Symmetric Locality: Definition and Initial Results

Giordan Escalona
Department of Computer Science
University of Rochester
Rochester, NY
gescalo2@u.rochester.edu

Dylan McKellips
Department of Computer Science
University of Rochester
Rochester, NY
dmckelli@u.rochester.edu

Chen Ding
Department of Computer Science
University of Rochester
Rochester, NY
cding@cs.rochester.edu

***Abstract*—In this paper, we characterize *symmetric locality*. In designing algorithms, compilers, and systems, data movement is a common bottleneck in high-performance computation, in which we improve cache and memory performance. We study a special type of data reuse in the form of repeated traversals, or re-traversals, which are based on the symmetric group. The cyclic and sawtooth traces are previously known results in symmetric locality, and in this work, we would like to generalize this result for any re-traversal. Then, we also provide an abstract framework for applications in compiler design and machine learning models to improve the memory performance of certain programs.**

***Index Terms*—locality, cache, algebraic topology, poset complexes, machine learning, algorithms**

I. Introduction

Since data movement is a common bottleneck in high-performance computation, locality is a crucial consideration in designing algorithms, programming, and compiler optimization. In practice, this means improving cache and memory performance. Hardware caches store recently accessed data blocks, so do operating systems for data pages. Locality of single data elements comes from data reuse.

In this paper, we consider repeated accesses to a collection of data. Each time the data is traversed again is called a *data re-traversal*. A re-traversal order may be identical, i.e., the same order each time, or different. This paper presents a theory of the locality of all re-traversal orders.

Two types of re-traversal order are common: the cyclic order and the sawtooth order. In a *cyclic* re-traversal, the order of access is the same as the previous traversal. In a *sawtooth* re-traversal, the order is reversed from the previous traversal. Since cache typically stores recently accessed data, the locality of access corresponds to recency. A measure of recency is the *reuse distance*, which is the amount of data accessed between two consecutive accesses to the same datum [1]. The reuse distance is first defined by Mattson et al. and called the LRU stack distance [2].

The cyclic order has the worst locality in that its reuse distance is the maximal possible. The cyclic traversal is often referred to as streaming access. It is widely used as a microbenchmark. For example, the STREAM benchmark is a standard test to measure the memory bandwidth on a computer [3]. STREAM contains four kernels that each traverse a different number of arrays in the cyclic order. This benchmark uses it so there is no cache reuse due to poor locality, and data must be transferred repeatedly from memory.

The sawtooth order has better locality than cyclic, and more precisely it has the best recency — the last accessed data is the first to be reused. Many techniques are designed to induce sawtooth data reuse, including the call stack of a running program, insert-at-front heuristic for free lists used by a memory allocator, and the move-to-front heuristic in list search [4].

We formulate the set of all re-traversal orders, of which cyclic and sawtooth are two members. The set of re-traversals of m objects corresponds to the set of $m!$ permutations of those objects. The cyclic order corresponds to the identity permutation, and the sawtooth order to the reversed identity permutation. The set of all permutations of m elements forms a symmetric group S_m as a graded poset (partially ordered set) in algebraic topology [5]. We characterize the locality of all possible traversal orders and call it the *symmetric locality*.

Symmetric locality generalizes cyclic and sawtooth orders and encompasses all possible traversal orders. It may be used in locality optimization: when improving the locality of repeated data traversals, and reordering is restricted, we can choose the one with the best symmetric locality. In particular, deep learning applications, including the transformer algorithm used in Generative AI systems, repeatedly access the same parameter space. Symmetric locality can be used to characterize the effect of different traversal orders.

The primary contributions of this paper as as follows:

- We characterize locality for symmetric groups, and prove an equivalence between locality ordering and inversion number of permutations
- We develop a novel algorithm for calculating reuse distance for traces
- We develop a novel algorithm for traversing symmetric groups with optimal locality

The rest of the paper is organized as follows. Section II gives a brief setup of the context and the main problems of the paper. Section III gives preliminaries in group theory and basic algebraic topology that is needed to understand Section IV, which contains the theoretical and experimental foundations for our symmetric theory of locality. Section V defines the chain-finding algorithm and the various total orderings needed for the algorithm. Section VI discusses the applications for

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00142

symmetric locality, including deep learning with permutation equivariant models and data.

II. Problem Statement

A. Outline

Let $\mathcal{M} = \{1, 2, 3, \ldots, m\}$, where m is the number of trace elements or distinct memory addresses. A trace $\mathcal{T} = (t_i)_i$ is a sequence of memory addresses such that each $t_i \in \mathcal{M}$. As a shorthand, we write $\mathcal{T} = A\,B$, where $A = (t_i)_{i=1}^{m}$ are the first m accesses, and $B = (t_i)_{i=m+1}^{2m}$ are the next m access after A. In particular, we assume that A and B each contain all of $\mathcal{M}$.

Definition 1 (Re-traversals)**.** Let $\sigma \in S_m$, where S_m is the symmetric group VIII-B of m elements, and σ is a permutation pm m elements as defined in the appendix VIII. We construct traces of the form $\mathcal{T} = A\,B$, where

$$A = (t_i)_{i=1}^{m} \qquad t_i = i$$
$$B = (t_i)_{i=m+1}^{2m} \qquad t_{m+i} = \sigma(t_i)$$

This yields $\mathcal{T} = A\,\sigma(A)$, and we can think of B as a reordering of A. We call such a trace a periodic trace or a **re-traversal**.

For the theory, we assume fully associative caches using least-recently used replacement (LRU). Modern caches may not use LRU, may be set associative, and often have multiple cache levels. We consider fully associative LRU cache for two reasons. First, it is amenable to theoretical analysis in reference to a symbolic cache size c. Second, LRU has the strongest theoretical guarantee in that no online algorithm has better amortized performance than LRU [4], and hence most cache policies use some variant of LRU.

Definition 2 (Miss Ratio)**.** For a given $c \in \mathbb{N}$, $mr(c; \mathcal{T})$ denotes the **miss ratio** at cache size c for trace $\mathcal{T}$. $MRC(\mathcal{T})$ is the miss ratio curve for trace $\mathcal{T}$:

$$MRC(\mathcal{T}) = \{(c, mr(c; \mathcal{T})) \,:\, \forall c \geq 0\}$$

We are interested in the following problems:

Problem 1.

Let $\mathcal{M} = \{1, 2, 3, \ldots, m\}$ and a cache size $c \leq m$. Let S_m be the symmetric group on m objects, and by observing $\sigma \in S_m$, where $B = \sigma(A)$, what is the miss ratio $mr(c)$ of $\mathcal{T} = A\,B \;\; \forall c > 0$?

Problem 2. For some program trace $\mathcal{T} = A\,B$, how can we optimize locality by reordering B? As in, if $\sigma(A) = B$, is there another $\tau \in S_m$ such that $\tau(A)$ is a valid trace of the program that preserves correctness as well as improves locality?

The first problem will be the running topic of the paper. The second problem covers applications with regard to machine learning, and will be discussed with more brevity in section VI.

B. Notation

Throughout the paper, we are dealing with traces of the form $\mathcal{T} = A\,B$. A can always be understood to be the generic trace of m elements in this fashion: $1, 2, 3, \ldots m$, and if it is not, we can use a relabeling argument. Since $B = \sigma(A)$, and σ is a bijective function by construction, we can understand any re-traversal by the permutation σ that generates it. Therefore, we will use either characterization interchangeably. Lastly, A is understood to be ordered as in the definition of $\mathcal{M}$, unless explicitly stated.

III. Preliminaries

A. Locality Measures

Closely related to $mr(c)$ is the concept of *cache hits*. There exists a hit, if at time i trace element t_i exists in the cache.

Definition 3. For a given $\mathcal{T} = A\,B$, $|A| = |B| = m$, let $hits_c(\mathcal{T}) \in \{0, 1, \ldots, c\}$ be the number of LRU cache hits that arise after traversing $\mathcal{T}$ with a cache of size c.

It may be helpful to refer to $hits_C(\mathcal{T})$, which is the m-sized vector:

$$hits_C(\mathcal{T}) = (hits_1(\mathcal{T}), hits_2(\mathcal{T}), \ldots, hits_m(\mathcal{T}))$$

For example, let $sawtooth_4$ refer to a well known trace of $m = 4$ data items: a, b, c, d, d, c, b, a. Upon inspection:

$$hits_C(sawtooth_4) = (1, 2, 3, 4).$$

The miss ratio for a given $\mathcal{T}$ and c is then given by $mr(c; \mathcal{T}) = 1 - \frac{hits_c(\mathcal{T})}{\#accesses}$. For the sake of convenience, we will use the $hits$ formula for an equivalent description of locality.

Definition 4 (Reuse Interval)**.** The **Reuse Interval** of an element in a trace is the number of memory accesses between two accesses of this element. For example, in the trace $abcabc$, the first a has reuse interval 3 as there are three accesses between the first and second a, and the second a has reuse distance ∞ as there is no third access.

Definition 5 (Reuse Distance)**.** The **Reuse Distance** of an element in a trace is the number of unique memory accesses between two accesses of this element. It is equivalent to LRU stack distance. [2] In the previous example, $abcabc$, the reuse distance is the same as reuse interval. However, in the trace $abccba$, the reuse distance of the first access of a would still be 3.

B. Algebraic Topology

A review of introductory group theory is covered in the Appendix VIII-A In this subsection, we introduce results studied in Algebraic Topology. Primarily, the classification of S_m as a Coxeter group[1] yields a natural partial order, the *Bruhat order*, defined in section IV. We show that this Bruhat order coincides with the classifications and orderings of re-traversals based on locality. This provides us a framework with which to analyze orderings.

[1]While S_m's identity as a Coxeter group implies the existence of the Bruhat order, we do not focus on the Coxeter group, but instead on the partial order.

C. Locality Poset

Definition 6. Let $\ell(\sigma)$ denote the *minimum* possible length of the cycle decomposition 3 of σ, ie if $\sigma = \sigma_1\sigma_2\sigma_3 \dots \sigma_n$ where each σ_i is an adjacent-disjoint 2-element swap, then $\ell(\sigma) = n$. *Example*: $(13) = (23)(12)(23) \implies \ell(13) = 3$.

Lemma 1. Let $\sigma \in S_m$. Pick $i < j$, if $\sigma(i) > \sigma(j)$, then the pair $(\sigma(i), \sigma(j))$ is an **inversion** of σ. Furthermore, ℓ is equivalent to the *number of inversions* of σ:

$$\ell(\sigma) = |\{(i,j) \,:\, \sigma(i) > \sigma(j)\}| \,. \text{ [6]}$$

Example: $\ell(sawtooth_4) = 6$. A simple way to find $\ell(\sigma)$ is to calculate $\sigma(A)$, then count how many (i,j) such that $\sigma(A)[j] > \sigma(A)[i]$ (using 1-indexing) For the trace $\sigma(A) = 2134$, consider the pair (1,2) which has $\sigma(A)[1] > \sigma(A)[2]$, which is an inversion.

Lemma 2. $\forall \tau \in S_m, \forall \sigma_i \in \mathcal{S}$, the generators of S_m,

$$\ell(\tau\sigma_i) = \begin{cases} \ell(\tau) + 1 & \tau(i) < \tau(i+1) \\ \ell(\tau) - 1 & \tau(i) > \tau(i+1) \end{cases} . \text{ [6]}$$

Example: Consider S_5. Let $\tau = (13)$, and $\sigma_3 = (34)$. Then, $\ell(\tau) = \ell((12)(23)(12)) = 3 \implies \ell(\tau\sigma_3) = 4$.

This lemma also extends to $T \subset S_m$, where T is the set of all simple swaps in S_m.

Let $H = (V, E)$ be a digraph (directed graph), such that $V = S_m$ for a given S_m. For a given $\sigma, \tau \in V$, we denote $\sigma \lhd_B \tau$, if $\exists t \in T$, a *swap* between only two elements such that $\tau = \sigma t$ and $\ell(\sigma) + 1 = \ell(\tau)$. Thus, we let E be the collection of all such relations:

$$E = \{(\sigma, \tau) \,:\, \sigma \lhd_B \tau\}.$$

Since E is generated by $\lhd_B$, we use either of these interchangeably.

Let λ be an edge labeler defined such that if Q is a totally ordered set,

$$\lambda : \{(\sigma, \tau) \,:\, \sigma \lhd_B \tau\} \to Q.$$

For a given starting point $x \in S_m$, we are interested in traversing or "ascending" the graph by picking the maximal or minimal edges wrt to λ. We call such paths **chains**, and the associated greedy algorithm *ChainFind*. This algorithm and λ will be discussed later in section V.

We would also like to note that H has origins from being defined as a *covering graph* using the *inversion* number. In this work, we refer to H as a covering graph interchangeably as the digraph defined here. The **Bruhat Order** $\lhd_B$ arises naturally from its classification as a covering graph. For formal background, refer to Appendix VIII-D.

IV. Symmetric Theory of Locality

In this section, we will motivate the usage of Coxeter-Symmetric groups and, more specifically, the Bruhat order to analyze how locality changes w.r.t. the symmetric group S_m. We show that the Bruhat order coincides with locality through theoretical proof and experimental results, and include additional miscellaneous results in VIII-F.

A. Inversions

Section III-B gives a definition of $\ell(\sigma)$, the inversion number of $\sigma \in S_m$. This quantity has been utilized to study efficiency of sorting algorithms. [7] The inversion number corresponds to lengths of chains on the covering graph formed by the Bruhat order.

B. Identity

By construction, the identity permutation, or the cyclic trace is considered the smallest element in $(S_m, \lhd_B)$. More formally, $\ell(e) = 0$, where e is the identity permutation. This trace, also known as the cyclic trace, has the worst locality, as well.

C. Reverse Identity

S_m is a poset defined by the Bruhat order, and in particular it is a graded poset. In a graded poset, its maximal possible chains from the lowest ordered element to the highest are all the same length, denoted $\ell(S_m)$. The sawtooth trace, which also refers to the reverse identity permutation, has the best locality and has the maximal ℓ, by the Bruhat order.

D. Locality

The foundation of the connection between the Bruhat Order imposed on S_m and the locality of a re-traversal is formally stated in the theorems below.

Theorem 1 (Reuse Distance Calculation)**.** *We provide an algorithm for calculating the cache hit vector iteratively, which also connects the reuse distance of an element with the inversion number of an element.*

We are caching with LRU stack distance, also known as reuse distance. The cache hit vector corresponds directly to reuse distance; $hits_c(\sigma)$ is exactly the number of elements with a reuse distance of c or smaller. This is a direct result of using LRU caching for counting cache hits, as reuse distance is also known as LRU stack distance.

We calculate the reuse interval of an element $a \in A$, and subtract the number of repeated elements between each access of a. Denote the rank $r(a)$ of a to be $n - a + 1$. Then, we see the reuse interval is $r(a) - 1 + i$, where i is the index of $a \in \sigma(A)$. The highest rank will be 1 in order for this to correspond with integers $1 \dots n$, and similarly we will index arrays starting at 1.

In order to convert this to reuse distance, we must subtract the number of repeated values. To do this we keep track of a binary vector c, which flips a bit at place r if the element at rank r is accessed. Then, at the time we access element a, the sum $\sum_{i=1}^{r(a)-1} c[i]$ will calculate the number of repeats we have seen (this also counts the inversion number induced by a). Taking this together, we have our formula for reuse distance $r - 1 + 1 - \sum_{i=1}^{r(a)-1} c[i]$. We increment the reuse distance histogram (rdh) and cache hit vector (chv) at this index by 1. Since a cache hit at size $i-1$ will also be a cache hit at size i, we add $chv[i-1]$ to $chv[i]$.

We provide an example below:

Algorithm 1 Reuse Distance Histogram

for $k \in \sigma(A)$ **do**
  $r \leftarrow n - k + 1$
  $c[r] \leftarrow 1$
  $repeats \leftarrow \sum_{i=0}^{r-1} c[i]$
  $rdh[r - 1 + i - repeats] \leftarrow +1$
  $chv[r - 1 + i - repeats] \leftarrow +1$
  $chv[i] \leftarrow +chv[i - 1]$
end for

Denote that we have seen 2, the element of rank 3

4	3	2	1	3	4	2	1	r
1	2	3	4	2	1	3	4	$A\,\sigma(A)$
				0	0	1	0	c
				0	0	0	0	RD Histogram
				1	2	3	4	Index
				0	0	0	0	Cache Hit Vector

Increment rdh and chv at index 3 - 1 + 1 - 0 = 3

4	3	2	1	3	4	2	1	r
1	2	3	4	2	1	3	4	$A\,\sigma(A)$
				0	0	1	0	c
				0	0	1	0	RD Histogram
				1	2	3	4	Index
				0	0	1	1	Cache Hit Vector

Denote that we have seen 1, the element of rank 4

4	3	2	1	3	4	2	1	r
1	2	3	4	2	1	3	4	$A\,\sigma(A)$
				0	0	1	1	c
				0	0	1	0	RD Histogram
				1	2	3	4	Index
				0	0	1	1	Cache Hit Vector

Increment chv and rd at index 4 - 1 + 2 - 1 , and sum the previous term from the chv

4	3	2	1	3	4	2	1	r
1	2	3	4	2	1	3	4	$A\,\sigma(A)$
				0	0	1	1	c
				0	0	1	1	RD Histogram
				1	2	3	4	Index
				0	0	1	2	Cache Hit Vector

Theorem 2 (Bruhat-Locality)**.** *Let S_m be the symmetric group of m elements. For $\sigma \in S_m$ and some $C \leq m$, we can compute $hits_C(\sigma) = hits_C(A\,B)$, then*

$$\sum_{c=1}^{m-1} hits_c(\sigma) = \ell(\sigma).$$

Proof. We prove this by considering the algorithm above, which connects the reuse distance histogram (and thus the cache hit vector) to the inversion number. We consider all variables in the equation $r-1+i-\sum_{i=1}^{r(a)-1} c[i]$ that determines the reuse distance of an element. In a swap $(a\,b)$, the rank of each element is unchanged, and the index of one is increased and the index of the other is decreased by an equal amount, so these values do not affect the reuse distance histogram.

Since we have imposed the Bruhat order, the swap is guaranteed to increase the inversion number of $\sigma(A)$ by exactly 1, and since $\sum_{i=1}^{r(a)-1} c[i]$ measures the inversion number of each element, the net increase of all sums across 1, resulting in a decrease by 1 of the index of an element. This improves the total reuse distribution (and the sum of cache hit vector) by 1.

□

Corollary 1. The below formula is equivalent to 2

$$\sum_{c=1}^{m} hits_c(\sigma) = m + \ell(\sigma).$$

After reading our initial proof, a colleague Donovan Snyder provided an alternative proof. We include an adaption of this this below.

Snyder Proof. For a permutation σ, define

$$\ell_a(\sigma) = |\{j : j > a, \sigma^{-1}(j) < \sigma^{-1}(a)\}|$$

We can think of this as the number of items that come before a that do not belong in the original permutation. Then, to count the number of elements with a reuse distance of c or less, we define the following:

$$h_c(\sigma) = |\{a : rd_\sigma(a) \leq c\} = \sum_{k=1}^{c} |rd_\sigma^{-1}(k)|$$

We then have

$$\begin{aligned}
\sum_{c=1}^{m} h_c(\sigma) &= \sum_{c=1}^{m}\sum_{k=1}^{c} |rd_\sigma^{-1}(k)| \\
&= \sum_{i=1}^{m} (m+1-i)|rd_\sigma^{-1}(i)| \\
&= (m+1)\sum_{i=1}^{m} |rd_\sigma^{-1}(i)| - \sum_{i=1}^{m} |rd_\sigma^{-1}(i)| \\
&= (m+1)m - \sum_{i=1}^{m} rd_\sigma(i) \\
&= m^2 + m - \sum_{i=1}^{n} (m-i) + \sigma^{-1}(i) - \ell_i(\sigma) \\
&= m^2 + m - \left(\frac{m(m-1)}{2} + \frac{(m+1)m}{2} - \ell(\sigma)\right) \\
&= m^2 + m - (m^2 - \ell(\sigma)) \\
&= \ell(\sigma) + m
\end{aligned}$$

□

Theorem 3. *If $\sigma \lhd_B \tau$, $mr(c;\sigma) \leq mr(c;\tau), \forall c \leq m$.*

Proof. By 2 and definition of $\lhd$:

$$\sum_{c=1}^{m-1} hits_c(\tau) = \left(\sum_{c=1}^{m-1} hits_c(\sigma) \right) + 1$$
$$\sum_{c=1}^{m-1} \left(hits_c(\tau) - hits_c(\sigma) \right) = 1$$

This implies $\exists! c' < m$ such that:

$$hits_{c'}(\tau) = hits_{c'}(\sigma) + 1$$

Therefore, $c \neq c'$, $mr(c;\sigma) = mr(c;\tau)$ and $mr(c';\sigma) < mr(c';\tau)$ □

As a result of 2 if $r_i(\mathcal{T})$ is the number of elements in $\mathcal{T}$ with reuse distance i or smaller, we have

$$\sum_{i=1}^{m-1} r_i(\mathcal{T}) = \ell(\sigma)$$

This statement is equivalent to Theorem 2. Note that since this sum truncates the highest possible reuse distance (m), a higher value is better for locality, as more elements with $r_i < m$ implies less elements with $r_i = m$. Then, since ℓ structures the partial order of S_m following the Bruhat order, we have a direct relation between the ordering imposed by ℓ and locality. That is, $\ell(\sigma) > \ell(\tau)$ implies that σ has better temporal locality than τ. This result and the theorems that lead to it are the basis of our symmetric theory of locality.

Theorem 4. *If σ is the optimal reordering for A such that $A\,\sigma(A)$ has better locality than $A\,\tau(A)$ for $\tau \neq \sigma$, then $\sigma(A)\,A$ has better locality than $\sigma(A)\,\tau(A)$ for $\tau \neq e$.*

Proof. Since Reuse Distance can be calculated backwards or forwards equivalently, $\sigma(A)\,A$ has the same temporal locality as $A\,\sigma(A)$, which is assumed to be optimal. Assume there was some other τ such that $\sigma(A)\,\tau(A)$ has better locality than $\sigma(A)\,A$. Then, by a relabeling argument,

$$\sigma(A)\,\tau(A) \overset{\text{locality}}{\simeq} A\,((\sigma\tau)\,(A))\,,$$

so $\sigma\tau$ has better locality than σ, which is a contradiction. □

E. Experimental Results

To support our symmetric theory of locality, we aggregate average cache miss ratio curves for each inversion number a permutation of a symmetric group can have in 1. To do this, we consider an element-wise average for each cache size.

We see that this depicts a clear trend and separation following the Bruhat ordering for symmetric groups, supporting the earlier theorems that prove a connection between the Bruhat ordering and locality. We also notice a decrease in convexity as ℓ approaches the maximum size. The trends continue for larger graph sizes, however graphs become hard to read for large cache and so we include only cache size up to 5.

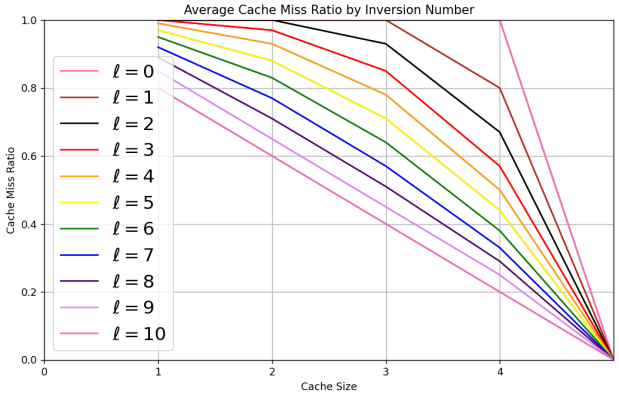

Fig. 1. Average Miss Ratio Curve by Inversion Number for S_5

V. ChainFind Algorithm

A. Chains & Labelings

The ChainFind algorithm is inspired by the construction of poset complexes from algebraic topology. [5] Let $H = (S_m, \lhd_B)$ be a covering system of S_m w.r.t. the Bruhat order. Let λ be a EL-labeling edge labeler. The following algorithm is a greedy weighted breadth first search: given a current path, it finds the best possible path among feasible edges w.r.t. Q, and iteratively builds this path until some stopping point. The full algorithm in its entirety is described in algorithm 2. The stack P returned is the chain starting from τ_0 to some τ_p, where $p = |P|, \tau_p \leq \tau_{rev}$. The next focus is to design good edge labelings of λ and study their consequences or variations.

With respect to the Bruhat order, all maximal chains in S_m are of length $O(m^2)$. [8] Algorithm 2 gives the best possible locality for a chain w.r.t. some totally ordered Q edge-labels, where each edge label is calculated from the locality of the edge's destination node. The covering relation ensures that the branching explored by the algorithm is limited by at most, $|T| = O(m)$, where T is the set of reflections of S_m. Therefore, the total runtime is $O(m^3)$.

The classification of the ChainFind algorithm as a greedy algorithm is proven in Appendix VIII-E. In the context of compiler design, where performance is critical, classifying the algorithm as greedy highlights its efficiency in making locally optimal decisions at each step.

In practice, finding chains should only be done for memory-loading intensive algorithms, For the proposed ChainFind algorithm, a running time of $O(m^3)$ may be too slow in cases that do not take advantage of intensive memory accesses. If the algorithm is being run multiple times, the chain finding algorithm can be done as part of a *Just-in-time* (JIT) scheme by caching (parts of) the chain during profiling, which massively reduces overhead. [9]

B. Locality Ordering

We propose several new and heuristic orderings that describe the locality of traces under permutation. For the purposes of

Algorithm 2 Chain Finding Algorithm

Require: m is the number of data access elements
Ensure: $\tau_0 \in S_m, \tau_0 \leq \tau_{rev}, \tau_{rev}$ is the reverse permutation.
  $chain \leftarrow Stack(\tau_0)$
  while $size\,(chain) < \frac{m(m+1)}{2}$ **do**
    $p_c \leftarrow peek(chain)$
    $E_{size} = \{y : p_c \lhd y\}$
    $next \leftarrow max(E_{size})$
    $push(chain, next)$
  end while
  return $chain$

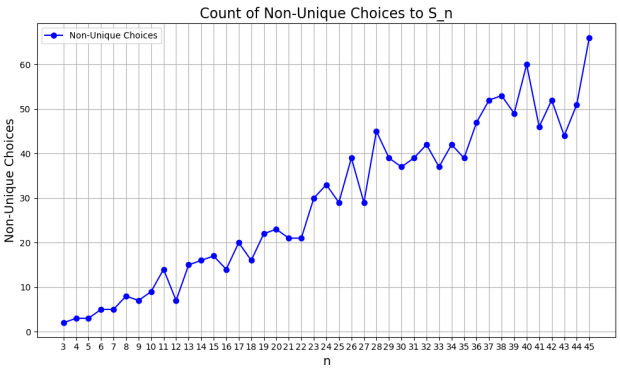

Fig. 2. Plot of λ^e: S_n vs. the count of arbitrary choices that ChainFind had to decide.

this section, let H represent the graph $(S_m, \lhd_B)$. Recall that $hits_c(\sigma)$ measures the locality of σ with cache size c.

Before we delve into a more in-depth study on possible locality orderings, we must discuss feasibility. We say that some trace is *feasible* if it is a possible trace that can be generated by a program. Due to topological dependencies and the AST structure of programs, not every trace is feasible for a given program. Thus, when computing the permutation of traces, we must make sure to stay within the feasible space.

Definition 7 (Feasibility)**.** An access trace is **feasible** if it can be generated from a given program P. Let $\mathcal{F}(P)$ be the space of feasible traces. *Infeasibilities* may occur when there exist topological dependencies within a program that create restrictions on which elements (or edges in our graph) we can access while maintaining program correctness. In this case we focus our attention on a subset of the graph of S_m.

In the context of making a labeler λ, we may define a binary *feasible* function Y:

$$Y(\mathcal{T}) = \begin{cases} 1 & \mathcal{T} \in \mathcal{F}(P) \\ 0 & \text{otherwise} \end{cases}$$

In terms of the lexicographically ordering, the chain generated should yield traces that are feasible. However, it is not necessarily *maximal*, but in the context of program optimization, that does not matter.

For mathematical compatibility, we assume henceforth that every possible trace is feasible.

1) Miss Ratio Labeling: A naive way to design the edge labeler is to simply consider the lexicographic ordering of $hits_C$ (and likewise, the corresponding miss ratio curve MRC).

$$\lambda(\sigma, \tau) = hits_C(\tau) = (hits_1(\tau), hits_2(\tau), \ldots, hits_m(\tau))$$

The ChainFind algorithm will make a decision with comparing the number of cache access hits for $c = 1$ among all τ. If the feasible traces are the same, then it will test $c = 2, 3, \ldots, m$. However, it is not a good labeling. Consider the counterexample at $\sigma = e$, and thus any $\sigma_i \in \mathcal{S}$ implies $e \lhd \sigma_i$. With respect to $c = 1$, we have $hits_1(s_i) = 0$, so $\lambda(e, s_i) \neq \lambda(e, s_j)$ for $i \neq j$.

One option to deal with this is to simply ignore it by creating an arbitrary tiebreaker, since as according to the current labeling, each of the feasible traces have equivalent locality. For compatibility with the good labeling property, the tiebreaker will have to be designed in a way to take into account the total ordering. Examples include a random tiebreaker and the σ_i that described the edge. In particular, the usage of σ_i is inspired by the *standard labeling* of the S_m as a Coexter group. [6]

2) Ranked Miss Ratio Labeling: Another option is to permute the miss ratio curve, which ranks the importance of cache size. Often, cache systems are hierarchical based on cache size, which also makes this option attractive. For $\psi \in S_m$, we define

$$\begin{aligned} \lambda^{\psi}(\sigma, \tau) &= hits_{\psi(C)}(\tau) \\ &= \left(hits_{\psi(1)}(\tau), hits_{\psi(2)}(\tau), \ldots, hits_{\psi(m)}(\tau)\right). \end{aligned}$$

Different cache sizes are preferred w.r.t. ϕ. For example, if we let $\psi(1) = m - 1$, then the counterexample previously mentioned does not need a tiebreaker. However, in practice, this does not alleviate the problem; we give an example of generating a chain from S_{11}, with $\psi = (1\ 10\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2)$, which essentially slides $hits_{c=10}$ value in front of the $hits_{C=11}$ vector. The total length of the chain is 66, but there is a factor of 9 different chains that could be made, as compared to $\psi = e$, where there is a factor of 14 different chains. Figure 2 shows that as the m from S_m increases, the factor of chains that could be made also roughly increases, which means that the chain generated from the algorithm is not distinct with respect to λ^e (as well as any $\phi \neq e$).

VI. Discussion

We present the following **open problem**:

Problem 3. Does there exist an EL-labeling (21) λ that is dependent *precisely* on locality? In other words, is the *ChainFind* algorithm that depends on this λ both optimal and computationally efficient?

To our knowledge, we do not know if it is possible to design such a λ. In order to find a *good labeling*, we attempted to propose several other orderings, where the most notable were

timescale locality [1] and *Data Movement Complexity* [10], among others. In order to advance this problem, we suggest first coming up with a proper *good labeling*, and in turn, transforming to a full *EL-labeling*.

The solution to problem 3 yields a particularly intriguing insight into the nature of program optimization concerning locality: a valid solution would suggest that the optimal strategy is greedy. This finding could open up new avenues for exploring the interplay between greedy algorithms and locality-aware optimization in caching systems.

It is also unclear if the construction of an EL-labeling does matter; maybe a good labeling suffices for most program optimizations.

For the rest of the section, we discuss several areas of exploration to take advantage of symmetric locality. In addition to the below sections, we would like to add that a positive effect of our *ChainFind* algorithm always takes program correctness into account with the addition of the feasibility boolean function Y.

A. Application To Deep Learning

1) Permutation Equivariance: Recently, permutation equivariant deep learning models have been developed for their flexibility, ability to address privacy [11], and handle weight-space tasks [12]. We define a function f to be permutation equivariant if, for some $\sigma \in S_n$:

$$\sigma f(x) = f(\sigma x)$$

Functions found to hold this property include element-wise operators, softmax, linear layers, normalization layers, attention, and others [11]. Since backward propogation is also permutation invariant, we can apply permutations to full models, including MLPs, CNNs, transformers [11], and GNNs [13].

However, this concept does not apply to all data types. The data must also be permutation invariant so it does not lose meaning when permuted. We define the notion of "order" for data to be reflect whether you can permute elements of data without information loss. Some data is completely unordered, like a set of stock prices, while other data is completely ordered, such as a novel. We will refer to unordered data as a set. We also encounter notions of partially ordered data, where the relative order of some elements must hold, but variance is permitted. Examples include a series of sentences where the sentences can be permuted but words within can not, or randomly sampled movements of particles over time where the order of the time stamps matters but the order of particles within the time stamps do not.

2) Optimizing Locality: We propose how to optimize locality for different orders of data. We investigate the case of linear layers (viewed as matrices) for MLPs, and assume a permutation equivariant activation function. We establish an essential theorem to assess repeated accesses of a sequence A. As a result of theorem 4, we can see that if $\sigma(A)$ is the optimal reordering of A, then the optimal reordering of accessing a sequence $A\,A\,A\,A\,A\ldots$ is $A\,\sigma(A)\,A\,\sigma(A)\,A\ldots$. Applying this to learning models, first time we encounter a linear layer, we compute without permutation. However, the second time we access a weight (in backpropogation), we access the tensor with regards to the optimal permutation while calculating loss. Then, upon seeing the weight again we resume initial order. Similarly, this concept can be applied to the key, value, and projection matrices in multihead-attention to greatly improve the memory efficiency of transformers. This can be applied to both training and deployment, as this optimization permits but does not necessitate backpropagation.

To illustrate the benefit of this, we provide a comparison of reuse distances for repeated $n \times m$ matrix (input to MLP) accessed in cyclic and sawtooth re-traversal order. In cyclic order, we see nm elements each with reuse distance nm, yielding a total reuse of n^2m^2. However, a sawtooth traversal would yield $\sum_{i=1}^{nm} i = \frac{nm(nm+1)}{2}$ reuse distance. The leading term is halved, leading to a significant improvement of temporal locality.

For unordered sets of data, the sawtooth permutation should be applied to optimize locality. Considering partially ordered data, we refer to the covering graph to inform our decision. The highest access order on the covering graph that preserves the partial order of the data should be selected to optimize locality. For completely ordered sets, we cannot permute data, so our optimization does not apply.

B. Instruction Locality

We proposed that this new theory can be applied under the guise of a compiler or JIT optimization. It would reorder program instructions when they are executed repeatedly. In the context of very simple programs, it is immediately applicable. Instruction scheduling, however, serves other purposes such as instruction level parallelism and register reuse. Also, typical loop bodies fit in instruction cache and may not benefit further from the reordering optimization.

C. Reordering Algorithms

Reordering algorithms, such as those used in preprocessing for GNNs [14], intend on optimizing ordering to maximize spatial and temporal locality by relabeling the nodes of graphs. Our analysis focuses on temporal locality, and could prove to be a useful tool to improve reordering efforts for subsets of graphs that undergo repeated traversals, such optimizing algorithms traversing as a set of vertices that share many neighbors.

D. Non-periodic Data Reuse

For the periodic trace $\mathcal{T} = A\,B$, each data is reused at most once. This is a problem as in real world caches, data is reused any arbitrary amount of times. To support this new theory, we would have to compare general traces to periodic traces.

E. Limitations

1) Non-periodic Data Reuse: One limitation is that the theory targets only data re-traversals. Modeling by permutations does not cover the extent of all real world cache accesses. The theory models fully associative LRU caches only and does not

consider the effect of parallelism. While memory performance can greatly benefit from latency hiding, e.g., prefetching, we focus on locality and consider only the data movement, not the running time.

VII. Final Remarks

We establish a theory of symmetric localityI, comparing data re-traversals and providing guidelines for optimization opportunities. Fundamentals of group theory, in particular the Bruhat order, are outlined to ground our work. We endow a concept of good labeling, and provide a chain finding algorithm for efficiently finding good labelings alongside an algorithm for efficiently computing reuse distance. Symmetric locality is particularly useful for programs with code or data dependencies, in which sawtooth traversal may not be applicable.

The theory of symmetric locality proposed is not complete. We outline possible future research directions. To motivate a usage to implement in applications in compiler designs or systems, the parallelism or scheduling problem is a very important issue to tackle. Not all programs have data reuse of its accesses being at most one, so the theory may not applicable for all programs. Expanding the applicable class size and addressing multi-threading will allow for a more comprehensive theory of symmetric locality.

Acknowledgments

We would like to thank the following for proofreading this paper: Jack Cashman, Leo Sciortino, Willow Veytsman, Woody Wu & Yiyang Wang. We wish to thank Donovan Snyder for his alternative proof of Theorem 2.

References

[1] L. Yuan, C. Ding, W. Smith, P. Denning, and Y. Zhang, "A relational theory of locality," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 3, aug 2019. [Online]. Available: https://doi.org/10.1145/3341109

[2] R. L. Mattson, J. Gecsei, D. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM System Journal*, vol. 9, no. 2, pp. 78–117, 1970.

[3] J. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE TCCA Newsletter*, 1995, http://www.cs.virginia.edu/stream.

[4] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, 1985.

[5] M. L. Wachs, "Poset topology: Tools and applications," 2006.

[6] D. Montrasio, "The bruhat order on the symmetric group," 2021.

[7] A. Elmasry and A. Hammad, "Inversion-sensitive sorting algorithms in practice," *ACM J. Exp. Algorithmics*, vol. 13, feb 2009. [Online]. Available: https://doi.org/10.1145/1412228.1455267

[8] J. Abello, "The majority rule and combinatorial geometry (via the symmetric group)," 2004, HAL Id: hal-00017901. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00017901

[9] M. Thom, G. W. Dueck, K. Kent, and D. Maier, "A survey of ahead-of-time technologies in dynamic language environments," in *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '18. USA: IBM Corp., 2018, p. 275–281.

[10] W. Smith, A. Goldfarb, and C. Ding, "Beyond time complexity: data movement complexity analysis for matrix multiplication," L. Rauchwerger, K. W. Cameron, D. S. Nikolopoulos, and D. N. Pnevmatikatos, Eds. ACM, 2022, pp. 32:1–32:12.

[11] H. Xu, L. Xiang, H. Ye, D. Yao, P. Chu, and B. Li, "Permutation equivariance of transformers and its applications," 2024.

[12] A. Zhou, K. Yang, K. Burns, A. Cardace, Y. Jiang, S. Sokota, J. Z. Kolter, and C. Finn, "Permutation equivariant neural functionals," 2023.

[13] V. G. Satorras, E. Hoogeboom, and M. Welling, "E(n) equivariant graph neural networks," 2022.

[14] M. Esfahani, P. Kilpatrick, and H. Vandierendonck, "Locality analysis of graph reordering algorithms," 11 2021, pp. 101–112.

VIII. Appendix

A. Groups

Definition 8. A group $\mathcal{G}$ is a set of elements with a binary operator $*$ satisfying the following relations:

- *Closure*: $\forall a, b \in \mathcal{G}, a * b \in \mathcal{G}$
- *Associativity*: $\forall a, b, c \in \mathcal{G}$, $(a * b) * c = a * (b * c)$
- *Identity*: $\exists! e \in \mathcal{G}$ such that $\forall a \in \mathcal{G}$, $e{*}a = a$ and $a{*}e = a$.
- *Invertibility*: $\forall a \in \mathcal{G}$, $\exists! b \in \mathcal{G}$ such that $a * b = e$ and $b * a = e$.

Definition 9. A subset $\mathcal{S}$ of elements of a group $\mathcal{G}$ is called a generator if all elements in $\mathcal{G}$ can be expressed as finitely many combinations of elements in $\mathcal{S}$ and the inverses of elements in $\mathcal{S}$.

Definition 10. A set $\mathcal{R}$ of relations is a set of constraints placed on elements. For example, a constraint could be that for elements $a, b \in \mathcal{G}$, $a * b = b * a$.

Definition 11. A presentation $\langle \mathcal{S} | \mathcal{R} \rangle$ is a set of generators and rules that generate a group. This means that to describe a group, all we would need are the set of its generators and its relations.

B. Symmetric Group

Definition 12. A permutation is a bijective mapping $\sigma : A \to A$ that often serves to rearrange elements of A.

Definition 13. A symmetric group S_m defined over a set of m objects is a group whose elements are permutations of these objects, and whose operator is function composition.

Definition 14. We can describe any permutation as a composition of k-cycles, which is a cycle with k elements. For example, (1,2, 3) is a 3-cycle, while (1,3)(2,4) is 2 2-cycles. We say that any cycle of 2 elements (a, b) is called a **transposition**.

Lemma 3. *Cycle Decomposition Theorem*: Any k-cycle $(a_1 \dots a_k)$ can be decomposed into not necessarily distinct series of transpositions:

$$(a_1 \dots a_k) = (a_1 a_k)(a_1 a_{k-1}) \dots (a_1 a_2).$$

While this decomposition is not unique, its parity is.

Definition 15. For the symmetric group S_m, let A be the set of **adjacent transpositions**, also known as swaps:

$$A = \{(i, i+1) : \forall i \in [m-1]\}.$$

For $i \in [0, m-1]$ we will refer to σ_i as the adjacent transposition $(i, i+1)$. A is also the set of **generators** of S_m; we can obtain any permutation by composing any finite number of swaps.

Definition 16. Let T represent the set of reflections of S_m:

$$T = \{\tau\sigma_i\tau^{-1} : \sigma_i \in A, \tau \in S_m\}$$

T describes swaps between any two elements.

Lemma 4. The symmetric group S_m is a Coxeter group with the set of reflections T [6]. The proof starts with first noticing that S_m is a presentation $\langle \mathcal{S}|\mathcal{R}\rangle$, such that

$$\mathcal{S} = A$$
$$\mathcal{R} = \begin{cases} \sigma_i\sigma_j = \sigma_j\sigma_i & |i-j| \geq 2 \\ \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & |i-j| = 1 \end{cases}$$

where A is the set of adjacent swaps. [6]

C. Partial Order

The following is also covered in [6].

Definition 17. Let $(P, \leq)$ be a set with a partial ordering $\leq$, that is $\forall x, y, z \in P$:

- *Reflexivity*: $x \leq x$
- *Transitivity*: If $x \leq y$ and $y \leq z$ implies $x \leq z$
- *Anti-symmetry*: If $x \leq y$ and $y \leq x$ implies $x = y$

Note, this doesn't imply that every single element is comparable. If this is the case, then the partial ordering becomes a total ordering.

If P is a nonempty set, then we say that $\alpha = \sup P$ is the supremum of P if $x \leq \alpha, \forall x \in P$. Likewise, we say that $\beta = \inf P$ is the infirmum of P if $\beta \leq x, \forall x \in P$. If both of these exist, then P is bounded.

D. Covering & Order

We seek to establish a covering system and order on the symmetric group S_m in order to institute structure for analysis. This yields a natural graph representation of S_m, which we wish to traverse while maintaining optimal locality at each step. This section establishes necessary building blocks to achieve these goals.

Definition 18. Let $\sigma, \tau \in S_m$, and let $T \subset S_m$ be the set of all transpositions (swaps). Then if $\tau = s_1 s_2 \dots s_p,\ s.t.\ s_j \in T$, we define an operator $\leq_B$ below:

$$\sigma \leq_B \tau \iff \sigma = s_{i_1} s_{i_2} \dots s_{i_q}$$

where $i_1, i_2, \dots, i_q$ is a subsequence of $1, 2, \dots, p$. The operator $\leq_B$ is defined as the **Bruhat Order**.

Definition 19. Let $H = (V, E)$ be a directed graph of S_m wrt to $\leq$, where $V = S_m$, and

$$E = \{(\sigma, \tau) \,:\, \ell(\tau) = \ell(\sigma) + 1 \wedge \sigma \leq_B \tau\}$$

In other contexts, H is also called a *covering system*. Since ℓ precisely constructs the edges, we also equivalently write $H = (S_m, \ell)$. By 2, the edge also corresponds to the $s_i \in T$ such that $\tau = \sigma s_i$.

Example: Let $\sigma = (13)$ and $\tau = (14)(13)$. $\ell(\sigma) = 3$ and $\ell(\tau) = 4$ by definition of inversion. In terms of the cycle decomposition, we can see that

$$\begin{aligned} \sigma &= (12)(23)(12) \\ \tau &= (12)(23)(34)(23)(12)(12)(23)(12) \\ &= (12)(23)(34)(23)(23)(12) \\ &= (12)(23)(12)(34) \end{aligned}$$

The decomposition of σ is a subsequence of τ, so $\sigma \leq \tau$. Also, the above elements make an edge in H, by inspecting the decomposition length ℓ.

To clarify notation, we also denote

$$x <_B y \iff x \leq_B y \wedge x \neq y$$
$$x \lhd_B y \iff x <_B y \wedge \ell(y) = \ell(x) + 1$$

E. Chains & EL-Labeling

Definition 20. A **chain** is a totally ordered collection of elements from S_m. A **saturated chain** is a chain with maximal possible length.

Let Q be a totally ordered set. We use the edge labeler λ below to describe the process of labeling edges with element of Q:

$$\lambda : \{(x, y) \,:\, x \lhd_B y\} \to Q$$

Definition 21. Let λ be a edge labeling function. It is an **EL-labeling** (Edge Lexicographic labeling)[2], if $\forall x, y \in S_m, x < y$:

1) There is exactly one saturated chain from x to y such that the labels are in a non-decreasing order.
2) For any 2 saturated chains, the labels of one chain is smaller than the other w.r.t. the dictionary order.

A chain with (2) w.r.t to any other is called a **minimal**[3] chain.

We would like to point out that the construction of an *EL-labeling* directly implies the optimality 21.1 and efficiency 21.2 of a greedy algorithm, we call this algorithm the *Chain-Find* algorithm.

Definition 22. Let Q be a totally ordered set (of labels). For a successor system H of S_m, we say that $\lambda : H \to Q$ is a *good labeling* if

$$\forall x, y, z \in H, x \lhd_B y \wedge x \lhd_B z,\ \lambda(x, y) = \lambda(x, z) \implies y = z$$

In other words, the multiple choices that increase the length of a chain are distinct wrt λ (or what is more commonly known as a bijective property).

Lemma 5. If λ is a good labeling, then 21.2 is fulfilled. [6]

A good labeling ensures that the chain finding algorithm retains its $O(m^3)$ running time. If a good labeling also satisfies 1, then the generated chain is optimal in terms of the λ defined.

[2]This property leads to another "nice" property known as EL-shellability.
[3]Maximal refers to the length of the chain. Minimal refers to its labels.

F. Miscellaneous Characterizations

The following observations provide additional facts obtained from working with re-traversals.

We can characterize each level by noting that possible cache hit vectors are all possible integer partitions for $\ell = n$. Furthermore, if we add all occurrences of each integer partition for a rank of the symmetric group, we find the Mahonian number $M(m, n)$, which counts how many permutations of m elements have n inversions. How to count the number of cache vectors with a specific integer partition is an open problem. Furthermore, if we were to integrate the normalized truncated cache miss vector (the normalized cache hit vector without the last element), we would note that vectos with the same inversion number evaluate to the same value, and the value of integrals drops from 1 (at the identity) to .5 (at sawtooth) with slop $\frac{1}{m(m-1)}$.