SIGACT News Online Algorithms Column 33

Rob van Stee
University of Siegen
Siegen, Germany

For this issue, Matthias Englert has contributed an alternative and simpler proof of a result by Gamzu and Segev, which was in *ACM Transactions on Algorithms* in 2009. The problem considered in this paper was the reordering buffer problem on the line. Gamzu and Segev were the first to give an $O(\log n)$-competitive algorithm for this problem, and there has been no improvement on this since then, leaving a gap with the best known lower bound of 2.154 by the same authors.

Matthias' proof shows that this result can be slightly improved (a smaller hidden constant) and simplified. Who is going to be the first to give a constant competitive algorithm, or show that this cannot be done?

As always, I would like to invite more contributions to this column, be it surveys, conference reports, or technical articles related to online algorithms and competitive analysis. If you are considering becoming a guest writer, don't hesitate to mail me at `rob.vanstee@uni-siegen.de`.

The reordering buffer problem on the line revisited

Matthias Englert

The reordering buffer problem (or also sorting buffer problem) was introduced by Räcke, Sohler, and Westermann in 2002 [13] and has been extensively studied since then. In this problem, a metric space is given[1] and a sequence of items arrive online. Each item is associated with a point in the metric space. We allow multiple items to be associated with the same point. An online algorithm can store up to k items in a buffer, but once the buffer is full, the algorithm has to process at least one of the items stored in the buffer. To process an item from the buffer, the algorithm moves a single server in the metric space to the point corresponding to that item. The goal is to minimize the total distance that the server has to travel to process the entire input sequence.

The problem is reasonably well understood for some metric spaces. For uniform metric spaces for example, a deterministic $O(\sqrt{\log k})$-competitive algorithm is known, which is close to the lower bound of $\Omega(\sqrt{\log k / \log\log k})$ [1]. Similarly, [4] gives a $O(\log\log k)$-competitive randomized online algorithm, which is asymptotically tight [1].

For other metric spaces however, the picture is less clear. We will refrain from listing all known results in detail, but there have been a number of papers investigating this online problem for different metrics spaces and settings [2, 3, 6, 7, 8, 9, 10, 11, 12]. However, in this column, we will focus on line metric spaces. The last notable result for this metric was obtained eleven years ago by Gamzu and Segev [10]. Their main result is a deterministic $O(\log n)$-competitive online algorithm for a line metric space with n evenly spaced points. In the reminder, we will sketch a slightly simplified and improved version of this result.

An $O(\log n)$-competitive algorithm for the n-point line

We consider a line metric spaces with n evenly spaced points. For simplicity, we refer to the points by integers from $\{1, \ldots, n\}$ so that the distance between two points p and q is $|p-q|$. We call an interval between two neighboring points p and $p+1$ a line segment and we say the server traverses the segment if the server moves from a point less or equal p to a point greater or equal $p+1$ or vice versa.

The central idea behind the online algorithm in [10] is the concept of a *doubling partition.*

Definition 1 (Doubling partition[2]). *A doubling partition of the line with respect to a point p is a partition of all remaining $n-1$ points into $m := 2\lceil \log(n+1) \rceil - 2$ many intervals $L_{\lceil \log(n+1) \rceil - 1}(p), \ldots, L_1(p), R_1(p), \ldots, R_{\lceil \log(n+1) \rceil - 1}(p)$, where*

$$L_i(p) = [p - 2^{i+1} + 2, p - 2^i + 1] \cap [1, n] \qquad \textit{and}$$
$$R_i(p) = [p + 2^i - 1, p + 2^{i+1} - 2] \cap [1, n] \ .$$

[1][13] introduces the problem for uniform metrics, but different and more general metric spaces were considered in subsequent works.

[2]For presentational reasons, our definition differs slightly from the one in [10].

Note that some intervals may be empty, but this does not cause any problems in the algorithm or analysis.

The online algorithm in [10] virtually partitions its buffer of size k into m equally sized parts so that each part contains the items for one of the intervals in the doubling partition. The algorithm is designed in such a way that none of these virtual sub-buffers overflow. That is, no interval in the doubling partition ever contains more than k/m items (except in some temporary intermediate configurations).

However, strictly maintaining such a property leads to slightly more complicated moves of the server than what would otherwise be required. Therefore, we give an alternative algorithm which avoids this virtual partition of the buffer. This also improves the competitive ratio slightly from $168 \cdot \lfloor \log n \rfloor + 124$ to $48 \cdot \lceil \log(n+1) \rceil - 36$.[3]

The online algorithm works in phases. At the start of a phase, the buffer is filled with items from the input sequence until it is full, i.e., contains k items. The server position is some point p. The algorithm then does the following.

1. Determine how many items in the buffer fall into each interval of the doubling partition with respect to p. Note that since the buffer contains k items and there are at most m intervals in the doubling partition, there must be some interval containing at least k/m of the items in the buffer. Say there is such an interval $R_i(p)$ to the right of p (the case where such an interval is to the left of p is handled symmetrically).

2. Move the server from p to $p + 2^i - 1$, i.e., to the left most endpoint of the interval.

3. Process all items in the buffer which fall into the interval $R_i(p)$, for this, it is sufficient to move the server once from $p + 2^i - 1$ to $p + 2^{i+1} - 2$. After this, move the server back to $p + 2^i - 1$.

Note that at the end of the input sequence, there may be a situation where there are not enough items left to completely fill the buffer. For simplicity, we could assume that the model does not require us to process these remaining items. If we are required to process them, we can do so (with a single sweep from the left most item to the right most one) at an additional cost no larger than twice that of an optimal offline algorithm.

For our analysis, we will pretend that the server moves in Step 3, from $p+2^i-1$ to $p+2^{i+1}-2$ and back, are not performed (but, magically, the items are still processed as before). Note that the server move in Step 2 incurs cost $2^i - 1$ and the total cost of the server moves in Step 3 is $2 \cdot (2^i - 1)$. Therefore, ignoring the server moves from Step 3 will result in total cost that is one third of the true cost.

We fix an arbitrary segment S of the line, say between point q and point $q+1$ and analyze how often the online algorithm traverses this segment versus an optimal algorithm. Given our new "pretended" server movements, i.e. excluding the movements in Step 3, we make two important observations:

(a) In every phase in which the online algorithm traverses S from left to right or from right to left, the online algorithm processes at least k/m items with positions greater or equal than $q+1$ (i.e. to the right of S) or with positions less or equal than q (i.e. to the left of S), respectively.

(b) After every phase in which the online algorithm traverses S, the final server position is on the other side of S. That is, if the server started left of S it will be to the right of S after the phase and vice versa.

[3]A more careful analysis can improve this further.

Furthermore, the following general lemma will help us to obtain a lower bound on the optimal offline cost.

Lemma 1. *Suppose we execute two (not necessarily online) algorithms A and B on the same input. Imagine they are synchronized: whenever A processes one item, B also processes exactly one. If, for a time interval, A's server is only located to the left (right) of S, then B can process at most $2k$ items that lie to the right (left) of S during this time interval.*

Proof. At the start of the considered time interval, B has at most k items in the buffer which can be located to the right (left) of S. Therefore, if at some point B has processed $2k + 1$ items at such positions, $k + 1$ of them must have arrived after the start of the time period. Algorithm A cannot have processed any of these $k + 1$ items since the server never visited any of the corresponding points after the start of the time interval. Since A can store at most k of these items in its buffer, this is a contradiction. □

Combining our two observations (a) and (b) with the lemma above gives us the following result.

Lemma 2. *If there is a time interval during which the online algorithm traverses S at least $4m + 2$ times, any optimal offline algorithm also has to traverse S at least once during the time interval.*

Proof. Assume for contradiction that the statement is not true. Without loss of generality assume that the server of the optimal offline algorithms is located to the left of S at the start of the time interval. Due to observation (b), every second time the online algorithm traverses S, this movement will be from left to right. Hence, according to observation (a), after traversing S $2\ell + 1$ times, the online algorithm has processed at least $\ell \cdot k/m$ items to the right of S. Using Lemma 1 with A being the optimal offline algorithm and B being the online algorithm, $\ell \cdot k/m$ cannot be greater than $2k$. Therefore $\ell \leq 2m$, which is a contradiction. □

This directly gives us the desired result.

Theorem 1. *The competitive ratio of the online algorithm is bounded by $48 \cdot \lceil \log(n+1) \rceil - 36 = O(\log n)$.*

Proof. The online algorithm never traverses a segment that an optimal offline algorithm never traverses, because to traverse a segment the input needs to contain at least one item to the left and one item to the right of the segment. Due to Lemma 2, for each segment, the online algorithm traverses the segment at most $2 \cdot (4m+2)$ times as often as an optimal offline algorithm. Hence, $2 \cdot (4m + 2)$ would be the competitive ratio of the online algorithm. However, so far we have ignored the cost of the server moves in Step 3 of the online algorithm. The correct bound on the competitive ratio is therefore $6 \cdot (4m + 2)$. □

Open problems

The obvious and intriguing open question is whether there is an online algorithm, either deterministic or randomized, that has a competitive ratio much better than $\Theta(\log n)$. The only known lower bound is $1 + 2/\sqrt{3} \approx 2.154$ [10].

It is not surprising that the algorithm presented here, like the version in [10], is $\Omega(\log n)$-competitive [14]. Let us for simplicity assume that $n = 2^a - 1$ for some $a \in \mathbb{N}, a > 2$ and that $2a - 2 \mid k$. Let the points be numbered $-\frac{n-1}{2}, -\frac{n-3}{2}, \ldots, \frac{n-1}{2}$. Then $m = 2a - 2$. The server starts in the middle, at position 0. Consider an input of the following form. At the beginning, there are $\frac{k}{m} + a - 2 = \frac{k}{2a-2} + a - 2$ requests at position 1 and at position -1. There are $\frac{k}{m} - 1$ requests at positions $2, 4, 8, ..., 2^{a-2}$ and $-2, -4, -8, ..., -(2^{a-2})$.

The algorithm will only move between positions 1 and -1. Every time that it serves requests at 1, $\frac{k}{m} + a - 2$ new requests arrive at -1, and vice versa. This continues for as long as we want. The algorithm therefore pays 2 to serve each group of $\frac{k}{m} + a - 2$ requests. Of course, the optimal solution is to first clear the line, at cost $2n$, and then to move back and forth between 1 and -1 only once every $k/2$ requests. Setting for instance $k = (a-2)(a-1)$, this gives in the long run a competitive ratio of $\frac{2}{3}(a-1) = \Omega(\log n)$.

More generally, it is known that a different type of analysis would be required for such a result. Specifically, our proof here as well as the proof in [10] is robust in terms of changes to the buffer size of the optimal offline algorithm. Specifically, say we compare the cost of the online algorithm with a buffer of size k to the cost of an optimal offline algorithm with a buffer of size $4k$, then these proofs still go through with appropriately adjusted multiplicative constants. However, a result by Barman, Chawla, and Umboh [5] implies that no proof with such a property can provide an upper bound better than $\Theta(\log n)$ on the competitive ratio.

Another question is what happens if the points on the line are not necessarily evenly spaced or the line metric is continuous. For this case, [10] gives an upper bound of $O(\log N \log\log N)$ on the competitive ratio, where N is the length of the input sequence. This can be improved to $O(\log N)$ (Cygan, Mucha, personal communication, 2011), but the gap to the lower bound of ≈ 2.154 remains large.

Finally, deviating slightly from the topic of this column about *online* algorithms, even the offline complexity of the problem appears to be open.

References

[1] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 607–616, 2011.

[2] Noa Avigdor-Elgrabli, Sungjin Im, Benjamin Moseley, and Yuval Rabani. On the randomized competitive ratio of reordering buffer management with non-uniform costs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 78–90, 2015.

[3] Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 13–21, 2010.

[4] Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm algorithm for reordering buffer management. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 2013.

[5] Siddharth Barman, Shuchi Chawla, and Seeun Umboh. A bicriteria approximation for the reordering buffer problem. In *Proceedings of the 20th European Symposium on Algorithms (ESA)*, pages 157–168, 2012.

[6] Matthias Englert and Harald Räcke. Reordering buffers with logarithmic diameter dependency for trees. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1224–1234, 2017.

[7] Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. *Theory of Computing*, 6(1):27–46, 2010.

[8] Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 627–638, 2005.

[9] Hossein Esfandiari, MohammadTaghi Hajiaghayi, Mohammad Reza Khani, Vahid Liaghat, Hamid Mahini, and Harald Räcke. Online stochastic reordering buffer scheduling. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 465–476, 2014.

[10] Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Trans. Algorithms*, 6(1), 2009.

[11] Rohit Khandekar and Vinayaka Pandit. Online sorting buffers on line. In *Proceedings of the 23rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 584–595, 2006.

[12] Matthias Kohler and Harald Räcke. Reordering buffer management with a logarithmic guarantee in general metric spaces. In *Proceedings of the 44th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 33:1–33:12, 2017.

[13] Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 820–832, 2002.

[14] Rob van Stee. Personal communication.