Notes on MTS

BY SAMBUDDHA ROY

Paper: An Optimal Online Algorithm for Metrical Task System

Authors: Allan Borodin, Nathan Linial, Michael Saks

Appears in: Journal of the ACM, 1992.

What they do:

They introduce the metrical task system (MTS) problem. This problem is an online problem. They show that in a deterministic setting, one may derive a *tight* $(2n-1)$ competitive ratio online algorithm.

However, when the algorithm is allowed to be randomized, they show that there is an *exponential* improvement: they give a $2 \cdot \log n$ factor competitive ratio algorithm for the MTS problem on a *uniform* metric, that has n points. They show correspondingly that this algorithm is nearly optimal: they exhibit a lower bound of $\log n$ for the MTS problem.

Definitions:

There will be n points in the (uniform) metric space. Given points i and j in the metric space, there is a well defined *distance* $d(i,j)$ that is 1 if $i \neq j$ and 0 otherwise.

The cost functions arrive in an online fashion: the cost function arriving at time t will be denoted as $c_t(\cdot)$. At any time t, when the cost function c_t arrives, the online algorithm has to make a decision as to whether to stay in the current state or to *change* state. Thus, the output of the online algorithm is a sequence of states for each of the timepoints that arrive. Let the time horizon be denoted as the interval from 1 to T, and let the cost functions be $c_1, \cdots, c_T$.

The algorithm's output is $\sigma(1), \sigma(2), \cdots, \sigma(T)$ where $\sigma(t)$ is the *state* the algorithm chooses to be in after seeing the cost function c_t (and thus all the cost functions $c_1, c_2, \cdots, c_t$). The overall **cost** of the algorithm is $\sum_t c_t(\sigma(t)) + \sum_t d(\sigma(t-1), \sigma(t))$. The cost component $\sum_t c_t(\sigma(t))$ is called the ***service cost*** and the component $\sum_t d(\sigma(t-1), \sigma(t))$ is called the ***movement cost***.

The online algorithm has to *compete* against the optimal *offline* algorithm that has entire visibility of the input.

In this writeup, as in their paper, let $H(n)$ denote the n^{th} Harmonic number, i.e.

$$\begin{aligned} H(n) &= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\ &\approx \ln n \end{aligned}$$

In the following we will only describe the upper and lower bounds for the randomized algorithm.

The Upper Bound:

Given a uniform metric, they give an online algorithm with an (expected) competitive ratio of at most $2H(n)$.

Their algorithm uses "continuous" time; however they then show that not much is lost via *discretizing* this continuous algorithm.

The overall algorithm is a randomized choice, at each time point, between various deterministic strategies. The deterministic strategies they consider are simple: the k^{th} deterministic strategy (for k between 1 and n) always chooses the k^{th} metric point. Call this algorithm A_k. To abuse notation, we will also refer to the algorithm A_k by the corresponding metric point k.

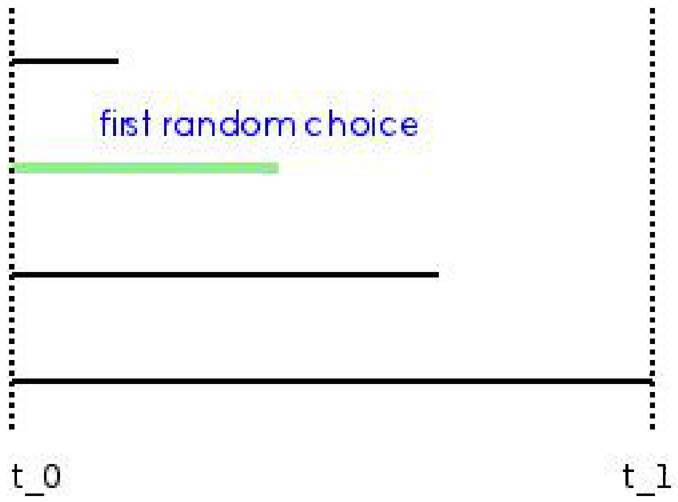
Let t_0 be the beginning, and the algorithm proceeds in *phases*. We define a phase as below.

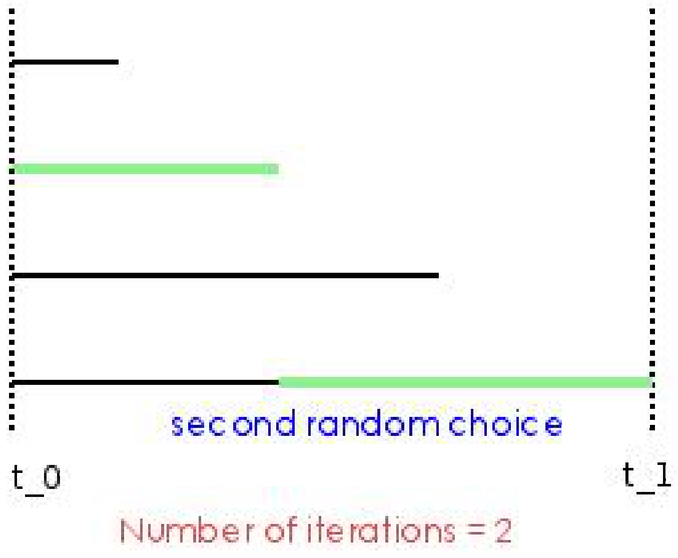
Phase i starts at time t_{i-1} and ends at time t_i. These times are defined inductively.

Consider the starting time t_{i-1}, and an algorithm A_k. Starting from time t_{i-1} until time t, the cumulative cost A_k pays against the *online* cost functions is $\sum_{j=t_{i-1}}^{t} c_j(k)$. Define $T(k)$ to be the (continuous) time-point t where this sum reaches 1. Note that this is *not* a *static* time-point; it depends entirely on the cost functions c_j's that arrive. Thus

$$\sum_{j=t_{i-1}}^{T(k)} c_j(k) \;=\; 1$$

Now, for each metric point k, there is such a time-point $T(k)$. Let t_i denote the *maximum* of these time-points $T(k)$ across all metric points. This ends the description of the phase between t_{i-1} and t_i. Also we say that A_k is "**saturated**" by time-point $T(k) \in [t_{i-1}, t_i]$.





Note that between t_{i-1} and t_i, the overall cost of every algorithm A_k is at least 1.

The online algorithm is as follows: follow an algorithm A_k till it is saturated (i.e. till time-point $T(k)$). At the time of saturation, there are some metric points that are already saturated, and some that are as yet unsaturated. We are at a juncture where the algorithm A_k that we were following turns out to be quite *expensive* already. Thus, we would like to change our current algorithm (i.e. state) by paying a movement cost. Each of the unsaturated metric points α have some partial cost i.e. $\sum_{j=t_{i-1}}^{T(k)} c_j(\alpha) < 1$; thus for instance, we may decide to move to the metric point α for which this quantity is the *least*. The rationale behind the "least" is so that we get to stay at the new metric point α for the longest time, while having paid not much cost as of now. However, given such a deterministic move, the adversary can fool us by making this specific α quite costly in the next few $c_j(\cdot)$'s thereby forcing us again to move from the metric point α. To hedge against this, the algorithm employs randomization. The algorithm jumps to a *random* unsaturated metric point.

Thus when the current metric point gets saturated, look at all the (currently) unsaturated algorithms A_r and *randomly* jump to one such algorithm and continue. See figure above.

At the end of this phase, when the overall saturation point t_i is reached, all the n algorithms A_k $(1 \leqslant k \leqslant n)$ are labelled unsaturated. The online algorithm starts afresh: it chooses a random algorithm A_r, and proceeds.

First, we observe a simple *lower bound* on the offline optimum.

Note 1. Any offline algorithm incurs a cost of at least 1 in a phase.

This is because any offline algorithm either makes a transition in the phase or not. If it makes a transition it incurs a movement cost of at least 1. If it does not make a transition, then it ends at t_i and pays an overall service cost of 1: this is because any algorithm A_k pays a service cost of at least 1 by the time-point t_i.

Lemma 2. *In a phase, the online algorithm incurs a cost of at most* $2H(n)$.

Proof. Denote by $f(k)$ the *expected* number of transitions until the end of the phase given that there are k unsaturated states remaining. Thus, if $k=1$, then $f(k)=1$.

The authors prove that $f(k)=H(k)$. Once this is proven the statement of the lemma is clear; the cumulative movement costs is upper bounded by $H(n)$. But by the nature of a phase, this implies that the total service cost in a phase is upper bounded by the cumulative movement cost in the phase. Thus, an overall bound of $2H(n)$ is achieved. The competitive ratio in a single phase is thereby $2H(n)$; summing over all the phases, we get the overall competitive ratio of $2H(n)$.

We now focus on the fact that $f(k)=H(k)$; this is actually a nice puzzle. We proceed to remodel this:

A puzzle:

The problem is this: at the beginning we have n numbers $1, 2, \cdots, n$ in sequence, and call these the numbers "surviving". In any *iteration* the following action is taken. We pick a random number r from the surviving numbers. Having done this, all the numbers to the *left of* (*and including)* the number r in the sequence are proclaimed *dead*. What is the *expected* number of *iterations* before the game ends (i.e. every number in the sequence is proclaimed dead)?

Some coarse intuition:

Suppose we were deterministic and went "binary": i.e. went to $n/2$ in the first step, then to $n/2+n/4$ in the second step, and so on, we would get $\log_2 n$ steps. In the puzzle, the process is random. But the binary scheme is a good *proxy* for the *expected behavior* of the random sequence of choices? Why is this? When we *randomly* choose (say in the very first step), the expected number of deaths is *half* the total number of surviving members, so it is as if we were going binary.

Formally...

However the above is just coarse, since the expectation we want is over the number of *iterations*. We claim that $f(k) = f(k-1) + 1/k$. The paper's nifty reasoning is the following. When there are only k surviving numbers, picking the very next surviving member happens with probability $1/k$, and then we have $k-1$ surviving members. Thus, we get that $f(k) = f(k-1) + \frac{1}{k}$, which leads to $f(k) = H(k)$.

However, if one is still uncomfortable with the above proof, the bruteforce proof is as follows. For computing $f(k)$, we know that in ***one*** more transition (which happens with probability $1/k$) we can get to $f(r)$, for $1 \leqslant r \leqslant k-1$. Thus, the expected value $f(k)$ satisfies:

$$\begin{aligned} f(k) &= \frac{(f(k-1)+1) + (f(k-2)+1) + \cdots + (f(1)+1)}{k} \\ &= \frac{k-1}{k} + \frac{f(k-1) + f(k-2) + \cdots + f(1)}{k} \end{aligned}$$

This simplifies to

$$k \cdot f(k) = (k-1) + [f(k-1) + f(k-2) + \cdots + f(1)]$$

Writing the same expression for $k-1$:

$$(k-1) \cdot f(k-1) = (k-2) + [f(k-2) + \cdots + f(1)]$$

and subtracting:

$$\begin{aligned} k \cdot f(k) - (k-1) \cdot f(k-1) &= 1 + f(k-1) \\ k \cdot f(k) - k \cdot f(k-1) &= 1 \\ i.e. f(k) &= f(k-1) + \frac{1}{k} \end{aligned}$$

This concludes the puzzle and the proof. □

Remarks:

One can consider the coarse intuition of the puzzle and ask: why doesn't the algorithm directly go after the middle. But then our algorithm is randomized, and since the cost functions are online, it doesn't have knowledge as to which specific metric point or algorithm A_k is the "middle". We make up for this lack of knowledge via randomization.

Notice that this framework already uses phases, and does the accounting phase by phase. This is the same as in the online primal dual framework where we compute ΔP_t and ΔD_t for the primal and dual in a single timeslot.

The Lower Bound:

We are attempting to lower bound the competitive ratio of randomized algorithms. There are certain ingredients in this regard.

- Yao's minimax principle. This is used commonly to analyze randomized online algorithms, and is expected here.

- Provide a probability distribution on the system of tasks. Given Yao's principle, this is analogous to the "gap example" in common parlance.

- Some facts from Renewal Theory. These arise via generalizing Poisson processes, with what are called arbitrary holding times. This arises from trying to prove specific properties of the distributions used in the proof.

Yao's Minimax Principle:

Yao's minimax principle is an easy corollary of Von Neumann's Minimax Theorem and says that for lower bounding the *expected* competitive ratio of a randomized algorithm, it suffices to consider deterministic algorithms with randomized inputs, taken from a specific distribution. Thus in the current context, the sequence of tasks that arrive will be *random* - we will specify the distribution soon.

The Distribution:

Given the n states, denote a task/job as J_k (for $1 \leqslant k \leqslant n$) where J_k has *cost* 1 at metric point k and 0 everywhere else. Call these tasks (i.e. the online *cost functions*) as "**unit** tasks". The distribution of tasks is as follows: at any time point, a random such unit task is chosen and is the next online task. This infinite task system will be denoted as $\mathcal{J}$. Also, let $\mathcal{J}_r$ denote the first r tasks of the system $\mathcal{J}$.

They start with an useful (and natural) lemma:

Lemma 3. *Given a task system and a probability distribution $\mathcal{D}$ on the set of infinite task sequences. In the following, we will take expectation wrt this distribution $\mathcal{D}$. Let $\mathbb{E}[c_O(\mathcal{J}_r)]$ denote the expected cost that* OPT *pays for the tasks in $\mathcal{J}_r$. Also, let m_r denote the minimum over all deterministic algorithms $\mathcal{A}$ of $\mathbb{E}[c_{\mathcal{A}}(\mathcal{J}_r)]$. Then the competitive ratio for the MTS problem has a lower bound of* $\limsup_{r\to\infty} \frac{m_r}{\mathbb{E}[c_O(\mathcal{J}_r)]}$.

In the following, we will *lower bound* m_r and *upper bound* $\mathbb{E}[c_O(\mathcal{J}_r)]$.

Given a deterministic algorithm $\mathcal{A}$ and a random collection of tasks $\mathcal{J}_r$ how do we estimate $c_{\mathcal{A}}(\mathcal{J}_r)$? The tasks correspond to the specific state that it indicates has cost 1 (exactly one state gets a cost of 1, and the other states incur a cost of 0). Notationally, at time t, let the algorithm $\mathcal{A}$ have state $\sigma(t)$. If the newest task s_t is such that $\sigma(t-1) = s_t$, then after the t^{th} cost function (i.e. the "task" s_t) has come along, the algorithm $\mathcal{A}$ has one of two options: if it stays in the same state *i.e.* $\sigma(t) = \sigma(t-1) = s_t$, it incurs a *service cost* of 1; if it *moves* from the state s_t to some other in order to prevent this service cost, it incurs a *movement cost* of 1. Eitherway, if $\sigma(t-1) = s_t$, the algorithm $\mathcal{A}$ pays a cost of 1. This event $\sigma(t-1) = s_t$ happens with a probability of $\frac{1}{n}$. Thus the expected overall cost $\mathbb{E}[c_{\mathcal{A}}(\mathcal{J}_r)] = \frac{r}{n}$.

The next step is the estimation of $\mathbb{E}[c_O(\mathcal{J}_r)]$. We have a very specific distribution of random tasks, and we have to estimate the *optimal offline* solution for this distribution. In this regard, the paper gives the following strategy. As above, we realise that the only time when we have to make a change or take a decision is when $\sigma(t-1) = s_t$, in short this is the only time when we have to pay a cost. When this happens, we would rather move to a new point - the question being *which new point*? Since $\sigma(t-1) = s_t$, every point other than s_t has cost 0 in the task at time t.

The overall motto for a potential algorithm tailored for this distribution is that if possible, *we will not want to pay any service cost, but rather pay movement costs.*

An Example:

Suppose we had 3 metric points $1, 2, 3$. Suppose also the random sequence of tasks was $1, 1, 2, 1, 2, 1, 3, 1, 2$. What would be a good way to go for the offline solution? The offline solution would want to start at 2 or 3 (so as to obviate the need to pay a cost of 1 by choosing 1). Suppose the solution chooses to start at 2. Then before the third time-point (i.e. when 2) comes along), the algorithm would need to change its state. Note the sequence $1, 1, 2, 1, 2, 1, 3, 1, 2$. If the algorithm changes its state to 3, then in the next 5 moves (indicated by the **red** part above), the algorithm needs to pay only *once*. This is because 3 does not appear anywhere in the **red** sequence apart from at the end. Conversely, suppose we are in a situation where $\sigma(t-1) = s_t$ and there is a subsequence of tasks following where every task (i.e. every state) appears at least once. Is it possible for any algorithm, given this initial condition (i.e. $\sigma(t-1) = s_t$) to not pay any cost on the subsequence? It is clear that this is not possible; every algorithm has to pay at least once. Our algorithm is designed to meet primarily this (implicit) lower bound.

With this preparation, we state the strategy (which in fact is optimal, but this fact is not necessary).

Offline Algorithm:

Given a time point t, let $a(t)$ be the *least* time-point such that $a(t) > t$ and such that $s_{t+1}, s_{t+2}, \cdots$, $s_{a(t)}$ contain *all* the states $1, 2, \cdots, n$. If $\sigma(t-1) \neq s_t$, then let $\sigma(t) = \sigma(t-1)$. If $\sigma(t-1) = s_t$, then let $\sigma(t) = s_{a(t)}$. Recall that here, we are considering an *offline* algorithm: thus, the algorithm has full visibility as to what happened at timepoints $t+1, t+2, \cdots$.

Analysis:

Interestingly, as we considered the offline algorithm, the rough picture is like this: the task sequence is dealt with in phases, where a phase consists of a subsequence where all the states appear ***at least once***, and it is ***minimal*** in this regard. "Minimal" in this context means that we are considering a subsequence where (1) all the states appear, and (2) the last state in the sequence appears exactly once. Now, this is precisely what Renewal Theory (see the wiki page *http://en.wikipedia.org/wiki/Renewal_ theory* deals with; and such phases/subsequences are what are called ***holding*** times.

By design, the algorithm pays a cost of 1 exactly when $\sigma(t-1) = s_t$. Let c_t be the cost incurred up to time t and let X_p be the time at which the total cost incurred reaches p. Then $c_t = \max\{p: X_p \leqslant t\}$. Note that the quantities $Y_p = X_p - X_{p-1}$ are *independent identically distributed* random variables; these Y_p's are the holding times. The paper uses the Elementary Renewal Theorem to prove that $\mathbb{E}[Y_p] = n \cdot H(n)$. Thus, the total expected cost $\mathbb{E}[c_O(\mathcal{J}_r)] \approx \frac{r}{n \cdot H(n)}$.

See the paper for full details of this last part.

Overall this completes the description of the $H(n)$ lower bound on the competitive ratio.