JACT Reviewer Response

10/13/2023: This document records the review responses obtained from a submission of the manuscript *Move Schedules: Fast persistence computations in coarse dynamic setting* to the *Journal of Applied and Computational Topology* (JACT).

Overall changes

In this iteration, we've revised the paper to improve readability, streamline the presentation, and address the concerns of the reviewers. Overall, the paper has not been altered dramatically in either content or structure---sections 1, 2, 4, and 5 have only had minor edits based on reviewer feedback. The main changes in this revision include:

- Various pseudocode has been re-included into the main paper from the appendix, per reviewer feedback
- The introductions of Sections 2.3 and 3 have been partially re-written to streamline introducing the main concepts
- Section 3.4 (schedule cost) has been partially re-written with added figures, per reviewer feedback
- All known grammatical errors have been fixed.

The corresponding point-by-point feedback is given below. We thank the reviewers for the constructive feedback given.

Reviewer 1

Thank you for the constructive review. We believe we've addressed the main concerns of the reviewer in the current revision. We remark that improving the accessibility and readability of the paper for a broader audience has been a priority in this revision. Please see the point-by-point responses for details.

Point by point responses

> As a reader with a TDA background, I find the notation and terminology used to describe permutations in the paper quite challenging to comprehend. For example, at page 15, the concept of the LCS of two filtrations is introduced... but later, the LCS of two permutations is extensively discussed in section 3.3.

- We have re-written the first outline section in Section 3 to make it clear that we are indeed just working with permutations of the set {1, ..., m} and how these translate to simplexwise filtrations.

> Additionally, I encountered confusion regarding the term "symbols in L" mentioned on page 22, as well as the discussion of array A on page 26.

- The use of "symbols" terminology was motivated by its widespread use in the context of edit operations on strings (see, for example, the paper we cite on the complexity of the LCS problem (ref. 31)); we agree that its use in this paper is superfluous and thus we've have removed its usage throughout, preferring pure set-theoretic terminology.

> Considering the authors' great examples for Vineyard algorithm 2.2 and 2.3, and that scheduling is the main contribution of this paper, I would love to see some examples. specifically:
>
> - In section 3.3, providing step-by-step demonstrations of constructing a schedule would be highly beneficial for readers seeking a more comprehensive understanding.
> - In section 3.4.2, offering examples illustrating the selection of move operations by the proxy objective would be immensely helpful in clarifying this process.
>
> By incorporating these practical examples, the paper's accessibility and value to a broader audience would be substantially enhanced.

- We've incorporated the reviewers feedback by including two additional figures demonstrating the greedy schedule construction.

- The first new figure (in 3.4.2) shows two tables, one illustrating how a schedule is constructed from a given LIS, and another showing what the proxy objective values are for each candidate element to consider moving. We also show the sorting as a k-layered bipartite graph for additional clarity to help readers relate the proxy objective function to something very concrete (crossing minimization).
- The second new figure (in 3.4.3) builds off the previous example, but gives a more visual demonstration of how the displacement "array" structure $\mathcal{A}$ relates to the footrule distance, and also how it enables us to devise an efficient algorithm for schedule construction.

We hope these changes make the paper more accessible to the broader audience

> On page 14, about the querying of non-zeros in Vineyard, I believe Move algorithm also needs to search for non-zeros to get list J. In my mind, it can take O(m log m) with a sparse matrix structure. Then in section 3.4.1, the authors state that "can be determined efficiently prior to any column operations- no more than O(m) time with row-oriented sparse matrices." Based on my experience, when computing the decomposition R = DV, the matrix structure used is often represented as lists of columns with sparse entries. This choice is typically made to reduce overheads during column operations. Therefore, if querying non-zeros is indeed as cheap as O(m) as stated by the authors, it would be highly beneficial to include a clear and detailed explanation about the specific matrix data structure employed.

- We opted for the lowest bound in the original statement which uses a row-oriented representation; it's true that a column oriented increases the complexity of the scanning by a logarithmic factor. Upon reconsideration, we feel the main take-a-way of the paragraph is that greedily minimizing the number of non-zeros to reduce can lead to an arbitrarily bad schedule. We've refactored the paragraph and removed the reference to row-oriented sparse matrices. (as an aside, we use the same matrix representation as vineyards; this is discussed in section 5)

> Secondly, for the experiment section, all figures that compares the performance of different algorithms like pHcol, Vineyards and Schedule are in terms of time index and number of column operations... it would be highly beneficial to include the results of the total time spent by each algorithm. This would involve considering the time taken for all necessary operations, including the construction of permutations and schedule. By doing so, readers will obtain a more accurate understanding of the practical efficiency of each algorithm in real-world scenarios.

- Thank you again for this detailed comment about improving the comparison. We agree that, for both vineyards and moves there is an overhead cost associated with their implementations---however, we believe that a *legitimate* "wall clock"-based evaluation between the algorithms may provide an unfair characterization. We find that although runtime comparisons can be useful, they typically have many variables that require careful tuning to ensure the comparison is meaningful. In the persistence setting, a few examples of such variables one need consider:
  - pHcol implementation run-times can vary by 2-3 orders of magnitude (see JavaPlex vs GUDHI vs Dionysus vs Ripser++, etc.); how would we account for the different "tricks" used by each implementation?
  - The runtime efficiency depends highly on very specific implementation details, e.g. ripser uses a robin-hood hashmap to store pivot entries, but other implementations opt for different hash table implementations. Not only do each of these varying in memory access time, they also exhibit different performance characteristics in the dynamic setting.
  - The runtime efficiency for phCol can vary significantly on the representations of the columns, and again these variations would need to be adequately explored in the dynamic setting. For example, the Persistent Homology Algorithm Toolbox [PHAT] allows for 8 (!) such representations for mod-2 arithmetic, all have different performances depending on the machine / size of the decomposition. Indeed, the bit-tree representation alone uses de Brujin sequences to reduce the access to pivot entries down to a couple bitwise operations, which is something that would have to be permuted in the dynamic setting---it's unknown whether this would be as fast as it currently is or if it would slow down the algorithm significantly (in contrast, permuting other column representations is straightforward, but less memory efficient).
  - The runtime efficiency of all of the above algorithms is highly machine-dependent; large servers have more cache, more processing power, and slower cores, representing a dramatically different compute environment than traditional research-grade computers.

    - The runtime efficiency of all the above algorithms is specific to characteristics of simplexwise filtration one considers (e.g. does one include the apparent pairs optimization, which only applies to flag filtrations? or the cohomology optimization, which is particularly powerful for other types of filtrations?)
- Rather than performing a runtime evaluation that minimizes these machine- and implementation-specific, we opted to simply show the implementation-independent performance and scalability characteristics by evaluating our method across many different applications, which we think is a fair comparison for a new algorithm. To recap our evaluation:
    - The second graph in the first figure in section 4 gives the reader a sense of the scalability of scheduling
    - The figure showing the CROCKER plot generation gives the reader a sense of how efficient scheduling can be under a relatively coarse discretization of the time domain (as exhibited by the Kendall distance)
    - The table from the bifiltration example compares all three algorithms across 5 different levels of coarsening. For transparency, this table also includes not just the (cumulative) number of column operations, but also the number of permutations!

As a small aside, even though we implemented the scheduling algorithm in Python and the column operations in C++, we did not observe the overhead associated with scheduling to be the main bottleneck of the computations---it was indeed the column operations themselves. Nonetheless, as we mentioned above, we do not believe runtime to be an informative comparison here.

> As the paper is based on the Move algorithm from Busaryev et al, the authors are suggested to give a clearer explanation about the move framework. For example, the Move algorithm restores V first and then R, but in the middle of page 13, the authors describe it as a single one step restoration.

- We've simplified the introduction of moves (section 2.3) and we've relocated the pseudocode for the move algorithms into the main content of the paper.

> In Section 3.2, using Spearman's Footrule Distance is clearer because Spearman Distance sometimes can be referred to 1 minus the Spearman rank correlation coefficient.

- We changed this by defining what we refer to as the Spearman distance, and we added a reference. As an aside, the spearman correlation coefficient effectively uses the footrule distance as an intermediate computation when the computed ranks between the observations are distinct (that is, the correlation coefficient is just a translated and normalized variant of the footrule distance)

> Inequality 21 uses the triangular inequality, so the sum should start with i = 0 to match up p.

- Modified the sum and added the special case for i = 0.

> In greyscale image, 0 means blackness, and 1 means whiteness. Using lower stars of pixel values and sublevel-set filtrations should give one 1-dimensional persistence pairs in the leftmost example. It can be fixed by set image = 1 - image.

- Added the term 'negative' to the figure caption.

> The motivation from this example is also confusing. It will be better if the authors can show something like the number of persistence pairs instead of calculating Betti numbers for each image.

- We've modified the language and added a footnote to make it clear we're simply considering the number of persistent pairs lying above the diagonal. Note that the number of persistent pairs in the grayscale image example is *constant* throughout the 1-parameter family (this includes pairs with 0-persistence, which need be maintained in the dynamic setting).

> In Algorithm 1, inside the second complexity symbol O(d2 log m), what is the d ?

- Added a reference for d.

> In the last paragraph of page 17, "the reduction of schedule planning to crossing detection implies the former can be that can be solved" should be "... can be that can be solved".... In section 3.3, "via the LIS of a single permutation p", *p* should be p'...On page 23, again, in equation 16, all p should be p* or p' or p¯ used in algorithm 5.... On page 26, "at any point during during the execution of Algorithm", remove one "during".... On page 33, in "a fixed matrix which upon application", needs a verb after which... On page 34 later, "Discrete Cosine Transform (DCT) basis, they a convenient basis" misses a verb.

- All fixed.

Reviewer 2

> I believe the paper contains very good and interesting results, but it needs a revision before being published.
>
> Pros: I really like the main idea of the paper: using the machinery of moves that allows one to efficiently recompute the R=DV decomposition after one cyclic permutation in the input filtration, the authors find a heuristic to decompose an arbitrary permutation of the filtration into a sequence of moves of provably minimal length and small cost (this is the heuristic part). The authors show experimentally that this idea works. The paper is as self-contained as possible and the authors work out some concrete examples which are helpful to the reader.
>
> Cons: The detailed list of the remarks follows. Sorry, it's not ordered by importance, so there are trivial things like grammar/typos and more serious issues. In general, I feel that the paper is a bit too verbose - it can be made shorter without losing virtually any content. The most important issues that are listed below are: the comment about 'MoveLeft' and some missing (or hard to find) details about the algorithm (GreedySchedule in Alg. 1).

Thank you for the constructive review. We believe we've addressed the main concerns of the reviewer in the current revision. We'd remark that verbosity is something we have tried to address in this revision, please see the point-by-point responses for details.

Point by point responses

> Also, I would appreciate if the authors formulated exactly what is the implementation they used in their experiments (e.g., did they really use treaps?).

- In practice, we did not use implicit treaps but rather we just compute prefix sums (e.g. `np.cumsum`) independently at every step of the scheduling on an array-based implementation (this is the $\mathcal{A}$ data structure in outlined in section 3.4.2). The code that implements the greedy minimization has a higher asymptotic complexity, but we observed in practice computing prefix sums is incredibly fast due to their implementations being vectorized.

> Page 1, abstract: I don't understand the $O(m \log \log m)$ estimate here. According to Thm. 1, this is the time it takes just to find d, the length of the optimal schedule. Computing the schedule itself costs $O(dm \log m)$ or $O(m \log m)$ per update. Shouldn't this be in the abstract?

- Yes, the $O(m \log \log m)$ complexity refers to the complexity of finding the length of the schedule and the LIS (only). We left out the complexity of constructing the schedule itself in the abstract as we envision there are situations where the algorithm used to actually construct the schedule may vary---depending on the particular data structure of choice supporting the operations outlined in 3.3 (e.g. implicit treaps, segment tree, vEB tree), the deterministic and expected time complexity may change. We've included the very general $O(dm \log m)$ complexity now in the abstract.

> Also, why 'simulation of persistence' (this expression will appear below, too)?

- We agree the phrase 'simulation of persistence' may be confusing to some, so we've replaced all sentences using the phrase with "computing persistence dynamically." As to why the phrasing was chosen originally, the intuition goes back to vineyards: one wants to compute persistence along a continuous homotopy h_t, for all $t \in [0, 1]$, and to do this vineyards is sufficient---but in practice the homotopy must be discretely approximated at every simplex (e.g. using PL, x-monotone, or polynomial curves). Thus, we think of the vineyards representation as a process of *simulating* a homotopy.

> Page 4, 'if the relations are available a-priori' – do the authors mean 'edges'?

- Yes, changed to edges for clarity.

> Page 7, 'we summarize our main results (Theorem 1)'. The main claim in theorem 1 is that one can efficiently compute only the length of the optimal schedule. Will the authors consider reorganizing?

- The theorem has been reorganized.

> Page 8: Although d can be $O(m)$, we provide evidence that $d\, m - sqrt(m)$. Well, $m - \sqrt(m) = O(m)$, why 'although'? Should it be 'Although d can be O(m) and we provide evidence that d ~ m - sqrt(m), d is much smaller in practice'?

- We are aware that $m - \sqrt{m} = O(m)$, which is why we omitted the big-O in the notation $d\, m - \sqrt{m}$. We amended the ordering of the statements to clarify this; it now states "We provide evidence that $d \sim m - \sqrt{m}$ in expectation for random filtrations...although this implies d can be $O(m)$ for pathological inputs, we give empirical results suggesting d can be much smaller in practice."

> Page 10: 'Despite this high algorithmic complexity, the number of column operations has been observed to be super-linear in practice' – this has to be rephrased, because there is no contradiction (order of m^2 column operations is super-linear, too).

- This statement has been removed.

> Sec. 2.3, Moves. Sorry, it might be my fault but I had difficulty understanding the procedure from the description.

- The description of the algorithm (section 2.3) has been simplified.

> I believe having the pseudocode right here will be helpful (after I read it in the appendix, I was happy with the algorithm). Especially since the algorithm is needed in the proof of Prop. 2, too.

- We previously moved much of pseudocode to the appendix in order to streamline the presentation of the concepts, as the pseudocode for both algorithms takes up an entire page, and the pseudocode for MoveRight is also available in Busaryev's paper; however, we agree that their inclusion may help readers understand the algorithm, so we have moved them back.

> Page 16, Alg. 1. a) The authors say 'Algorithm 1 may be easily modified to be completely online, keeping at most two filtrations and one decomposition in memory'. I agree, but I believe it would be equally easy and more natural to state it in this form immediately. The change is in modifying the meaning of \rho_i: while now it is the permutation taking f_1 to f_i, it should be replaced with the permutation taking f_{i-1} to f_i.

- We agree! The algorithm is now presented in the much simpler "online form", which only requires as input a pair of filtrations and a single D = RV decomposition. The original batch-processing presentation of pseudocode reflected more closely the implementation we used (we found it more efficient to compute a batch of schedules, concatenate them, and then give them to the moving algorithm to process sequentially, saving the diagrams between each schedule execution).

> b) Is it correct that the authors use both MoveRight and MoveLeft in their implementation? I believe it is necessary, and Alg. 1 reflects that; however, it raises some questions about the sequel. I'll get back to this point.

- Yes!

> c) Line 6 calls the function GreedySchedule. There is no such algorithm in the paper. There is Alg. 5 in the appendix, LCS-sort, but there is no greedy heuristic in it. Also, it computes the LIS internally, while GreedySchedule takes the longest increasing sequence as a second argument. Please fix this.

- The "GreedySchedule" algorithm wasn't included in part because it corresponded to simply modifying line 6 in Alg. 5 (choosing the next 'symbol' / simplex to move) with equation (22). In this revision, we've not only made this much more explicit, but we've also simplified the notation of the inner schedule construction algorithm.

> Note that $m_{i,j}$ is defined here and used in Alg. 5 at the very end of the paper without explanation.

- The usage in Alg. 5 was intentional as is defined in equations (10) and (11); a reference to those equations has been added in Alg. 5.

> Page 21: Do I understand correctly that this is what is going on in the proof of Corollary 1: $E(m - |LCS(p,q)|) = m - 2\sqrt(m) - cm^{1/6} + o(m^{1/6})$ by the known result of Baik for $E|LIS(p)|$, therefore it is less than $m - \sqrt(m)$? So any function which is asymptotically less than $2\sqrt(m) - cm^{1/6} + o(m^{1/6}$ would do?

- The expectation result implies that $\mathbb{E}[LIS(p)]$ for random $p \in S_m$ is something slightly smaller than $2\sqrt{m}$, thus we expect the size of the LCS between random permutations $p, q \in S_m$ to be slightly larger than $m - 2\sqrt{m}$, which we simplify by stating "... no larger than $m - \sqrt{m}$."

> Another question : Corollary 1 speaks about two random filtrations, which the authors interpret as 'two permutations chosen uniformly at random' (explaining why this is okay in light of Boissonat's result). But the longest increasing sequence result by Baik is for a single permutation drawn uniformly at random. Is it obvious that, if p and q are drawn from S_m uniformly at random, then $p \circ q^{-1}$ (the permutation we should consider for LIS) is also uniformly distributed? Well, for symmetry reasons it seems to be true and obvious, but, given the level of details in the other parts of the paper, the authors may want to include this as well.

- We agree that $p \circ q^{-1}$ is uniform, and we agree the proof strategy for Corollary 1 outlined by the reviewer would match the level of detail of the rest of the paper. That said, in our struggle to keep the length of the paper manageable we chose to present the proof in a succinct and informal way.

> Page 22, 'It is clear from Corollary 1' - wrong reference. Probably, from the proof of proposition 3 (Prop. 3 itself does not say anything about LIS).

- Yes, this used to be its own corollary. The proof of prop 3 is indeed the reduction we're referring too. We've amended the statement to explicitly point to the proof.

> General comment on the greedy heuristic: does most of the reasoning in section 3.4 apply to the case of MoveRight only? The starting point is the term \sum |I_k| + |J_k|, which only applies to moving simplex i to simplex j if i < j. What about the MoveLeft? Can it be analyzed theoretically? Does it occur frequently in the authors' experiments? Can the freedom that the authors have in choosing the schedule of the optimal length be used to minimize the number of left moves and, if I want to implement this approach, should I do that or would it only harm me?

- We actually included a discussion and small analysis of MoveLeft, including some empirical comparisons, in the original draft of the paper. To answer your questions successively:
  - > "does most of the reasoning ... apply to the case of MoveRight only?"

    - Yes, due to theoretical limitations / the lack of a donor concept, the statement only holds for MoveRight.
  - > "What about the MoveLeft? Can it be analyzed theoretically?"

    - We have a small discussion of MoveLeft in the appendix, which was moved to shorten the paper. The pseudocode for MoveLeft has been relocated along with MoveRight for transparency. We made several attempts at analyzing the move left action (since it is not included in Busaryev's paper) and we found it could be expressed in a similar way as MoveRight in the sense of restoring columns; however, the notion of the donor concept seems unique to MoveRight. There is no way we know of, using either row or column operations, to symmetrically translate the notion of moving a simplex later in the filtration to moving it earlier.
  - > Does it occur frequently in the authors' experiments?

    - Though this depends greatly on the filtrations being compared, in general we didn't notice any large discrepancies between the frequency with which the heuristic chooses to move right or left
  - > Can the freedom that the authors have in choosing the schedule of the optimal length be used to minimize the number of left moves and, if I want to implement this approach, should I do that or would it only harm me?

    - We originally performed many experiments attempting to validate exactly this; that is, we tried to empirically measure the number of columns operations move left (ML) incurred vs move right (MR), seeking to improve the heuristic by preferring one or the other. Surprisingly, we found that it wasn't the discrepancy between balancing MR vs ML that heavily influenced the number of operations but rather it was the quality of the heuristic at getting close to the Kendall distance (in terms of crossings), which we tried to demonstrate in the performance comparison in the Figure from section 3.1 (left side). Intuitively, one could use only MR to update filtration via selection sort (at the expense of requiring more than d moves)---this is what Busaryev did actually, which we show with the dotted red line in the figure. We observed that, since R is typically quite sparse, the additional cost of ML was not large (say no more than 2.5x that of MR). To keep the paper at a reasonable length, we did not include this comparison.

> Sec. 4.1: I find the phrase 'simulate persistence' unusual. Does it just mean 'compute persistent homology'?

- See the comment above; we've removed the work simulate.

> As for the second test (page 28): was the Algorithm 1 used verbatim, or was the scheduling performed to move from one snapshot to the next one?

- In practice we used a batch implementation, such that given an initial filtration (K, f_0) and some number $b > 0$, we construct the schedules for (f_0, f_1), (f_1, f_2), ..., (f_{b-1}, f_b) in advance and concatenate them. The *scheduling* is still performed pairwise, but performed in a batch-wise manner to amortize the overhead associated with switching between scheduling and performing column operations. This was motivated by the observation that in practice the column operations always dominate the compute time, so to maximize cache efficiency we collect as many move operations ahead of time as space allowed.

> Page 32, 4.3.1: Is \kappa from [10] the same as defined in Theorem 1? I suspect not, [10] says it's coarseness. Please clarify.

- Fixed; apologies, they are not the same quantities (though they are related in a non-trivial way, which is what may have caused the confusion).

> Page 34: 'they a convenient basis for D may be obtained' - is this supposed to be 'they prove that a convenient basis ...' ?voronoi -> Voronoi 7-spheres volume -> 7-sphere's volume In order reveal -> In order to reveal

- All fixed.

> vineyards is, moves requires -> either 'vineyards/moves algorithm/approach is/requires', or 'vineyards are/moves require'. Or make vineyards and moves boldface or italics, then they are visually perceived as names.

- All references to vineyards and moves as methodologies have been refactored with italics to address this.

> Page 45. We recall an important claim .. - What is the claim? Seems to be missing, it should say something about positive/negative simplices.

- Removed.

> Page 48. Well, in A.2.1 the authors mention \kappa as coarseness parameter! Also, the naming clash between the structure T used to compute the sorting and the barcode template T is unfortunate.

- Fixed. We now use $\mathcal{U}$ for the set like data structure, and ν for the output sensitive bound.

> Page 8: 'an simplexwise' -> a simplexwise ... Page 17, the paragraph after Def. 1: 'the former can be that can be solved optimally' - this sentence should be fixed. ... Page 19: 'small changes ... affects' -> affect. .. Page 20: 'allows use to exploit' -> allows us to exploit ... Let T denote a ordered set-like -> an ordered set-like ... Page 27: equation (22) may solved -> equation (22) can be solved ... Page 35: 'Projecting the image data onto the first two basis vectors of leads' - basis vectors of what?... Page 36: 'the number of permutations applied throughout the encountered along the traversal of the dual graph' - please fix this sentence.... that naively computing -> than naively computing.... Page 37: if the filtrations in the parameterization family is nearby -> are nearby ... Page 43 , A1.1.1 'After setting V is set' -> After setting V... 'Since there exists complexes' -> Since there exists a complex / there exist complexes .. kendall -> Kendall

- All typos and known grammar issues fixed