On optimally partitioning a text to improve its compression

(hidden inside: speeding up dynamic programming solutions via approximation)

Rossano Venturini
Joint work with Paolo Ferragina and Igor Nitto

European Symposium on Algorithms, 2009
Denote with $C$ a generic lossless compression algorithm

**Definition**

If $P$ is a partition of input text $T$ into contiguous substrings, say $T = T_1 T_2 \cdots T_k$, define its compression cost as:

$$\text{Cost}(P) = \sum_{i=1}^{k} |C(T_i)|$$
On optimally partitioning a text to improve its compression

Introduction

Optimal Text Partitioning

Denote with $C$ a generic *lossless compression algorithm*

**Definition**

If $P$ is a partition of input text $T$ into contiguous substrings, say $T = T_1 T_2 \cdots T_k$, define its *compression cost* as:

$$\text{Cost}(P) = \sum_{i=1}^{k} |C(T_i)|$$

**Problem (Optimal Partitioning)**

*Find a partition $P_{\text{opt}}$ of $T$ into contiguous substrings with minimum compression cost:*

$$\text{Cost}(P_{\text{opt}}) = \min_P \text{Cost}(P)$$
Optimal Partitioning is solvable through DP in $O(n^3)$ time:

- Need to run $C$ over $\Theta(n^2)$ substrings of average size $\Theta(n)$
On optimally partitioning a text to improve its compression

Introduction

Simplified Optimal Partitioning

Optimal Partitioning is solvable through DP in $O(n^3)$ time:

- Need to run $\mathcal{C}$ over $\Theta(n^2)$ substrings of average size $\Theta(n)$

To achieve an efficient solution we relax the original problem in 2 ways:

1. Replace exact compress size $|\mathcal{C}(T_i)|$ with entropy-based estimate $\tilde{\mathcal{C}}(T_i)$ of $\mathcal{C}$’s compressed output
Simplified Optimal Partitioning

Optimal Partitioning is solvable through DP in $O(n^3)$ time:

- Need to run $C$ over $\Theta(n^2)$ substrings of average size $\Theta(n)$

To achieve an efficient solution we relax the original problem in 2 ways:

1. Replace exact compress size $|C(T_i)|$ with entropy-based estimate $\tilde{C}(T_i)$ of $C$'s compressed output

2. Admit $(1 + \epsilon)$-approximate solutions $\tilde{P}$ with:

$$\text{Cost}(\tilde{P}) \leq (1 + \epsilon)\text{Cost}(P_{opt})$$

where $\epsilon$ is a user-defined positive constant
Empirical Entropy: 0-th order

Introduced for the analysis of text compressors [Manzini ’99] (very similar to information-theoretic entropy):

Definition (0-th order empirical entropy)

\[
H_0(T) = \sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n}{n_c}
\]

where \( n_c \) is the number of occurrences of \( c \) in \( T \).

Provides bounds on \( |C_0(T)| \) for any 0-order compressor, like Arithmetic or Huffman:

\[
n H_0(T) \leq |C_0(T)| \leq n H_0(T) + f_C(n, |\Sigma|) = \tilde{C}_0(T)
\]

(e.g. when \( C = Arithmetic \) set \( f_C(n, |\Sigma|) = |\Sigma| \log n \))
Empirical Entropy: $k$-th order

Let us call context of a symbol of $T$ the sequence of $k$ symbols preceding it.

**Definition (k-th order empirical entropy [Manzini ’99])**

$$H_k(T) = \sum_{u \in \Sigma^k} \frac{|u_T|}{n} H_0(u_T)$$

$u_T :=$ string of all symbols of $T$ having $u$ as context.

Quantifies how many bits we need to encode a symbol given its context of size $k$.

Provides bounds on output size of higher-order compressors:

$$n H_k(T) \leq |C(T)| \leq n H_k(T) + f_k^C(n, |\Sigma|) = \tilde{C}(T)$$
Advantages of Optimal Partitioning

Why partitioning before compression?
Advantages of Optimal Partitioning

Why partitioning before compression?

**Example**

Let $C$ be a 0-order compressor (e.g. Arithmetic) and let the input text be $T = 0^{n/2}1^{n/2}$. Then

$$|C(T)| = nH_0(T) + O(\log n) = n + O(\log n)$$

while

$$|C(0^{n/2})| = |C(1^{n/2})| = O(\log n)$$

Similar examples exist for higher order compressors.
(Again) Reduction to shortest-path computation

*Optimal Partitioning* reduces to a shortest path computation over DAG $G(T)$:

- A vertex $v_i$ for each text position $i$ of $T$
- An edge $(v_i, v_j)$ with $i < j$ with assigned cost $c(v_i, v_j) = |\tilde{C}(T[i, j - 1])|$

(Again) There is a 1-to-1 correspondence between $v_1$-to-$v_{n+1}$ path in $G(T)$ and partitions of $T$:

- Each edge $(v_i, v_j)$ represents substring $T[i, j - 1]$.

(Again) Number of edges in $G(T)$ is $\Theta(n^2)$
On optimally partitioning a text to improve its compression

The graph $\mathcal{G}(T)$

Can we use previous speed up (pruning strategy)?
Can we use previous speed up (pruning strategy)?

Edges costs outgoing any vertex \( v_i \) must be increasing.
Can we use previous speed up (pruning strategy)?

Edges costs outgoing any vertex $v_i$ must be increasing.

- OK! For any vertex $v_i$:
  \[ 0 < c(v_i, v_{i+1}) \leq c(v_i, v_{i+2}) \leq \ldots \leq c(v_i, v_{n+1}) \]
Can we use previous speed up (pruning strategy)?

Edges costs outgoing any vertex \( v_i \) must be increasing.

- OK! For any vertex \( v_i \):
  \[
  0 < c(v_i, v_{i+1}) \leq c(v_i, v_{i+2}) \leq \ldots \leq c(v_i, v_{n+1})
  \]

How many distinct costs outgoing from any vertex?

- Too much! \( n \) distinct costs in worst case.

Previous solution requires \( O(n \cdot n) \) time!
On optimally partitioning a text to improve its compression

The graph $G(T)$

**Idea!**

Let’s force the number of distinct costs to be (arbitrarily) small!
Idea!

Let’s force the number of distinct costs to be (arbitrarily) small!

Denote by $G_\epsilon(T)$ the DAG obtained from $G(T)$ by modifying its edges costs.

- Cost of $(v_i, v_j)$ becomes $(1 + \epsilon)^t$ iff
  $(1 + \epsilon)^{t-1} < c(v_i, v_j) \leq (1 + \epsilon)^t$
Idea!

Let's force the number of distinct costs to be (arbitrarily) small!

Denote by $G_\epsilon(T)$ the DAG obtained from $G(T)$ by modifying its edges costs.

- Cost of $(v_i, v_j)$ becomes $(1 + \epsilon)^t$ iff
  \[(1 + \epsilon)^{t-1} < c(v_i, v_j) \leq (1 + \epsilon)^t\]

How many distinct costs outgoing from any vertex?
On optimally partitioning a text to improve its compression

The graph $G(T)$

Idea!

Let’s force the number of distinct costs to be (arbitrarily) small!

Denote by $G_\epsilon(T)$ the DAG obtained from $G(T)$ by modifying its edges costs.

- Cost of $(v_i, v_j)$ becomes $(1 + \epsilon)^t$ iff
  $$(1 + \epsilon)^{t-1} < c(v_i, v_j) \leq (1 + \epsilon)^t$$

How many distinct costs outgoing from any vertex?

- Just $O(\log_{1+\epsilon} n)$ distinct costs in worst case. Maximum cost was smaller than $n \log \sigma$ bits.
Idea!

Let’s force the number of distinct costs to be (arbitrarily) small!

Denote by $G_\epsilon(T)$ the DAG obtained from $G(T)$ by modifying its edges costs.

- Cost of $(v_i, v_j)$ becomes $(1 + \epsilon)^t$ iff
  $$(1 + \epsilon)^{t-1} < c(v_i, v_j) \leq (1 + \epsilon)^t$$

How many distinct costs outgoing from any vertex?

- Just $O(\log_{1+\epsilon} n)$ distinct costs in worst case. Maximum cost was smaller than $n \log \sigma$ bits.

Thus, shortest path in $G_\epsilon(T)$ can be computed in $O(n \log_{1+\epsilon} n)$ time (using previous solution).
The graph $G(T)$ is good enough

Theorem (Theorem)

$$E_{G(T)}[n + 1] \leq E_{G_\epsilon(T)}[n + 1] \leq (1 + \epsilon) E_{G(T)}[n + 1]$$

where $E_G[j]$ denotes shortest path distance in graph $G$ from $v_1$ to $v_j$
Proof.

By induction.
Proof.

By induction.

\[ E_{G_\epsilon(T)}[1] \leq (1 + \epsilon) \ E_g(T)[1] \]
Proof.

By induction.

- \( E_{G_\epsilon}(T)[1] \leq (1 + \epsilon) \ E_g(T)[1] \)
- Let us assume that \( E_{G_\epsilon}(T)[i] \leq (1 + \epsilon) \ E_g(T)[i] \) is true for any \( i < j \).
Proof.

By induction.

- \( E_{G_\epsilon(T)}[1] \leq (1 + \epsilon) \ E_{G(T)}[1] \)
- Let us assume that \( E_{G_\epsilon(T)}[i] \leq (1 + \epsilon) \ E_{G(T)}[i] \) is true for any \( i < j \).

We want to prove \( E_{G_\epsilon(T)}[j] \leq (1 + \epsilon) \ E_{G(T)}[j] \).
Proof.

By induction.

- \( E_{G_\epsilon(T)}[1] \leq (1 + \epsilon) \ E_{G(T)}[1] \)
- Let us assume that \( E_{G_\epsilon(T)}[i] \leq (1 + \epsilon) \ E_{G(T)}[i] \) is true for any \( i < j \).
  We want to prove \( E_{G_\epsilon(T)}[j] \leq (1 + \epsilon) \ E_{G(T)}[j] \).
  Let \((v_k, v_j)\) be the edge used to reach \(v_j\) in the SP of \(G(T)\).
Proof.

By induction.

- \( E_{G\epsilon}(T)[1] \leq (1 + \epsilon) E_{G(T)}[1] \)
- Let us assume that \( E_{G\epsilon}(T)[i] \leq (1 + \epsilon) E_{G(T)}[i] \) is true for any \( i < j \).

We want to prove \( E_{G\epsilon}(T)[j] \leq (1 + \epsilon) E_{G(T)}[j] \).

Let \((v_k, v_j)\) be the edge used to reach \( v_j \) in the SP of \( G(T) \).

We know that \( c_{\epsilon}(v_k, v_j) \leq (1 + \epsilon)c(v_k, v_j) \).
Proof.

By induction.

- $E_{G\epsilon(T)}[1] \leq (1 + \epsilon) E_{G(T)}[1]$
- Let us assume that $E_{G\epsilon(T)}[i] \leq (1 + \epsilon) E_{G(T)}[i]$ is true for any $i < j$.
  
We want to prove $E_{G\epsilon(T)}[j] \leq (1 + \epsilon) E_{G(T)}[j]$.
Let $(v_k, v_j)$ be the edge used to reach $v_j$ in the SP of $G(T)$.
We know that $c_{\epsilon}(v_k, v_j) \leq (1 + \epsilon)c(v_k, v_j)$.
Thus,

$E_{G\epsilon(T)}[j] \leq E_{G\epsilon(T)}[k] + c_{\epsilon}(v_k, v_j)$
Proof.

By induction.

- \( E_{G_\epsilon(T)}[1] \leq (1 + \epsilon) \ E_{G(T)}[1] \)

- Let us assume that \( E_{G_\epsilon(T)}[i] \leq (1 + \epsilon) \ E_{G(T)}[i] \) is true for any \( i < j \).

We want to prove \( E_{G_\epsilon(T)}[j] \leq (1 + \epsilon) \ E_{G(T)}[j] \).

Let \((v_k, v_j)\) be the edge used to reach \( v_j \) in the SP of \( G(T) \).

We know that \( c_\epsilon(v_k, v_j) \leq (1 + \epsilon)c(v_k, v_j) \).

Thus,

\[
E_{G_\epsilon(T)}[j] \leq E_{G_\epsilon(T)}[k] + c_\epsilon(v_k, v_j) \\
\leq (1 + \epsilon) \ E_{G(T)}[k] + (1 + \epsilon) \ c(v_k, v_j) = (1 + \epsilon) \ E_{G(T)}[j]
\]
Shortest Path in $G_{\epsilon}(T)$

We can find a $(1 + \epsilon)$-approximate optimal partition of $T$ by computing a $v_1$-to-$v_{n+1}$ shortest path in $G_{\epsilon}(T)$.

- This takes $O(n \log_{1+\epsilon} n)$ time with previous (exact) solution
Shortest Path in \( G_\epsilon(T) \)

We can find a \((1 + \epsilon)\)-approximate optimal partition of \( T \) by computing a \( v_1\)-to-\( v_{n+1} \) shortest path in \( G_\epsilon(T) \).

- This takes \( O(n \log_{1+\epsilon} n) \) time with previous (exact) solution.

However, generating \( G_\epsilon(T) \) is not a trivial task. We cannot:

- Materialize \( G_\epsilon(T) \) all-at-once would require super-linear storage space.
- Scan \( T[i, j - 1] \) to compute \( c_\epsilon(v_i, v_j) \) would require \( \Theta(n^3) \) time.
On optimally partitioning a text to improve its compression

The graph $G(T)$

**Shortest Path in $G_\epsilon(T)$**

We can find a $(1 + \epsilon)$-approximate optimal partition of $T$ by computing a $v_1$-to-$v_{n+1}$ shortest path in $G_\epsilon(T)$.

- This takes $O(n \log_{1+\epsilon} n)$ time with previous (exact) solution.

However, generating $G_\epsilon(T)$ is not a trivial task. We cannot:

- Materialize $G_\epsilon(T)$ all-at-once would require super-linear storage space.
- Scan $T[i, j - 1]$ to compute $c_\epsilon(v_i, v_j)$ would require $\Theta(n^3)$ time.

Solution:

We dynamically generate $G_\epsilon(T)$ as vertices are examined during shortest-path computation. Each cost is computed in $O(1)$ amortized time.
Theorem

We can find an \((1 + \epsilon)\)-optimal partition of \(T\) with respect to a \(k\)-th order compressor in \(O(n \log_{1+\epsilon} n)\) time and \(O(n)\) space, where \(\epsilon\) is any user-defined positive constant.
On optimally partitioning a text to improve its compression

Summarizing

Dynamic Programming speed ups

\[ E[j] = \min_{i<j} (E[i] + c(i, j)) \]

Assume you can compute any \( c() \) in \( O(1) \).

If \( c(i, j) \leq c(i, j + 1) \) for any \( i, j \) \((i < j)\), an \((1 + \epsilon)\) approximation of \( E[n] \) can be computed in \( O(n \log_{1+\epsilon} M) \) time and \( O(n) \) space, where \( M \) is the maximum cost and \( \epsilon \) is a user-defined parameter.
Dynamic Programming speed ups

\[ E[j] = \min_{i<j} (E[i] + c(i, j)) \]

Assume you can compute any \( c() \) in \( O(1) \).

- If \( c(i, j) \leq c(i, j + 1) \) for any \( i, j \) \( (i < j) \), an \( (1 + \epsilon) \) approximation of \( E[n] \) can be computed in \( O(n \log_{1+\epsilon} M) \) time and \( O(n) \) space, where \( M \) is the maximum cost and \( \epsilon \) is a user-defined parameter.

Any problem?
Thank you!