CSE 291: Unsupervised learning **Spring 2008**

Lecture 6 — Online and streaming algorithms for clustering

6.1 On-line k-clustering

To the extent that clustering takes place in the brain, it happens in an *on-line* manner: each data point comes in, is processed, and then goes away never to return. To formalize this, imagine an endless stream of data points $x_1, x_2, \ldots$ and a k-clustering algorithm that works according to the following paradigm:

```
repeat forever:
   get a new data point x
   update the current set of k centers
```

This algorithm cannot store all the data it sees, because the process goes on ad infinitum. More precisely, we will allow it space proportional to k. And at any given moment in time, we want the algorithm's k-clustering to be close to the optimal clustering of all the data seen so far.

This is a tall order, but nonetheless we can manage it for the simplest of our cost functions, k-center.

6.2 The on-line k-center problem

The setting is, as usual, a metric space $(\mathcal{X}, \rho)$. Recall that for data set $S \subset \mathcal{X}$ and centers $T \subset \mathcal{X}$ we define $\text{cost}(T) = \max_{x \in S} \rho(x, T)$. In the on-line setting, our algorithm must conform to the following template.

```
repeat forever:
   get x ∈ X
   update centers T ⊂ X, |T| = k
```

And for all times t, we would like it to be the case that the cost of T for the points seen so far is close to the optimal cost achievable for those particular points. We will look at two schemes that fit this bill.

6.2.1 A doubling algorithm

This algorithm is due to Charikar, Chekuri, Feder, and Motwani (1997).

```
        T ← {first k distinct data points}
        R ← smallest interpoint distance in T
        repeat forever:
           while |T| ≤ k:
(A)           get new point x
              if ρ(x,T) > 2R : T ← T ∪ {x}
(B)        T' ← { }
           while there exists z ∈ T such that ρ(z,T') > 2R:
              T' ← T' ∪ {z}
           T ← T'
(C)        R ← 2R
```

Here $2R$ is roughly the cost of the current clustering, as we will shortly make precise. The lemmas below describe invariants that hold at lines (A), (B), and (C) in the code; each refers to the *start of the line* (that is, *before* the line is executed).

Lemma 1. *All data points seen so far are (i) within distance $2R$ of T at (B) and (ii) within distance $4R$ of T at (C).*

Proof. Use induction on the number of iterations through the main loop. Notice that (i) holds on the first iteration, and that if (i) holds on the pth iteration then (ii) holds on the pth iteration and (i) holds on the $(p+1)$st iteration. □

To prove an approximation guarantee, we also need to lower bound the cost of the optimal clustering in terms of R.

Lemma 2. *At (B), there are $k+1$ centers at distance $\geq R$ from each other.*

Proof. A similar induction. It is easiest to simultaneously establish that at (C), all centers are distance $\geq 2R$ apart. □

We now put these together to give a performance guarantee.

Theorem 3. *Whenever the algorithm is at (A),*

$$\text{cost}(T) \leq 8 \cdot \text{cost}(\text{optimal } k \text{ centers for data seen so far}).$$

Proof. At location (A), we have $|T| \leq k$ and $\text{cost}(T) \leq 2R$.

The last time the algorithm was at (B), there were $k+1$ centers at distance $\geq R/2$ from each other, implying an optimal cost of $\geq R/4$. □

Problem 1. A commonly-used scheme for on-line k-means works as follows.

```
initialize the k centers t_1, ..., t_k in any way
create counters n_1, ..., n_k, all initialized to zero
repeat forever:
   get data point x
   let t_i be its closest center
   set t_i ← (n_i t_i + x)/(n_i + 1) and n_i ← n_i + 1
```

What can be said about this scheme?

6.2.2 An algorithm based on the cover tree

The algorithm of the previous section works for a pre-specified value of k. Is there some way to handle all k simultaneously? Indeed, there is a clean solution to this; it is made possible by a data structure called the *cover tree*, which was nicely formalized by Beygelzimer, Kakade, and Langford (2003), although it had been in the air for a while before then.

Assume for the moment that all interpoint distances are ≤ 1. Of course this is ridiculous, but we will get rid of this assumption shortly. A cover tree on data points $x_1, \ldots, x_n$ is a rooted infinite tree with the following properties.

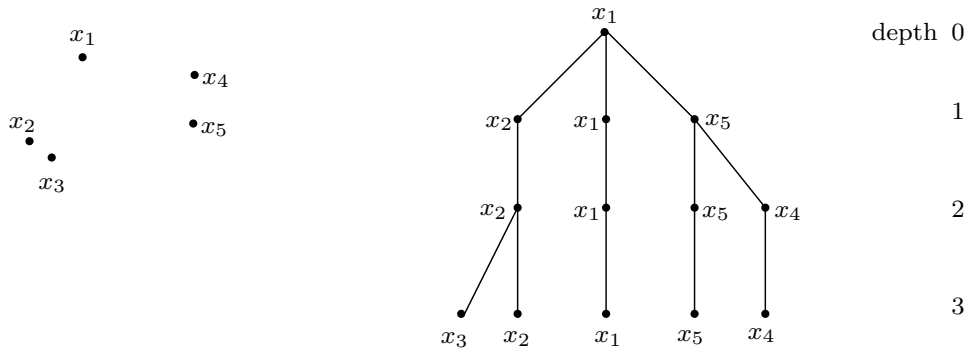
1. Each node of the tree is associated with one of the data points x_i.
2. If a node is associated with x_i, then one of its children must also be associated with x_i.

3. All nodes at depth j are at distance at least $1/2^j$ from each other.

4. Each node at depth $j+1$ is within distance $1/2^j$ of its parent (at depth j).

This is described as an infinite tree for simplicity of analysis, but it would not be stored as such. In practice, there is no need to duplicate a node as its own child, and so the tree would take up $O(n)$ space.

The figure below gives an example of a cover tree for a data set of five points. This is just the top few levels of the tree, but the rest of it is simply a duplication of the bottom row. From the structure of the tree we can conclude, for instance, that x_1, x_2, x_5 are all at distance $\geq 1/2$ from each other (since they are all at depth 1), and that the distance between x_2 and x_3 is $\leq 1/4$ (since x_3 is at depth 3, and is a child of x_2).



What makes cover trees especially convenient is that they can be built on-line, one point at a time. To insert a new point x: find the largest j such that x is within $1/2^j$ of some node p at depth j in the tree; and make x a child of p. Do you see why this maintains the four defining properties?

Once the tree is built, it is easy to obtain k-clusterings from it.

Lemma 4. *For any k, consider the deepest level of the tree with $\leq k$ nodes, and let T_k be those nodes. Then:*

$$\text{cost}(T_k) \leq 8 \cdot \text{cost}(\text{optimal } k \text{ centers}).$$

Proof. Fix any k, and suppose j is the deepest level with $\leq k$ nodes. By Property 4, all of T_k's children are within distance $1/2^j$ of it, and its grandchildren are within distance $1/2^j + 1/2^{j+1}$ of it, and so on. Therefore,

$$\text{cost}(T_k) \;\leq\; \frac{1}{2^j} + \frac{1}{2^{j+1}} + \cdots \;=\; \frac{1}{2^{j-1}}.$$

Meanwhile, level $j+1$ has $\geq k+1$ nodes, and by Property 3, these are at distance $\geq 1/2^{j+1}$ from each other. Therefore the optimal k-clustering has cost at least $1/2^{j+2}$. □

To get rid of the assumption that all interpoint distances are ≤ 1, simply allow the tree to have depths $-1, -2, -3$ and so on. Initially, the root is at depth 0, but if a node arrives that is at distance ≥ 8 (say) from all existing nodes, then it is made the new root and is put at depth -3.

Also, the creation of the data structure appears to take $O(n^2)$ time and $O(n)$ space. But if we are only interested in values of k from 1 to K, then we need only keep the top few levels of the tree, those with $\leq K$ nodes. This reduces the time and space requirements to $O(nK)$ and $O(K)$, respectively.

6.2.3 On-line clustering algorithms: epilogue

It is an open problem to develop a good on-line algorithm for k-means clustering. There are at least two different ways in which a proposed scheme might be analyzed. The first supposes at each time t, the algorithm sees a new data point x_t, and *then* outputs a set of k centers T_t. The hope is that for some constant $\alpha \geq 1$, for all t,

$$\text{cost}(T_t) \leq \alpha \cdot \text{cost}(\text{best } k \text{ centers for } x_1, \ldots, x_t).$$

This is the pattern we have followed in our k-center examples.

The second kind of analysis is the more usual setting of the on-line learning literature. It supposes that at each time t, the algorithm announces a set of k centers T_t, *then* sees a new point x_t and incurs a loss equal to the cost of x_t under T_t (that is, the squared distance from x_t to the closest center in T_t). The hope is that at any time t, the total loss incurred upto time t,

$$\sum_{t' \leq t} \min_{z \in T_t} \|x_t - z\|^2$$

is not too much more than the optimal cost achievable by the best k centers for $x_1, \ldots, x_t$.

Problem 2. Develop an on-line algorithm for k-means clustering that has a performance guarantee under either of the two criteria above.

6.3 Streaming algorithms for clustering

The *streaming* model of computation is inspired by massive data sets that are too large to fit in memory. Unlike on-line learning, which continues forever, there is a finite job to be done. But it is a very big job, and so only a small portion of the input can be held in memory at any one time. For an input of size n, one typically seeks algorithm that use memory $o(n)$ (say, $\sqrt{n}$) and that need to make just one or two passes through the data.

On-line	Streaming
Endless stream of data	Stream of (known) length n
Fixed amount of memory	Memory available is $o(n)$
Tested at every time step	Tested only at the very end
Each point is seen only once	More than one pass may be possible

There are some standard ways to convert regular algorithms (that assume the data fits in memory) into streaming algorithms. These include divide-and-conquer and random sampling. We'll now see k-medoid algorithms based on each of these.

6.3.1 A streaming k-medoid algorithm based on divide-and-conquer

Recall the k-medoid problem: the setting is a metric space $(\mathcal{X}, \rho)$.

Input: Finite set $S \subset \mathcal{X}$; integer k.

Output: $T \subset S$ with $|T| = k$.

Goal: Minimize $\text{cost}(T) = \sum_{x \in S} \rho(x, T)$.

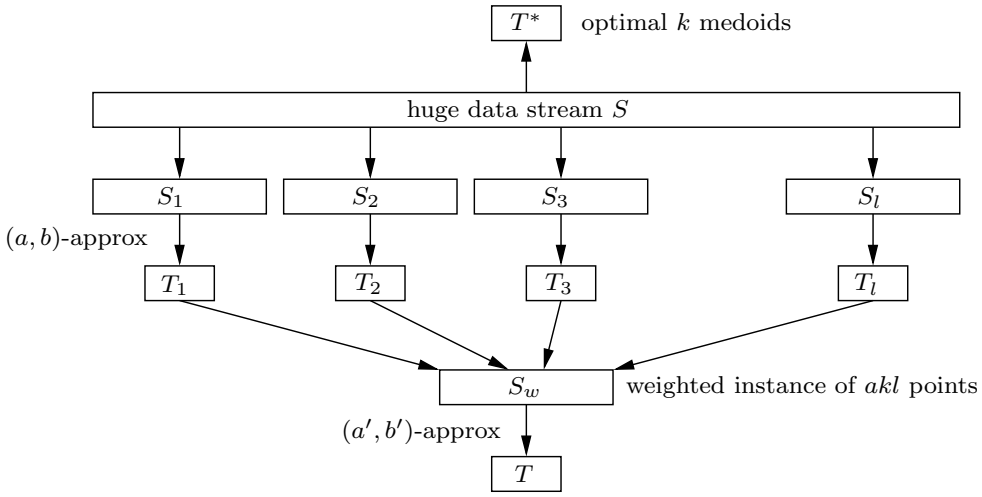


Figure 6.1. A streaming algorithm for the k-medoid problem, based on divide and conquer.

Earlier, we saw an LP-based algorithm that finds $2k$ centers whose cost is at most 4 times that of the best k-medoid solution. This extends to the case when the problem is *weighted*, that is, when each data point x has an associated weight $w(x)$, and the cost function is

$$\text{cost}(T) = \sum_{x \in S} w(x)\rho(x, T).$$

We will call it a $(2, 4)$-approximation algorithm. It turns out that (a, b)-approximation algorithms are available for many combinations (a, b).

A natural way to deal with a huge data stream S is to read as much of it as will fit into memory (call this portion S_1), solve this sub-instance, then read the next batch S_2, solve this sub-instance, and so on. At the end, the partial solutions need to be combined somehow.

```
Divide S into groups S_1, S_2, ..., S_l
for each i = 1, 2, ..., l:
   run an (a, b)-approximation alg on S_i to get ≤ ak medoids T_i = {t_i1, t_i2, ...}
   suppose S_i1 ∪ S_i2 ∪ ··· are the induced clusters of S_i
S_w ← T_1 ∪ T_2 ∪ ··· ∪ T_l, with weights w(t_ij) ← |S_ij|
run an (a', b')-approximation algorithm on weighted S_w to get ≤ a'k centers T
return T
```

Figure 6.1 shows this pictorially. An interesting case to consider is when stream S has length n, and each batch S_i consists of $\sqrt{nk}$ points. In this case, the second clustering problem S_w is also of size $\sqrt{nk}$, and so the whole algorithm operates with just a single pass and $O(\sqrt{nk})$ memory.

Before analyzing the algorithm we introduce some notation. Let $T^* = \{t_1^*, \ldots, t_k^*\}$ be the optimal k medoids for data set S. Let $t^*(x) \in T^*$ be the medoid closest to point x. Likewise, let $t(x) \in T$ be the point in T closest to x, and $t_i(x)$ the point in T_i closest to x. Since we are dividing the data into subsets S_i, we will need to talk about the costs of clustering these subsets as well as the overall cost of clustering S. To

this end, define

$$\begin{aligned}\text{cost}(S', T') &= \text{cost of medoids } T' \text{ for data } S' \\ &= \begin{cases} \sum_{x \in S'} \rho(x, T') & \text{unweighted instance} \\ \sum_{x \in S'} w(x)\rho(x, T') & \text{weighted instance} \end{cases}\end{aligned}$$

Theorem 5. *This streaming algorithm is an $(a', 2b + 2b'(2b + 1))$-approximation.*

The a' part is obvious; the rest will be shown over the course of the next few lemmas. The first step is to bound the overall cost in terms of the costs of two clustering steps; this is a simple application of the triangle inequality.

Lemma 6. $\text{cost}(S, T) \leq \sum_{i=1}^{l} \text{cost}(S_i, T_i) + \text{cost}(S_w, T)$.

Proof. Recall that the second clustering problem S_w consists of all medoids t_{ij} from the first step, with weights $w(t_{ij}) = |S_{ij}|$.

$$\begin{aligned}\text{cost}(S, T) = \sum_{i=1}^{l} \sum_{x \in S_i} \rho(x, T) &\leq \sum_{i=1}^{l} \sum_{x \in S_i} (\rho(x, t_i(x)) + \rho(t_i(x), T)) \\ &= \sum_{i=1}^{l} \text{cost}(S_i, T_i) + \sum_{i=1}^{l} \sum_{j} |S_{ij}| \rho(t_{ij}, T) \\ &= \sum_{i=1}^{l} \text{cost}(S_i, T_i) + \text{cost}(S_w, T).\end{aligned}$$

□

The next lemma says that when clustering a data set S', picking centers from S' is at most twice as bad as picking centers from the entire underlying metric space $\mathcal{X}$.

Lemma 7. *For any $S' \subset \mathcal{X}$, we have*

$$\min_{T' \subset S'} \text{cost}(S', T') \leq 2 \min_{T' \subset \mathcal{X}} \text{cost}(S', T').$$

Proof. Let $T' \subset \mathcal{X}$ be the optimal solution chosen from $\mathcal{X}$. For each induced cluster of S', replace its center $t' \in T'$ by the closest neighbor of t' in S'. This at most doubles the cost, by the triangle inequality. □

Our final goal is to upper-bound $\text{cost}(S, T)$, and we will do so by bounding the two terms on the right-hand side of Lemma 6. Let's start with the first of them. We'd certainly hope that $\sum_i \text{cost}(S_i, T_i)$ is smaller than $\text{cost}(S, T^*)$; after all, the former uses way more representatives (about akl of them) to approximate the same set S. We now give a coarse upper bound to this effect.

Lemma 8. $\sum_{i=1}^{l} \text{cost}(S_i, T_i) \leq 2b \cdot \text{cost}(S, T^*)$.

Proof. Each T_i is a b-approximation solution to the k-medoid problem for S_i. Thus

$$\begin{aligned}\sum_{i=1}^{l} \text{cost}(S_i, T_i) &\leq \sum_{i=1}^{l} b \cdot \min_{T' \subset S_i} \text{cost}(S_i, T') \\ &\leq \sum_{i=1}^{l} 2b \cdot \min_{T' \subset \mathcal{X}} \text{cost}(S_i, T') \\ &\leq \sum_{i=1}^{l} 2b \cdot \text{cost}(S_i, T^*) = \text{cost}(S, T^*).\end{aligned}$$

The second inequality is from Lemma 7. □

Finally, we bound the second term on the right-hand side of Lemma 6.

Lemma 9. $\text{cost}(S_w, T) \leq 2b'(\sum_{i=1}^{l} \text{cost}(S_i, T_i) + \text{cost}(S, T^*))$.

Proof. It is enough to upper-bound $\text{cost}(S_w, T^*)$ and then invoke

$$\text{cost}(S_w, T) \;\leq\; b' \cdot \min_{T' \subset S_w} \text{cost}(S_w, T') \;\leq\; 2b' \cdot \min_{T' \subset \mathcal{X}} \text{cost}(S_w, T') \;\leq\; 2b' \cdot \text{cost}(S_w, T^*).$$

To do so, we need only the triangle inequality.

$$\begin{aligned} \text{cost}(S_w, T^*) \;=\; \sum_{i,j} |S_{ij}| \rho(t_{ij}, T^*) \;&\leq\; \sum_{i,j} \sum_{x \in S_{ij}} (\rho(x, t_{ij}) + \rho(x, t^*(x))) \\ &=\; \sum_{i} \sum_{x \in S_i} (\rho(x, t_i(x)) + \rho(x, t^*(x))) \\ &=\; \sum_{i} \text{cost}(S_i, T_i) + \text{cost}(S, T^*). \end{aligned}$$

□

The theorem follows immediately by putting together the last two lemmas with Lemma 6.

Problem 3. Notice that this streaming algorithm uses two k-medoid subroutines. Even if both are perfect, that is, if both are $(1, 1)$-approximations, the overall bound on the approximation factor is 8. Can a better factor be achieved?

6.3.2 A streaming k-medoid algorithm based on random sampling

In this approach, we randomly sample from stream S, taking as many points as will fit in memory. We then solve the clustering problem on this subset $S' \subset S$. The resulting cluster centers should work well for all of S, *unless* S contains a cluster that has few points (and is thus not represented in S') *and* is far away from the rest of the data. A second round of clustering is used to deal with this contigency.

The algorithm below is due to Indyk (1999). Its input is a stream S of length n, and also a confidence parameter δ.

```
let S' ⊂ S be a random subset of size s
run an (a,b)-approximation alg on S' to get ≤ ak medoids T'
let S'' ⊂ S be the (8kn/s) log(k/δ) points furthest from T'
run an (a,b)-approximation alg on S'' to get ≤ ak medoids T''
return T' ∪ T''
```

This algorithm returns $2ak$ medoids. It requires two passes through the data, and if s is set to $\sqrt{8kn \log(k/\delta)}$, it uses memory $O(\sqrt{8kn \log(k/\delta)})$.

Theorem 10. *With probability $\geq 1 - 2\delta$, this is a $(2a, (1 + 2b)(1 + 2/\delta))$-approximation.*

As before, let $T^* = \{t_1^*, \ldots, t_k^*\}$ denote the optimal medoids. Suppose these induce a clustering $S_1, \ldots, S_k$ of S. In the initial random sample S', the large clusters — those with a lot of points — will be well-represented. We now formalize this. Let L be the large clusters:

$$L = \{i : |S_i| \geq (8n/s) \log(k/\delta)\}.$$

The sample S' contains points $S_i' = S_i \cap S'$ from cluster S_i.

Lemma 11. *With probability* $\geq 1-\delta$*, for all* $i \in L$*:* $|S_i'| \;\geq\; \frac{1}{2} \cdot \frac{s}{n} \cdot |S_i|$.

Proof. Use a multiplicative form of the Chernoff bound. □

We'll next show that the optimal centers T^* do a pretty good job on the random sample S'.

Lemma 12. *With probability* $\geq 1-\delta$*,* $\text{cost}(S', T^*) \leq \frac{1}{\delta} \cdot \frac{s}{n} \cdot \text{cost}(S, T^*)$.

Proof. The expected value of $\text{cost}(S', T^*)$ is exactly $(s/n)\text{cost}(S, T^*)$, whereupon the lemma follows by Markov's inequality. □

Henceforth assume that the high-probability events of Lemmas 11 and 12 hold. Lemma 12 implies that T' is also pretty good for S'.

Lemma 13. $\text{cost}(S', T') \leq 2b \cdot \frac{1}{\delta} \cdot \frac{s}{n} \cdot \text{cost}(S, T^*)$.

Proof. This is by now a familiar argument.

$$\text{cost}(S', T') \;\leq\; b \cdot \min_{T \subset S'} \text{cost}(S', T) \;\leq\; 2b \cdot \min_{T \subset \mathcal{X}} \text{cost}(S', T) \;\leq\; 2b \cdot \text{cost}(S', T^*)$$

and the rest follows from Lemma 12. □

Since the large clusters are well-represented in S', we would expect centers T' to do a good job with them.

Lemma 14. $\text{cost}\left(\cup_{i \in L} S_i, T'\right) \;\leq\; \text{cost}(S, T^*) \cdot \left(1 + 2(1+2b)\frac{1}{\delta}\right)$.

Proof. We'll show that for each optimal medoid t_i^* of a large cluster, there must be a point in T' relatively close by. To show this, we use the triangle inequality $\rho(t_i^*, T') \leq \rho(t_i^*, y) + \rho(y, t'(y))$, where $t'(y) \in T'$ is the medoid in T' closest to point y. This inequality holds for all y, so we can average it over all $y \in S_i'$:

$$\rho(t_i^*, T') \;\leq\; \frac{1}{|S_i'|} \sum_{y \in S_i'} \left(\rho(y, t_i^*) + \rho(y, t'(y))\right).$$

Now we can bound the cost of the large clusters with respect to medoids T'.

$$\begin{aligned}
\text{cost}\left(\cup_{i \in L} S_i, T'\right) &\leq \sum_{i \in L} \sum_{x \in S_i} \left(\rho(x, t_i^*) + \rho(t_i^*, T')\right) \\
&\leq \text{cost}(S, T^*) + \sum_{i \in L} |S_i| \rho(t_i^*, T') \\
&\leq \text{cost}(S, T^*) + \sum_{i \in L} \frac{|S_i|}{|S_i'|} \sum_{y \in S_i'} \left(\rho(y, t_i^*) + \rho(y, t'(y))\right) \\
&\leq \text{cost}(S, T^*) + \frac{2n}{s} \sum_{y \in S'} \left(\rho(y, t^*(y)) + \rho(y, t'(y))\right) \\
&= \text{cost}(S, T^*) + \frac{2n}{s} \left(\text{cost}(S', T^*) + \text{cost}(S', T')\right)
\end{aligned}$$

where the last inequality is from Lemma 11. The rest follows from Lemmas 12 and 13. □

Thus the large clusters are well-represented by T'. But this isn't exactly what we need. Looking back at the algorithm, we see that medoids T'' will take care of S'', which means that we need medoids T' to take care of $S \setminus S''$.

Lemma 15. $\text{cost}(S \setminus S'', T') \leq \text{cost}(S, T^*) \cdot \left(1 + 2(1+2b)\frac{1}{\delta}\right)$.

Proof. A large cluster is one with at least $(8n/s)\log(k/\delta)$ points. Therefore, since there are at most k small clusters, the total number of points in small clusters is at most $(8kn/s)\log(k/\delta)$. This means that the large clusters account for the vast majority of the data, at least $n - (8kn/s)\log(k/\delta)$ points.

$S \setminus S''$ is exactly the $n-(8kn/s)\log(k/\delta)$ points *closest* to T'. Therefore $\text{cost}(S \setminus S'', T') \leq \text{cost}\left(\cup_{i \in L} S_i, T'\right)$. □

To prove the main theorem, it just remains to bound the cost of S''.

Lemma 16. $\text{cost}(S'', T'') \leq 2b \cdot \text{cost}(S, T^*)$.

Proof. This is routine.

$$\text{cost}(S'', T'') \;\leq\; b \cdot \min_{T \subset S''} \text{cost}(S'', T) \;\leq\; 2b \cdot \min_{T \subset \mathcal{X}} \text{cost}(S'', T) \;\leq\; 2b \cdot \text{cost}(S'', T^*) \;\leq\; 2b \cdot \text{cost}(S, T^*).$$

□

Problem 4. The approximation factor contains a highly unpleasant $1/\delta$ term. Is there a way to reduce this to, say, $\log(1/\delta)$?