ABSTRACT

Given a region $S$ comprised of locations that each have a time series of length $|T|$, the Persistent Change Windows (PCW) discovery problem aims to find all spatial window and temporal interval pairs $(S_i, T_i)$ that exhibit persistent change of attribute values over time. PCW discovery is important for critical societal applications such as detecting desertification, deforestation, and monitoring urban sprawl. The PCW discovery problem is challenging due to the large number of candidate patterns, the lack of monotonicity where sub-regions of a PCW may not show persistent change, the lack of predefined window sizes for the ST windows, and large datasets of detailed resolution and high volume, i.e., spatial big data. Previous approaches in ST change footprint discovery have focused on local spatial footprints for persistent change discovery and may not guarantee completeness. In contrast, we propose a space-time window enumeration and pruning (SWEP) approach that considers zonal spatial footprints when finding persistent change patterns. We provide theoretical analysis of SWEP’s correctness, completeness, and space-time complexity. We also present a case study on vegetation data that demonstrates the usefulness of the proposed approach. Experimental evaluation on synthetic data show that the SWEP approach is orders of magnitude faster than the naive approach.

Categories and Subject Descriptors

H.2.8 [Database Applications]: Spatial databases and GIS

General Terms

Algorithms, Performance, Experimentation

Keywords

Spatiotemporal data mining, pattern discovery, enumeration and pruning

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

BIG SPATIAL ’13, November 05 - 08 2013, Orlando, FL, USA
Copyright 2013 ACM 978-1-4503-2534-9/13/11 ...$15.00
http://dx.doi.org/10.1145/2534921.2534928.

1. INTRODUCTION

Given a region $S$ comprised of locations that each have a time series of length $|T|$, and a change rate threshold, the problem of Persistent Change Window (PCW) discovery aims to find all (window, interval) pairs $(S_i, T_i)$ that exhibit persistent change over time. For example, Figure 2 shows a sample spatiotemporal data field with 16 locations. Each location is associated with a series of 4 values indicating the vegetation cover at each time step. Given a minimum average change rate across time steps, the PCW discovery may find a rectangular window (e.g., the nine locations to the left-top corner) and a time interval (e.g., time steps 1 through 4) where a persistent degradation of the land cover occurs.

PCW discovery is important to a number of societal applications. Ecologists, for example, may be interested in identifying regions where the landscape progresses through different stages and continuously transforms in appearance. Urban planners and policy makers may be interested in finding regions where the farmland grows rapidly to assess urban sprawl and food production. Climate scientists may be interested in finding regions where a persistent decrease in precipitation occurs to assess the severity of droughts. The explosion of planetary environmental data in recent years created new opportunities for answering the above questions. For example, Google’s Earth Engine is comprised of trillions of scientific measurements dating back almost 40 years [1]. Figure 1 shows an example in Brazil from 1984 to 2012 where the effect of deforestation of the Amazon becomes more pronounced over time. Identifying geographic areas that may be exhibiting persistent change patterns, however, is a labor intensive task for domain experts dependent on visual analysis of the data. Efficient computational approaches to help in early identification of such regions may facilitate countermeasures or prevention techniques such as reforestation, where depleted forests and woodlands may be restocked.

Increasingly, however, the size, variety, update rate, and combinatorics (e.g., the enumeration space of candidate patterns) of spatial datasets such as Google Earth Engine [1] exceed the capacity of commonly used spatial computing and database technologies to learn, manage, and process the data with reasonable effort. We believe that this data, which we call Spatial Big Data (SBD), represents the next frontier in several domains including Climate Science, Ecology, and Urban Planning. In addition to data from Google’s Earth Engine [1], examples of emerging SBD datasets include Unmanned Aerial Vehicle (UAV) data, LiDAR data, etc.
PCW discovery is challenging for the following reasons. First, there are a huge number of candidate patterns to consider when trying to determine the solution. For example, if the given spatial window $S$ is comprised of $M \times N$ locations, where each location has a time-series of length $T$, the number of candidates is $M^2 \times N^2 \times T^2$. For a moderate resolution remote sensing data tile (e.g., MODIS 250 NDVI, 4800 by 4800 pixels for 13 years), the total number of candidate may reach $10^{16}$. If we consider all the tiles in the dataset or finer resolution dataset (e.g., Landsat 30m resolution) at the global scale, the candidate space will exceed $10^{30}$, i.e., “big combinatorics”. Second, PCW lacks monotonicity; regions of interest may be comprised of sub-regions that are not interesting, making computational techniques such as apriori-based pruning and dynamic programming inapplicable. For example, in Figure 1, there are several regions without deforestation inside and around the regions with deforestation, which illustrates the lack of monotonicity. Third, the size of an ST window may vary, without a maximum length. For example, deforestation in Brazil has spanned over 230,000 square miles since 1970 [2]. Finally, the data volume is potentially large when considering attributes such as vegetation cover, temperature, precipitation, etc., over hundreds of years from different global climate models and sensor networks. The volume of such datasets will range from terabytes to petabytes.

Previous approaches on Spatiotemporal (ST) change footprint discovery have focused on discovering abrupt change point with local (e.g., time series at individual raster cells or pixels) or zonal footprints. Local-footprint based methods include time series change point detection [5,10–12] techniques, which aim at finding a point in a time series when a shift in statistical parameter occurs. For example, CUSUM [11,12] keeps a cumulative score of the log-likelihood ratio of distribution parameter (e.g., mean), and flags the change when the score exceeds a threshold. Zonal techniques such as spatiotemporal scan statistics [7,8] takes the aggregated attributes series (e.g., total number of disease count) in each spatial area, and finds an area and time point where a change in the underlying distribution is most likely to occurs (e.g., outbreak). These related techniques may find the most likely change pattern, but do not guarantee completeness of the results. In addition, they may not directly solve the problem but only provide assistance to manual efforts.

In addition to the above work, a large body of change detection techniques [6,9,13] and softwares [3,4] developed in remote sensing research have focused on finding pixel-wise changes across a few (typically two) snapshots of satellite images. These techniques, though having the ability to find changes across multiple snapshots, lack the ability to discover time intervals with arbitrary length and non-monotonic changes. In addition, these techniques output pixel-wise changes rather than zonal, collective summarization of change footprints, and may require intensive human labor for post-processing and visual analysis when dealing with big data.

Other work expands the temporal change footprint to intervals or periods of interest [15]. However, these work are still local-footprint based, and have not been extended to handle persistent change patterns with zonal spatial footprint. In contrast, in this paper we propose a completely automatic and complete computational approach to discover persistent change patterns with a zonal spatial footprints. Table 1 summarizes the classification of related work in this area.

Table 1: Classification of Related Work

<table>
<thead>
<tr>
<th>Spatiotemporal (ST) Change</th>
<th>Spatial footprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>Zonal</td>
</tr>
<tr>
<td>Temporal footprint</td>
<td></td>
</tr>
<tr>
<td>Across few snapshots</td>
<td>Remote sensing change detection [6,9,13]</td>
</tr>
<tr>
<td>Long Interval (persistent)</td>
<td>Interesting sub-path discovery [15]</td>
</tr>
</tbody>
</table>

Contributions: To address the above limitations of related work, we propose a space-time (ST) window enumeration and pruning (SWEP) approach that considers zonal spatial footprints when finding ST windows. It is completely automatic and guarantees the completeness of results. In summary, our contributions are as follows:

- We formally define the persistent change window (PCW) discovery problem.
- We propose an ST window enumeration and pruning approach (SWEP) as a computational solution to PCW discovery problem.
- We provide theoretical analysis of the correctness, completeness, and space/time complexity of the proposed method.
- Experiments on a synthetic datasets with various settings show that SWEP leads to orders of magnitude computational savings over the naive approach.
We present a case study using vegetation cover data to evaluate the effectiveness of SWEP.

Scope and Outline: This paper is focused on ST windows with persistent changes. Visual analytics and other semi-automatic or manual techniques for land cover change analysis are beyond the scope of this paper. The proposed approach is validated using case study of vegetation cover change. However, we do not discuss the details (e.g., causes or impacts) of desertification, deforestation, etc. The rest of this paper is organized as follows: Section 2 presents the basic concepts and problem statement of PCW. Section 3 outlines a naive approach to solving PCW and presents our proposed SWEP algorithm. In Section 4, we provide theoretical analysis of the correctness, completeness, and space/time complexity of SWEP. Section 5 presents a case study that shows the effectiveness of SWEP using vegetation cover data. Section 6 outlines the experimental evaluation. Section 7 concludes the paper and discusses future work.

2. BASIC CONCEPTS AND PROBLEM STATEMENT

In this section, we introduce several key concepts in the Windows of Persistent Change (PCW) discovery problem and give a formal problem statement.

2.1 Basic Concepts

Relevant definitions to our problem statement and proposed approaches are as follows:

Spatial Time-Series: A spatial field, $S$, is a partition of a region of geographic space, forming a finite tessellation of spatial objects or locations. An example of a spatial field is shown in Figure 2. As can be seen, the spatial field is comprised of 16 locations, each having different values at different time instants. A temporal framework, $T$, is a partition of a time interval into sub-intervals and a time-series is a computable function from $T$ to a finite attribute domain, $A$. A spatial time-series is $S \times T$, which may be thought of as a spatial field where each location $s_i \in S$ has a time-series. In Figure 2, the time-series for the spatial location in the upper left corner is $[10, 9, 6, 3]$, indicating different values at that location (e.g., vegetation cover) at time instants 1 through 4.

Spatial and spatiotemporal window: A spatial window $S_i$ in a spatial field $S$ with $M \times N$ locations is defined as $S_i = [x_{1i}, x_{2i}] \times [y_{1i}, y_{2i}]$, where $[x_{1i}, x_{2i}] \subseteq [1, M], [y_{1i}, y_{2i}] \subseteq [1, n]$, i.e., rectangular areas. Similarly, a spatiotemporal (ST) window is a subspace $S_i \times T_i$ where $S_i$ is a spatial window and $T_i$ is a time interval: $T_i = [t_1, t_n] \subseteq T$.

Spatial Aggregated Time-Series, $T_{S_i}$: Each location $s_j$ in a spatial field $S$ has a time-series $T_{s_j} = [x(s_j, 1), x(s_j, 2), ..., x(s_j, t)]$. A spatial aggregated time-series $T_{S_i}$ over spatial window $S_i$ is a series of aggregated value of the locations in $S_i$. For example, $T_{S_i}$ can be defined as the sum (or average, etc) of time series in each location, i.e., $T_{S_i} = \sum_{s_j \subseteq S_i} x(s_j, 1), \sum_{s_j \subseteq S_i} x(s_j, 2), ..., \sum_{s_j \subseteq S_i} x(s_j, t)$. The spatial aggregated time-series for all 16 locations in Figure 2 is [160, 146, 134, 121], where the value for each time instant $t_i$ represents the sum of all the values of all the locations in $t_i$. For example, the value for time instant 1 is 160 because all 16 locations in the spatial field have a value of 10.

Average Change Rate (ACR): For a spatiotemporal window $S_i \times T_i$ (where $T_i = [t_1, t_n]$), the average change rate is defined as the average percentage of change (i.e., decrease or increase) in the spatial aggregated time series $T_{S_i}$ over period $T_i$. Formally, it can be expressed as $ACR(S_i, T_i) = \frac{|T_{S_i}(t_1) - T_{S_i}(t_n)|}{T_{S_i}(t_1) / (n-1)}$, where $T_{S_i}(t_1)$ and $T_{S_i}(t_n)$ are the first and nth value of the spatial aggregated time series of window $S_i$. In Figure 2, the change (decrease) rate between time instants $t_1$ and $t_2$ for all 16 locations is 8.8%. The ACR in Figure 2 for all 16 locations and all 4 time instants is 8.13%, which represents the decrease rate between the first and last value, divided by the length: $(160-121)/160/3 = 8.13%$. An example of how the average change rate may be used is to determine the average rate at which total vegetation cover in an area (e.g., the Amazon) decreases over a certain period (e.g., the last 3 decades). Similarly, it can be used to determine the average increase rate.

Persistent Change Window (PCW): Given a threshold $r$ of minimum average change rate (ACR), a persistent change window is a spatiotemporal window $S_i \times T_i$ where $ACR(S_i, T_i) \geq r$.

2.2 Problem Statement

The problem of windows of persistent change (PCW) can be expressed as follows:

Given:
- A spatial field $S$ with $M \times N$ locations that has a time series of values of length $|T|$
- A threshold $r$ of average change rate (ACR)
- A minimum window size $S_{min}$ (optional)
- A minimum time length $t_{min}$ (optional)

Find:
- All persistent change windows $(S_i, T_i)$

Objective:
- Reduce computational cost

Constraints:
- $|S_i| \geq S_{min}$ and $|T_i| \geq T_{min}$
- $(S_i, T_i)$ is not a subset of any other pair $(S_j, T_j)$ such that $S_i \subseteq S_j \land T_i \subseteq T_j$
- Completeness and Correctness

The inputs for PCW discovery include a spatial field, an average change rate threshold, and minimum window and time length sizes as defined previously. The output is all window-interval pairs whose ACR exceeds the given threshold. The objective is to reduce computational cost. The problem has three constraints. First, window and time intervals require user-specified minimum sizes, which allows flexibility in ignoring small window-interval pairs which may not be of interest. A second constraint is that $(S_i, T_i)$ pairs must not be subset of any other window-interval pairs. This avoids duplication in that the information in a smaller $(S_j, T_j)$ pair that is a subset of $(S_i, T_i)$ is fully captured by the larger or dominant pair. The third constraint is completeness (i.e., all relevant $(S_i, T_i)$ pairs are discovered) and correctness (i.e., all discovered $(S_i, T_i)$ pairs are indeed persistent change windows as outlined in the problem statement).

Example: Figure 2 and Figure 3 show input and output examples of PCW. The input, shown in Figure 2, consists of a spatial field comprised of 16 locations, where each location has a time series of 4 values. The ACR threshold is set to
15%, and $S_{\min}$ and $T_{\min}$ are set to 9 and 4, respectively. The output, shown in Figure 3, is the highlighted window (9 locations) across all 4 time instants. The highlighted $(S_i, T_j)$ pair has an ACR of 16%, which satisfies the threshold. The ACR is based on the spatial aggregated time-series for the highlighted 9 locations in the spatial field, which is [90, 79, 66, 53]. The total change rate over this time period is ($90-53)/90=52%$. The ACR is the average of the decrease rate over 3 years, i.e., $52%/3 = 17.3\%$.

An important difference between the proposed problem and previous work in the vast domain of remote sensing [6, 9, 13] is that in PCW all region and time-interval pairs are considered. This is very different from calculating persistent changes on a pixel-by-pixel basis or calculating persistent changes for a pre-defined region. If the work of looking at every region and time-interval pair is not done, important windows across space and time may be missed. For example, in Figure 3, the ACR between time instants $t_1$ and $t_2$ for the pixel in the first row, second column is 10% whereas the ACR for the pixel in the first row, third column is 0% for the same period. Individually, only the former pixel might seem interesting. However, PCW allows us to analyze these pixels (and every other combination of pixels) as a group in space and time. This advantage provides early identification of regions of deforestation, urban sprawl, etc.

3. PROPOSED APPROACH

This section describes the naive algorithm and our proposed ST window enumeration and pruning (SWEP) approach for solving the ST windows of persistent change (PCW) discovery problem.

3.1 Naive Approach

We first present an intuitive and brute-force solution as the baseline. The pseudo code for the naive approach (Algorithm 1) consists of three main steps. Step 1 generates all window-time interval pairs, $(S_i, T_j)$ such that each spatial window is of a minimum size $S_{\min}$ and each time interval is of a minimum size $T_{\min}$. The aggregated time series for each window-time interval pair is also calculated at this point. In Step 2, all window-time interval pairs whose average change rate exceeds the given threshold $r$ are saved as potential candidates. Finally, Step 3 returns all candidates that are not subsets of any other candidate.

**Algorithm 1 Naive PCW Algorithm**

**Input:**
- A spatial field $S$ with $M \times N$ locations that has a time series of values of length $|T|$,
- A threshold $r$ of average change rate (ACR),
- A minimum region size $S_{\min}$,
- A minimum time length $T_{\min}$

**Output:**
All persistent change windows $(S_i, T_j)$ such that $|S_i| \geq S_{\min}$ and $|T_j| \geq T_{\min}$ and $(S_i, T_j)$ is not a subset of any other pair $(S_l, T_k)$ such that $S_l \subset S_j \land T_l \subset T_j$

**Algorithm:**
1. Enumerate all $(S_i, T_j)$ (window, time interval) pairs and generate the aggregated time series for each $(S_i, T_j) \in S \times T$ such that $S_i \subset S_j \land T_i \subset T_j$
2. **Candidates** ← all $(S_i, T_j)$ pairs whose ACR $\geq r$
3. **return** all $(S_i, T_j) \in \text{Candidates}$ that are not subsets of any other $(S_l, T_k) \in \text{Candidates}$

The main limitation of the naive approach is its high time complexity, where steps 1 and 2 require $O(M^3 N^3 T^3)$ and...
step 3 requires up to \(O(M^4N^4T^4)\). The reason for this is that all ST windows are enumerated, and each pair needs to be compared to eliminate dominated ones, which gets expensive quickly. We expound on the details of time complexity and other theoretical properties in the Theoretical analysis section. Next, we describe our proposed approach.

3.2 The ST Window Enumeration and Pruning Approach

In order to reduce computational cost, we propose a space-time window enumeration and pruning (SWEP) approach, which enumerates all the ST windows in such an order that no redundant evaluation is performed.

Fast computation of spatial aggregation: A primary optimization can be done to reduce the time for computing the interest measure. The spatial aggregation in each spatial window requires repeated scan of each location's value. We propose to precompute a lookup table so as to enable a constant-time lookup of the aggregate value for each spatial window and the ACR score in each pattern evaluation.

The SUM of any spatial window \([(x, y)]\) stores the SUM of spatial window \([(1 \leq x, y)]\) at time \(t\). The SUM of any spatial window \([(x_1, y_1), (x_2, y_2)]\) thus can be computed using four cells in the table in constant time:

\[
\sum([(x_1, y_1), (x_2, y_2)]) = \sum([(1, 1), (x_2, y_2)]) - \sum([(1, 1), (x_2 - 1, y_2)]) - \sum([(1, 1), (x_2, y_1 - 1)]) + \sum([(1, 1), (x_2 - 1, y_1 - 1)]).
\]

Figure 4 illustrates this relationship where the desired area is the bottom-right. The lookup table also can be constructed based on this idea with a linear scan of each value, resulting in a \(O(MNT)\) time cost. A similar idea has been used in image processing and computer vision (i.e., integral image [14]). In the following part, we focus on the enumeration and pruning of ST windows, and ignore the time cost of computing the ACR score in each pattern evaluation.

An alternative representation of ST window: An ST window \((S, T)\) can be uniquely identified by the 3-dimensional coordinate of two locations, namely, the left-bottom-near (LBN) location, and the right-top-far (RTF) location. For example, an ST window \([(1, 1), (10, 10), (3, 4)]\) can be represented as \((1, 10, 3), (1, 10, 4)\). Figure 5 illustrates the LBN and RTF locations of an ST window.

For simplicity when comparing two 3-D locations, we define the following relationship.

**Definition 1.** Given two ST locations \(A = (x_1, y_1, t_1)\) and \(B = (x_2, y_2, t_2)\), \(A \gg B\) (\(B \ll A\)) if \(x_1 \geq x_2 \land y_1 \geq y_2 \land t_1 \geq t_2\). An ST windows \(W_1 = \langle LBN_1, RTF_1 \rangle\) dominates all the ST windows \(W_2 = \langle LBN_2, RTF_2 \rangle\), s.t. \(LBN_1 \ll LBN_2 \land RTF_1 \gg RTF_2\).

General idea of the SWEP algorithm: The SWEP algorithm enumerates all the ST windows by examining all pairs of LBN and RTF locations using a two-level enumeration process. The algorithm first enumerates all the LBN locations (the inner loop). For each LBN location, the algorithm enumerates all the valid RTF locations to find Persistent Change Window (PCW) (inner loop). The enumeration order is designed in such a way that (1) an ST window \(W\) is evaluated only when all of the ST windows \(W'\) are evaluated, where \(W \subset W'\); and (2) if an ST window \(W\) is identified as a PCW pattern, then no subset of \(W\) should be evaluated.

In the outer loop, a breadth-first traversal is performed to enumerate the LBN location \((x_1, y_1, t_1)\) of the candidate ST windows. For each LBN location, in the inner loop, the RTF location is also enumerated among a proper subset of all the locations. Each pair of \((LBN, RTF)\) locations that form a PCW will be sent to output. The details of the algorithm are described below.

Enumerating LBN locations: The algorithm enumerates the LBN locations in a breadth-first manner, starting from the nearest LBN location \((1, 1, 1)\). The enumeration space is all the ST locations in the dataset. This space can be modeled as a 3-D directed lattice graph, where each node represents an ST location, and each directed edge connects neighboring ST locations with a one-unit difference in one of the three dimensions. The direction of each edge, aligned with one of the three dimensions, is from a location with a smaller coordinate to a location with a larger coordinate in that dimension. In such a lattice graph, each non-boundary node has three child nodes, one along each dimension, and three parent nodes, one along each dimension as well. A breadth-first enumeration guarantees that locations with smaller coordinates are always visited before locations with larger coordinates. Figure 7(a) illustrates the process of breadth-first enumeration of the LBN location.

Enumerating ST windows with a fixed LBN: We now consider the problem of enumerating ST windows with a fixed LBN location. In order to determine a unique ST window, we need both LBN and RTF locations. So we propose to enumerate all the valid RTF locations and evaluate the ST windows formed by the given LBN and each RTF. Similar to the LBN enumeration process, the RTF enumeration also can be viewed as a breadth-first traversal on the 3-D lattice graph. The traversal starts from the farthest RTF location \((M, N, T)\), and proceeds towards the given LBN location. Figure 7(b) illustrates the traversal space and the process. The only difference of this traversal space from the previous LBN traversal space is that all the directed edges are in the opposite direction, i.e., pointing from a node to all its ST neighboring locations with smaller coordinates.

The enumeration rules are as follows: A node (candidate RTF) is enumerated only if all of its parent nodes are visited, and none of them form a PCW pattern with the given LBN node. The total number of parents of each node in the lattice graph will be 0 (root node), 1 (boundary nodes along two dimensions), 2 (boundary nodes along one dimension), or 3 (all inner nodes). If the candidate RTF node forms a PCW window with the current LBN node, this PCW will be output and none of the RTF node's children will be enumerated. For example, as illustrated in Figure 7(c), the orange node (RTF) and the LBN node form a PCW pattern. None of the blue nodes (successors of RTF) will be visited. Only non-successors of RTF (white nodes) are enumerated.

In addition, there are three conditions under which a node will not be enumerated (thus pruned), (1) the node to the left, lower, or near side of the given LBN node. This makes sure that all the ST windows have positive volume; (2) the corresponding ST window formed by this node and the given LBN does not satisfy the minimum area or minimum time length constraints; (3) there exists another pair of LBN and RTF locations \((LBN_i, RTF_i)\) such that \((LBN, RTF) \subset (LBN_i, RTF_i)\). The first two conditions are easy to check, while the third one is challenging. We show in the following part that the third condition can be guaranteed by pre-decide the enumeration space for each LBN.

The enumeration space of RTF for each LBN: To address the third challenge above, we consider the follow-
The enumeration space of RTF for a particular LBN location can be represented as the complement set of all locations that are covered by the RTF enumeration process for that LBN. Formally, if LBN is visited before LBN in the enumeration, then LBN is not covered by any RTF location of LBN. So LBN << LBN. So ∩{LBN, RTF} ⊂ PCW. This means RTF should not be part of any existing PCWs. The conclusion is proved.

A careful review of the term ∩{LBN, RTF} may reveal that it is actually the set of locations examined but not selected to output in the RTF enumeration process for LBN. We define this set of locations as the "covered space" of LBN. The covered space of each LBN can be obtained during the enumeration of RTF. For example, in the scenario illustrated in Figure 6, the node labeled RTF forms a PCW with LBN. The covered space of the current LBN (in green) is thus the nine blank nodes to the back of the grid structure. Children of LBN will only need to enumerate RTF among these locations to find PCW.

Due to the transitive property of the << relationship, and the nature of breadth-first enumeration of LBN locations, if LBN << LBN2, then LBN << LBN2’s children. This helps us simplify the representation of the enumeration space. Thus we finally derive the following lemma.

**Lemma 2.** The enumeration space of RTF for a particular LBN location is the intersection of the covered space of all its LBN’s parents. Formally it can be written as:

\[
\text{enumeration}\_\text{space}(LBN) = \cap\{\text{covered}\_\text{space}(LBN) | LBN_i \in \text{parents}(LBN)\}.
\]

**Proof.** Suppose there exist a RTF location RTF in the enumeration space for the current LNB. If there exists some PCW = (LBN, RTF) such that RTF ⊂ PCW, we have RTF >> RTF. Since PCW is already generated, LBN is visited before LBN in the enumeration. So LBN << LBN. So ∩{LBN, RTF} ⊂ PCW. This means RTF should not be part of any existing PCWs. The conclusion is proved.
Algorithm 2 shows the process of the RTF enumeration with a fixed LBN location. The candidate enumeration space of RTF for each LBN location is implemented as a $M \times N \times T$ 0-1 array where 1 represents the location that needs to be enumerated. The algorithm also uses an array ($nVst$) to record the number of parents visited of each node. As described previously, if this number reaches the total number of parents (computed by n_parents()), this node will be enumerated (Lines 14-18).

The pseudo code of the entire SWEP algorithm is presented in Algorithm 3. A list of 3-D arrays ($C_{Space}$) is kept as the covered space of each LBN that has been enumerated so far, where each array is a 0-1 array of size $M \times N \times T$. Since the covered space may be the same for different LBNs, another pointer array ($Link_{spc}$) is established to link each LBN to the corresponding version of the covered space. In the outer loop, all the LBN locations are enumerated (Lines 4-16). In the inner loop, the RTF locations are enumerated (Line 8) by running Algorithm 2. Before entering the inner loop, the enumeration space ($e_{space}$) of LBN is generated by taking the intersection of the covered space of its parents ($Find_{E_{space}}$). After the inner loop is finished, the $C_{Space}$ list and $Link_{spc}$ table are updated with the LBN’s covered space (recorded while enumerating RTF). If it is same as an existing versions of covered space, just update the corresponding pointer in $Link_{spc}$. Otherwise add the new covered space to $C_{space}$.

4. THEORETICAL ANALYSIS

In this section we analyze the correctness, completeness, and computational complexity of the propose algorithms.

**Lemma 3.** The SWEP algorithm is correct. Correctness means that all the ST windows discovered by the algorithm are dominant PCWs based on the definitions.

**Proof.** According Algorithm 3, each ST window in the output is evaluated against the threshold. Now we show that none of the ST windows in the output are subset of others. According to Lemma1 and Lemma2, this is guaranteed as long as the steps are followed. So the SWEP algorithm is correct. □

**Lemma 4.** The SWEP algorithm is complete. The completeness means that all the PCWs which are not subsets of others in the given dataset are reported by the SWEP algorithm.

**Proof.** This is to say that the SWEP algorithm only skip ST windows that are subset of other PCWs. This is obviously true for the enumeration of RTFs with a fixed LBN due to the nature of breadth first traversal. In the enumeration of different LBNs, according to the proof of Lemma 2, later generated PCWs will not be subset of any existing PCWs, as has been proved by the lemma. Obviously they can not be superset of earlier generated PCWs, either due to the order of their LBN nodes. This means the SWEP algorithm is complete. □

**Time and Space complexity:** The time complexity of the naive algorithm is $O(M^2N^2T^2)$ in the first two step. If considering the time for spatial aggregation, the total complexity would reach $O(M^3N^3T^2)$. Suppose there are $k$ PCWs generated, in the third step, the total time cost for eliminating the dominated PCWs is $O(k^2)$ (for pairwise comparison). In the worst case $k = O(M^2N^2T^2)$ and this complexity is $O(M^4N^4T^4)$. For space complexity, in the best case, no PCWs are generated. The space cost will be $O(1)$. In the worst case, all the candidate PCWs need to be stored and the space complexity would reach $O(M^2N^2T^2)$.

For the SWEP algorithm, the time complexity is $O(1)$ in the best case, if the entire ST window is a PCW pattern. The algorithm terminates after the first evaluation. In the worst case, no PCW pattern exists and all of the ST windows need to be examined. This lead to $O(MNT)$ time to enumerate all the LBN locations, and $O(MNT)$ time in each round to enumerate all the RTF locations. The total complexity is $O(M^2N^2T^2)$. If we consider the time for computing the lookup table, the best case time complexity will be $O(MNT)$. For space complexity, in the best case, all of the LBNs have the same covered space, which reduced the total memory cost to $O(MNT)$. In the worst case, each LBN will have a different covered_space, which lead to $O(M^2N^2T^2)$ memory cost.

5. CASE STUDY

This section presents a case study of the proposed approach on a vegetation cover dataset. The goal of this section is to show that the SWEP approach can discover meaningful ST persistent change windows which corresponds to
known phenomenon in particular areas.

5.1 Dataset and Settings:

In the case study, we use Normalized Difference of Vegetation Index (NDVI) data, measuring the vegetation cover extend, from the NASA MODIS project (MOD13Q1). The dataset has a spatial resolution of 250m, with a 16-day temporal resolution ranging from 2000 to 2012. The value ranges from 0 to 1 indicating more vegetation cover. We run the proposed algorithm on selected areas in Saudi Arabia to discover potential ST windows with a fast change of total vegetation cover. In order to get rid of the seasonality affect, we picked the snapshots of the same time of each year (July 27/28) and generated a spatial time series dataset. Figure 8 (a) shows the study area and its position in a world map. The data for this area is 200 by 300 pixels by 13 years.

In this example, we employ average as the spatial aggregate function in order to make the measure of different spatial windows comparable. It can be computed in the same way as we illustrated in Section 3. We select the average change rate (ACR) threshold as 0.1, meaning that the selected ST window should have a persistent increase of mean vegetation cover at an average rate of 10% per year. In order to reduce the trivial patterns discovered, we require that the PCWs should at least have 200 pixels in area and 4 years in time length.

![Figure 8: The study area and one discovered PCW highlighted in three snapshots of the MODIS NDVI data](image)

![Figure 9: Observations in the same area from Google Time lapse [1]](image)

![Figure 10: Spatial aggregated time series of the discovered PCW.](image)

Table 2: Comparison of time and space complexity of the two algorithms

<table>
<thead>
<tr>
<th></th>
<th>Naive approach</th>
<th>SWEP approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time complexity</td>
<td>$O(M^2N^2T^3)$</td>
<td>$O(MNT)$</td>
</tr>
<tr>
<td>Space complexity</td>
<td>$O(1)$</td>
<td>$O(MNT)$</td>
</tr>
</tbody>
</table>

![Algorithm 3: ST Window Enumeration and Pruning (SWEP) Algorithm](image)
5.2 Results: Irrigation in Saudi Arabia

The irrigation in Saudi Arabia has led to a significant increase in farmland and vegetation in recent decades. This process has been featured as one of the best-known land cover change process by Google Time-lapse [1]. Figure 9 (b-d) shows the snapshots of the observations from Google Time-lapse in 2001, 2006, and 2012 respectively.

The algorithm discovered a Persistent Change Window (PCW) consisting of (1) a rectangular spatial area with 100 by 50 pixels to the north of the study area, and (2) a time period from 2001 to 2012. using given threshold settings. The spatial footprint of the PCW is shown in Figure 8. The spatial aggregated (average) NDVI time series of this PCW is shown in Figure 10, in the highlighted box. The average change rate (ACR) of this pattern is 11.5% per year during the above period. This discovery shows that in the highlighted area, during 2001-2012, there was a persistent and rapid increase of vegetation cover. This result can also be verified by observations from Google Time-lapse shown in the previous figures, where a clear expansion of green land can be seen from 2001 through 2012. Note that there are also increase of NDVI out side the discovered PCW. However, due to our constraint on the minimum change rate, a larger PCW may not have a significant enough change rate to be displayed. But one could simply reset the threshold and discover these candidates.

The above results shows the effectiveness of the proposed approach in finding significant persistent change windows.

6. EXPERIMENTAL EVALUATION

In this section, we present experimental evaluation results on a synthetic dataset. The goals of the experiments are (1) compare the time cost of the two algorithms with respect to dataset size, and (2) to compare their run time under different pattern distribution in the data.

6.1 Experiment Setup

Figure 11 illustrates the setup of the experiments. We implement the Naive algorithm and SWEP algorithm in a simulator. We use a synthetic dataset to feed the simulator. We manipulate the value distribution so that the size of the PCW is controlled in each run. In the experiments, we vary the spatial window side length (M or N) and the time length (T) to test the scalability of the algorithms. In addition to the above parameters, we also use a new measure named Pattern Volume Ratio (PVR) to evaluate the performance under different data distribution. The Pattern Volume Ratio represents the ratio of (1) the volume of the largest WCP in the dataset, and (2) the volume of the entire dataset (M × N × T). For example, the PVR in the sample dataset given in Figure 2 is \((3 \times 3 \times 4)/(4 \times 4 \times 4) = 0.56\). This ratio is used as an indicator of the computational savings of SWEP algorithm, where PVR → 0 indicates the worst case (PCWs are small) and PVR=1 indicates the best case (entire dataset is a PCW). All results are measured in seconds of CPU time.

The algorithms are implemented and tested in Matlab v2013a. The platform is a HP ProLiant BL280c G6 blade servers, with a quad-core 2.8 GHz Intel Xeon X5560 processor and 24 GB shared memory, running Linux system.

6.2 Results and Analysis

We first evaluate the computational time with respect to the size of the spatial window. We assume that the spatial window is a square (M=N), and vary the side length from 10 to 50. The area thus varies from 100 to 2500. The time length is fixed at 20. We first test a scenario close to the worst case, where the PVR value is fixed at 0.1. Figure 12(a) shows the run time of the two algorithms. A clear trend can be observed. The run time of the naive algorithm increase at a much higher speed than the SWEP algorithm. The savings become significant after N≥ 30 and reaches as much as 80% when N=50. We then test a scenario close to the best case, where the PVR value is fixed at 0.95. The largest PCW is almost as large as the entire dataset. Figure 12(b) shows the run time of the two algorithms. As can be seen, the run time of the naive algorithm increase exponentially, while the SWEP algorithm stays linear and orders of magnitude faster than the naive algorithm.

We next test the computational time of the two algorithms on increasing time length (T). We fix the spatial window as 50 × 50 = 2500, and increase the time length from 10 to 50. We also test the best case (PVR = 0.95) and the worst case (PVR = 0.1). As shown in Figure 12(c), in the worst case, the two algorithm both have subquadratic trends. The SWEP algorithm is always significantly faster than the naive algorithm, with up to 70% speedup. In the best case shown in Figure 12(d), however, the near-constant time SWEP is orders of magnitude faster than the the naive algorithm who increases linearly.

Finally, we evaluate the run time of the two algorithms on a fixed dataset with different PVR. Intuitively, a larger PVR favors the SWEP algorithm. The dataset is 50 × 50 × 50 = 125000 locations. The PVR varies from 0.1 (near-worst case) to 1 (best case), with a step of 0.1. As can be seen in Figure 12(e), The run time of the SWEP algorithm keeps decreasing since the computational savings is increasing. However, the naive algorithm has a slightly increasing time since the total number of candidates increases, which lead to a longer time for Step 3 in Algorithm 1. The SWEP algorithm always outperforms the naive algorithm with huge computational savings.

7. CONCLUSION AND FUTURE WORK

This work explored the problem of persistent change window (PCW) discovery. This problem is important for critical societal applications such as detecting desertification, deforestation, and urban sprawl. However, this problem is computationally challenging because of the large number of candidate patterns, the lack of monotonicity where sub-regions of a region of interest may not be interesting, the lack of predefined window sizes for region-time interval pairs, and large datasets of detailed resolution and high volume. We proposed an ST window enumeration and pruning approach
SWEP as a computational solution to PCW. SWEP is novel because unlike previous approaches that focus on local spatial footprints, it uses zonal spatial footprints when finding region-time interval pairs such as deforestation in the Amazon over decades. Experiments on synthetic datasets showed that SWEP leads to computational savings over the naive approach without affecting result quality. We also presented a case study using vegetation cover data to evaluate the effectiveness of SWEP and theoretical analysis to validate its correctness, completeness, and space/time complexity. In future work, we would like to enhance our case studies by discovering more well recorded patterns (e.g., those shown by Google Time Lapse [1]) to validate the effectiveness of our approach. In addition, we would like to explore other algorithmic designs and cloud computing solutions to improve the efficiency of the current approach. We also plan to extend the current approach to discover interesting spatiotemporal patterns with irregular spatial footprints.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1029711, IIS-1320580, 0940818 and IIS-1218168 as well as USDOD under Grant No. HM1582-08-1-0017, and HM0210-13-1-0005. We would like to thank Kim Koffolt and the members of the University of Minnesota Spatial Database and Spatial Data Mining Research Group for their comments, and the Minnesota Supercomputing Institute for computational facility support.

9. REFERENCES