A Derivation of Dual Formulation (5)

Let C denote the feasible set in problem (2). We have $d_i(x_i) = \max_{y_i} \{\langle y_i, x_i\rangle - d_i^*(y_i)\}$. We can rewrite problem (2) as

$$\begin{aligned}\min_{x\in C} \sum_{i=1}^n d_i(x_i) &= \min_{x\in C} \left\{\sum_{i=1}^n \max_{y_i} \{y_i x_i - d_i^*(y_i)\}\right\} \\ &= \min_{x\in C} \max_{y} \left\{\sum_{i=1}^n (y_i x_i - d_i^*(x_i^*))\right\} \\ &= \max_{y} \min_{x\in C} \left\{\sum_{i=1}^n (y_i x_i - d_i^*(y_i))\right\} \\ &= \max_{y} \left\{\sum_{i=1}^n (-d_i^*(y_i)) + \min_{x\in C} \langle y, x\rangle\right\}.\end{aligned}$$

Let us focus on the $\min_{x\in C}\langle y, x\rangle$ term. If we let $Y_i = y_i - y_{i+1}$ for $i \in [n-1]$ and $Y_n = y_n$, we have $y_i = \sum_{l=i}^n Y_k$. This gives us

$$\begin{aligned}\langle y, x\rangle &= \langle y, c\rangle + \langle y, x - c\rangle \\ &= \langle y, c\rangle + \sum_{i=1}^n y_i(x_i - c_i) \\ &= \langle y, c\rangle + \sum_{i=1}^n \left(\sum_{k=i}^n Y_k\right)(x_i - c_i) \\ &= \langle y, c\rangle + \sum_{k=1}^n \left(\sum_{i=1}^k (x_i - c_i)\right) Y_k.\end{aligned}$$

If any Y_k is larger than 0 for any $k \in [n-1]$, then $\inf_x\langle y, x\rangle = -\infty$; we can set $x_i = c_i$ for $i \notin \{k, k+1\}$, $x_k \to -\infty$ and $x_{k+1} = c_k + c_{k+1} - x_k$. This means that we require $Y_k \le 0$ for all k (i.e. $y_{i+1} \ge y_i$). So $\min_x\langle y, x\rangle = \langle y, c\rangle$, obtained by setting $x_i = c_i$ for all i.

B Proofs Omitted From Main Paper

B.1 Section 3 Proofs

Lemma 3.1. *(Suehiro et al., 2012) Let x' be the projection of z onto the permutahedron under a uniformly separable Bregman divergence ϕ. Suppose $z_1 \ge z_2 \ge \ldots \ge z_n$. Then, we have $x'_1 \ge x'_2 \ge \ldots \ge x^*_n$.*

Proof. Let x be a point in the permutahedron where $z_i > z_j$ but $x_i < x_j$. The difference between the objective obtained by swapping the points x_i and x_j is given by:

$$\begin{aligned}&\Delta_\phi(x_i, z_i) + \Delta_\phi(x_j, z_j) - \Delta_\phi(x_i, z_j) - \Delta_\phi(x_j, z_i) \\ =& -\nabla\phi(z_i)(x_i - x_j) - \nabla\phi(z_j)(x_j - x_i) \\ =& -(\nabla\phi(z_i) - \nabla\phi(z_j))(x_i - x_j) > 0,\end{aligned}$$

so swapping the terms decreases the objective further and x is not the projection of z. □

Lemma 3.2. *Let x' be the projection of z onto the permutahedron under a uniformly separable Bregman divergence defined by a sign-invariant ϕ. Then $\operatorname{sgn}(x'_i) = \operatorname{sgn}(z_i)$ for all i. Furthermore, if $|z_1| \ge |z_2| \ge \ldots \ge |z_n|$, we have $|x'_1| \ge |x'_2| \ge \ldots \ge |x^*_n|$.*

Proof. We will show that if $\operatorname{sgn}(x_i) \neq \operatorname{sgn}(z_i)$ we can improve the objective by setting x_i to 0, implying x_i is not optimal. By the sign-invariance of ϕ, we have $\nabla\phi(u) = -\nabla\phi(-u)$, which means $\nabla\phi(0) = 0$. By the strict convexity of ϕ, we know $\operatorname{sgn}(\nabla\phi(z_i))$ is an increasing functions, so $\operatorname{sgn}(\nabla\phi(z_i)) = \operatorname{sgn}(z_i)$. The change in objective after swapping is

$$\begin{aligned}&\Delta_\phi(x_i, z_i) - \Delta_\phi(0, z_i) \\ =& \phi(x_i) - \phi(0) - \nabla\phi(z_i)(x_i - 0) \\ \ge& \phi(x_i) - \phi(0) - \nabla\phi(0)(x_i - 0) > 0,\end{aligned}$$

where the last line follows from strict convexity of ϕ.

The proof of the second part is similar to the proof of lemma 3.1. □

Theorem 3.3. *Let $y^A \in \mathbb{R}^n$ be an optimal solution to problem (5). We get an optimal solution to problem (6) by truncating the positive values of y^A to zero.*

Proof. We will first show that there is an optimal solution y^B to problem (6) such that if $y_i^A > 0$, then $y_i^B = 0$. Let y^C be any optimal solution to problem (6) and let S be the set of indices i where $y_i^A > 0$ and $y_i^C < 0$. Suppose S is nonempty. By the monotonicity of the y vectors, we know that S is an interval of indices $\{a, a+1, \ldots, b\}$, and $y_i^C = 0$ for $i > b$ and $y_i^A < 0$ for $i < a$. Let v denote the vector that is $y_i^A - y_i^C$ for $i \in S$ and 0 otherwise, and note that v is nonnegative. We will now compare $\sum_{i\in S} f_i(y_i^A)$ and $\sum_{i\in S} f_i(y_i^C)$.

- If $\sum_{i\in S} f_i(y_i^A) > \sum_{i\in S} f_i(y_i^C)$, then we can pick some $\epsilon > 0$ such that $y^A - \epsilon v$ is a valid solution for problem (5) that has a lower objective than y^A, a contradiction.
- If $\sum_{i\in S} f_i(y_i^A) < \sum_{i\in S} f_i(y_i^C)$, we get a similar contradiction to the optimality of y^C.

Hence, the two sums must be equal, and we can now pick $\delta > 0$ such that $y^C + \delta v$ has one less negative term. This reduces the size of set S by one, and we can repeat the process until we obtain a y^B where if $y_i^A > 0$, then $y_i^B = 0$.

We can now assume we have an optimal solution y^B to problem (6) such that if $y_i^B < 0$, then $y^A \le 0$. Let

k denote the largest index where $y_k^B < 0$. We can form two new vectors y^D and $y^E - y^D$ is y^A with all values truncated to be less than or equal to zero, and y_i^E is y_i^B for $i \le k$ and y_i^A for $i > k$. y^D and y^E are feasible for problem (6) and problem (5) respectively. If y^D is not optimal for problem (6) one can show that y^E has a lower objective value than y^A, contradicting the optimality of y^A for problem (5). □

B.2 Section 4 Proofs

Lemma 4.3. $\texttt{PoolV}_{\phi,z}(S)$ *satisfies*

$$\sum_{i\in S} (\nabla\phi)^{-1}(\gamma + \nabla\phi(z_i)) = \sum_{i\in S} c_i. \tag{9}$$

Proof. We will set the derivative of $\sum_{i\in S} f_i(\gamma)$ to zero:

$$\begin{aligned}
0 &= \nabla_\gamma \left(\sum_{i\in S} d_i^*(\gamma) - \gamma \sum_{i\in S} c_i \right) \\
&= \sum_{i\in S} \nabla_\gamma \left(d_i^*(\gamma)\right) - \sum_{i\in S} c_i \\
&= \sum_{i\in S} \nabla_\gamma \left(\gamma x_i' - \phi(x_i') + \nabla\phi(z_i) x_i'\right) - \sum_{i\in S} c_i \\
&= \sum_{i\in S} \left(x_i' + \gamma\nabla_\gamma(x_i') - \nabla\phi\left((\nabla\phi)^{-1}(\gamma + \nabla\phi(z_i))\right)\nabla_\gamma(x_i') \right. \\
&\quad \left. + \nabla\phi(z_i)\nabla_\gamma(x_i')\right) - \sum_{i\in S} c_i \\
&= \sum_{i\in S} x_i' - \sum_{i\in S} c_i \\
&= \sum_{i\in S} (\nabla\phi)^{-1}(\gamma + \nabla\phi(z_i)) - \sum_{i\in S} c_i,
\end{aligned}$$

yielding the desired equality. □

B.3 Section 5 Proofs

Lemma 5.3. *Consider adjacent intervals I_1, I_2 and vector y where y_{I_1} and y_{I_2} are the optimal solution to dual problem (5) when restricted to only the indices in I_1 and I_2 respectively. The output of $y_{I_1\cup I_2}$ of* $\texttt{Merge}_f(I_1, I_2, y_{I_1\cup I_2})$ *gives the optimal solution to problem (5) when restricted to the indices in $I_1 \cup I_2$.*

Proof. Suppose we have adjacent intervals I_1, I_2, and $y_{I_1\cup I_2}$ such that each of y_{I_1}, y_{I_2} is the optimal solution to problem (5) over just the I_1 indices and just the I_2 indices) respectively. We will show that the optimal solution to the problem (5) over the $I_1 \cup I_2$ indices can be obtained from $y_{I_1\cup I_2}$ via at most a single pooling operation, and that $\texttt{Merge}_{\{f_i\}}$ finds the right elements to pool. Note that throughout the proof, we will exploit the strict convexity of f_i, especially when we refer to Lemma 4.2.

If $y_{I_1.\text{end}} \le y_{I_2.\text{start}}$, then for any $\gamma_{\text{test}} < y_{I_1.\text{end}}$, S_{test} only contains indices i for $y_i \ge \gamma_{\text{test}}$. Since $I_1.\text{end} \in S_{\text{test}}$ and the value $y_{I_1.\text{end}}$ satisfies $\gamma_{\text{test}} < y_{I_1.\text{end}}$, we have $\texttt{PoolV}_f(S_{\text{test}}) > \gamma_{\text{test}}$ by Lemma 4.2. A similar fact holds for $\gamma_{\text{test}} > y_{I_1.\text{end}}$, and the $\texttt{Merge}_f$ algorithm terminates at $\gamma_{\text{test}} = y_{I_1.\text{end}}$ or $y_{I_2.\text{start}}$ and no elements are pooled together, as desired.

Now suppose $y_{I_1.\text{end}} > y_{I_2.\text{start}}$. We can prove the correctness of this algorithm by showing that the pooling choice the $\texttt{Merge}_f$ subroutine takes can be obtained by $\texttt{PAV}$ applied to just $I_1 \cup I_2$ given values $y_{I_1\cup I_2}$. Let S_{PAV} denote the elements of $I_1 \cup I_2$ pooled together by the $\texttt{PAV}$ algorithm when applied to $y_{I_1\cup I_2}$. S_{PAV} is an interval that is a subset of $\{i \in I_1 \mid y_i \ge \texttt{PoolV}_f(S_{PAV})\} \cup \{i \in I_2 \mid y_i \le \texttt{PoolV}_f(S_{PAV})\}$. At each iteration of the main loop in $\texttt{Merge}_f$, we will show that $\texttt{PoolV}_f(S_{PAV})$ is contained in $[\min(\mathcal{Y}), \max(\mathcal{Y})]$. This holds initially since Lemma 4.2 means that $\texttt{PoolV}_f(S_{PAV})$ must be between $\min_{i\in I_1\cup I_2} y_i$ and $\max_{i\in I_1\cup I_2} y_i$.

Suppose we have chosen $\gamma_{\text{test}} < \texttt{PoolV}_f(S_{PAV})$. We want to show that $\texttt{PoolV}_f(S_{\text{test}}) > \gamma_{\text{test}}$, which will mean that we make the correct choice of which half of $\mathcal{Y}$ to discard. If $\texttt{PoolV}_f(S_{\text{test}}) \ge \texttt{PoolV}_f(S_{PAV})$, we are done. Suppose $\texttt{PoolV}_f(S_{\text{test}}) < \texttt{PoolV}_f(S_{PAV})$. Then, we can define the following three consecutive intervals: $S_1 = S_{\text{test}} \setminus S_{PAV}$, $S_2 = S_{\text{test}} \cap S_{PAV}$, and $S_3 = S_{PAV} \setminus S_{\text{test}}$ such that $S_{\text{test}} = S_1 \cup S_2$ and $S_{PAV} = S_2 \cup S_3$. Figure 4 illustrates these sets.

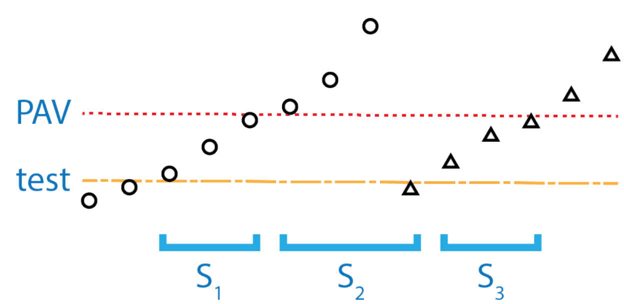

Figure 4: An example of S_1, S_2, and S_3 when $\gamma_{\text{test}} < \texttt{PoolV}_f(S_{PAV})$.

Note that every element in y_{S_1} is larger than or equal to γ_{test}, so Lemma 4.2 implies

$$\texttt{PoolV}_f(S_1) \ge \gamma_{\text{test}}. \tag{10}$$

A similar argument shows that $\texttt{PoolV}_f(S_3) \le \texttt{PoolV}_f(S_{PAV})$. As for $\texttt{PoolV}_f(S_2)$, Lemma 4.2 implies that $\texttt{PoolV}_f(S_2) \ge \texttt{PoolV}_f(S_{PAV})$ since $S_{PAV} = S_2 \cup S_3$. Hence, we have

$$\texttt{PoolV}_f(S_2) \ge \texttt{PoolV}_f(S_{PAV}) > \gamma_{\text{test}}. \tag{11}$$

By combining inequalities (10) and (11) and applying the lemma again, we get $\texttt{PoolV}_f(S_{\text{test}}) > \gamma_{\text{test}}$.

This shows that the correct half of elements are omitted from the search range in the next iteration of `Merge`. We can apply the same reasoning to $\gamma_{\text{test}} > \texttt{PoolV}_f(S_{PAV})$. Eventually, $\mathcal{Y}$ gets reduced until it has at most two elements, which leaves only three candidate sets of S_{test} to try out. □

Proposition 5.5. *The running time of* `MergeAndPool` *is $O(n \log n)$ for uniformly separable Bregman divergences ϕ with incremental* $\texttt{PoolV}_{\phi,z}$ *cost.*

Proof. `MergeAndPool` pairs off and merges pairs of intervals in each round, and there are $O(\log n)$ rounds in total. We will show that each call to $\texttt{Merge}_{\phi,z}$ takes $O(n)$ time.

We can find the $\lceil |\mathcal{Y}|/2 \rceil$th smallest value in two ordered sequences in $O(n)$. At each iteration, we halve the range we are selecting over, so the selection takes linear time in total. For the interval S_{test}, we half the number of elements are are changing at the ends of S_{test}. Since $\texttt{PoolV}_{\phi,z}$ can be computed incrementally, all the S_{test}-related work takes linear time in aggregate. □

Lemma 5.7. *We can sort the entries of vector z into d groups in $O(n \log d)$ time such that the ith group has n_i elements and for each z_j in group i and z_k in group $i+1$, we have $z_j \geq z_k$.*

Proof. We can apply a quicksort-like procedure where at each iteration we select the pivot that partitions the elements into two sets of roughly the same number of groups. There are $O(\log d)$ iterations and each iteration takes $O(n)$. □

Theorem 5.9. *We can compute the projection x' onto the permutahedron $\mathcal{PH}(c)$ under any incremental uniformly separable Bregman divergence in time $O(n \log d)$.*

Proof. We first show that running time of `MergeAndPool` when we provide a partition with d groups is $O(n \log d)$. There are $O(\log d)$ iterations of the outer loop in `MergeAndPool` with one call to $\texttt{Merge}_{\phi,z}$ for each pair of intervals in each iteration. We will show that each call to $\texttt{Merge}_{\phi,z}$ takes $O(|I_1| + |I_2|)$ time. Picking the $\lceil |\mathcal{Y}|/2 \rceil$th smallest value of $\mathcal{Y}$ can be done in linear time using the efficient selection algorithm. The construction of S, and computation of $\texttt{PoolV}_{\phi,z}$ can be done in linear time. After each iteration of the loop in $\texttt{Merge}_{\phi,z}$, the search space halves, so the amount of work required halves.

The correctness of the output follows directly from the fact that the $\texttt{Merge}_{\phi,z}$ subroutine will make the same choice of elements to pool together no matter how the vector $y_{I_1 \cup I_2}$ is permuted. In particular, this returns the same results as in the case where the indices are fully sorted. □

B.4 Section 6 Proofs

The proof of Theorem 6.3 follows directly from the next lemma, which is the ϵ-close analogue of Lemma 5.3.

Lemma B.1. *Consider adjacent intervals I_1, I_2, and let y' denote the vector where y'_{I_1} and y'_{I_2} are the solutions to problem (5) when restricted to only the indices in I_1 and I_2, respectively. The output of* $\texttt{Merge}_f(I_1, I_2, \mathcal{L}(y'))$ *is $\mathcal{L}(y'')$, where y'' is the solution to problem (5) when restricted to the indices in $I_1 \cup I_2$.*

Proof. Firstly, we note that Lemma 5.3 does not depend on whether the sets used to form S_{test} are created using an inequality or strict inequality. Secondly, that lemma demonstrates that given any γ_{test}, we can correctly determine if $\texttt{PoolV}_f(S_{PAV})$ is higher or lower using $\texttt{PoolV}_f(S_{\text{test}})$, and using the derivative in ϵ-$\texttt{Merge}_f$ has the same effect. Finally, we note that the sets S_{test} formed in ϵ-$\texttt{Merge}_f$ are the same regardless of whether we are given y' or $\mathcal{L}(y')$ as the input. Together, by using Lemma 5.3, these imply that ϵ-$\texttt{Merge}_f$ is able to correctly determine the two lattice points in $\{\epsilon k \mid k \in \mathbb{Z}\}$ that $\texttt{PoolV}_f(S_{PAV})$ is between. Once these two points are found, the algorithm rounds down $\texttt{PoolV}(S_{PAV})$, thereby obtaining $\mathcal{L}(y'')$. □

C Experiments on Scaling Effects of `MergeAndPool`

To show how `MergeAndPool` scales in practice and to demonstrate that the empirical performance of the algorithm aligns with the theory, we performed a set of simple experiments implemented in Julia 0.4.5. The results are shown in Figure 5.

D An Efficient Implementation of `PAV`

We now describe a linked-list based implementation of the `PAV` algorithm for solving the dual problem (5).

Algorithm 5 Pool Adjacent Violators Algorithm (`PAV`)

Input: strictly convex function $\phi_i : \mathbb{R} \to \mathbb{R}$, sorted $z \in \mathbb{R}$
{Initialize Algorithm}
$S_{\text{prev}} \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
  $S_{\text{curr}} \leftarrow \{i\}$
  $S_{\text{curr}}.\min \leftarrow \texttt{PoolV}_{\phi,z}(S_{\text{curr}})$
  Update pointers for S_{curr} and S_{prev}
  $S_{\text{prev}} \leftarrow S_{\text{curr}}$
end for
set after $\{n\} \leftarrow \emptyset$

{Main Loop}
$S_{\text{prev}} = \emptyset$, $S_{\text{curr}} = \{1\}$, $S_{\text{next}} = \{2\}$
while $S_{\text{next}} \neq \emptyset$ **do**
  if $S_{\text{curr}}.\min > S_{\text{next}}.\min$ **then**
    $S_{\text{curr}} \leftarrow (S_{\text{curr}} \cup S_{\text{next}})$ and update pointers
    $S_{\text{curr}}.\min \leftarrow \texttt{PoolV}_{\phi,z}(S_{\text{curr}})$
    $S_{\text{next}} \leftarrow$ set after S_{curr}
    while $S_{\text{prev}} \neq \emptyset$ and $S_{\text{prev}}.\min > S_{\text{curr}}.\min$ **do**
      $S_{\text{curr}} \leftarrow (S_{\text{prev}} \cup S_{\text{curr}})$ and update pointers
      $S_{\text{curr}}.\min \leftarrow \texttt{PoolV}_{\phi,z}(S_{\text{curr}})$
      $S_{\text{prev}} \leftarrow$ set before S_{curr}
    end while
  end if
  $S_{\text{prev}} \leftarrow S_{\text{curr}}$, $S_{\text{curr}} \leftarrow S_{\text{next}}$, $S_{\text{next}} \leftarrow$ set after S_{next}
end while

{Output Solution}
while $S_{\text{curr}} \neq \emptyset$ **do**
  for $i \in S_{\text{curr}}$ **do**
    $y_i \leftarrow S_{\text{curr}}.\min$
  end for
  $S_{\text{curr}} \leftarrow$ set before S_{curr}
end while
return y

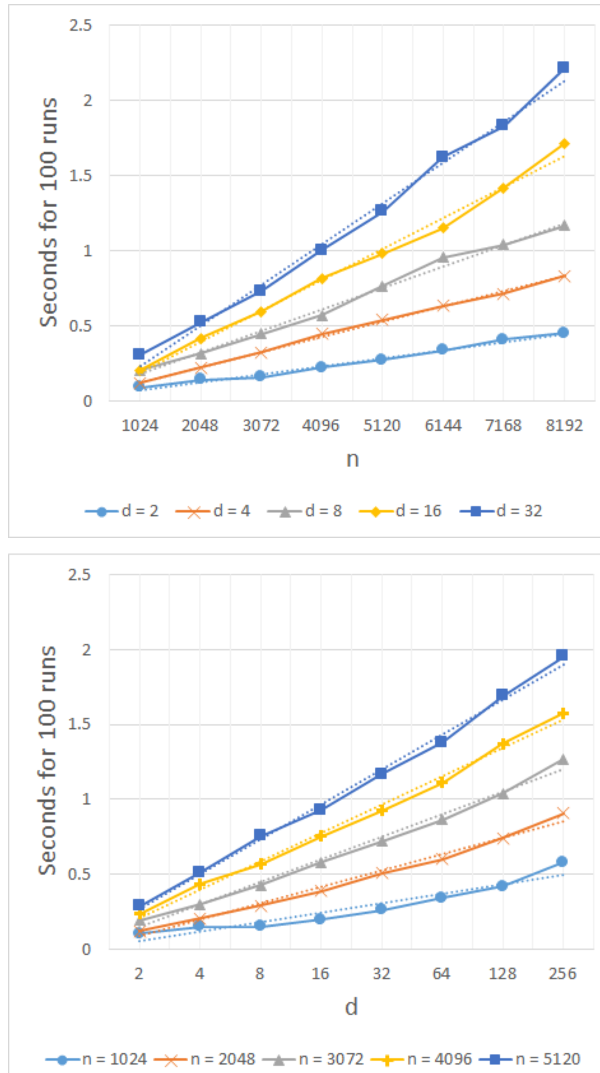

Figure 5: Running times of `MergeAndPool` when varying n and initial number of intervals d. The first graph varies n along the x axis, while the second has d (log-scale) along that axis. The complexity scales linearly with n and $\log d$.