1]\).

### 3.1. Implementation.

The usual implementation of 1-dimensional DFA-based pattern matching code represents a 1-dimensional DFA as a table indexed by states (i.e., the row index) and by alphabet symbols (i.e., the column index). For an \( n \)-dimensional DFA, we take a similar table-based approach: the column index refers to one of the (finitely many) \((n-1)\)-dimensional DFA which define the \( n \)-dimensional DFA. For example, in the 2-D case, the traditional operation of matching a symbol (in the 1-D case) is replaced by simulating a 1-D DFA to decide a match in the appropriate direction. The code for the \( n \)-dimensional DFA was implemented using the C++ language.

**\( n \)-dimensional regular expression parser.** To make our system practical, we also implemented a parser capable of reading \( n \)-dimensional regular expressions, such as those shown in (2.1), (2.2) and (2.3), in a linearized ASCII form. For example equation (2.2) may be written as

\[
[1 \ 0] [1 \ 0^5] [1^6] [1 \ 0^5] [1 \ 0]
\]

Here blank spaces denote concatenation, circumflex (\(^\)\)) denotes repetition. In addition, vertical bar (\(|\)) denotes set union. We use the acronym \textsc{mdre} to refer to our computer implementation of the ideas presented in this paper.

\(2\)The reader is referred to \([15]\) for further details.
3.2. Target Search in a 2-D Example. We illustrate 2-D expression matching on a 256 $\times$ 256 simulated scene shown in Figure 2 (left). Let the dimensions be ordered as: (vertical, horizontal). This ordering corresponds conveniently to row-column indexing where the pixels in the scene are arranged in a matrix. If we use the symbols 1 for an occupied pixel, and 0 for an un-occupied pixel, the inverted “T” target objects seen in Figure 2 can be specified by the regular expression:

$$R = [1\,1\,0]^4 [1^17\,0]^3 [1\,1\,0]^4$$

Alternately, the ordering on the dimensions can be chosen to be (horizontal, vertical). In this case, a regular expression appropriate to the target object is:

$$Ra = [1^4\,1^2\,0^4\,1^3\,0^4\,1^4\,0^4\,1^4]$$

MDRE successfully finds the seven occurrences of the object specified by the regular expressions above (using either dimension ordering), and records a bounding box for each instance. The result of finding the target object is illustrated in Figure 2 (right).

Next, we apply MDRE and 3D regular expressions to search for objects of specific geometry in LiDAR data sets.

4. 3D Regular Expressions for Implicit Geometry Data Sets. We consider a sample (real) implicit geometry data set and give a 3D regular expression for finding light poles in the observed region. This dataset is a tile from a larger collection of measurements of Ottawa, Canada obtained using ground and air-based LiDAR equipment at 15cm resolution [11]. The regular expression given in (4.1) is best understood by visualizing a stack of squares measuring 6 $\times$ 6 voxels at the base and extending 9 voxels vertically. Within the 6 $\times$ 6 square, in the central 4 $\times$ 4 region at least one occupied voxel is required. Alternates (set unions denoted by “|”) within the regular expression are used to accommodate sampling issues, including aliasing, noise, and variations (e.g., attachments) in the shape of the poles. Additionally, we require 1 voxel of unoccupied space surrounding the central 4 $\times$ 4 region. The complete expression is given in (4.1):
The current implementation converts the regular expression into an equivalent 3D DFA and performs lexical analysis (pattern finding) by simulating the 3D DFA on an IG data set. Figure 3 (left) illustrates the light poles identified by this process.

![Light poles and trees recognized in an IG data set](image)

We now turn our attention to a more difficult target: trees. The issue here is that there is no obvious or easily defined pattern that we can definitively declare to be the unambiguous characterization of a tree. On examining a few LiDAR/IG data sets, it is clear that the lower portion of tree trunks are not well represented in the 3D data sets, apparently because the trunk is mostly occluded by the upper branches and leaves. A 3D regular expression for recognizing trees is given in (4.2). The basic idea is to look for a region in which:

- most (but not all) voxels in the region are occupied, and
- the region is an appropriate distance from the ground.

The structure of the expression in (4.2) is similar to the structure of (4.1): the expression is organized as a stack of $9 \times 9$ squares. The initial portion of (4.2), containing $[[011]]^6$, acts to lift the stack of $9 \times 9$ squares 6 voxel units above the ground level. The remaining portion of (4.2) uses alternates (set unions) and concatenation to specify three consecutive regions (in the vertical direction):

1. a $5 \times 5$ central region in which at least $3/5$ of the voxels are occupied,
2. a $3 \times 3$ central region in which at least $1/3$ of the voxels are occupied, and
3. a $9 \times 9$ unoccupied region.
A number of patterns could be defined which could use alternate characterizations of the appearance of trees in IG data sets.

\[
R_{\text{tree}} = \left[ \left[ \left( 0 | 1 \right)^{\ast} \right] \right]^{6}
\]

A sample IG scene together with the identified trees is illustrated in Figure 3 (right). We observe that not all trees are found in this example. The 3D regular expression can be made more inclusive (i.e., allow more alternates) in an effort to recognize more trees. However, patterns that become too inclusive tend to falsely identify other scene objects as trees. Preliminary investigation suggests that our 3D regular expressions can be modified to incorporate probabilistic measures to better allow for natural variations in the target objects and to avoid classification errors due to noise. Probabilistic measures also hold the possibility of allowing us to simplify the regular expressions: variations in the target object will be built in to the probabilistic framework rather than a large number of alternates within the regular expression itself.

5. Hamming distance for DFAs. A regular expression may fail to match an input data set due to errors (noise) in the LiDAR data set. We observe that for the implicit geometry data sets studied here, insertion errors and deletion errors do not occur because of the method by which an implicit geometry representation is computed from a raw point-cloud data set. For this reason, we use the Hamming distance measure instead of the Levenshtein distance [9, 10]. I.e., we consider only substitution errors. Mismatches between a multidimensional DFA and the input data are detected only during 1D DFA matching. For this reason, it is sufficient to define Hamming distance between a DFA and a string in the 1D case.

To further simplify our analysis, we consider regular expressions which do not include the Kleene closure operation; the resulting DFA graphs contain no cycles.

\[3\text{In practice the Kleene operation is rarely helpful to express a pattern, since objects of interest are never infinitely extensible.}\]
Let $x$ denote a 1D input string, and let
\[ S(M, x) = \{ w \mid w \in L(M) \text{ and } |w| = |x| \} \]
where $L(M)$ denotes the language accepted by DFA $M$. We define the Hamming distance between a string $x$ and a DFA $M$ to be:
\begin{equation}
  d(x, M) = \begin{cases}
    \min_{w \in S(M, x)} h(x, w) & \text{if } S(M, x) \neq \emptyset \\
    +\infty & \text{otherwise}
  \end{cases}
\end{equation}
where $h(x, w)$ is the Hamming distance between strings $x$ and $w$.

We can use equation (5.1) to accept a (noisy) string $x$ if and only if
\begin{equation}
  d(x, M) \leq \tau
\end{equation}
for some (nonnegative integer) threshold $\tau$ of tolerable error count. Computing $d(x, M)$ is not as trivial as one would hope. An algorithm to find the minimum number of substitutions needed for acceptance can not proceed by performing a substitution at the point where the first symbol in error is detected. It is possible that a fault in a previously scanned symbol, e.g., symbol $b$ in position $t$, caused the DFA to transition to an incorrect state $q$. Continuing from state $q$, several symbols may match correctly, followed by a large number of mis-matched symbols. However, correcting the single erroneous symbol in position $t$ could lead to a state $q'$ from which no further errors are encountered.

At first glance, it would appear that backtracking is necessary, leading to a computationally inefficient method. Fortunately, for a given tolerance $\tau$, we can construct a polynomial time algorithm for deciding equation (5.2) without resorting to backtracking.

### 5.1. Hamming Automata

Given a string of symbols $w$ and a positive integer $\tau < |w|$, the corresponding Levenshtein automata accepts any string $y$ which can be formed from $w$ using $\tau$ or fewer insertion, deletion and substitution operations (i.e., within Levenshtein distance $\tau$). A simple modification of the Levenshtein idea can be used to limit editing operations to substitution errors only, resulting in an automata which will accept any string within a chosen Hamming distance. We propose the term **Hamming Automata** to refer to a finite automata which accepts all strings $y$ which are within Hamming distance $\tau$ of string $w$. A Levenshtein automata (left) and a Hamming automata (right) are shown in Figure 4 for the string $w =$ “Lucky” with $\tau = 2$. A transition labeled by a question mark (?) denotes a transition on any symbol, except those symbols explicitly shown. Observe that while the Levenshtein automata is an NFA with $\epsilon$-transitions, the Hamming automata is a simple DFA.

A polynomial time algorithm to decide equation (5.2) is given below.

**Input:** A DFA $M$, a string $x$ and a positive integer $\tau$.

**Output:** “Accept” if $d(x, M) \leq \tau$, “reject” otherwise.

**Method:**
1. Construct a Hamming automata $H$ for input string $x$ and tolerance $\tau$. Note: This construction can be done in time $O(|x|\tau)$.
2. If $L(M) \cap L(H)$ is non-empty, then accept $x$, otherwise reject.
The implementation of step 2 above is straightforward. Given two DFA $M_1$ and $M_2$ with $n_1$ and $n_2$ states respectively, a DFA $\hat{M}$ recognizing $L(M_1) \cap L(M_2)$ can be constructed in time $O(n_1 n_2)$ using a well known algorithm; see [15]. Depth first search of a DFA (treated as a directed graph) may be used to decide if there exists a path from the start state to any accepting state.

6. Conclusions. We have presented a novel and practical approach for efficient representation and detection of geometric objects in $n$ dimensional space by means of $n$-dimensional regular expressions and corresponding finite automata. A fundamental advantage of regular expressions over context free or bag grammars is that every practical question regarding regular languages, e.g. equivalence of two different regular expressions, can be computed in polynomial time. In contrast, many practical questions regarding context free and bag grammars, e.g. equivalence of two different context free or bag grammars, is Turing undecidable [15].

Higher order regular and context free languages have been studied in the literature, specially within the theoretical computer science field, see e.g. [2, 13, 16] and references therein. Our notation is unique however, in that it allows one to represent complex classes of $n$-dimensional object geometries in compact form. Equations (4.1) and (4.2) illustrate this point. A large number of strings can match these regular expressions in a given dataset as demonstrated in Figures 3 and 3. To the best of our knowledge, the work presented here is the first of its type to demonstrate the practical application of regular expressions for object detection in $n$-dimensional spaces.

7. Future Work. We outline a few directions in which the work presented here may be extended for object detection and retrieval purposes.

1. Fused image datasets, such as co-registered LiDAR and HSI datasets [5] may be processed by extending the alphabet of the $n$-dimensional DFA to also include a larger but finite set of features representative of the fused dataset.

2. Scale invariance may be introduced into the object detection algorithm by applying $n$-dimensional DFAs at various spatial scales and selection scale automatically by minimizing the Hamming distance. Similarly, rotation invariance may be introduced in several ways. For example, one may first detect the principal direction using a histogram of gradients approach for a region of interest and then rotating the coordinate space appropriately to match the regular expression.

3. Hamming automata may be extended to provide a distance metric between two $n$-dimensional DFAs (or regular expressions). Discrete optimization techniques, often used in machine learning approaches, may utilize such distance criteria for learning $n$-dimensional regular expressions from training data.
REFERENCES