Using Bitmap Index for Interactive Exploration of Large Datasets

INFO3504 Presentation

Nick Armstrong
Daniel Collis

University of Sydney

Table of Contents

Bitmap Indexes

Bitmap Indexes:

Bitmap Indexes

Bitmap Indexes:

- Are an old idea

Bitmap Indexes

Bitmap Indexes:

- Are an old idea
- Are efficient for big data and data warehouse solutions with read-only records

Bitmap Indexes

Bitmap Indexes:

- Are an old idea
- Are efficient for big data and data warehouse solutions with read-only records
- Do not reorder data, unlike some other algorithms

What Are They?

For those who weren't in last week's lecture, they

What Are They?

For those who weren't in last week's lecture, they are a way to represent records and fields as a grid (essentially).

What Are They?

For those who weren't in last week's lecture, they are a way to represent records and fields as a grid (essentially).

Just imagine...

Imagine a table, with a bunch of records (R_x) and an enumeration ($A_x = \{1, 2, 3\}$):

What Are They?

For those who weren't in last week's lecture, they are a way to represent records and fields as a grid (essentially).

Just imagine...

Imagine a table, with a bunch of records (R_x) and an enumeration ($A_x = \{1, 2, 3\}$):

	Value	A_1	A_2	A_3
R_1	3			
R_2	1			
R_3	1			
R_4	2			
R_5	3			

What Are They?

For those who weren't in last week's lecture, they are a way to represent records and fields as a grid (essentially).

Just imagine...

Imagine a table, with a bunch of records (R_x) and an enumeration ($A_x = \{1, 2, 3\}$):

	Value	A_1	A_2	A_3
R_1	3	0	0	1
R_2	1	1	0	0
R_3	1	1	0	0
R_4	2	0	1	0
R_5	3	0	0	1

What Are They?

The example table mentioned before can become really really big with multiple attributes and multiple rows.

What Are They?

The example table mentioned before can become really really big with multiple attributes and multiple rows.

Lots of attributes!

It could look something like this:

	Value	A_1	A_2	A_3
R_1	3	0	0	1
R_2	1	1	0	0
R_3	1	1	0	0
R_4	2	0	1	0
R_5	3	0	0	1

	Value	B_1	B_2	B_3
R_1	1	1	0	0
R_2	1	1	0	0
R_3	3	0	0	1
R_4	2	0	1	0
R_5	3	0	0	1

What Are They?

The example table mentioned before can become really really big with multiple attributes and multiple rows.

Lots of attributes!

It could look something like this:

	Value	A_1	A_2	A_3
R_1	3	0	0	1
R_2	1	1	0	0
R_3	1	1	0	0
R_4	2	0	1	0
R_5	3	0	0	1

	Value	B_1	B_2	B_3
R_1	1	1	0	0
R_2	1	1	0	0
R_3	3	0	0	1
R_4	2	0	1	0
R_5	3	0	0	1

How to determine if a record matches multiple attributes? Use a bitwise-AND!

What Are They?

The example table mentioned before can become really really big with multiple attributes and multiple rows.

Lots of attributes!

It could look something like this:

	Value	A_1	A_2	A_3
R_1	3	0	0	1
R_2	1	1	0	0
R_3	1	1	0	0
R_4	2	0	1	0
R_5	3	0	0	1

	Value	B_1	B_2	B_3
R_1	1	1	0	0
R_2	1	1	0	0
R_3	3	0	0	1
R_4	2	0	1	0
R_5	3	0	0	1

How to determine if a record matches multiple attributes? Use a bitwise-AND!

This is quite a quick and efficient operation.

Bitmap Compression

Bitmap Indexes are great, efficient, and all. But, they can become very large!

Bitmap Compression

Bitmap Indexes are great, efficient, and all. But, they can become very large! The researchers of this paper had two data samples:

Bitmap Compression

Bitmap Indexes are great, efficient, and all. But, they can become very large! The researchers of this paper had two data samples:

- 1.6 GiB

Bitmap Compression

Bitmap Indexes are great, efficient, and all. But, they can become very large! The researchers of this paper had two data samples:

- 1.6 GiB
- 33 GiB

Bitmap Compression

Bitmap Indexes are great, efficient, and all. But, they can become very large! The researchers of this paper had two data samples:

- 1.6 GiB
- 33 GiB

Storing these in a typical database with a `Bitmap Index` took $3\times$ the size, similar to when using a `B-Tree Index`

Bitmap Compression

Bitmap Indexes are great, efficient, and all. But, they can become very large! The researchers of this paper had two data samples:

- 1.6 GiB
- 33 GiB

Storing these in a typical database with a `Bitmap Index` took $3\times$ the size, similar to when using a `B-Tree Index` = Not ideal!

Bitmap Compression

What can we do?

Bitmap Compression

What can we do?

- Run-Length Encoding

Bitmap Compression

What can we do?

- Run-Length Encoding
- Word-Aligned Hybrid (WAH)

Word-Aligned Hybrid (WAH)

A patent exists for it:

Publication number	US6831575 B2
Publication type	Grant
Application number	US 10/701,655
Publication date	Dec 14, 2004
Filing date	Nov 4, 2003
Priority date ?	Nov 4, 2002
Fee status ?	Lapsed
Also published as	US20040090351
Inventors	Kesheng Wu, Arie Shoshani, Ekow Otoo
Original Assignee	The Regents Of The University Of California
Export Citation	BiBTeX, EndNote, RefMan

Patent Citations (2), Referenced by (15), Classifications (9), Legal Events (7)

External Links: USPTO, USPTO Assignment, Espacenet [1]

[1] https://www.google.com/patents/US6831575

[2] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1214955

Word-Aligned Hybrid (WAH)

A patent exists for it:

Publication number	US6831575 B2
Publication type	Grant
Application number	US 10/701,655
Publication date	Dec 14, 2004
Filing date	Nov 4, 2003
Priority date	Nov 4, 2002
Fee status	Lapsed
Also published as	US20040090351
Inventors	Kesheng Wu, Arie Shoshani, Ekow Otoo
Original Assignee	The Regents Of The University Of California
Export Citation	BiBTeX, EndNote, RefMan

Patent Citations (2), Referenced by (15), Classifications (9), Legal Events (7)

External Links: USPTO, USPTO Assignment, Espacenet [1]

Using Bitmap Index for Interactive Exploration of La

Kesheng Wu, Wendy Koegler, Jacqueline Chen, and Arie Shos

Abstract

Many scientific applications generate large spatio-temporal datasets. A common way of exploring these datasets is to identify and track regions of interest. Usually these regions are defined as contiguous sets of points whose attributes satisfy some user defined conditions, e.g. high temperature regions in a combustion simulation. At each time step, the regions of interest may be identified

velocity throughout a volume of are then identified as meeting so these properties, e.g. regions in t is very hot. Datasets can have hu as pressure, concentrations, etc. be defined based on any of the at well as combinations of differen usually has to explore a number before proceeding to subsequent [2]

[1] https://www.google.com/patents/US6831575

[2] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1214955

What Exactly Is It?

Differers from Run Length Encoding in **two** major ways:

What Exactly Is It?

Differers from Run Length Encoding in **two** major ways:

- It only records long groups of 0s or 1s using Run Length Encoding (the short groups are represented *literally*)

What Exactly Is It?

Differs from Run Length Encoding in **two** major ways:

- It only records long groups of 0s or 1s using Run Length Encoding (the short groups are represented *literally*)
- It requires groups of a certain size so that operations on the compressed data are efficient

How Does It Work?

Understand: What Is A Fill?

A group of consecutive identical bits.
A fill with only 0s is called a 0-fill and a fill with only 1s is called a 1-fill.

Understand: Word-Size?

The word size is dependent on the computer architecture. On x86 machines (32-bit), the word size is 32-bits. On x86_64/amd64 machines, the word size is 64-bits.

How Does It Work?

1. Split the bitmap index into multiple rows of size: `word-length` − 1
2. If the row is **NOT** entirely made up of either 1s or 0s (i.e. not a 0-fill or a 1-fill), then keep as is (*literal*) and prefix row with a 0
3. Otherwise, determine how many following rows are also fill rows. Prefix the row with a 1 (fill bit), mark the second bit with the type of bit that is to be filled, then the rest of the bits are how many subsequent rows also match this criteria

How Does It Work?

1. Split the bitmap index into multiple rows of size: `word-length` $- 1$
2. If the row is **NOT** entirely made up of either 1s or 0s (i.e. not a 0-fill or a 1-fill), then keep as is (*literal*) and prefix row with a 0
3. Otherwise, determine how many following rows are also fill rows. Prefix the row with a 1 (fill bit), mark the second bit with the type of bit that is to be filled, then the rest of the bits are how many subsequent rows also match this criteria

Who understands all of that?

How Does It Work?

Let's go back to our example:

Just imagine...

Imagine a table, with a bunch of records (R_x) and an enumeration ($A_x = \{1, 2, 3\}$):

	Value	A_1	A_2	A_3
R_1	3	0	0	1
R_2	1	1	0	0
R_3	1	1	0	0
R_4	2	0	1	0
R_5	3	0	0	1

How Does It Work?

Let's go back to our example:

Just imagine...

Imagine a table, with a bunch of records (R_x) and an enumeration ($A_x = \{1, 2, 3\}$):

	Value	A_1	A_2	A_3
R_1	3	0	0	1
R_2	1	1	0	0
R_3	1	1	0	0
R_4	2	0	1	0
R_5	3	0	0	1

Let's perform WAH on A_1.

How Does It Work?

Let's go back to our example:

Just imagine...

Imagine a table, with a bunch of records (R_x) and an enumeration ($A_x = \{1, 2, 3\}$):

	Value	A_1	A_2	A_3
R_1	3	0	0	1
R_2	1	1	0	0
R_3	1	1	0	0
R_4	2	0	1	0
R_5	3	0	0	1

Let's perform WAH on A_1. Let's turn it on its side: $A_1 = 01100$

How Does It Work?

Preamble:

- Let's turn it on its side: $A_1 = 01100$
- Assume that our word size is 16 bits.
- Assume that there are 60 tuples (i.e. $R_1, R_2, \cdots, R_{31}, R_{32}$)

How Does It Work?

Preamble:

- Let's turn it on its side: $A_1 = 01100$
- Assume that our word size is 16 bits.
- Assume that there are 60 tuples (i.e. $R_1, R_2, \cdots, R_{31}, R_{32}$)

$$
\begin{aligned}
A_1 = \quad & 011001001011010 \\
& 000000000000000 \\
& 000000000000000 \\
& 100110110100101
\end{aligned}
$$

How Does It Work?

Preamble:

- Let's turn it on its side: $A_1 = 01100$
- Assume that our word size is 16 bits.
- Assume that there are 60 tuples (i.e. $R_1, R_2, \cdots, R_{31}, R_{32}$)

$$A_1 = \begin{array}{l} 011001001011010 \\ 000000000000000 \\ 000000000000000 \\ 100110110100101 \end{array}$$

Result from applying WAH:

$$A_1 = 0 \quad 011001001011010$$

How Does It Work?

Preamble:

- Let's turn it on its side: $A_1 = 01100$
- Assume that our word size is 16 bits.
- Assume that there are 60 tuples (i.e. $R_1, R_2, \cdots, R_{31}, R_{32}$)

$$
\begin{aligned}
A_1 = \quad & 011001001011010 \\
& 000000000000000 \\
& 000000000000000 \\
& 100110110100101
\end{aligned}
$$

Result from applying WAH:

$$
\begin{aligned}
A_1 = \quad & 0 \quad 011001001011010 \\
& 10 \quad 00000000000010
\end{aligned}
$$

How Does It Work?

Preamble:

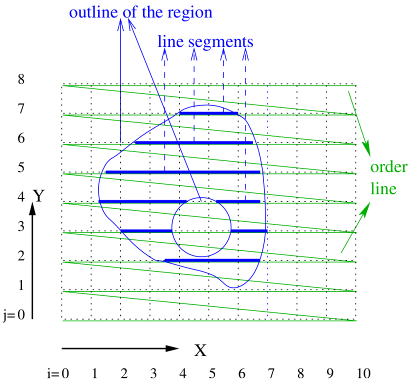
- Let's turn it on its side: $A_1 = 01100$
- Assume that our word size is 16 bits.
- Assume that there are 60 tuples (i.e. $R_1, R_2, \cdots, R_{31}, R_{32}$)

$$
\begin{aligned}
A_1 = \quad & 011001001011010 \\
& 000000000000000 \\
& 000000000000000 \\
& 100110110100101
\end{aligned}
$$

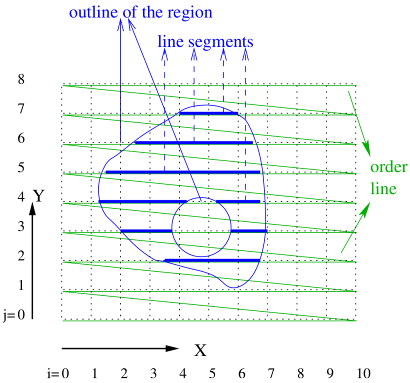
Result from applying WAH:

$$
\begin{array}{lll}
A_1 = & 0 & 011001001011010 \\
& 10 & 00000000000010 \\
& 0 & 100110110100101
\end{array}
$$

Example From Paper

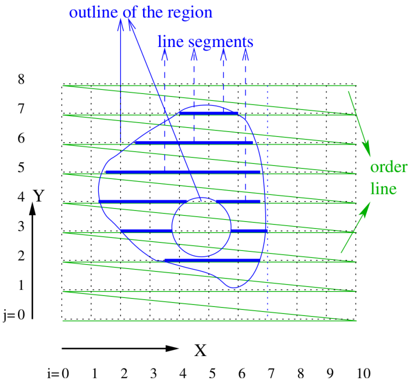


Example From Paper



Run-Length Encoding: 26×0, 3×1, 6×0, 2×1, 2×0, 1×1, 6×0, 3×1, 1×0, 1×1, 6×0, 5×1, 7×0, 4×1, 8×0, 3×1, 15×0

Example From Paper



Run-Length Encoding: 26×0, 3×1, 6×0, 2×1, 2×0, 1×1, 6×0, 3×1, 1×0, 1×1, 6×0, 5×1, 7×0, 4×1, 8×0, 3×1, 15×0

WAH Encoding (in hex):
`0000001C 0640E81F 00F00E00 00000000`

Versus Run Length Encoding

- In theory, it is not as compressible as Run Length Encoding.

Versus Run Length Encoding

- In theory, it is not as compressible as Run Length Encoding. But in practice, it takes less space than the Run Length Encoding scheme, especially if many of the groups are small

Versus Run Length Encoding

- In theory, it is not as compressible as Run Length Encoding. But in practice, it takes less space than the Run Length Encoding scheme, especially if many of the groups are small
- Logical operations on WAH compressed bitmaps can work *directly* on the compressed data and *generate compressed answers*. Because of this, the time to perform these operations is proportional to the sizes (bytes) of the bitmaps involved.

Versus Run Length Encoding

- In theory, it is not as compressible as Run Length Encoding. But in practice, it takes less space than the Run Length Encoding scheme, especially if many of the groups are small
- Logical operations on WAH compressed bitmaps can work *directly* on the compressed data and *generate compressed answers*. Because of this, the time to perform these operations is proportional to the sizes (bytes) of the bitmaps involved.
- We can generate bitmap indices efficiently. Using WAH compression, it is possible to only insert bits that are 1 when creating a bitmap index. In practice, generating a bitmap index has the computational complexity of $O(N\log(b))$ (where b is number of bitmaps generated), compared to generating an uncompressed bitmap index: $O(Nb)$ (significantly worse).

Feature Tracking

Given some kinds of data we want to identify and track regions of interest.

Feature Tracking

Given some kinds of data we want to identify and track regions of interest.

- Numerical simulations of scalar or vector fields on some domain.
  - E.g. The temperature and velocity in a volume of air.

Feature Tracking

Given some kinds of data we want to identify and track regions of interest.

- Numerical simulations of scalar or vector fields on some domain.
  - E.g. The temperature and velocity in a volume of air.
- Stored as a collection of snapshots of a 3D grid over time.

Feature Tracking

Given some kinds of data we want to identify and track regions of interest.

- Numerical simulations of scalar or vector fields on some domain.
  - E.g. The temperature and velocity in a volume of air.
- Stored as a collection of snapshots of a 3D grid over time.
- Regions of interest are regions that satisfy certain properties
  - E.g. High temperature regions in a combustion simulation.

Feature Tracking

Given some kinds of data we want to identify and track regions of interest.

- Numerical simulations of scalar or vector fields on some domain.
  - E.g. The temperature and velocity in a volume of air.
- Stored as a collection of snapshots of a 3D grid over time.
- Regions of interest are regions that satisfy certain properties
  - E.g. High temperature regions in a combustion simulation.

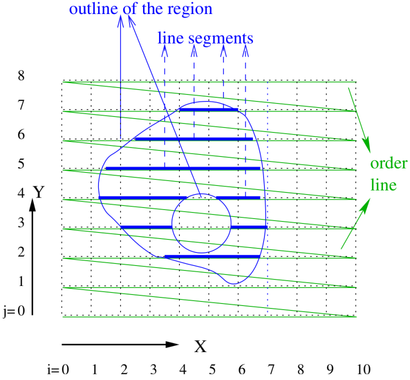
To visualise this data we need the boundaries of connected regions, and how they change in time.

Region Growing

After using bitmap indexes to find the points satisfying the conditions we need to identify the connected regions.

Region Growing

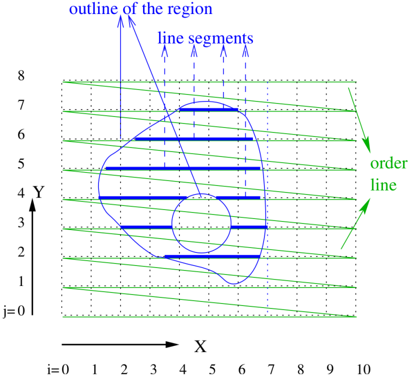
After using bitmap indexes to find the points satisfying the conditions we need to identify the connected regions.



- Compressed bitmap converted into a list of line segments

Region Growing

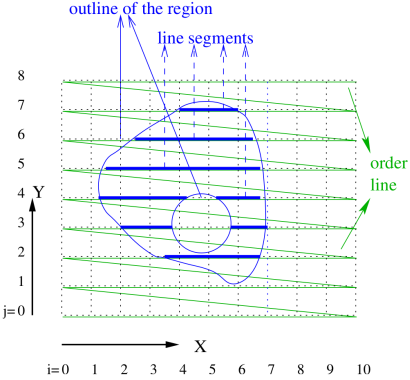
After using bitmap indexes to find the points satisfying the conditions we need to identify the connected regions.



- Compressed bitmap converted into a list of line segments
- Line segments are aligned along the x-axis

Region Growing

After using bitmap indexes to find the points satisfying the conditions we need to identify the connected regions.



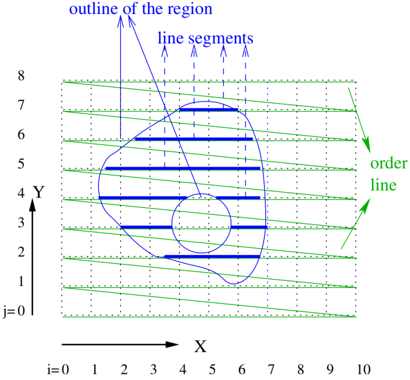
- Compressed bitmap converted into a list of line segments
- Line segments are aligned along the x-axis
- Discovers line segments in order of j and k

Region Growing

Once we have the line segments, we assign them to connected regions.

Region Growing

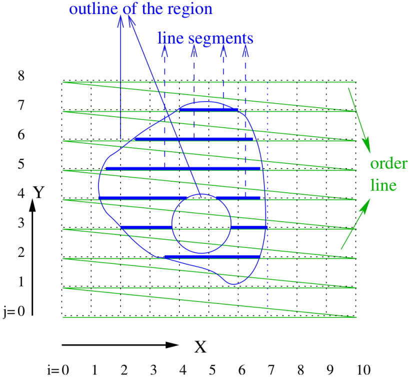
Once we have the line segments, we assign them to connected regions.



- For each line segment with coordinates (j, k) we compare with all line segments with coordinates (j-1, k), (j, k-1), and maybe (j-1, k-1)

Region Growing

Once we have the line segments, we assign them to connected regions.



- For each line segment with coordinates (j, k) we compare with all line segments with coordinates (j-1, k), (j, k-1), and maybe (j-1, k-1)
- If the line segments overlap then they are connected

Region Growing

To generate a smooth boundary we calculate the positions of the exposed points and their outside neighbours and interpolate.

Region Growing

To generate a smooth boundary we calculate the positions of the exposed points and their outside neighbours and interpolate.

- Most steps in region growing take $O(s)$ time. s is the size of the boundary.

Region Growing

To generate a smooth boundary we calculate the positions of the exposed points and their outside neighbours and interpolate.

- Most steps in region growing take $O(s)$ time. s is the size of the boundary.
- Some take $O(s_l)$ time. s_l is the number of line segments.

Region Growing

To generate a smooth boundary we calculate the positions of the exposed points and their outside neighbours and interpolate.

- Most steps in region growing take $O(s)$ time. s is the size of the boundary.
- Some take $O(s_l)$ time. s_l is the number of line segments.
- Overall takes $O(s)$ time.
  - vs $O(v)$ time for a straightforward algorithm.

Region Tracking

With spatio-temporal datasets we want to be able to track the evolution of regions of interest over time.

Region Tracking

With spatio-temporal datasets we want to be able to track the evolution of regions of interest over time.

- We produce bitmaps for each connected region identified during region growing.

Region Tracking

With spatio-temporal datasets we want to be able to track the evolution of regions of interest over time.

- We produce bitmaps for each connected region identified during region growing.
- These bitmaps are used to determine the overlap of regions in neighbouring time steps.

Region Tracking

With spatio-temporal datasets we want to be able to track the evolution of regions of interest over time.

- We produce bitmaps for each connected region identified during region growing.
- These bitmaps are used to determine the overlap of regions in neighbouring time steps.
- Can do a bitwise AND operation and count the number of 1s.

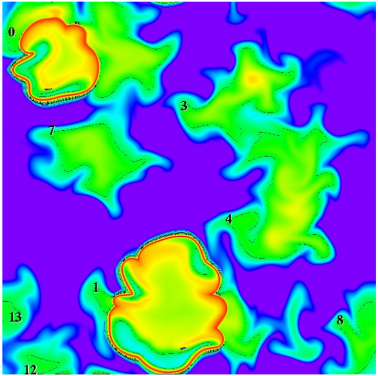
Region Tracking

With spatio-temporal datasets we want to be able to track the evolution of regions of interest over time.

- We produce bitmaps for each connected region identified during region growing.
- These bitmaps are used to determine the overlap of regions in neighbouring time steps.
- Can do a bitwise AND operation and count the number of 1s.
- Has a time and space complexity of $O(s_1 + s_2)$ which is optimal.
  - Compared to $O(v_1 v_2)$ for a straightforward approach.

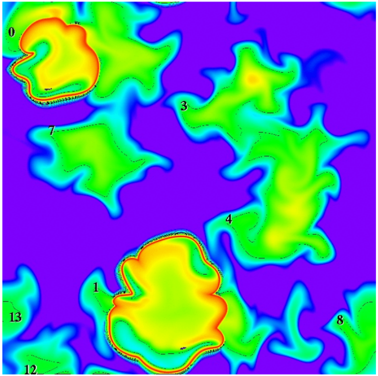
Region Tracking

A simple approach assigns numbers to regions which identifies them over time.



Region Tracking

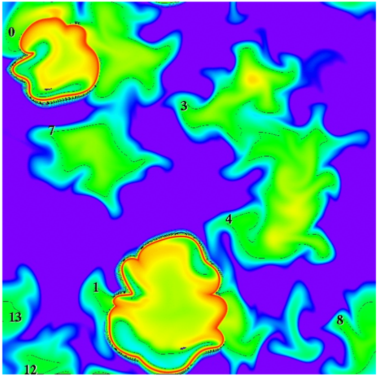
A simple approach assigns numbers to regions which identifies them over time.



- At the first time step, regions are arbitrarily numbered.

Region Tracking

A simple approach assigns numbers to regions which identifies them over time.



- At the first time step, regions are arbitrarily numbered.
- A region in time step p gets assigned the number of the region in time step $p-1$ with maximal overlap
  - or an arbitrary number if it does not overlap.

Thanks!

Slides can be found online:

`http://tinyurl.com/info3504-bitindex`