Efficient Parameterized Algorithms for Data Packing

Krishnendu Chatterjee, Amir Goharshady, Nastaran Okati, Andreas Pavlogiannis

January 22, 2019
Overview

- Data Packing is a classical problem in Cache Management
Overview

- Data Packing is a classical problem in Cache Management
- NP-hard and Hard-to-approximate $\implies$ Heuristics are used
Overview

- Data Packing is a classical problem in Cache Management
- NP-hard and Hard-to-approximate → Heuristics are used
- The heuristics provide no guarantee of optimality
Overview

- Data Packing is a classical problem in Cache Management
- NP-hard and Hard-to-approximate $\rightarrow$ Heuristics are used
- The heuristics provide no guarantee of optimality

- We reduce Data Packing to a graph problem
Overview

- Data Packing is a classical problem in Cache Management
- NP-hard and Hard-to-approximate → Heuristics are used
- The heuristics provide no guarantee of optimality

- We reduce Data Packing to a graph problem
- We show that this problem can be solved in linear time if the underlying graph has a specific structural property
Overview

- Data Packing is a classical problem in Cache Management
- NP-hard and Hard-to-approximate $\rightarrow$ Heuristics are used
- The heuristics provide no guarantee of optimality

- We reduce Data Packing to a graph problem
- We show that this problem can be solved in linear time if the underlying graph has a specific structural property
- We experimentally show that graphs obtained from many common algorithms have this property
Overview

- Data Packing is a classical problem in Cache Management
- NP-hard and Hard-to-approximate → Heuristics are used
- The heuristics provide no guarantee of optimality

- We reduce Data Packing to a graph problem
- We show that this problem can be solved in linear time if the underlying graph has a specific structural property
- We experimentally show that graphs obtained from many common algorithms have this property
- → We provide the first positive theoretical result for Data Packing
The Setting

- A two-level memory system with:
  - large but slow main memory
  - small but fast cache
- The cache can hold up to $m$ blocks (pages)
- Each block can hold up to $p$ data items
- When accessing a data item, its block must be in the cache
The Goal

- A sequence $R$ of accesses to data elements is given
- The goal is to minimize cache misses over $R$
- $N := |R|$, $n :=$ number of distinct data items
Two Distinct Problems

- Minimizing cache misses can be broken in two parts:
  - **Paging**: Choosing which block to evict from the cache when a cache-miss occurs (**LRU**, **FIFO**, etc.)
  - **Data Packing**: Choosing a data placement scheme, i.e. choosing how to divide the data items into blocks and which data items to put together

- Data Packing is the focus of this work
Access Graph

\[ R = \langle a, b, c, a, b, b, d, b, d, e, c, b, f \rangle \]
$R = \langle a, b, c, a, b, b, d, b, d, e, c, b, f \rangle$
Access Hypergraph

\[ R = \langle a, b, c, a, b, b, d, b, d, e, c, b, f \rangle \]
Nice Tree Decompositions
**Previous Results**

**Theorem (Lavaee, POPL 2016)**

Assuming either LRU or FIFO as the replacement policy, we have the following hardness results:

- **For any** $m$ **and any** $p \geq 3$, **Data Packing** is NP-hard.
- **Unless** $P=NP$, **for any** $m \geq 5$, $p \geq 2$ **and any constant** $\epsilon > 0$, there is no polynomial algorithm that can approximate the **Data Packing** problem within a factor of $O(N^{1-\epsilon})$. 
Our Results

- **Theorem 2.1**: NP-hard
  - Linear-time: $q = \lfloor \frac{m - 1}{p} + 2 \rfloor$
  - Hard to Approximate: $q = \lfloor \frac{m - 5}{p} + 1 \rfloor$

Graph:
- x-axis: $m$
- y-axis: $q$
- Blue area: Linear-time Theorem 4.1
- Green area: NP-hard Theorem 4.2
- Red area: Hard to Approximate Theorem 4.3
- Yellow area: NP-hard Theorem 2.1
- Orange area: Hard to Approximate Theorem 2.1
Minimum-weight $p$-partitioning

$m = 1, p = 2$

$R = \langle a, b, c, a, b, b, d, b, d, e, c, b, f \rangle$

Cross edge: An edge that goes from one partition to another
States

- Let $G = (V, E)$ and $A \subseteq V$. A state over $A$ is a pair $(\varphi, sz)$ where:
  - $\varphi$ is a partitioning of $A$ in which every equivalence class has a size of at most $p$
  - $sz$ is a size enlargement function $sz : A/\varphi \rightarrow \{0, \ldots, p - 1\}$ that maps each equivalence class $[v]_\varphi$ to a number which is at most $p - |[v]_\varphi|$
**States**

**Realization.** We say that a $p$-partitioning $\psi$ realizes the state $s = (\varphi, sz)$ over $A$, if

- $\psi$ partitions the vertices in $A$ in the same manner as $\varphi$
- if a partition $[v]_\psi$ of $\psi$ intersects $A$, then $[v]_\psi$ contains as many vertices from outside of $A$ as fixed by $sz$.

**Compatibility.** Two states are compatible iff there exists a $p$-partitioning that realizes both of them.
**The Algorithm**

**Step 0: Initialization.** We define several variables at each node of our tree decomposition $T$. For every $t \in T$ and every state $s$ over the boundary $X_t$, we define a variable $dp[t, s]$ and initialize it to $\infty$.

Invariant. $dp[t, s] = \text{The minimum total weight of cross edges over all } p\text{-partitionings of } G_t \text{ that realize } s$. 
The Algorithm

Step 1: Computation of $dp$. The $dp$ variables are computed in a bottom-up order. Each $dp$ value at a tree node $t$ can be computed based on the $dp$ values at its children.

- If $t$ is a Leaf: $dp[t, s] = 0$;
- If $t$ is a Join node with children $t_1$ and $t_2$:
  \[
dp[t, s] = \min_{sz_1 + sz_2 = sz} \dp[t_1, (\varphi, sz_1)] + \dp[t_2, (\varphi, sz_2)];
\]
- If $t$ is an Introduce Vertex node, introducing $v$, with a single child $t_1$:
  \[
dp[t, s] = \dp[t_1, (\varphi|X_{t_1}, sz|X_{t_1})];
\]
- If $t$ is an Introduce Edge node, introducing $e$, with a single child $t_1$:
  \[
dp[t, s] = \dp[t_1, s] + w(e, \varphi),
\]
  where $w(e, \varphi)$ is equal to $w(e)$ if $e$ is a cross edge in $\varphi$ and 0 otherwise;
- If $t$ is a Forget Vertex node, forgetting $v$, with a single child $t_1$:
  \[
dp[t, s] = \min_{s' \equiv_s} \dp[t_1, s'].
\]
The Algorithm: Introduce Edge Nodes

\[ dp[t, s] = dp[t_1, s] + w(e, \varphi) \]
The Algorithm: Join Nodes

\[ dp[t, (\varphi, sz)] = \min_{sz_1 + sz_2 = sz} dp[t_1, (\varphi, sz_1)] + dp[t_2, (\varphi, sz_2)] \]
The Algorithm

Step 2: Computing the Output. The algorithm computes and return the following output:

$$\min_{s \in S_{X_r}} dp[r, s].$$
Theorem

If the access graph has constant treewidth, then Data Packing can be solved in linear time.
If the access graph has constant treewidth, then Data Packing can be solved in linear time.
Experimental Results

- Inner-product of two vectors
- Computation of Fibonacci Numbers
- Insertion Sort
- Random Insertions in a Heap
- Random Binary Searches on a Sorted Array
- Closest Pair of Points in 2D
## Comparison of the Number of Cache Misses

<table>
<thead>
<tr>
<th></th>
<th>Linear Algebra</th>
<th>Sorting</th>
<th>Dynamic Programming</th>
<th>Recursion</th>
<th>String Matching</th>
<th>Computational Geometry</th>
<th>Trees</th>
<th>Sorted Arrays</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>CCDP</td>
<td>129.12</td>
<td>114.65</td>
<td>113.24</td>
<td>128.57</td>
<td>115.04</td>
<td>135.27</td>
<td>136.16</td>
<td>136.8</td>
<td>122.1</td>
</tr>
<tr>
<td>CPACK</td>
<td>138.77</td>
<td>106.95</td>
<td>110.62</td>
<td>124.02</td>
<td>114.4</td>
<td>140.31</td>
<td>123.04</td>
<td>127.75</td>
<td>124.61</td>
</tr>
<tr>
<td>CPACK+/GPart+/Capri+</td>
<td>139.07</td>
<td>148.78</td>
<td>121.9</td>
<td>139.13</td>
<td>101.87</td>
<td>135.01</td>
<td>117.38</td>
<td>136.08</td>
<td>119.78</td>
</tr>
<tr>
<td>Sampling</td>
<td>106.28</td>
<td>152.71</td>
<td>118.95</td>
<td>175.78</td>
<td>115.4</td>
<td>128.06</td>
<td>142</td>
<td>154.69</td>
<td>115.2</td>
</tr>
<tr>
<td>k-Distance</td>
<td>146.32</td>
<td>170.54</td>
<td>122.88</td>
<td>167.49</td>
<td>114.55</td>
<td>131.12</td>
<td>143.15</td>
<td>161.18</td>
<td>131.23</td>
</tr>
</tbody>
</table>

![Bar Chart](image)
Conclusion