Lemma 4.1.10. *The CountSketch sketch as above but with $B \geq 15k$ guarantees that for any i,* query(i) *returns a value $x_i \pm \|x_{tail(k)}\|_2/\sqrt{k}$ w.p. $\geq 1 - \delta$.*

4.2 Graph sketching

In this section we show the perhaps surprising result that linear sketching can be used to solve combinatorial problems on graphs. Specifically, we consider a dynamic (multi)graph on n vertices in which vertices can be both inserted and deleted. This model can be faithfully represented in the strict turnstile model, where x has dimension $\binom{n}{2}$; x_e specifies the presence (or number of copies) of edge e in the graph. An insertion of edge e then corresponds to the turnstile update $(e, +1)$, whereas an edge deletion corresponds to the update $(e, -1)$. Solving graph problems naively would require remembering x exactly, i.e. $\Omega(n^2)$ memory in the worst case. In this section we describe the "AGM sketch" [AGM12], which shows that dynamic spanning forest can be solved using $O(n \log^c n)$ bits of memory with success probability $1 - 1/poly(n)$. The AGM sketch achieves the exponent with $c = 3$, though in these notes we describe a simpler version achieving $c = 4$. One can also obtain bounds based on the failure probability as a separate parameter δ; see [NY19]. Note that being able to query the dynamic spanning forest allows for solving many other problems, such as finding an s-t path, global connectivity, or s-t connectivity. It is furthermore known that the $\Omega(n \log^3 n)$ bound is tight; any algorithm that reports a spanning forest with probability at least even 1% must use this much memory [NY19]. Yu recently showed a stronger lower bound in the distributed sketching model, in which all vertices and a "referee" share public randomness each vertex knows only its own neighborhood and must send a short message to the referee; he showed that even if the query is not required to output an entire spanning forest but simply a single bit indicating whether the graph is connected, the average message length must be $\Omega(\log^3 n)$ bits [Yu21] (the AGM sketch provides a matching upper bound for spanning forest in this model as well). After the paper [AGM12], linear sketching has been proven useful for a wide variety of dynamic graph problems; see [McG14] for a survey.

The AGM sketch uses certain other data structures as subroutines, which we describe first. Both these data structures operate in the general turnstile model.

4.2.1 k-sparse recovery

Recall $support(x) \subseteq [n]$ denotes the indices i such that $x_i \neq 0$. The parameter k is given at the beginning of the stream, and there is a single query. The answer to a query is as follows: if $|support(x)| \leq k$, the query simply returns x exactly (the indices in the support together with their values); otherwise, the query response can be arbitrary. We show that this problem can be solved deterministically using $O(k \log(nM))$ bits of memory if we are promised that all update amounts are integers and no entry of x is ever larger than M in magnitude. Note with this restriction it suffices to solve the problem over $\mathbb{F}_p$ for any prime $p > M$, as any x_i will equal $x_i \mod p$. The solution we discuss will also require $p \geq n$, and thus we will work over any prime $p > \max\{M, n\}$. The total space will be at most $2k\lceil \log p \rceil$ bits.

Recall in linear sketching we maintain Πx in memory for some $\Pi \in \mathbb{R}^{m \times n}$ (in this case $\Pi \in \mathbb{F}_p^{m \times n}$)). k-sparse recovery is possible iff $\Pi x \neq \Pi y$ for x, y distinct k-sparse vectors, which is equivalent to $\Pi(x - y) \neq 0$. Noting $x - y$ is $2k$-sparse, this requirement is thus equivalent to $\Pi z \neq 0$ for any $2k$-sparse vector z. If S denotes the support of z, then $\Pi z = \Pi_S z$, where Π_S is the $m \times |S|$ submatrix of Π keeping only the columns indexed by S. Thus our requirement for Π is that all of its $m \times 2k$ submatrices have full column rank. We will pick $m = 2k$, so these are square submatrices; thus having full column rank is equivalent to $\det(\Pi_S) \neq 0$ for all $2k \times 2k$ submatrices

of Π. We will specifically pick Π to be the transpose of a Vandermonde matrix. Specifically, pick $x_1 \neq x_2 \neq \cdots \neq x_n \in \mathbb{F}_p$ (this is why we require $p \geq n$, to guarantee at least n distinct elements in $\mathbb{F}_p$) and set $\Pi_{i,j} = x_j^{i-1} \mod p$ for $i, j \in [n]$. For concreteness, we could pick $x_j = j$. The following known fact, which we will not prove here, implies that any $2k \times 2k$ submatrix of Π has nonzero determinant.

Fact 4.2.1. *Let $A \in F^{r \times r}$ be such that $A_{i,j} = x_j^{i-1}$ for $i, j \in [r]$ for some field F. Then*

$$\det(A) = \prod_{1 \leq i < j \leq n} (x_i - x_j)$$

The above fact is usually written for $A^\top$, but note $\det(A) = \det(A^\top)$ for any A. Note Fact 4.2.1 implies $\det(A) \neq 0$ if the x_i are distinct.

Lemma 4.2.2. *Suppose $A \in F^{m \times n}$ is such that every $m \times 2k$ submatrix of A has full column rank, where F is a field. Then there is an algorithm running in $\binom{n}{2k} \cdot poly(n)$ to recover x given $y = Ax$ for any k-sparse x.*

Proof. We loop over all $S \subset [n]$ of size exactly $2k$ (our guess for a set containing the support of x) and compute $x' = \Pi_S^{-1} y$. If x' is k-sparse, then we form x as the n-dimensional vector x with $x_S = x'$ and $x_{[n] \setminus S} = \vec{0}$ and return x. □

Remark 4.2.3. For the particular scheme we propose, it is possible to actually recover x from Πx in $O(k^2 \, polylog(p))$ time via an algorithm called *syndrome decoding*, though we will not cover it here.

4.2.2 SupportFind

In this problem, we would like a randomized data structure which, if $x = 0$, reports `null`. Otherwise, if $x \neq 0$ it should return some $i \in support(x)$ with probability at least $1 - \delta$ (with probability δ it is allowed to behave arbitrarily). There are no promises regarding the vector x; it may or may not be sparse, yet the query algorithm should still succeed.

We describe an algorithm, the JST sketch, for this problem due to [JST11] which uses $O(\log(1/\delta) \log^2 n)$ bits of memory if all entries in x are promised to be integers which are at most $poly(n)$ in magnitude. It is known that this bound is optimal even if the entries of x are promised to always be either 0 or 1 [KNP+17].

The JST sketch uses the geometric sampling technique of Subsection 2.2.3. Specifically, we pick a hash function $h : [n] \to [\log_2 n]$ with $\mathbb{P}(h(i) = j) = 1/2^j$ (other than for $j = \log_2 n$, in which case we have $\mathbb{P}(h(i) = j) = 1/2^{j-1}$ so that the probabilities add to one). For now we assume h is a perfectly random hash function, though we discuss in Remark 4.2.5 how this can be relaxed using bounded independence. For each $j \in [\log_2 n]$, we also instantiate a k-sparse recovery data structure A_j from Subsection 4.2.1 with $k = C \log(1/\delta)$ for a sufficiently large constant $C > 0$ and with $M = poly(n)$.

To process $\mathsf{update}(i, \Delta)$, we simply call $A_{h(i)}.\mathsf{update}(i, \Delta)$. To process a query, we loop from $j = \log_2 n$ down to 1 and for each such j call $A_j.\mathsf{query}()$. For the first (i.e. largest) value of j for which the query is not the zero vector, we return any index in the support of the query response. See Subsection 4.2.2 for pseudocode.

Theorem 4.2.4. *If $x = 0$, `null` is returned with probability 1. Otherwise, the probability some $i \in support(x)$ is returned is at least $1 - \delta$.*

```
for j = log_2 n, ..., 1:
    z ← A_j.query()
    if z ≠ 0:
        return any i such that z_i ≠ 0
return null // x is the zero vector
```

Proof. The case $x = 0$ is clear, as all A_j will return the zero vector when queried. Also clear is the case $|\mathit{support}(x)| \le k := C\log(1/\delta)$, since every A_j will receive a vector with support size at most that of x (and thus will return its received vector exactly), and at least one of the A_j must receive a nonzero vector if $x \neq 0$ since every index of x is hashed to exactly one A_j.

For the remainder of the proof we thus focus on the case that $\mathit{support}(x) \ge k$. Let t denote $|\mathit{support}(x)|$. Let $x^{(j)}$ denote the vector x where we zero out all coordinates i such that $h(i) \neq j$, and define the random variable $T_j := |\mathit{support}(x^{(j)})|$. For some (large) constant c such that $1 < c < C$, let $j^* \in [\log_2 n]$ be such that $c\log(1/\delta) \le t/2^{j^*} < 2c\log(1/\delta)$. Such j^* must exist since $t > C\log(1/\delta)$. We define two events: $\mathcal{E}_1$ is the event $\max_{j \ge j^*} T_j \le k$, and $\mathcal{E}_2$ is the event $T_{j^*} \ge 1$. Note if both events occur, then our query output is guaranteed to be correct. This is because $\mathcal{E}_1$ implies $A_j.\mathsf{query}()$ will correctly return $x^{(j)}$ for all $j \ge j^*$, and $\mathcal{E}_2$ implies that at least one of these $x^{(j)}$ is nonzero (since in particular $x^{(j^*)} \neq 0$). We then show $\mathbb{P}(\neg\mathcal{E}_1 \vee \neg\mathcal{E}_2) \le \delta$, by the union bound.

We bound $\mathbb{P}(\neg\mathcal{E}_1)$ itself by a union bound over $j \ge j^*$. Note $\mathbb{E}\,T_j = t/2^j$. Then $\mathbb{P}(T_j > k) = \mathbb{P}(T_j > (k2^j/t)\cdot\mathbb{E}\,T_j) < (k2^j/t)^{-C'k}$ (see Eq. (1.7)). Summing over $j \ge j^*$ gives a geometric series dominated by its largest term, which is the term for j^*, which is $(k2^{j^*}/t)^{-C'k} \le (k/(c\log(1/\delta))^{-C'k} = (C/c)^{-C'k} < \delta/2$ for C sufficiently larger than c. $\mathbb{P}(\neg\mathcal{E}_2)$ is also bounded by the Chernoff bound. We have $\mathbb{E}\,T_{j^*} \in [c\log(1/\delta), 2c\log(1/\delta)]$. Thus $\mathbb{P}(\neg\mathcal{E}_2) = \mathbb{P}(T_{j^*} = 0)$, which is at most $\delta/2$ by the Chernoff bound. □

Remark 4.2.5. It is a useful fact to know that tail bounds imply moment bounds and vice versa. In one direction, if we have a bound on all moments $\|Z\|_p$ then we have a tail bound via Markov's inequality: $\mathbb{P}(|Z| > \lambda) < \inf_p\{\lambda^{-p}\|Z\|_p^p\}$ (recall $\|Z\|_p := (\mathbb{E}\,|Z|^p)^{1/p}$). The value $p \ge 1$ can be chosen to minimize the right hand side. In the other direction, $\|Z\|_p^p = \int_0^\infty px^{p-1}\,\mathbb{P}(|Z| > x)dx$ via integration by parts. Thus a tail bound on $|Z|$ yields moment bounds. Now, since the Chernoff bound gives strong tail bounds, one can use this correspondence to obtain the implied moment bounds on $|\sum_i X_i - \mu|$ for all $p \ge 1$, and from those moment bounds re-derive the Chernoff bound itself by choosing p optimally based on λ and μ, which is determined by p-wise independence of the X_i if p is an even integer. The punchline is that if one were to carry out this calculation exercise, one would find that whenever the Chernoff bound yields tail probability δ, it sufficed to choose $p = O(\log(1/\delta))$, so that the X_i could be $O(\log(1/\delta))$-wise independent (see also [BR94, SSS95], which take different approaches to showing this). This observation allows one to select the h in the JST sketch from an $O(\log(1/\delta))$-wise independent family, so that it only takes $O(\log(1/\delta)\log n)$ bits to represent.

4.2.3 AGM sketch

Before describing the AGM sketch, we first design a non-streaming algorithm. Imagine that we proceed in $R = \log_2 n$ rounds. We start each round with a partition that is a refinement of the partition of vertices into connected components, and in the first round each vertex is in its own partition. Now, at the beginning of each round we ask each partition (which we henceforth call a *super-vertex*) to identify an edge leaving it and entering another partition. At the end of the