A Tunable Compression Framework for Bitmap Indices

Gheorghi Guzun, Guadalupe Canahuate

The University of Iowa

David Chiu

Washington State University

Jason Sawin

University of St. Thomas

Overview

- Big Data and bitmap indexing
- Background on bitmap compression
  - WAH, PLWAH, EWAH, COMPAX.
- Variable Aligned Length (VAL) Framework
  - Bitmap Compression – Segment Length Selection
  - Query Execution – between bitmaps encoded with different lengths
  - User defined parameter to control trade-offs
- Evaluation
- Conclusions and Future work

Big Data and Bitmap Indices

- Big Data enables deeper understanding of things but comes at a cost
  - Disk space, memory and bandwidth needs grow very fast
  - When used properly, bitmap indices improve both compression and query times
- A Bitmap Index is a binary vector that represents which elements fall into a category or have certain property
  - Used in data warehousing applications with large amounts of data and ad hoc queries, but a low level of concurrent DML transactions.
- Advantages:
  - Leverage fast bit-wise operations to resolve queries
  - Highly compressible (hybrid run-length encoding)

Bitmap Example

Tuple	**Name**	**Age**	**Salary**	**City**	**Favorite Drink**	**...**
t1	Sara	42	65,000	Beaverton	Ninkasi	
t2	Julie	50	130,000	West Linn	Bridgeport	
t3	Tom	21	25,000	Portland	Boneyard	
...						

Discretize (binning) and bit assignment

	Name			**Age**			
Tuple	Sara?	Julie?	Tom?	Age <= 25	25 < Age < 50	50 <= Age	...
t1	1	0	0	0	1	0	...
t2	0	1	0	0	1	0	
t3	0	0	1	1	0	0	
...							

Bitmap Compression

- Goal:
  - Reduce bitmap size
  - Minimize overhead in query execution time
    - Execute query without explicit decompression
- Run-length encoders
  - Word-Aligned Hybrid Code (WAH) – widely adopted
    - Two types of words - simple
    - Word Aligned → Faster CPU time
    - EWAH, PLWAH, CONCISE, COMPAX are word-aligned based

133 Bits	1,20*0,4*1,78*0,30*1
31-bit groups	1,20*0,4*1,6*0 62*0 10*0,21*1 9*1
groups in hex	400003C0 00000000 00000000 001FFFFF 000001FF
WAH (hex)	400003C0 80000002 001FFFFF 000001FF

Literal Word Fill Word

WAH Compression – Weak points

- Often uses too many bits for encoding runs
  - 30 bits for W=32 and 62 bits for W=64
  - For W=64, longest compressible run in one word is $63*2^{62}$
  - PLWAH, CONCISE, COMPAX try to improve by exploiting this waste of bits
- Not all run-lengths are equal
  - Most real-world datasets are skewed
  - Sorted bitmaps have even greater contrasts in run-lengths

- **Solution – Variable number of bits to compress runs**

Variable Length Compressions

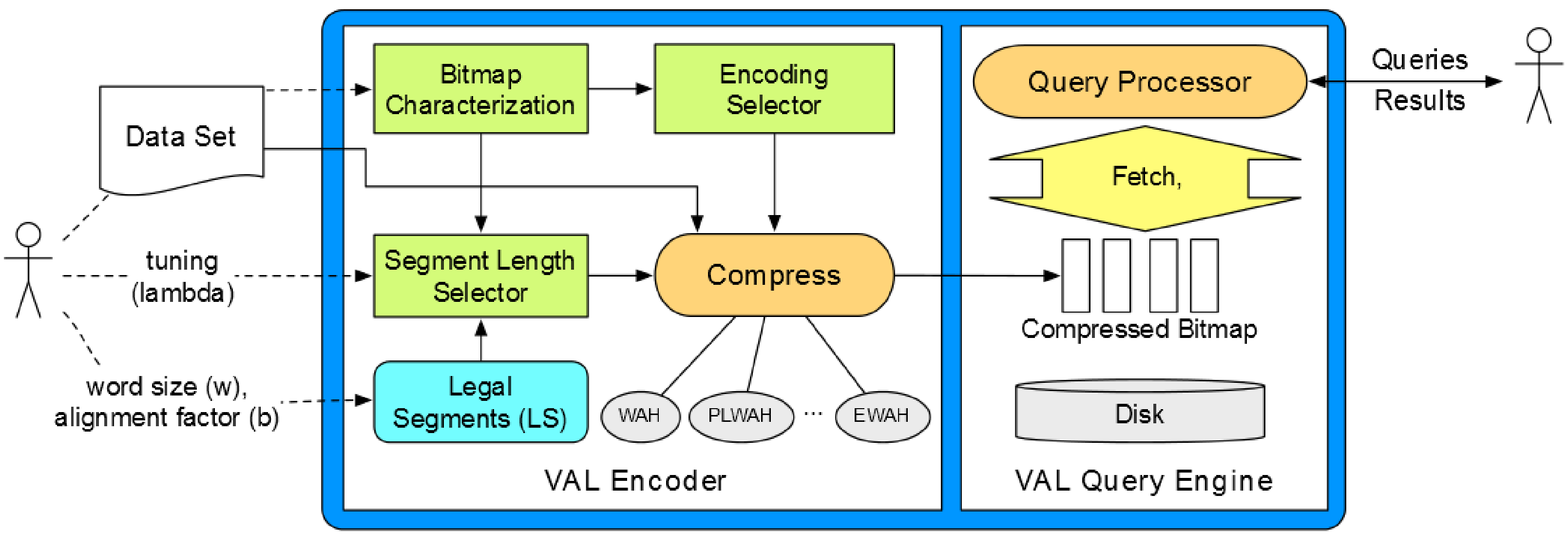
- Variable Length Compression (VLC)
  - Uses variable number of bits (variable segment length) for compression
  - Looser alignment requirements – slows down queries dramatically
- PWAH (Partitioned WAH)- WAH using shorter segment lengths (partitions)
  - Encodes entire dataset with one single segment length
  - Does not provide query algorithms between bitmaps encoded with different segment lengths

VAL- Variable Aligned Length (Challenges)

- Improve Compression
  - Use variable number of bits (segments) to compress runs $s < W$
- Keep the word encapsulation structure
  - Reading entire words streamlines memory accesses
- Efficient query algorithms
  - Enable queries between bitmaps encoded using a different “s”
  - Maintain alignment between segments

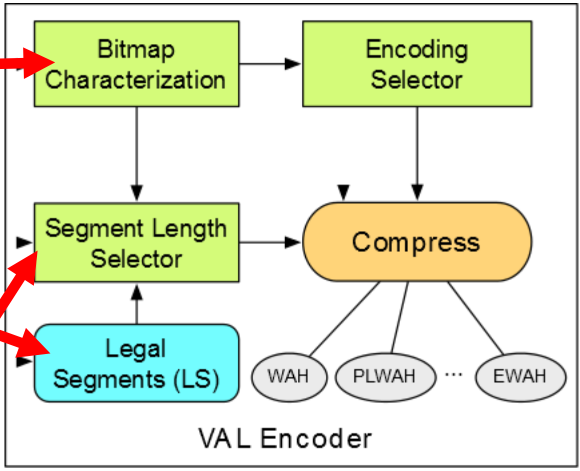
The VAL Framework

- Integrates existing compression techniques – Uses VAL
  - Based on the observation that most existing word aligned based encodings use the essentially the same query algorithm



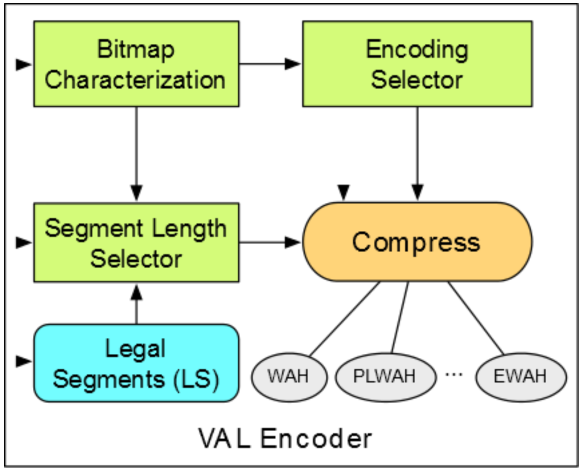
Segment Length Selection

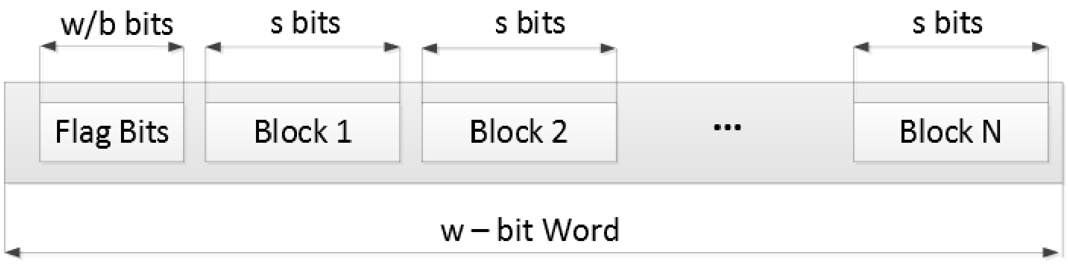
- Scan data
  - Collect bit-density, bitmap size, run-lengths
- Legal Segment Lengths
  - $LS = \{2^i \times (b-1) | 0 \le i \le (log_2(w) - log_2(b))\}$
  - e.g. for W=64, LS can be {15, 30, 60}
- Segment Length Selector
  - $\begin{cases} s_{c+i}, & if \ \frac{size(B,s_c)\times(1+\lambda)^{1+i+\lambda}}{i+1} \ge size(B, s_{c+i}) \\ s_c, & Otherwise \end{cases}$
  - $s_c = arg \min_{s_i \in LS}\{size(B, s_i)\}$

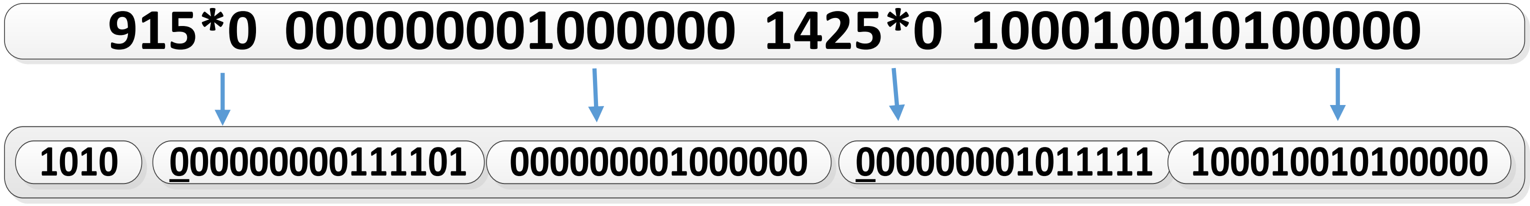


Compression

- Compresses given selected method and segment length
- Encapsulates blocks into words
- Each word contains flag-bits for it’s blocks
- Each bitmap has a header where the segment length is stored

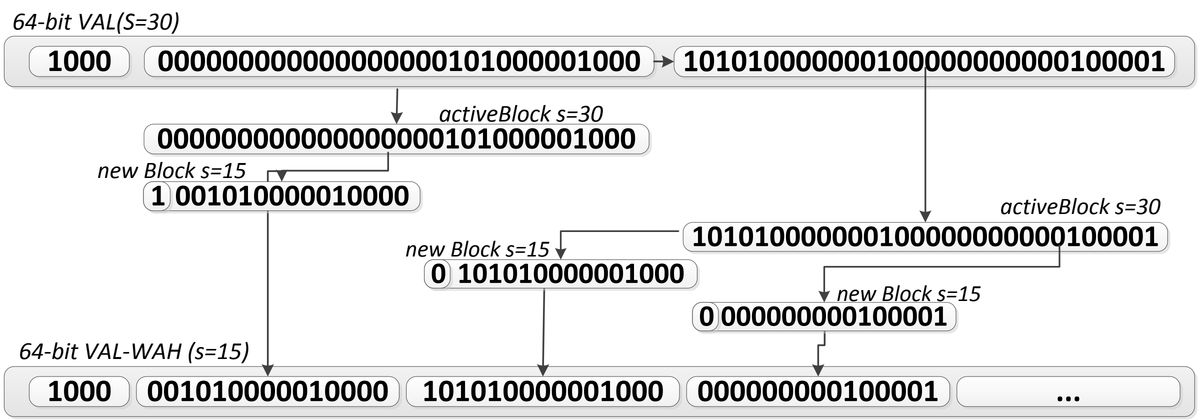


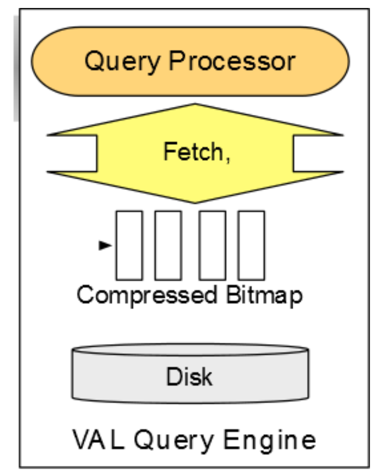




General Query Algorithm

- One general logical operation algorithm for all encodings
  - Iterate the two bitmaps and operate block by block
  - Convert between segment lengths if necessary

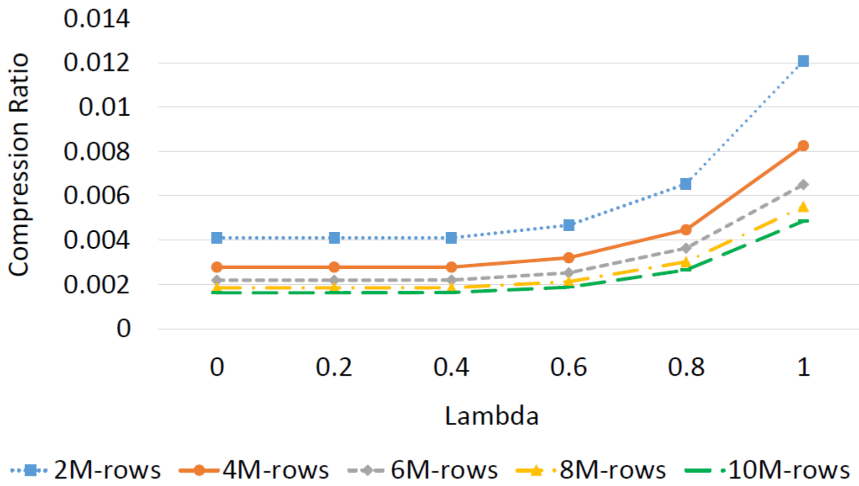




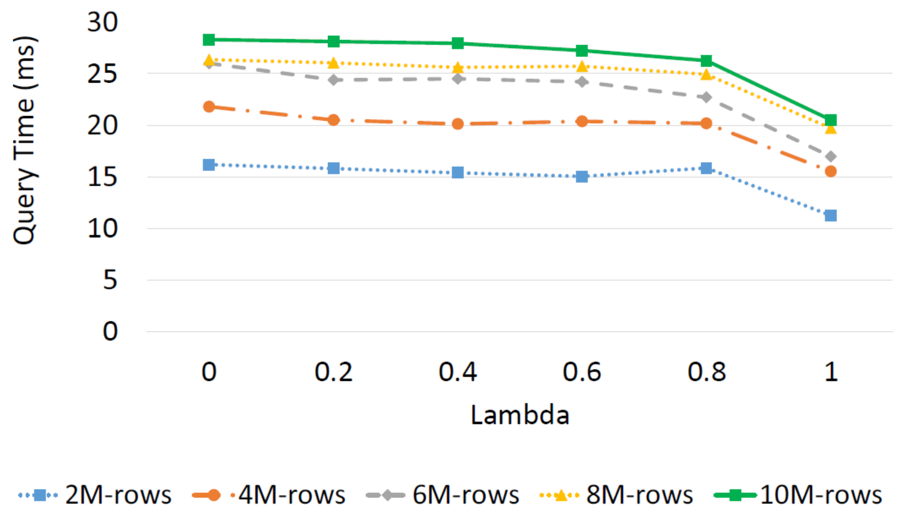
Experimental Setup

- Implemented VAL-WAH as proof of concept
- Run on 64-bit Intel i7-2600 3,2 GHz 8MB cache and 8 GB RAM
- Synthetic datasets
  - Use zipf distribution with skew 0; 1;2.
  - Sorted and non-sorted. For sorted use gray-code sorting
- Real datasets
  - Kddcup, Berkeley Earth

Effects of lambda

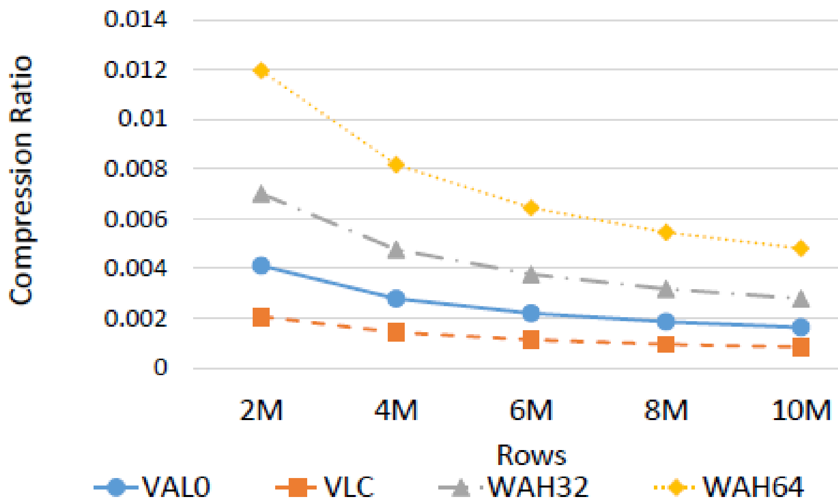


(a) Synthetic Data (zipf2) - Sorted: Compression Ratio

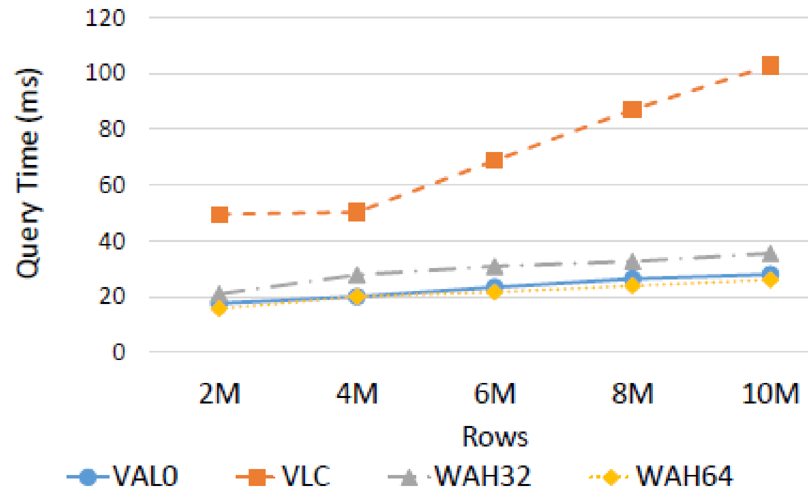


(b) Synthetic Data (zipf2) - Sorted: Query Time

Segment alignment

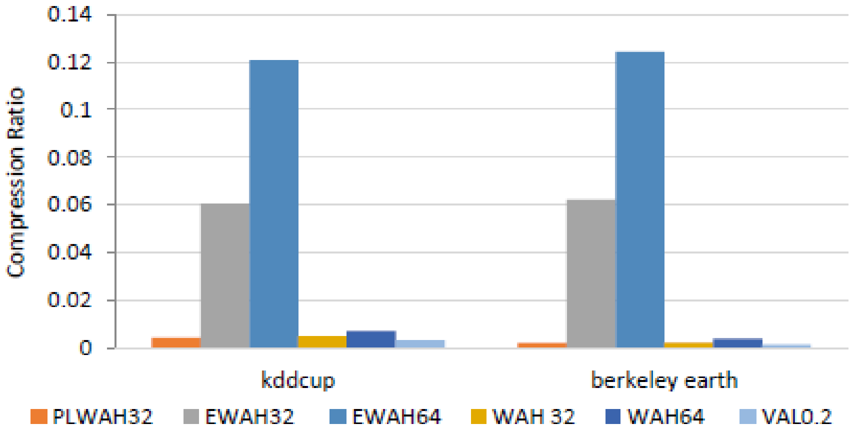


(a) Synthetic Data (`zipf2`) - Sorted: Compression Ratio

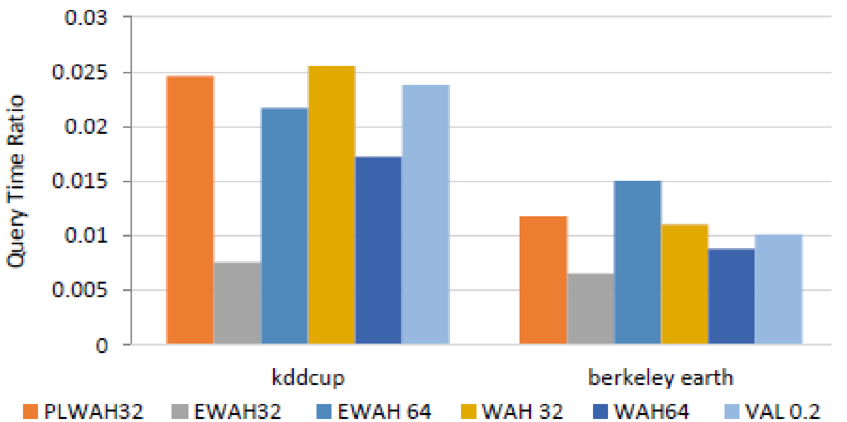


(b) Synthetic Data (`zipf2`) - Sorted: Query Time

Real Data



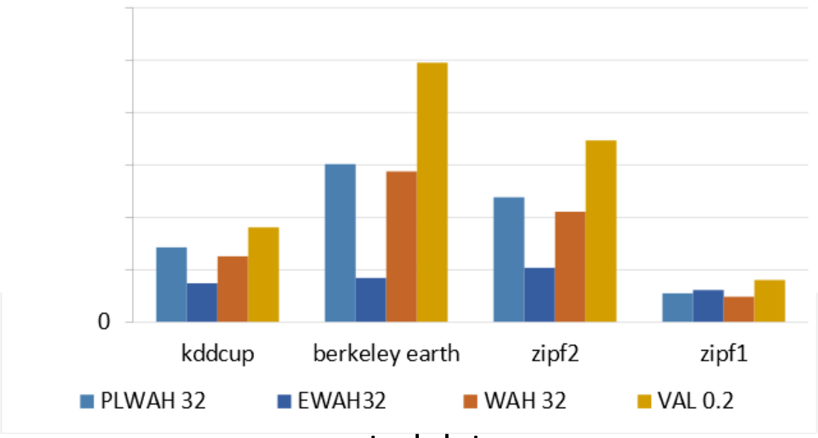
(a) Compression Ratio: Real Data



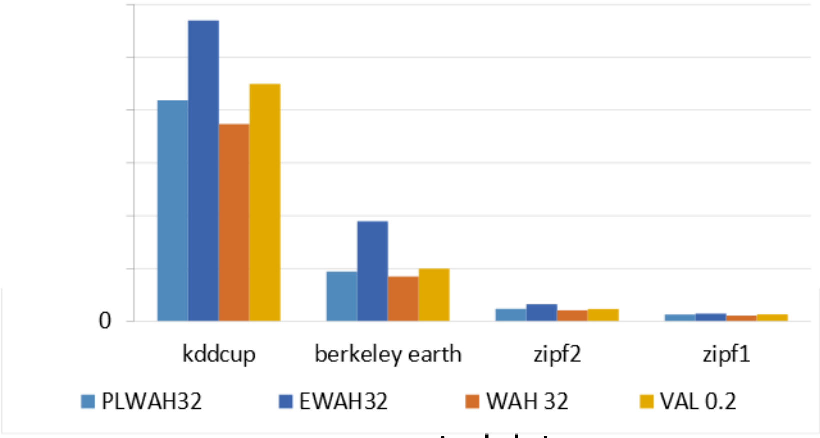
(b) Query Time Ratio: Real Data

Combined gain

- To help simplify discussion on trade-off:
    - $gain = \frac{1}{H_m} = \frac{queryTimeRatio + compRatio}{2 \times queryTimeRatio \times compRatio}$



sorted data



non-sorted data

Summary

- We developed the VAL framework
  - Uses variable segment lengths for better compression
  - Maintains alignment for faster query processing
  - Introduces a tuning parameter to adjust trade-offs
- Future work
  - Implement other word-aligned encodings within VAL
  - Develop algorithms for selecting encoding methods