Online algorithms with pairwise-permuted inputs

Andrew Twigg*

August 20, 2004

Keywords: On-line algorithms

1 Introduction

Imagine a network $\mathcal{N}$ with delays, such that when a message sequence $a, b, c, \ldots$ is sent over the network, elements of the sequence may individually experience delays. This is a simplified model of UDP transmission, which is often used for broadcasting streams of data where low transmission overhead is important (for now, let's assume that the stream is lossless, ie the maximum delay experienced by any element is finite). We say that $\mathcal{N}$ has delay of d if the maximum amount of time any element can be delayed for is d. Clearly, the set of d-delayed versions of a stream σ is a subset of the permutations of the elements of σ. For example, a 1-delayed version of $\sigma = \sigma_1, \sigma_2, \sigma_3, \ldots$ is $\sigma' = \sigma_1, \sigma_3, \sigma_2, \ldots$.

Now imagine an on-line algorithm ALG that receives an input stream σ' over the network, as in Figure 1. The network can be considered a set of links and the network chooses (adversarially) which link to send each element over. We investigate the effect of the network on the competitive ratio of ALG, by comparing the worst-case performance of ALG(σ') to ALG(σ). We look at on-line algorithms for three well-studied problems: minimum spanning tree, steiner tree and static list-accessing.

*Address: Computer Laboratory, 15 JJ Thomson Avenue Cambridge UK e-mail: andrew.twigg@cl.cam.ac.uk

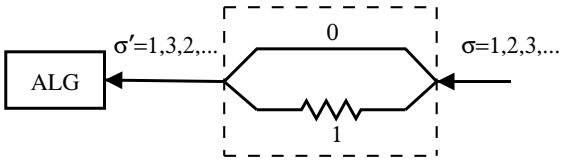

Figure 1: An on-line algorithm ALG receives its input via a network. The top and bottom links incur delays of 0,1 respectively. The network can choose adversarially, for each element, the link to send it over

2 Minimum Spanning Tree

The first example is the on-line minimum spanning tree problem. Given a graph G with vertices V, and input sequence $\sigma = v_1, v_2, \ldots, v_i$, the algorithm must construct a spanning tree T_i, where the set of vertices in $T_i = \{v_1, v_2, \ldots, v_i\}$. The cost of the algorithm on σ is the sum of edge weights used (edges can be deleted at zero cost).

Algorithm GREEDY. The algorithm GREEDY connects v_i to the closest node $\arg\min_{v \in T_i} d(v, v_i)$ currently in the spanning tree. Hence, it never removes edges. The following theorem is well-known.

Theorem 1 *The greedy minimum spanning tree algorithm is $O(\log n)$-competitive for any weighted graph over n vertices, and nothing on-line can beat $\Omega(\log n)$.*

It is reasonably easy to show that a lower bound, i.e. that the 'cost' of a 1-delayed adversarial network to GREEDY for the on-line minimum spanning tree problem is at least $\frac{3}{2}$.

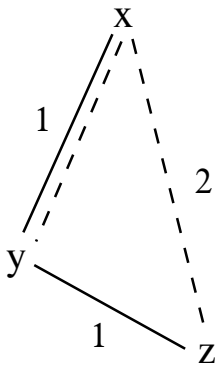

Figure 2: Illustrating Theorem 2. The solid edges illustrate the tree constructed by GREEDY$[x, y, z]$ whilst the dashed edges show the tree constructed by GREEDY$[x, z, y]$

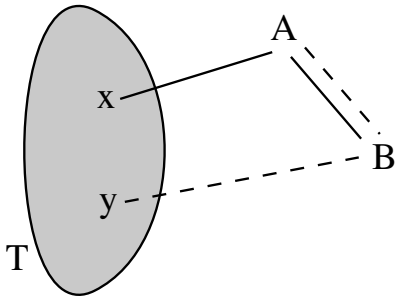

Figure 3: Illustrating Theorem 3. The tree T exactly spans the vertices in σ. The solid edges are added by GREEDY on σ, A, B while the dashed edges are added by GREEDY on σ, B, A

Theorem 2 *For the on-line minimum spanning tree problem, the competitive ratio of GREEDY on 1-delayed inputs is at least* $3/2$ *times the competitive ratio of GREEDY on 0-delayed inputs.*

Proof. We need to exhibit a sequence σ and a 1-delayed version σ', such that the cost ratio is $3/2$. Consider Figure 2, where $\sigma = x, y, z$ and $\sigma' = x, z, y$, and assume wlog that $d(x, y) \leq d(z, y)$. Then GREEDY on σ connects x, y and y, z, yet on σ' it connects x, z then x, y. It is clear that GREEDY$(\sigma') = \frac{3}{2}$GREEDY(σ). □

To provide an upper bound, we use a lemma which applies to on-line algorithms for constructing (minimum) spanning trees.

Lemma 1 *For the on-line minimum spanning tree problem, the cost of serving future requests* $\sigma_{i+1}, \sigma_{i+2}, \ldots$ *is independent of the edges in the current tree* T_i*.*

Proof. The proof follows immediately from the definition of the spanning tree, i.e. T_i contains exactly $\{v_1, v_2, \ldots, v_i\}$ and so has no paths $v \to w \rightsquigarrow x$ where $w \notin \sigma$. □

Theorem 3 *For the on-line minimum spanning tree problem, the competitive ratio of GREEDY on 1-delayed inputs is as most* $3/2$ *times the competitive ratio of GREEDY on 0-delayed inputs.*

Proof. By Lemma 1, it suffices to show that the cost of adding any pair of vertices B, A to a tree T, incurs at most $3/2$ times the cost of adding A, B to the *same tree* T. If A is not the closest element to B, then the ordering of A, B is unimportant since GREEDY(σ) $=$ $d(x, A) + d(y, B)$ $=$ GREEDY(σ') where x is the closest element to A and y is the closest element to B. Hence we can assume wlog that $d(A, B) \leq d(t, B)$ for any $t \in T$.

Let x be the closest element of T to A, and similarly y for B. Then GREEDY on σ connects x, A then A, B, and on σ' it is forced to take the long way round and connects y, B followed by B, A, as in Figure 3. If $x = y$ then it is clear that GREEDY$(\sigma') \leq \frac{3}{2}$GREEDY$(\sigma)$.

Assume that $x \neq y$, then from Figure 3, $d(y, B) \leq d(A, B) + d(x, A)$ if y is to really be the closest element of T to B. Since A, B are closest to each other, then $d(A, B) \leq d(x, A)$ which implies that GREEDY$(\sigma) \geq$

$2d(A,B)$, hence $d(A,B) \leq \frac{1}{2}\text{GREEDY}(\sigma)$. We have

$$\begin{aligned}\text{GREEDY}(\sigma') &= d(y,B) + d(A,B) \\ &\leq 2d(A,B) + d(x,A) \\ &= d(A,B) + \text{GREEDY}(\sigma) \\ &\leq \frac{3}{2}\text{GREEDY}(\sigma).\end{aligned}$$

□

3 Minimum Steiner Tree

The on-line Steiner tree problem is similar to the minimum spanning tree, but the Steiner tree T_i can include nodes not in σ, i.e. $\{v_1, v_2, \ldots, v_i\} \subseteq T_i$, and the subtree T_i must include T_{i-1} as a subgraph. The configuration of a Steiner tree algorithm is the current tree T_i. The cost for changing from T_i to T_{i+1} is the sum of weights of edges added, known as the distance $d(T_i, T_{i+1})$. Hence $d(\cdot)$ is a metric on the space of configurations.

The on-line Steiner tree problem is a special case of the file allocation problem, where $D = 1$, only read requests are allowed, and the algorithm is forced to replicate on every read request.

The GREEDY algorithm simply connects the requested node to the closest node in the tree, via the shortest path (rather than single edge). Since it never removes edges, the cost GREEDY(σ) is simply the weight of the final tree T_l. The optimum cost is the weight of a minimum Steiner tree spanning the nodes requested. We start by recalling a well-known result about the competitiveness of GREEDY.

Theorem 4 *The greedy Steiner tree algorithm is strictly $\lceil \log n \rceil$-competitive for any weighted graph over n vertices, and nothing on-line can beat $\frac{1}{2}\log n$.*

Now we present a lower bound of 3 for the approximation ratio for 1-delay networks for GREEDY.

Theorem 5 *For the on-line minimum Steiner tree problem on infinite metric spaces, the competitive ratio of GREEDY on 1-delayed inputs is at least 3 times the competitive ratio of GREEDY on 0-delayed inputs.*

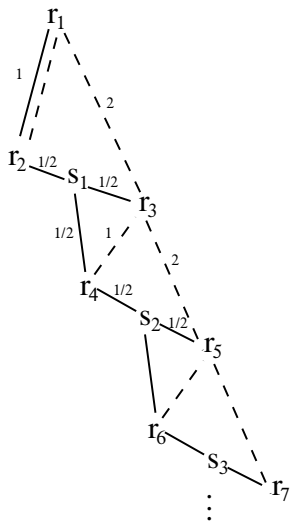

Figure 4: Illustrating the lower bound of 3 for the greedy on-line Steiner tree algorithm. 5.

Proof. Figure 4 shows the result of the request sequence $\sigma = r_1, r_2, r_3, r_4, r_5, r_6, \ldots, \theta$ and $\sigma' = r_1, r_3, r_2, r_5, r_4, r_7, r_6, \ldots, \theta$ where $\theta = r_2 \rightsquigarrow r_3, s_1 \rightsquigarrow r_4, r_4 \rightsquigarrow r_5, s_2 \rightsquigarrow r_6, r_6 \rightsquigarrow r_7 \ldots$. Clearly, σ' is a 1-delayed version of σ. Label each triangle t_i from the top, so that $t_1 = \{r_1, r_2, r_3\}$, $t_2 = \{r_3, r_4, r_5\}$, $t_3 = \{r_5, r_6, r_7\}, \ldots$. We will show that each triangle except the first, forces GREEDY on σ' to incur at least 3 times the cost of GREEDY on σ.

The cost to GREEDY of serving σ is the cost of the solid lines, and it gets to serve θ at zero cost. Hence $\text{GREEDY}(\sigma) = 2 + \frac{3}{2} + \frac{3}{2} + \cdots$. The cost to GREEDY of serving σ' is the cost of serving each of the dashed triangles. The first triangle costs 4, since both algorithms start at r_1, and each successive triangle is served at cost $\frac{9}{2}$, which is the cost of the dashed lines and the solid lines. Hence $\text{GREEDY}(\sigma') = 4 + \frac{9}{2} + \frac{9}{2} + \cdots$.

The lower bound of 3 follows, since the cost ratio is $\text{GREEDY}(\sigma') = 3 \cdot \text{GREEDY}(\sigma) - 2$, where the additive constant depends on the size of the first triangle $r1, r_2, r_3$. □

The following upper bound shows that the 'cost' of a 1-delayed network to GREEDY is 3, ie our bounds

are tight. Surprisingly, the adversarial network can force greedy to do no better than a minimum spanning tree, by building disjoint paths at each step. The main difficulty in analysing GREEDY is that Lemma 1 no longer holds for on-line Steiner tree constructions. Over σ, the algorithm could use a path $u \to v \rightsquigarrow w$, and so v would be in the tree constructed from σ but not in the tree constructed from σ'. Future requests could then utilise v, at zero cost to the algorithm on σ.

Theorem 6 *For the on-line minimum Steiner tree problem on infinite metric spaces, the competitive ratio of GREEDY on 1-delayed inputs is at most* 3 *times the competitive ratio of GREEDY on 0-delayed inputs. That is, GREEDY*$(\sigma') \leq 3$*GREEDY*$(\sigma) - \alpha$*, where* α *is some constant.*

Proof. The proof is based on a potential argument, where T_σ may include edges not in $T_{\sigma'}$, and these have a potential'. The existence of a potential function Φ with the following properties implies the result:

- The potential function is non-negative, $\Phi \geq 0$;
- Whenever GREEDY on σ incurs a cost of x, the potential increases by $\Delta\Phi \leq 3x$;
- Whenever GREEDY on σ' incurs a cost x, the potential decreases by $-\Delta\Phi \geq x$.

Informally, we want the potential to be the maximum cost that GREEDY on σ can be forced to incur, on a request sequence θ which GREEDY on σ' can serve at zero-cost.

We will show that the potential of a path $u \rightsquigarrow v$ is $d(u, v)$ if both endpoints u, v are in $T_{\sigma'}$. The idea is to construct a walk $u \rightsquigarrow v$ (abusing notation -fix) that forces GREEDY on σ to incur a cost of $d(u, v)$, even if already touches both endpoints, by repeatedly requesting the midpoint of the remaining edges. The sequence is to request the midpoint of u, v, say ω_1, incurring at least $\frac{1}{2}d(u, v)$. Assume it was served by adding the edges along $u \rightsquigarrow \omega_1$ to $T_{\sigma'}$. Then request the midpoint of v, ω_1, say ω_2, and so on. The total cost incurred is $d(u, v)(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots) = d(u, v)$.

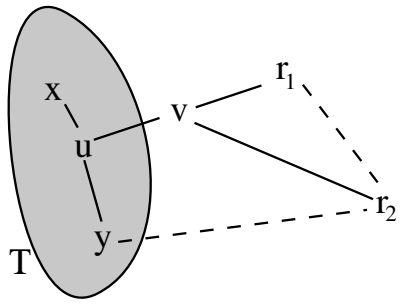

Figure 5: Illustrating Lemma 2.

The potential function is then naturally defined as the sum of all edges in $\mathcal{T} = T_\sigma - T_{\sigma'}$, that is $\Phi = \sum_{(u,v)\in\mathcal{T}} d(u, v)$.

Lemma 2 *Whenever GREEDY on* σ *incurs a cost of* x*, the potential increases by* $\Delta\Phi \leq 3x$*;*

Proof. Consider the situation as in Figure 5, and wlog let the edge (y, x) be the largest-cost edge in $\mathcal{T}$ (if it is not the largest-cost edge, the potential can only be increased by a smaller amount). Now consider the sequence σ, r_1, r_2, and let GREEDY on σ connect $(u, v), (v, r_1)$ and (v, r_2) (since v may be on the shortest path from u to r_1), as shown by the solid lines in the figure. It incurs the cost $d(u, v) + d(v, r_1) + d(v, r_2) \leq d(v, r_2) + d(u, r_1)$.

The request sequence $r_2, r_1, u \rightsquigarrow r_1, v \rightsquigarrow r_2$ will force GREEDY on σ' to incur maximum cost, at zero-cost to GREEDY on σ, and hence provides a bound on the potential Φ. Let the potential before ALG moves be $\Phi' + d(y, x)$, since we assumed that (y, x) was the largest edge in $\mathcal{T}$. There are two cases to consider:

Case 1 - GREEDY on σ increases the length of the largest edge in $\mathcal{T}$. Then, $d(u, r_1) \geq d(u, y)$ and so the new potential is given by

$$\begin{aligned}\Phi &= \Phi' + \big(3d(v, r_2) + 2d(u, r_1) + d(u, y)\big) + d(u, r_1)\\ &\leq \Phi' + 3\big(d(v, r_2) + d(u, r_1)\big) + d(u, y).\end{aligned}$$

The increase in potential is then given by

$$\begin{aligned}\Delta\Phi &\leq 3\left(d(v,r_2)+d(u,r_1)\right)+d(u,y)-d(y,x)\\ &\leq 3\left(d(v,r_2)+d(u,r_1)\right)-d(u,x)\\ &\leq 3\left(d(v,r_2)+d(u,r_1)\right).\end{aligned}$$

Case 2 - GREEDY on σ does not increase the length of the longest edge in $\mathcal{T}$. Assume wlog that $d(u,y) = d(u,x)$, since both endpoints are in $T_{\sigma'}$. By cutting the edge (y,x) at u, the potential is reduced by at most $\frac{1}{2}d(y,x)$, and leaves (u,y) as the longest edge. The new potential is given by

$$\begin{aligned}\Phi &= \Phi' + \big(d(y,r_2)+d(r_1,r_2)+d(u,v)+d(v,r_1)\\ &\quad +d(v,r_2)\big)+d(u,y)\\ &\leq \Phi' + 3d(v,r_2)+2d(u,v)+2d(v,r_1)+2d(u,y)\\ &\leq \Phi' + 3d(v,r_2)+2d(u,r_1)+2d(u,y).\end{aligned}$$

The increase in potential is then given by

$$\Delta\Phi \leq 3d(v,r_2)+2d(u,r_1)+2d(u,y)-d(y,x). \quad (1)$$

Since y is the closest vertex in $T_{\sigma'}$ to r_2 (by assumption), then we also have that $d(y,r_2) \leq d(v,r_2) + d(x,v) \leq d(v,r_2)+d(x,u)+d(u,v)$. A similar argument to the above shows that $\Phi \leq \Phi' + 3d(v,r_2) + 2d(u,r_1) + 2d(u,x)$, and so we also have that

$$\Delta\Phi \leq 3d(v,r_2)+2d(u,r_1)+2d(u,x)-d(y,x). \quad (2)$$

The increase in potential is bounded above by both (1) and (2), so by a balancing argument we can remove the assumption $d(u,y) = d(u,x)$ and so $\Delta\Phi \leq 3\big(d(v,r_2)+d(u,r_1)\big)$. □

Lemma 3 *Whenever GREEDY on σ' incurs a cost x, the potential decreases by $-\Delta\Phi \geq x$*

Proof. In both cases, GREEDY on σ' adds the paths $y \rightsquigarrow r_2$ and $r_1 \rightsquigarrow r_2$. Since these are disjoint from T_σ, then the length of the largest edge in $\mathcal{T}$ is unaltered. Hence the potential is decreased by at least $d(y,r_2)+d(r_1,r_2) \leq -\Delta\Phi$. □

The result follows, since the total potential remaining after any 1-delayed sequence σ' is bounded by at most 3 times the cost incurred by GREEDY on σ. □

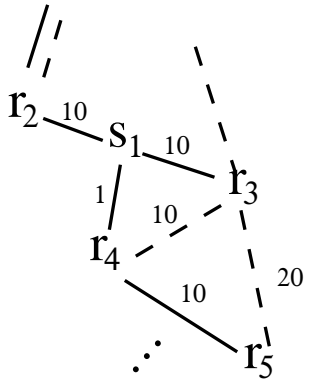

Figure 6: An example of a 1-delayed construction which can force GREEDY on σ' to incur > 3 times the cost to GREEDY on σ. The costs are $11+30 = 41$ and 11, respectively. The problem is that the construction can only be repeated with a cost ratio which exponentially approaches 3, as the longest edge $\in \mathcal{T}$ has halved.

The tricky part of the upper bound is in showing that the short-term gain of adding longer edges (since one can then build triangles that force a cost ratio of strictly greater than 3) is offset by the loss in potential, and sufficiently so, that it can be taken as a constant, rather than a fraction of the total cost.

For example, one can build triangles as in Figure 6 that force a cost ratio of strictly greater than 3, but this construction can only be repeated with an exponentially diminishing cost ratio, so some longer edges need to be added again. The cost of adding these edges does not justify the increased cost ratio of such a construction.

There are some interesting special cases of this upper bound, for example where we consider the two trees T_σ and $T_{\sigma'}$ (built by GREEDY on σ and its 1-delayed σ' respectively), but then let them both follow the same sequence θ after this point. By a similar (though simplified) argument, the total cost GREEDY on $\sigma'\theta$ is a 2-approximation to GREEDY on $\sigma\theta$. That is, the entire potential gained over σ' can be taken as a constant if the sequence θ is sufficiently long.

4 List Accessing

Suppose we have a linked list containing n items. We receive a request to access a particular item x, say. To find x, we must walk through the list until we find it, so if x is at position i we incur a cost of $O(i)$ for serving the request. After accessing the item, we may return it at to any position in the list, for free.[1] The aim of reorganizing the list is to reduce the search time in the future, for example if an item is likely to be requested soon, it is wise to reinsert it closer to the front. The problem is inherently on-line in that the request sequence is not known in advance, so the goal is to find an algorithm which minimizes the worst-case search costs.

The list accessing problem is relevant since the linked-list structure is often-used in practice, since it is simple and memory need not be pre-allocated contiguously. Furthermore, and our main interest in including this problem, is that on-line algorithms for the problem can be directly explocted to produce simple and efficient data compression schemes.

Before looking at some list accessing algorithms, it is useful to consider the following positive theorem, which is easy to see.

Theorem 7 *If ALG is c-competitive against OPT, then it is also c-competitive against itself with networks of arbitrary finite delay. Hence it is c^2-competitive against OPT (when OPT operates on the 0-delayed sequence).*

Proof. A network with arbitrary finite delays can produce any permutation of the input sequence σ. Fortunately, this is the same worst-case measure as the original competitive ratio between ALG and OPT, hence arbitrary permutations can elicit this worst-case behaviour. □

Although the theorem is a positive result, in the sense that competitiveness is a property of the algorithm and not the network it receives its requests via, this is not surprising given the pessimistic nature of the competitive ratio. It would be interesting to investigate a stochastic analogue of the theorem.

There are some interesting points not covered by the above theorem. For example, it is easy to see that for the MST problem, operating in a non-metric space, the GREEDY algorithm is not competitive against itself even with 1-delays (imagine the triangle of Figure 2 with edges of length 1,1 and an arbitrarily large hypoteneuse). Fortunately, this does not contradict the theorem since GREEDY is not competitive against OPT for non-metric spaces.

4.1 Algorithm Move-to-Front (MTF)

After accessing or inserting an item, move it to the front of the list, without changing the relative order of the other items.

We will count the cost of accesses in the following way, known as the partial cost model. When ALG accesses an item x at position i, it pays i. This counts the number of comparisons ALG makes while searching for x. Among these i comparisons, $i-1$ are with items different from x (called negative comparisons). The last comparison with x is called a positive comparison, and the number of such comparisons is the same for all algorithms. It makes sense then to ignore the number of positive comparisons, and only count negative comparisions, known as the partial cost model. The model where all comparisons are counted is the full cost model.

Theorem 8 *For the on-line static list accessing problem, the competitive ratio of MTF on 1-delayed inputs is at least 2 times the competitive ratio of MTF on 0-delayed inputs.*

Proof. Consider the initial list x, y of length 2. We construct a cruel pair of sequences σ and a 1-delayed version σ'. Let $\sigma = x, y, y, x, \ldots$ and $\sigma' = y, x, y, x, \ldots$. On σ, the first element is always accessed exactly once, then the second element, for a cost of 2 for each phase x, y, y, x of length 4. Serving σ' forces MTF to always access the second element, incurring a cost of 2 for each phase y, x of length 2. Hence, $\mathrm{MTF}(\sigma') \leq 2 \cdot \mathrm{MTF}(\sigma)$ for all 1-delayed σ'. □

[1] Actually, we are only interested in algorithms which return it to either adjacent nodes or the head or tail of the list, in which case this operation would be free in a linked-list implementation.

Theorems 7 and 8 together immediately imply that for 1-delayed networks, 2 is an upper bound of the ratio of MTF(σ') to MTF(σ), since we know that MTF is 2-competitive against OPT. Furthermore, since 1-delays are a special case of arbitrary delays, this bound must also be tight for networks with arbitrary finite delays.

It is a positive result that, even under arbitrary delays, the competitive ratio of MTF to OPT (on 0-delays) is asymptotically unchanged, but a negative result in that this worst-case can be easily achieved with just 1-delays (pairwise permutations).

In the full cost model, MTF is 2-competitive against OPT, but the above argument appears to give a ratio of only $\frac{4}{3}$, but the sequences $\sigma = x, y, y$ and $\sigma' = y, x, y$ give $\frac{3}{2}$, and it appears that $\frac{3}{2}$ is a tight bound, although the best upper bound is $\frac{5}{3}$.

Theorem 9 *The competitive ratio of MTF on 1-delayed inputs is at most $\frac{5}{3}$ times the competitive ratio of MTF on 0-delayed inputs in the full cost model.*

Proof. Assume that both algorithms start with the same list. We need to consider two types of event partitioning the request sequences. The first is where two elements are swapped, and the second is the usual case of a request to a single element. Hence we need to show that $\mathrm{MTF}_{\sigma'}(y,x) + \Delta\Phi \leq \frac{5}{3}\mathrm{MTF}_{\sigma}(x,y)$ and $\mathrm{MTF}_{\sigma'}(x) + \Delta\Phi \leq \frac{5}{3}\mathrm{MTF}_{\sigma}(x)$, where Φ_i is a potential function with $\Phi_0 = 0$ and $\Phi_i \geq 0$ for all i. Let Φ_i be the number of inversions in $\mathrm{MTF}_{\sigma'}$'s list with respect to MTF_{σ}'s list, just after both have finished processing the ith event (noting that an event may be a pair of permuted requests).

We first consider the simple case of a single request x to both algorithms. Consider the lists before the request. We will view the lists (configurations) by considering the inversions that we are interested in. There are three types of inversions - $\triangle$-inversions precede x in $\mathrm{MTF}_{\sigma'}$'s list but are behind x in MTF_{σ}'s list, and the opposite for $\triangledown$-inversions. $\square$-inversions are those elements which precede x in both lists (and so are not inversions in the initial configuration). We first consider the simple case of a single request x to both algorithms. Consider the lists before the request to x. These initial configurations can be viewed as

$$\begin{aligned} \mathrm{MTF}[\sigma'] &: \{\square, \triangle\}, x, \{\triangledown\} \\ \mathrm{MTF}[\sigma] &: \{\square, \triangledown\}, x, \{\triangle\} \end{aligned}$$

Now let $\mathrm{MTF}_{\sigma'}$ serve x. All the $\triangle$-inversions are destroyed, but the $\square$-inversions are created. When MTF_{σ} serves x, all the $\square$-inversions and $\triangledown$-inversions are destroyed, giving the intermediate configurations

$$\begin{aligned} \mathrm{MTF}[\sigma', x] &: x, \{\square, \triangle, \triangledown\} \\ \mathrm{MTF}[\sigma, x] &: x, \{\square, \triangledown, \triangle\} \end{aligned}$$

Hence, all inversions wrt x (i.e. $\{\triangle, \triangledown\}$-inversions) are destroyed after both algorithms have served x, which is to be expected since moving an element to the front preserves the relative order of the rest of the list.

Let k_x be the number of items preceding x in both MTF_{σ} and $\mathrm{MTF}_{\sigma'}$'s lists (and similarly, k_y for y). Let l_x be the number of items preceding x only in $\mathrm{MTF}_{\sigma'}$'s list (and similarly, l_y for y). Then, the cost to $\mathrm{MTF}_{\sigma'}$ is $k_x + l_x + 1$, and the cost to MTF_{σ} is at least $k_x + 1$. Since the number of $\triangle$-inversions is l_x (the number of $\triangledown$-inversions is ≥ 0), we have

$$\begin{aligned} \mathrm{MTF}_{\sigma'}(x) + \Delta\Phi &\leq k_x + l_x + 1 \\ &\quad -(\triangle\text{-inversions}) - (\triangledown\text{-inversions}) \\ &\leq k_x + 1 \\ &\leq \mathrm{MTF}_{\sigma}(x). \end{aligned}$$

This case also implies that, for serving the same sequence σ, MTF incurs no penalty (in the competitive ratio against itself) for starting on arbitrarily different lists, so it might be said to be 'self-synchronizing'.

Next, we consider the case where two elements are served in alternate order. There are two subcases, depending on whether the pair of elements $\{x, y\}$ is an inversion in the configurations prior to the requests.

Subcase 1 - $\{x, y\}$ is an inversion. We will consider the sequences σ, y, x and σ', x, y. Assume wlog that y precedes x in $\mathrm{MTF}_{\sigma'}$'s list (otherwise we can request σ, x, y and σ', y, x). Let us consider the inversions wrt x. The configurations can be viewed as

$$\begin{aligned} \mathrm{MTF}[\sigma'] &: \{\square, \triangle\}, y, \{\square, \triangle\}, x, \{\triangledown\} \\ \mathrm{MTF}[\sigma] &: \{\square, \triangledown\}, x, \{\triangle\}, y, \{\triangle\} \end{aligned}$$

$\triangle$-inversions are elements preceeding x in $\mathrm{MTF}_{\sigma'}$'s list, but are behind x in MTF_{σ}'s list, and the opposite for $\triangledown$-inversions.

When $\mathrm{MTF}_{\sigma'}$ moves x to the front, it destroys all $\triangle$-inversions and creates $\square$-inversions. The action of MTF_{σ} on y has no effect on all the $\{\square, \triangle, \triangledown\}$-inversions wrt x since the relative order of remaining elements is preserved. The intermediate configurations are now

$$\begin{aligned} \mathrm{MTF}[\sigma', x] &: x, \{\square, \triangle\}, y, \{\square, \triangle, \triangledown\} \\ \mathrm{MTF}[\sigma, y] &: y, \{\square, \triangledown\}, x, \{\triangle\} \end{aligned}$$

When MTF_{σ} moves x to the front of its list, it destroys all $\{\square, \triangledown\}$-inversions, and we have the final configurations

$$\begin{aligned} \mathrm{MTF}[\sigma', x, y] &: y, x, \{\square, \triangle, \triangledown\} \\ \mathrm{MTF}[\sigma, y, x] &: x, y, \{\square, \triangle, \triangledown\} \end{aligned}$$

A similar argument considering the inversions wrt y shows that all the inversions wrt x (call this Φ^x) and wrt y (Φ^y) are destroyed, and that the inversion $\{x, y\}$ is preserved. Since the number of $\triangle$-inversions is l_x (the number of $\triangledown$-inversions is ≥ 0), this gives $\Delta\Phi = \Phi^x + \Phi^y \leq -(l_x + l_y)$.

The cost to MTF_{σ} of serving y, x is at least $(k_x + 1) + (k_y + 2)$, since moving x to the front pushes y back by one place. Similar reasoning shows that the cost to $\mathrm{MTF}_{\sigma'}$ of serving x, y is $(k_x + l_x + 1) + (k_y + l_y + 2)$. Hence

$$\begin{aligned} \mathrm{MTF}_{\sigma'}(x, y) + \Delta\Phi &\leq (k_x + k_y + l_x + l_y + 4) \\ &\quad -(l_x + l_y) \\ &= k_x + k_y + 4 \\ &\leq \frac{5}{3}\mathrm{MTF}_{\sigma}(y, x). \end{aligned}$$

The last step follows since it must be that $k_x + k_y \geq 1$, since it is impossible for both x and y to be at the front of a list (and to both be served at cost 1). Hence

$$\begin{aligned} \frac{5}{3}\mathrm{MTF}_{\sigma}(y, x) &\geq (k_x + k_y) + \frac{2}{3}(k_x + k_y) + 5 \\ &\geq (k_x + k_y) + \frac{17}{3}. \end{aligned}$$

Subcase 2 - $\{x, y\}$ is not an inversion. We will consider the sequences σ, x, y and σ', y, x. We can assume that x preceeds y in both lists, as the alternative is dominated by this case (since MTF_{σ} would do strictly worse on x, y than $\mathrm{MTF}_{\sigma'}$ on y, x). Let us consider the inversions wrt x. The configurations can then be viewed as

$$\begin{aligned} \mathrm{MTF}[\sigma'] &: \{\square, \triangle\}, x, \{\triangledown\}, y, \{\triangledown\} \\ \mathrm{MTF}[\sigma] &: \{\square, \triangledown\}, x, \{\triangle\}, y, \{\triangle\} \end{aligned}$$

When MTF_{σ} moves x to the front, it destroys all $\triangle$-inversions and creates $\square$-inversions (all inversions are with respect to x). The intermediate configurations are now

$$\begin{aligned} \mathrm{MTF}[\sigma', y] &: y, \{\square, \triangle\}, x, \{\triangledown\} \\ \mathrm{MTF}[\sigma, x] &: x, \{\square, \triangle, \triangledown\}, y, \{\triangle\} \end{aligned}$$

When $\mathrm{MTF}_{\sigma'}$ moves x to the front, it destroys all $\square$-inversions and all $\triangledown$-inversions, but $\{x, y\}$ is a new inversion and we have the final configurations

$$\begin{aligned} \mathrm{MTF}[\sigma', y, x] &: x, y, \{\square, \triangle, \triangledown\} \\ \mathrm{MTF}[\sigma, x, x] &: y, x, \{\square, \triangle, \triangledown\} \end{aligned}$$

A similar argument considering the inversions wrt y shows that all the inversions wrt x (call this Φ^x) and wrt y (Φ^y) are destroyed, and that the single inversion $\{x, y\}$ is created. Hence $\Delta\Phi = \Phi^x + \Phi^y + 1 \leq -(l_x + l_y) + 1$.

The cost to MTF_{σ} of serving x, y is at least $(k_x + 1) + (k_y + 1)$. The cost to $\mathrm{MTF}_{\sigma'}$ of serving y is $k_y + l_y + 1$ (as before), but this pushes x back by one, so the cost of serving x is $k_x + k_x + 2$. Hence

$$\begin{aligned} \mathrm{MTF}_{\sigma'}(y, x) + \Delta\Phi &\leq (k_x + k_y + l_x + l_y + 3) \\ &\quad -(l_x + l_y) + 1 \\ &= k_x + k_y + 4 \\ &\leq \frac{5}{3}\mathrm{MTF}_{\sigma}(x, y). \end{aligned}$$

To see the last step, it must be that $k_x + k_y \geq 1$, since it is impossible for both x and y to be at the front of a list (and to both be served at cost 1). Hence

$$\begin{aligned} \frac{5}{3}\mathrm{MTF}_{\sigma}(x, y) &\geq (k_x + k_y) + \frac{2}{3}(k_x + k_y) + \frac{10}{3} \\ &\geq (k_x + k_y) + 4. \end{aligned}$$

□

As a verification of of Theorem 7, the following argument shows that MTF is a 2-approximation of itself for arbitrary delays. Let the list be $x_1, x_2, \ldots, x_l$ initially. Then $\mathrm{MTF}(x_1, x_2, \ldots, x_l) = \frac{l(l+1)}{2}$ and $\mathrm{MTF}(x_l, x_{l-1}, \ldots, x_1) = l^2$. For a sufficiently large number of insertions, the ratio is $\lim_{l\to\infty} 2\frac{l}{l+1} = 2$.

4.2 Algorithm Transpose (TRANS)

After accessing or inserting an item, transpose (swap) it with the immediately preceding item.

We know that TRANS is not competitive for the dynamic list accessing problem, but this does not necessarily imply that it cannot achieve a constant factor approximation in the face of delayed networks. Unfortunately, TRANS cannot remain competitive with itself even with 1-delays (better: it cannot approximate itself?), as illustrated by the following result.

Theorem 10 *Algorithm TRANS is not competitive with itself on 1-delayed networks.*

Proof. The proof relies on the following observation. Consider the list x_1, x_2, x_3. Given the request sequence x_2, x_3, TRANS produces the list x_2, x_3, x_1, i.e. the pair x_2, x_3 has been 'shuffled' one place to the front. Now consider the 1-delayed request sequence x_3, x_2. TRANS transposes x_3 and x_2 twice, leaving the list unchanged. Hence this can be repeated to shuffle a pair of elements to the front, while leaving them at the back for TRANS on σ'.

Consider the list $x_1, x_2, \ldots, x_l$. The cruel request sequence consists of two phases. In the first phase, the last two elements x_{l-1} and x_l are 'shuffled' to the front of the list to obtain $x_{l-1}, x_l, x_1, \ldots, x_{l-2}$, using the construction above. In the second phase, TRANS on σ repeatedly serves $x_{l-1}, x_l, x_l, x_{l-1}, \ldots$ at a cost of 2 for each quartet of requests (in effect, it behaves as OPT). Throughout both phases, the 1-delayed sequence $\sigma' = x_l, x_{l-1}, \ldots$ is served at a cost of $2(l-1)$ for each pair of requests. It is easy to check that σ' is indeed a 1-delayed version of both phases of σ. The cost ratio for the second phase is then $4(l-1)/2 = 2(l-1)$.

The cost of the shuffle phase is absorbed into the additive constant, but for completeness we analyse its cost. Each shuffle of elements at positions $j-1$ and j costs $2j-3$. Hence the total cost for the $l-2$ shuffles is

$$\begin{aligned}\sum_{i=1}^{l-2} 2l - (2i+1) &= 2\left(\sum_{i=1}^{l-2} l - \sum_{i=1}^{l-2} i\right) - (l-2)\\ &= l(l-2)\end{aligned}$$

During the shuffle phase, TRANS on σ' incurs a cost of $2(l-1)(l-2)$, for a cost ratio of $2(l-1)/l = 2\left(1-\frac{1}{l}\right)$. Of course, the second phase forces this constant cost ratio to be taken into the additive constant.

Since there is no a priori bound on the length of the list, the above argument, coupled with a sufficiently long sequence of initial insertions, establishes the result. □

5 Other models

It would be good to investigate alternative, less adversarial models for the network. Two ideas seem particularly interesting:

Distributional. This currently considers what happens when the network chooses the links adversarially. But what about having a probability distribution on the links?

Nash equilibria. Real-world networks are not adversarial, rather their behaviour is determined by many individual agents, each wishing to minimise its own latency. The stable behaviour of the network is then defined by the Nash equilibria. Indeed, the Nash equilibria also arise via distributed shortest-path algorithms. These Nash equilibria are precisely the fixed points of the shortest-path protocol in which all nodes of the network define the length of their incident edges as their current delay (the shortest paths computed are a function of how routers define edge length). What is the cost to the on-line algorithm of receiving a sequence sent through a network at a worst-case Nash equilibrium?