Hall-type theorems for fast dynamic matching and applications

Bruno Bauwens
National Research University Higher School of Economics, Faculty of Computer Science, Moscow, Russia

Marius Zimand
Department of Computer and Information Sciences, Towson University, Baltimore, MD.

Abstract
We show that in bipartite graphs a large expansion factor implies very fast dynamic matching. Coupled with known constructions of lossless expanders, this gives a solution to the main open problem in a classical paper of Feldman, Friedman, and Pippenger [7].

Application 1: storing sets. We construct 1-query bitprobes that store a dynamic subset \( S \) of \( \{1, \ldots, N\} \) as a bitstring. A membership query reads a single bit, whose location is computed in time \( \text{poly}(\log N, \log(1/\varepsilon)) \) and is correct with probability \( 1 - \varepsilon \). Elements can be inserted and removed efficiently in time \( \text{quasipoly}(\log N) \). Previous constructions were static: membership queries have the same parameters, but each update requires the recomputation of the whole data structure, which takes time \( \text{poly}(\#S \log N) \). Moreover, the size of our scheme is smaller than the best known constructions for static sets.

Application 2: switching networks. We construct explicit constant depth \( N \)-connectors of essentially minimum size in which the path finding algorithm runs in time \( \text{quasipolynomial in } \log N \). In the non-explicit construction in [7] and in the explicit construction of Wigderson and Zuckerman [19] the runtime is exponential in \( N \).

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Matching, bipartite graphs, online algorithms, lossless expanders, one bit-probe, rearrangeable networks, connectors

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Related Version https://arxiv.org/abs/2204.01936

Funding Bruno Bauwens: This research was funded by RSF grant number 20–11–20203 (Sections 4 and 5)
Marius Zimand: Partially supported by NSF grant CCF 1811729. Sections 1–3 and 6–8
1 Introduction

A bipartite graph has offline matching up to $K$ elements if every set of $K$ left nodes can be covered by $K$ pairwise disjoint edges. A graph has $\epsilon$-expansion up to $K$ if every subset $S$ with at most $K$ left nodes has at least $\epsilon \cdot \#S$ right neighbors. Hall’s theorem states that a graph has offline matching up to $K$ elements if and only if it has 1-expansion up to $K$.

We consider a dynamic matching problem in which an opponent may remove edges from the matching and select new left nodes that need to be covered. We use a relaxed notion, in which a right node may be incident on up to $\ell$ edges. This does not harm in many applications if $\ell$ is small, say logarithmic, because we may use $\ell$ copies of the graph.

Dynamic matching game. The game with parameters $K$ and $\ell$ is played on a graph. Two players, called Requester and Matcher, know this graph and alternate turns. Together they maintain a subset $M$ of edges. Requester starts. At his turn, he can remove zero or more edges. After this, $M$ should contain at most $K - 1$ edges. Also, he must select a left node $x$. At her turn, Matcher may add an edge to $M$. After this, $x$ should be incident on an edge of $M$, and each right node must be incident on at most $\ell$ edges from $M$. If these conditions are not satisfied, then Matcher looses.

Definition. A graph has dynamic matching up to $K$ elements with load $\ell$, if Matcher has a strategy in the above game in which she never looses. If the load $\ell$ is omitted, then $\ell = 1$ is assumed.

The following graphs have offline matching but not dynamic matching up to 2 elements (and with load 1).

Feldman, Friedman, and Pippenger have shown that if a graph has 2-expansion up to $2K$, then it has dynamic matching up to $K$. By a similar argument, 1-expansion up to $K$ implies dynamic matching up to $K$ with load 3, see Appendix C. Unfortunately, matches are computed in time exponential in $K$. An important open question from [7] is to find a variant with an efficient matching algorithm.

We show that a graph with expansion factor equal to a large fraction of the left degree (i.e., a lossless expander) has fast dynamic matching with small load.

Theorem 1.1. If a graph with $N$ left elements and left degree $D$ has $(\frac{2}{3}D + 2)$-expansion up to $K$, then it has dynamic matching up to $K$ with load $O(\log N)$. Moreover, there exists a data structure that uses $O(D \log N)$ time to process each retraction and compute each match.

The running time for computing and retracting a match is double exponentially faster than in [7]. Combined with known explicit constructions of lossless expanders, we obtain graphs with right size $K \cdot \text{quasipoly}(\log N)$ and quasipoly$(\log N)$ time dynamic matching up to $K$.

In the literature, dynamic matching also refers to the problem in which the edges of a left node are only given when the left node appears in a request, see for example [9]. The objective is to update fast while keeping the size of the matching close to the maximal offline size.
Hall-type theorems for fast dynamic matching and applications

The idea of the proof is to use 2 matching strategies. The first, already wins the dynamic matching game, but its evaluation requires time poly\( (N) \), see Proposition 4.1. The second is a greedy one that runs in time \( O(D \log N) \) as required, but can not assign matches for a few problematic left nodes. Fortunately, a small subset containing these nodes can be identified well in advance, i.e., many requests before such a problematic request might happen. During these requests, the slow computations are performed in small chunks in parallel with runs of the greedy allocation algorithms. The memory of the slow algorithm is stored in the datastructure.

In the first application we use a related but incomparable result. Let \( S \) be a subset of left nodes of a graph with left degree \( D \). An \( \varepsilon \)-rich matching for \( S \) is a set of edges in which each node in \( S \) is covered at least \((1 - \varepsilon)D\) times. In a similar way as above \( \varepsilon \)-rich matching up to \( K \) is defined. This is stronger if \((1 - \varepsilon)D > 1\). On the other hand, we restrict Requester: he must retract an edge at most \( T \) rounds after being added to the matching.

We show that graphs with \((((1 - \varepsilon)D)\)-expansion have \((2\varepsilon)\)-rich dynamic matching with load \( O(\log(NT)) \). From this we obtain explicit graphs with \( O(\varepsilon)\)-rich dynamic matching up to \( K \) (with load 1). Moreover, the right size is \( K \cdot \text{quasipoly}((\log(NT)) \), which is almost optimal, see Corollary 3.3. A slightly weaker result was proven in [4, Corollary 2.13], but the proof in terms of matchings is less technical.

**Application 1: one bitprobe storage schemes for dynamic sets.**

The goal is to store a subset \( S \) of a large set \( \{1, \ldots, N\} \) to answer membership queries “Is \( x \) in \( S \)?”. Let \( K = \#S \) be the size. A simple way is to store \( S \) in a sorted list. This requires \( K[\log N] \) bits of memory, and given \( x \), one can determine whether \( x \) is in \( S \) by reading \((\lceil \log K \rceil + 1) \cdot \lceil \log N \rceil \) bits from the table. An alternative is to have a table of \( N \) bits and set bit \( x \) equal to 1 if and only if \( x \in S \). Now the query “Is \( x \in S \)?” can be answered by reading a single bit. Also, one can insert or delete an element by modifying a single bit. The cost is that the table is long, since typically \( N \gg K \). We show that the advantages of the latter approach can be obtained with a data structure whose size is close to \( K \log N \).

A **one bitprobe storage scheme** (also called a **bit vector**) is a data structure that answers a membership query “Is \( x \) in \( S \)” by reading a single bit. It is a fundamental data structure introduced by Minsky and Papert in their book on perceptrons [13]. See [6] for more historic references.

In [6], lossless expanders are used to build one bitprobe storage schemes with short tables in which membership queries are answered probabilistically with small error \( \varepsilon \). Using a non-explicit expander they obtain storage size \( O(\frac{1}{\varepsilon}K \log N) \). Note that this is close to the lower bound \( K \log N - O(1) \) for any set data structure. They also have an explicit construction achieving storage size \( O(\frac{1}{\varepsilon^2}K \log N)^2 \). Ta-Shma [14] and Guruswami, Umans, and Vadhan [8, Theorem 7.4] give explicit constructions with smaller storage size. In all these schemes, making a membership query, (i.e., finding the location in the data structure of the bit that is probed), takes time \( \text{poly}(\log(N/\varepsilon)) \).

These one bitprobe storage schemes work for **static sets**, in the sense that any updating of \( S \) requires the recomputation of the entire data structure, which takes time

---

\[ \text{Such one bitprobe storage schemes are different from Bloom filters which store an approximation of the set. More precisely, a Bloom filter stores a superset } S' \text{ of the intended } S. \text{ Thus for every } x \in S' - S \text{ (the false positives) the error probability of the query \"Is } x \text{ in } S' \text{?" is 1, and for } x \in S \text{ or in } U - S' \text{ the error probability is 0 (and the probability over the choice of the hash functions used by the Bloom filter that an element is in } S' - S \text{ is } \varepsilon \text{).} \]
poly(K log(N/ε)). We obtain explicit one bitprobe storage schemes for dynamic sets. Membership queries also take time poly(log(N/ε)). Insertion and deletion of an element takes time quasipoly(log(N/ε)). The storage size is smaller than in the previous explicit schemes for static sets provided ε ≥ 1/K^1/log^2 K, see table 1. Full definitions and proofs are given in appendix B.

storage size

|          | reference |
|----------|-----------|
| O(K · log N · (1/ε)^2) | [6]       |
| O((K · log N · 1/ε)^2) | [6]       |
| K · exp(O((log log N)/ε)^2)) | [17]      |
| K · poly((log N)/ε) · exp(√(log((log N)/ε) · log K)) | [8]       |
| K · poly(log N) · exp(O((1/ log K) · log log K)) | Theorem 3.1 |

Table 1 One bitprobe storage schemes. The first scheme is non-explicit, the others explicit. The last is for dynamic sets, the others for static sets.

All previous explicit one bitprobes required lossless expanders with a special “list-decoding” property (see [8, Th.7.2]), while our approach works with any lossless expander. Thus future improvements in explicit lossless expanders will give better dynamic one bitprobes. This feature of our method also opens the possibility of implementations that are attractive in practice (see remark on page 17).

Application 2: non-blocking networks.

This area concerns switching networks for the transfer of information between a large number of nodes. In such graphs, the aim is to connect certain input nodes to certain output nodes using node disjoint paths. This area was the motivation for introducing expander graphs [2, 12].

An N-network is a directed acyclic graph in which N nodes are called inputs and N nodes are called outputs. A rearrangeable N-network is such a network in which for every 1-to-1 function f from outputs to inputs, there exist N node disjoint paths that connect each output node f(i) to the input node i.

For example, a fully connected bipartite graph with left and right sets of size N defines a rearrangeable N-network with N^2 edges. The goal is to construct networks with a minimal number of edges. Since there are N! different mappings, the minimum is at least log N!, which is at least N(log N − 2) by Stirling’s formula.

We use a generalized variant of rearrangeability, in which an input can be connected with several outputs, but an output can still be connected only with a single input. In terms of broadcasting, this means that several output nodes can listen to the same input node. Moreover, the connection problem needs to be solved dynamically. This can be defined in 2 closely related ways (called in the literature strict-sense non-blocking connector and wide-sense non-blocking connector). See [10] for more background. We use the second one, which is weaker.

Connection game. The game is played on an N-network. Two players, called Requester and Connector, both know the network and alternate turns. They maintain a set of at most N trees. The root of each tree must be an input and the leaves must be outputs. The trees must be node disjoint. Initially, the set of trees is empty. Requester starts.
At his turn, Requester may remove zero or more trees. Afterwards, he may select an input \( x \) and an output \( y \) such that \( y \) does not lay on any of the trees.

At her turn, Connector may create or extend a tree so that the above conditions are satisfied. Afterwards, there should be a tree in which \( x \) is the root and \( y \) is a leaf.

\[ \text{Definition.} \text{ A wide-sense non-blocking generalized } N\text{-connector (or simply } N\text{-connector) is an } N\text{-network in which } \text{Connector has a winning strategy in the above game.} \]

A fully connected bipartite graph is an \( N\)-connector. An \( N\)-connector was given in [7] that has \( O(N \log N) \) edges. This is optimal within a constant. The graph is explicit but the path finding algorithm (which is the algorithm that computes Connector’s replies) is very slow. Afterwards, in [1] an explicit \( N\)-connector is constructed of size \( O(N \log N) \) in which also the runtime of the path finding algorithm is \( O(\log N) \), and this is optimal within a constant factor. See [1] or [10, chapter 2] for more history.

The depth of a network is the length of the longest path between an input and an output. We focus on constant depth \( N\)-connectors. In [15] it is shown that \( N\)-connectors of depth \( t \) have at least \( tN^{1+1/t} \) edges. In [7], non-explicit constructions of \( N\)-connectors are given of size \( O(N^{1+1/t} \log^{1-1/t} N) \), but again the path finding algorithm runs in time exponential in \( N \). They ask whether a generalized connector exists with small size and an efficient path finding algorithm. They do not specify explicitly, but “small size” is usually considered to be a value that is \( N^{1+1/t} \cdot \log N \), see [19], and “efficient” should ideally mean that the runtime is \( \log N \). Some explicit constant-depth \( N\)-connectors are known with path finding algorithms running in time \( \log N \), but their size is not optimal, see [10, chapter 2]. For instance, for odd \( t \), the Clos network of depth \( t \) has size \( \Theta_t(N^{1+2/(t+1)}) \).

In [19, Th. 5.4] an explicit construction of size \( N^{1+1/t} \exp((\log \log N)^{O(1)}) \) was obtained, but the path finding algorithm is the same slow one from [7].

In appendix A, we present a non-explicit constant depth \( N\)-connector whose size is optimal up to factors \( \log N \) and with a path finding algorithm running in time \( \log N \).

\[ \text{Corollary 1.2.} \text{ For all } t \text{ and } N, \text{ there exists an } N\text{-connector of depth } t \text{ and size } N^{1+1/t} \text{ poly}(t \log N) \text{ with a poly}(\log N) \text{ time path finding algorithm.} \]

An \( N\)-connector is explicit if the \( i\)-th neighbor of a node is computed in time \( \log N \). We present an explicit connector with small size and a path finding algorithm running in \( \log N \) time.

\[ \text{Corollary 1.3.} \text{ For all } t \text{ and } N, \text{ there exists an explicit } N\text{-connector of depth } t, \text{ size } tN^{1+1/t} \exp(O(\log^2 \log N)), \text{ with a path finding algorithm with runtime } t \exp(O(\log^2 \log N)). \]

## 2 Fast dynamic matching with \( T\)-expiration

In this section we present matching strategies for games in which Requester is restricted. Recall the dynamic matching game with parameters \( K \) and \( \ell \). Requester and Matcher maintain a set \( M \) of edges. They alternate turns, and at their turn they do the following.

- Requester removes edges from \( M \) so that \( |M| \leq K – 1 \). He also selects a left node \( x \).
- We say that he requests \( x \).
- Matcher replies by adding an edge to \( M \) that is incident on \( x \). After this, every left node must lie on at most \( \ell \) edges in \( M \).

The aim of Matcher is to provide correct replies indefinitely. In the following 3 variants of the game the task of Matcher is easier.
In the incremental matching game, Requester can not remove edges from \(M\). Note that such a game lasts for at most \(K\) rounds, and Matcher wins if he can reply \(K\) times.

In the \(T\)-round matching game, the game is terminated after \(T\) rounds.

In the \(T\)-expiring matching game, Requester must remove the edge added in a round \(i\) during one of the rounds \(i + 1, \ldots, i + T\).

We say that a graph has incremental matching, respectively, \(T\)-round matching, and \(T\)-expiring matching if Matcher has a winning strategy in the corresponding games.

Note that incremental matching up to \(K\) and \(K\)-round matching are equivalent. Also, \(T\)-expiring matching implies \(T\)-round matching.

Examples. Consider the 2 graphs in the introduction. Recall that the left graph has offline matching up to 2. This graph does not have incremental matching up to 2, because if the middle node is selected first, then 1 of the 2 other nodes can not be matched.

The right graph does have incremental matching up to 2. But it does not have 3-round matching up to 2, and hence, also no 2-expiring matching up to 2.

Definition. Consider a graph with \(N\) left nodes and left degree \(D\). We say that the graph has fast dynamic matching if Matcher has a strategy for which replies can be computed in time \(O(D \log N)\).

In [14, p229 bottom] and [3, Corollary 2.11] it is proven that 1-expansion up to \(K\) implies fast incremental matching up to \(2K\) with load \(2 + \log K\). In the remainder of this section we prove the following extension of this result.

Proposition 2.1. If a graph has 1-expansion up to \(K\), then it has \(T\)-expiring fast matching up to \(K\) with load \(O(\log( KT ))\).

An \(\ell\)-clone of a graph \(G\) is a graph obtained from \(\ell\) copies of \(G\) by identifying the left nodes.

Remarks.
- A graph \(G\) has \(e\)-expansion if and only if an \(\ell\)-clone has \((e\ell)\)-expansion.
- A graph has matching with load \(\ell\) of any of the above types if and only if an \(\ell\)-clone has such a matching with load 1.

The proposition follows from these remarks and lemmas 2.3 and 2.4 below. Lemma 2.3 is a variant of the result from [14, Section 2.3], which we prove first.

Lemma 2.2. If a graph has 1-expansion up to \(K\), then a \((1 + \lceil \log K \rceil)\)-clone has fast incremental matching up to \(K\).

Proof. Let the copies of the clone be ordered. Node \(y\) is a free neighbor of \(x\) if the edge \((x, y)\) is not in the matching.

Matching strategy. Given a request \(x\), select the first copy in which \(x\) has a free neighbor, and match \(x\) to any free neighbor in this copy.

For \(K = 1\), correctness is trivial. For larger \(K\), we use induction on \(\lceil \log K \rceil\). Assume the statement is already proven for some value of \(K\). We need to prove that with 1 more copy, we obtain incremental matching up to \(2K\).

Fix a moment in the game. Let \(M'\) be the set of edges in \(M\) that belong to the first copy. Let \(R\) be the set of requests whose matches do not belong to \(M'\). The total number of requests is \(#M' + #R\), and this is bounded by \(2K\) during the incremental matching. It remains to show that \(#R \leq #M'\), since this implies \(#R \leq K\) and the result follows by the inductive hypothesis.
Let $N(R)$ denote the neighbors of $R$ in the first copy. Note that $N(R)$ is covered by edges in $M'$, because if request $x$ is not matched in the first copy, then its neighbors $N(x)$ are covered by $M'$ by choice of the algorithm. By 1-expansion, we have

$$\#R \leq \#N(R) \leq \#M'. \quad \blacktriangle$$

**Lemma 2.3.** Let $T/K$ be a non-negative power of 2. If a graph has 3-expansion up to $K$, then a $(1 + \lfloor \log T \rfloor)$-clone has $T$-round fast matching.

**Proof.** The matching strategy is the same as above. Thus for $T = K$, correctness is already proven. For larger $T$, we consider requests in blocks of length $2K$. It suffices to show that while processing each such block, at least $K$ matches are assigned using the first copy.

Fix a block and consider a moment during the processing of its requests. Let $M'$ be the set of all edges of the first copy that at some point have been present in the matching since the processing of the block started (and might still be present). Note that $\#M' \leq 3K$, because at the start of the block at most $K$ edges can be present, and at most $2K$ requests are processed during the block. In fact, we have $\#M' < 3K$ until the last request is processed.

Let $R$ be the set of requests in the current block that were matched outside the first copy. We show that $\#R < K$ after adding each next match, except perhaps after the last request.

Again, let $N(R)$ denote the set of neighbors of $R$ in the first copy. As in the previous lemma, $N(R)$ is covered by $M'$, thus $\#N(R) \leq \#M'$. Since $\#R < K$ was true during the previous step, after 1 more match, we have $\#R \leq K$. By 3-expansion up to $K$, we conclude that

$$3\#R \leq \#N(R) \leq \#M' < 3K,$$

and hence $\#R < K$. \blacktriangle

**Lemma 2.4.** If a graph has $T$-round matching, then a 2-clone has $T$-expiring matching.

**Proof.** The matching algorithm processes $T$ rounds on the first copy, then the next $T$ rounds on the other copy, then again $T$ rounds on the first one, and so on. At each switch, the matching has no edges in the copy because of $T$-expiration. \blacktriangle

### 3. $\varepsilon$-rich matching

We consider matchings in which a left node is matched to most of its right neighbors, and present an explicit family of graphs that have dynamic such matchings with $K$-expiration. This is used to construct one bitprobe storage schemes.

Given a graph with left degree $D$, an $\varepsilon$-rich matching for a set $S$ of left nodes is a set of edges in which each node in $S$ is incident on at least $(1 - \varepsilon)D$ edges. The dynamic $\varepsilon$-rich matching game is defined in the same way as before, but now, Requester needs to remove edges such that at most $K - 1$ left nodes are covered, and when he selects a left node $x$, Matcher needs to cover $x$ with $(1 - \varepsilon)D$ different edges. A graph has $\varepsilon$-rich matching if it has a winning strategy in the dynamic $\varepsilon$-rich matching game. Graphs with incremental and $T$-expiring $\varepsilon$-rich matchings are defined similarly.

The product of 2 graphs with the same left set $L$ and right sets $R_1$ and $R_2$ is the graph with left set $L$ and right set $R_1 \times R_2$ in which a left node $x$ is adjacent to $(y_1, y_2) \in R_1 \times R_2$ if $x$ is adjacent to both $y_1$ and $y_2$ in the respective graphs.

**Proposition 3.1.** If a graph with degree $D$ has $((1 - \varepsilon)D)$-expansion up to $K$, and another graph has $\varepsilon'$-rich matching up to $4 + 2\log K$, then their product has $K$-expiring $(2\varepsilon + \varepsilon')$-rich matching.
Remark. This is easily generalized from K-expiring to T-expiring matching, provided the 2nd graph has $\varepsilon'$-rich matching up to $4 + 2\log \max\{K, T\}$.

To prove Proposition 3.1, we first adapt Lemma 2.4 for incremental matching.

Lemma. If a graph has $((1 - \varepsilon)D)$-expansion up to $K$, then a $(1 + [\log K])$-clone has incremental $(2\varepsilon)$-rich matching up to $K$.

Proof. Matching algorithm given a request $x$. Select the first copy in which $x$ has a fraction $1 - 2\varepsilon$ of free neighbors and add these neighbors to $M$.

For $K = 1$, correctness is trivial. For the sake of induction, assume that the graph has expansion up to $2K$ and that this algorithm computes incremental matches up to $K$. We show that with 1 more clone it also computes incremental matches up to $2K$ by allocating at least half of its matches in the first copy.

Let $F$ be the set of requests for which the first copy was used and let $R$ be the set of other requests. Let $N(F)$ and $N(R)$ be the sets of their neighbors in the first copy.

$$N(F \cup R) = N(F) \cup \bigcup_{r \in R} (N(r) \setminus N(F)).$$

By choice of the algorithm, we have $\#N(r) \setminus N(F) \leq (1 - 2\varepsilon)D$, because otherwise $r$ would have enough free neighbors to be matched in the first copy. By expansion up to $2K$, we have

$$(1 - \varepsilon)D(\#F + \#R) \leq \#N(F \cup R) \leq D\#F + (1 - 2\varepsilon)D\#R.$$

After a calculation, we conclude that $\#R \leq \#F$. Thus for at least half of the requests, the first copy is used.

To finish the proof of the proposition, we need to decrease the load in lemma 2.4 from $O(\log K)$ to 1, and for this we apply the following.

Lemma. Assume a first graph has $\varepsilon$-rich matching up to $K$ with load $\ell$, and a second graph has $\varepsilon'$-rich matching up to $\ell$. Then the product has $(\varepsilon + \varepsilon')$-rich matching up to $K$. If the matching in the former 2 graphs is with $T$-expiration, so is the matching in the product graph.

Proof. Let $G$ and $G'$ be the graphs in the lemma. The matching strategy in $G \times G'$ runs the strategy in $G$, and for each right node in $G$, it runs the matching strategy of $G'$ separately. Each time the strategy of $G$ adds an edge to a right node $y$ in $G$, the left node makes a request in the $y$-copy of the $G'$-strategy. By definition of load $\ell$, this produces an $\varepsilon'$-rich matching in the $y$-copy of $G'$.

The union of all edges in all copies of $G'$ forms a set $M$ that satisfies the definition of $((1 - \varepsilon)(1 - \varepsilon'))$-rich matching, because given a request, $G$'s strategy produces $(1 - \varepsilon)D$ edges covering neighbors $y$, and for each such $y$, the $y$-copy produces edges on $(1 - \varepsilon')D'$ neighbors. Since $(1 - \varepsilon)(1 - \varepsilon') \geq 1 - \varepsilon - \varepsilon'$, the lemma is proven.

Finally, we apply the proposition to explicit graphs. The first one is an explicit expander based on [8], and the second one is a standard hash code with prime numbers, see for example [1] lemma 2.4 and appendix D for a proof.

Theorem 3.2 ([11, Th 18]). For all $\varepsilon$, $N$, and $K$, there exists an explicit graph with left size $N$, $((1 - \varepsilon)D)$-expansion up to $K$, left degree $D = (\log N)^{O(1)}(\frac{1}{2} \log K)^{O(\log \log K)}$, and right size $\text{poly}(D \log N)$. 

CVIT 2016
Lemma. For all $\varepsilon$, $N$, and $K$, there exists an explicit graph with left size $N$, right size $K^2 \cdot \text{poly}(\frac{1}{\varepsilon} \log N)$, and $\varepsilon$-rich matching up to $K$.

Corollary 3.3. For all $\varepsilon > 0$, $K$ and $N$, there exists an explicit graph with left degree $D = (\log N)^{O(1)}(\frac{1}{\varepsilon} \log K)^{O(\log \log K)}$ and right size $K \text{ poly}(D, \log N)$, that has $\varepsilon$-rich matching up to $K$ with $K$-expiration. Moreover, with an additional data structure the matching algorithm runs in time $O(D \log N)$.

4 Polynomial time dynamic matching

Proposition 4.1. If a graph with left size $N$ and left degree $D$ has $(\frac{2}{3}D + 2)$-expansion up to $K+1$ then it has a dynamic matching algorithm up to $K$ in which each match is computed in time $\text{poly}(N)$.

This algorithm is used in the faster $O(D \log N)$-time algorithm in Theorem 1.1. The proof uses 2 technical lemmas. Given a set $S$ of left nodes, we call a right node private for $S$ if it has precisely 1 left neighbor in $S$. Let $\mathcal{N}(S)$ be the set of neighbors of $S$.

Lemma. The number of private neighbors of $S$ is at least $2\# \mathcal{N}(S) - D\#S$.

Proof. We need to lower bound the number $p$ of private neighbors of $S$. The number of vertices in $\mathcal{N}(S)$ that are not private, equals $\# \mathcal{N}(S) - p$. There are $D\#S$ edges with an endpoint in $S$. For each such edge, the right endpoint is either private or has at least 2 neighbors in $S$. Hence

$$D\#S \geq p + 2(\# \mathcal{N}(S) - p).$$

The lower bound of the lemma follows by rearranging.

The following lemma holds for graphs satisfying the assumption in the proposition.

Lemma. Let $Y$ be a subset of right nodes with $\#Y \leq 2K + 1$. If a set $P$ contains only left nodes $x$ with $\# \mathcal{N}(x) \cap Y \geq D/3$, then $\#P \leq K$.

Proof. Suppose $P$ contains at least $K + 1$ elements, and let $S$ be a subset of $P$ of size exactly $K + 1$. By expansion, $S$ has at least $(\frac{2}{3}D + 2)\#S$ right neighbors. By assumption on $P$, each of its nodes has at most $\frac{2}{3}D$ neighbors outside $Y$. Thus,

$$\# \mathcal{N}(S) \cap Y \geq (\frac{2}{3}D + 2)\#S - \frac{2}{3}D\#S = 2\#S = 2(K + 1).$$

But this contradicts $\#Y \leq 2K + 1$.

Proof of Proposition 4.1 The idea of the matching algorithm is to assign a “virtual match” to left nodes for which at least $D/3$ neighbors are matched. Note that there are 2 types of matches to which we refer as standard and virtual matches. In the $D/3$ bound, we count both types of matches. Virtual matches are treated as actual matches and other nodes can not be matched to it.

Left nodes with at least $D/3$ matched neighbors are called critical. A virtual match will be assigned to a left node $x$ if and only if $x$ is critical and has no standard match.

---

Footnote: In fact, with a similar proof it is enough to assume $(\frac{2}{3}D + 2)$-expansion up to $K$ instead of $K + 1$. For this we define nodes to be critical if they have at least $D/3 + 1$ matched neighbors instead of $D/3$, and follow the proof with extra $+1$’s and $-1$’s where needed.
Matching algorithm when given a request $x$. If the request is a critical node, then its virtual match $y$ is returned, and thus $y$ is now a standard match. Otherwise, $x$ is matched to any neighbor $y$ that does not have a match (of either type). Below it is explained that such a neighbor exists.

After this, there might be critical nodes which do not have a match (of either type). Let $S$ be the set of such nodes. Virtual matches for these elements are assigned 1 by 1 as follows.

Select an unmatched right node $y$ that has exactly 1 neighbor in $S$. Below we explain that such a $y$ always exists. Let $x$ be this unique neighbor. Remove $x$ from $S$ and declare $y$ to be the virtual match of $x$. Add to $S$ all new critical nodes without a match. Keep repeating until $S$ is empty. (This must happen, because an element can be added at most once.)

Matching algorithm when a match $(x, y)$ is retracted. If $x$ is critical, then declare $y$ to be a virtual match. Otherwise, retract the match and retract all virtual matches of left nodes with less than $D/3$ matches. This finishes the description of the matching algorithm.

Note that updates require $\text{poly}(N, D)$ amount of computation. We may assume that $D \leq N$, since otherwise the proposition is trivial. Hence, the runtime is $\text{poly}(N)$. In the presentation of the algorithm 2 claims were made. Note that if these claims are true, then the dynamic matching algorithm always produces matches and the proposition is proven.

The first claim is easy: every non-critical node always has an unmatched neighbor. Indeed every non-critical node has less than $D/3$ matched neighbors and hence more than $2D/3 > 0$ unmatched ones.

The second claim is the hardest to prove, and we first prove 2 other claims.

Proof that at any moment, at most $K$ nodes are critical. In the above algorithm, matches are added 1 by 1. Assume that just before allocating a match there are at most $K$ critical nodes. Then the number of standard and virtual matches is at most $K + K$ (and in fact, it is 1 less, but this doesn’t matter). Let $Y$ be the set of matched right nodes with the new match included, thus $\#Y \leq 2K + 1$. By the second lemma, there are still at most $K$ critical nodes.

Proof that at any moment, all nodes in $S$ have exactly $\lceil D/3 \rceil$ matched neighbors. By construction a node is placed in $S$ when it has at least $D/3$ neighbors. This condition is checked each time after a match is assigned, thus when a node is added to $S$, it has exactly $\lceil D/3 \rceil$ neighbors. As long as $S$ is nonempty, a virtual match $y$ is given to a left node $x$ such that $y$ has no other neighbor in $S$, and then $x$ is removed. Thus for all other nodes in $S$, the number of matched neighbors remains the same.

Proof that in the above matching algorithm, an unmatched node $y$ exists that has exactly 1 left neighbor in $S$. Since all nodes in $S$ are critical, we have $\#S \leq K$. By the assumption on expansion, $\#N(S) \geq (\frac{2}{3}D + 2)\#S$. By the first lemma, $S$ has at least $(\frac{1}{3}D + 4)\#S$ private neighbors. At most $\lceil \frac{1}{3}D \rceil \#S$ of the private neighbors can be matched, by the previous point. Hence, at least $3\#S$ right nodes are private and unmatched. Thus, if $\#S \geq 1$, the required right node $y$ exists.

5 Fast dynamic matching

We finish the proof of the main result, Theorem 1.1. The matching strategy combines an $O(D \log N)$ time greedy strategy from section 2 with the $\text{poly}(N)$ time strategy of section 4. The greedy strategy allocates most matches, while the polynomial one is used for a few problematic requests that are anticipated well in advance.
Recall that in fast dynamic matching we use a data structure to compute matches. We consider a relaxed notion of fast matching that besides algorithms to compute matches and handle retractions, also has a preparation algorithm. This algorithm is run at regular intervals and does not need to be fast. This algorithm prepares the data structure for fast computation of future matches.

**Definition.** We say that a graph with left size $N$ and left degree $D$ has fast dynamic matching with $T$-step preparation if there exists a dynamic matching algorithm that computes matches and processes retractions in time $O(D \log N)$. Moreover, each time after $T$ matches have been assigned, it runs a preparation algorithm that takes $O(T)$ time.

**Remark.** A 2-clone of such a graph has fast dynamic matching because blocks of $T$-subsequent requests can alternatingly be given to the copies: while one copy is used for assigning matches, the other can run its preparation algorithm (in little chunks at each request).

Let $G$ and $G'$ be graphs with vertices $V$ and $V'$, and with edges $E$ and $E'$. The union of $G$ and $G'$ is the graph with vertices $V \cup V'$ and edges $E \cup E'$.

**Lemma.** Consider two graphs with the same left set of size $N$. If the first has $(\frac{1}{2} D + 3)$-expansion up to $K$ and the second has polynomial time dynamic matching up to $2K$, then their union has fast dynamic matching up to $K$ with load $O(\log N)$ and poly$(N)$-step preparation.

Before proving the lemma, we show that it implies the main result.

**Proof of Theorem 1.** Let $D$ be the degree of the graph in the assumption of the theorem, and let $G$ be a 2-clone of it. Graph $G$ has degree $2D$ and expansion $2(\frac{3}{2} D + 2) \geq \frac{1}{2}(2D) + 3$ up to $K$.

By proposition Proposition 4.1, graph $G$ has polynomial-time dynamic matching up to $K - 1$. By applying the lemma to $G \cup G = G$, this graph has dynamic matching up to $(K - 1)/2$ with load $O(\log N)$ and poly$(N)$ preparation time.

By the remark above, a 2-clone of $G$ has dynamic matching up to $(K - 1)/2$ with load $O(\log N)$. Hence, a 4-clone has dynamic matching up to $K - 1$ with the same load.

Therefore, the original graph of the theorem has matching up to $K - 1$ with $O(\log N)$ by multiplying by $8 = 4 \cdot 2$ the constant hidden in $O(\cdot)$. Finally, adding 1 more match increases the load by at most 1. We obtained matching up to $K$ with load $O(\log N)$. The theorem is proven, except for the proof of the lemma.

**Proof of the lemma.** Let $F$ be the graph with $(\frac{1}{2} D + 3)$-expansion and let $G$ be the graph with polynomial time dynamic matching. We may assume that their right nodes are disjoint, because this affects the load by at most a factor 2. Let $T$ be a polynomial of $N$ that we determine later.

The preparation algorithm precomputes matches for nodes that are at-risk (the precise definition is below). The preparation and retraction algorithms share a queue containing matches in $G$.

To process the first $T$ requests, we apply the fast algorithm from Lemma 2.3 using $F$. Since $F$ has 3-expansion, we obtain matching with load $O(\log T)$.

**Preparation algorithm.** Retract all matches from the queue. Also retract all precomputed matches from the previous run of the preparation algorithm that are not matches in $G$.

We call a right node of $F$ disabled if it is matched. The others are called enabled. Let $A$ be the set of left nodes with at least $D/2$ disabled neighbors (the at-risk nodes). Compute...
the induced subgraph $F'$ of $F$ containing the left nodes not in $A$ and the enabled right nodes. The set $A$ and this graph are fixed until the next run of the preparation algorithm and will be used in the fast computation of matches below.

Precompute matches in $G$ for all nodes in $A$ that do not already have a match. Do this by generating requests 1-by-1 in any order. We soon explain that this is successful.

**Computing a match for $x$.** If $x$ is in $A$, return its precomputed match in $G$. Otherwise, run the fast $T$-round algorithm from Lemma 2.3 on the graph $F'$ computed in the previous preparation phase. (We soon explain that $F'$ has 3-expansion up to $K$.)

**Retracting $(x, y)$.** If $(x, y)$ is in $G$, then add the edge to the queue. Otherwise, remove the edge.

The value of $T$ is chosen to be a polynomial in $N$ large enough so that the preparation algorithm can be performed in time $T$. By Lemma 2.3, the running times of the two algorithms for computing and, respectively, retracting a match satisfy the conditions.

Above, 2 claims were made that need a proof. After this, the lemma is proven, because by construction, the load of all nodes in $G$ is bounded by 1, and for $F$ it is bounded by $1 + \lfloor \log T \rfloor$.

**Proof that $F'$ has 3-expansion up to $K$.** Let $S$ be a set of left nodes in $F'$ of size at most $K$. By expansion in $F$, the set has at least $(\frac{1}{2}D + 3)\#S$ neighbors in $F$. By choice of $A$, each element in $S$ has at most $\frac{1}{2}D$ disabled neighbors in $F$. Thus the number of neighbors in $F'$ is at least

$$(\frac{1}{2}D + 3)\#S - \frac{1}{2}D\#S = 3\#S.$$ 

**Proof that the polynomial-time matching algorithm precomputes matches for all the nodes in $A$.** First we show that $\#A < K$. Suppose that $\#A \geq K$ and let $S$ be a subset of $A$ with exactly $K$ elements. Let $M$ be the set of matches in $F$. Each node in $S$ has at most $\frac{1}{2}D$ neighbors that are not covered by $M$. Hence, the number of neighbors of $S$ in $F$ is at most

$$\#M + (\frac{1}{2}D)\#S.$$ 

By the expansion of $S$ in $F$ and because $\#M \leq K$, we conclude that

$$(\frac{1}{2}D + 3)\#S \leq \#N(S) \leq K + \frac{1}{2}D\#S.$$ 

Hence, $3\#S \leq K$, but this contradicts $\#S = K$.

The claim follows because less than $2K$ precomputed matches can exist simultaneously. Indeed, there are less than $K$ matches computed in the current run and also at most $K$ matches from previous runs that became actual matches and have not been retracted. \hfill \blacktriangleleft

# 6 Discussion

Table 2 summarizes various Hall-type results which state that a certain expansion property implies a certain type of matching.

For the case of $\varepsilon$-rich matching, we have shown (the result is not included in this version) that any graph with $(1 - \varepsilon)D$-expansion up to $2K$ has $T$-expiration dynamic $(2\varepsilon, O(\log(KT)))$-rich matching up to $K$, in which a matching assignment/retraction can be done in time $D \cdot \text{poly}(\log N) \cdot \log T$. 

CVIT 2016
Hall-type theorems for fast dynamic matching and applications

| expansion up to $K$ | matching up to $K$ | load | runtime per match | reference |
|---------------------|---------------------|------|-------------------|-----------|
| 1                   | offline             | 1    | $N/\Lambda$       | Hall’s Theorem |
| 1                   | dynamic             | 3    | $N^{K+O(1)}$      | [7, Prop. 1] |
| $T$-expiration      | dynamic             | O($\log(KT)$) | poly($\log(NT), D$) | Proposition 2.1 |
| $2D/3 + 2$          | dynamic             | 1    | poly($N, D$)      | Proposition 4.1 |
| $2D/3 + 2$          | dynamic             | O($\log N$) | O($D \log N$)    | Theorem 1.1 |

Table 2: A graph with $N$ left nodes and left degree $D$ satisfying the condition in column 1 has matching with the features in columns 2, 3, and 4. The 4th column is the worst case runtime for finding or retracting one matching assignment.

There exist explicit graphs with 1-expansion up to $K$ that have degree $D = \text{poly}(\log N)$ and right size $\#R = K \cdot \text{poly}(\log N)$. Such graphs can be obtained from the dispersers in [16] (for details see [3, Section 3.2]). In contrast, there exist explicit graphs with $(2D/3 + 2)$-expansion up to $K$ with $D = \text{poly}(\log N) \cdot 2^{O((\log \log K)^2)}$ and $\#R = K \cdot \text{poly}(D, \log N)$ ([11, Th. 18]).

**Open question.** Given the difference of the degree $D$ between explicit graphs with 1-expansion up to $K$ and graphs with $(2D/3 + 2)$-expansion up to $K$, it is interesting to find out whether graphs of the former type have fast (or at least polynomial-time) dynamic matching like the graphs of the latter type. We suspect that this is not the case, but proving such a result requires some hardness assumption because if $P = NP$, the algorithm from [7, Prop. 1] (implicitly invoked in the second line of the table) runs in time $\text{poly}(N, D)$.

**References**

1. Sanjeev Arora, F. Thomson Leighton, and Bruce Maggs. On-line algorithms for path selection in a nonblocking network. *SIAM Journal on Computing*, 25:149–158, 1996.
2. L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1973.
3. Bruno Bauwens, Anton Makhlin, Nikolai K. Vereshchagin, and Marius Zimand. Short lists with short programs in short time. *Computational Complexity*, 27(1):31–61, 2018. doi:10.1007/s00037-017-0154-2
4. Bruno Bauwens and Marius Zimand. Universal almost optimal compression and Slepian-Wolf coding in probabilistic polynomial time. *J. ACM*, 70(2):1–33, 2023. (arxiv version posted in 2019).
5. M. Blum, R.M Karp, O. Vornberger, C.H. Papadimitriou, and M. Yannakakis. The complexity of checking whether a graph is a superconcentrator. *Information Processing Letters*, 13(4,5):164–167, 1981.
6. Harry Buhrman, Peter Bro Miltersen, Jaikumar Radhakrishnan, and Srinivasan Venkatesh. Are bitvectors optimal? In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, May 21-23, 2000, Portland, OR, USA, pages 449–458. ACM, 2000. doi:10.1145/335305.335357
7. Paul Feldman, Joel Friedman, and Nicholas Pippenger. Wide-sense nonblocking networks. *SIAM J. Discret. Math.*, 1(2):158–173, 1988. doi:10.1137/0401018
8. Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *J. ACM*, 56(4), 2009.
9. Monika Henzinger, Shahbaz Khan, Richard Paul, and Christian Schulz. Dynamic matching algorithms in practice. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy*
A Constant-depth connectors with fast path finding algorithms

Graphs with dynamic matching up to $K$ can be composed into $N$-connectors of constant depth $t$. Moreover, the number of edges is close to optimal: if the left degree is $D$ and the right size is at most $CK$, then the number of edges is minimal within a factor $O(tCD)$. The following is a modification of [7, Proposition 3.2].

\textbf{Proposition A.1.} Let $N$ be a $t$-th power of an integer. Assume that for some $C$ and $D$, for all integers $c < t$, we have graphs with $CN^{c+1/t}$ left nodes, at most $CN^{c/t}$ right nodes, left degree at most $D$, and dynamic matching up to $N^{c/t}$. Then, there exists an $N$-connector of depth $t$ with at most $tCDN^{1+1/t}$ edges.

Optimality follows from the lower bound $\Omega(N^{1+1/t})$ given in [15, Proposition 2.1].

\textbf{Remark.} To compute a path (or extend a tree), at most $t$ matches need to be computed. Thus, for matching obtained from Theorem 1.1, this requires $O(tD \log N)$ time.

First the construction for rearrangeable networks is given. An $(N, N')$-network is a directed acyclic graph with $N$ input and $N'$ output nodes. Recall that the network is rearrangeable if every 1-to-1 mapping from outputs to inputs can be realized using node disjoint paths. The following 2 lemmas are directly obtained from the definitions.

\textbf{Lemma.} Consider a graph with $N$ left and $N'$ right nodes that has offline matching up to $K$. The concatenation of this graph with a rearrangeable $(N', K)$-network is a rearrangeable $(N, K)$-network.

\textbf{Lemma.} Consider $B$ rearrangeable $(N, N')$-networks with the same set of inputs and disjoint outputs. The union of these $B$ networks is a rearrangeable $(N, BN')$-network.
Proof of the proposition. Let $N = B^t$ for an integer $B$.

Construction of a rearrangeable $(CN, N)$-network. For every $c \leq t$, we construct a $(CB^c, B^c)$-rearrangeable network of depth $c$ recursively. For $c = 1$, we use the complete bipartite graph with left size $CB$ and right size $B$.

Suppose for some $c \geq 1$, we already have such a network $H$. First obtain an $(CB^{c+1}, B^c)$-network of depth $c + 1$ by introducing $CB^{c+1}$ input nodes, denoted by the set $I$ and connect them to $H$ according to a graph with matching up to $B^c$ in the statement of the proposition. Then merge $B$ copies of this graph having the same set $I$ of inputs and disjoint sets of outputs.

The rearrangeability property follows from the 2 lemmas above.

Proof that the network has at most $tCB^{t+1}$ edges. We prove this by induction on $t$. For $t = 1$, the network is fully connected and has at most $(CB) \cdot B \leq tCB^{t+1}$ edges.

Let $t \geq 2$ and assume that the construction of a $(CB^{t-1}, B^{t-1})$-connector of depth $t - 1$ contains at most $(t - 1)CB^t$ edges. The $(CB^t, B^t)$-network consists of $B$ such connectors and $B$ graphs with $CB^t$ left nodes. Thus, the total number of edges is at most

$$B \cdot (DCB^t + (t - 1)CB^t) = tCB^{t+1}.$$ 

With exactly the same construction, connectors are obtained. The matching game on $(N, N')$-networks is defined in precisely the same way, and using this game, $(N, N')$-connectors are defined. We adapt the 2 lemmas above.

- If a graph with sizes $N$ and $N'$ has dynamic matching up to $K$, then its concatenation with an $(N', K)$-connector is an $(N, K)$ connector.
- The union of $C$ output disjoint $(N, N')$-connectors is an $(N, CN')$-connector.

For the first, when Requester provides an input–output pair $(i, o)$, then request a match $i'$ for $i$ in the first graph, and generate a request $(i', o)$ in the connector. Since there are $K$ outputs, at most $K$ matches are simultaneously needed.

The second is easy to understand, since the path finding (or better tree extension) algorithms of separate copies do not interfere. Both claims together provide the connectors of the proposition. ▶

Corollary 1.3 follows by applying this construction to the matching algorithm from Theorem 1.1, applied to the lossless expander obtained from Theorem 3.2 as follows. Choose $\varepsilon = 1/4$ and left size $N^2$. For each $K \leq N$, the expander of Theorem 3.2 satisfies:

$$\max\{\text{left degree}, \frac{\#\text{right set}}{K}\} \leq \text{poly}(\log N \exp O(\log^2 \log N)) \leq \exp(O(\log^2 \log N)),$$

and this is bounded by $N$ for large $N$. Let $C = \varepsilon$ the right side. From this expander, we only use the first $CN^{(c+1)/t}$ of the $N^2$ left nodes and drop the others. This yields the graphs satisfying the conditions of Proposition A.1 with $D = \exp(O(\log^2 \log N))$.

For non-explicit constructions, we can use an expander with smaller degree. In fact, a random graph has good expansion properties as explained for example in [18, Th. 6.14] or [4, Appendix C].

Lemma. For each $N$ and $K$, there exists a $(\frac{3}{2}D)$-expander up to $K$ with left degree $D = O(\log N)$, left size $N$ and right size $O(KD)$.

Corollary 1.2 follows from Theorem 1.1 instantiated with this expander.
### One bitprobe storage scheme for dynamic sets

The goal is to store a $K$-element set $S \subseteq [N]$, where typically $K \ll N$. A one bitprobe (or bit vector) storage scheme is a data structure in which queries “Is $x$ in $S$?” are answered probabilistically by reading a single bit. Previous constructions for one bitprobes are for static sets. We show that graphs that admit $\varepsilon$-rich matching can be used to obtain one bitprobe storage schemes for dynamic sets: the data structure also allows for efficient insertions and deletions from $S$.

A static one bitprobe is a data structure that is described by a size $s$ and a probabilistic algorithm $\text{pos}$ mapping $[N]$ to $[s]$, which selects a bit to answer a membership query. Let $\mathbf{x} \in \{0, 1\}^*$ be 1 if $x \in S$ and 0 otherwise.

**Formal requirement for a one bitprobe with parameters $N, K, \varepsilon$.**

For all $S \subseteq [N]$ with $\#S \leq K$ there exists $v \in \{0, 1\}$ such that for all $x \in [N]$,

$$\Pr[v_{\text{pos}}(x) = [x \in S]] \geq 1 - \varepsilon.$$  

A dynamic one bitprobe additionally has an update function for adding and removing elements from the set. A history is a list of integers that describes these operations chronologically, where a positive integer $i$ represents the addition of $i$ to the set, and $-i$ its removal.

For a history $h \in \mathbb{Z}^*$, let $\text{set}(h)$ be the set of elements that remain after the sequence of operations (thus it is the set of positive entries $i$ in $h$ with no appearance of $-i$ at their right). In the definition of one bitprobes we consider histories that at any moment encode sets of size at most $K$.

**Definition.** History $h \in \mathbb{Z}^*$ is $(N, K)$-legal if $|h| \in [N]$ and $\# \text{set}(\tilde{h}) \leq K$ for each prefix $\tilde{h}$ of $h$.

**Definition.** A dynamic one bitprobe with parameters $N, K, \varepsilon$ consists of

- a size $s$,
- a deterministic algorithm $\text{upd} : \mathbb{Z} \times \{0, 1\}^s \to \{0, 1\}^s$, and
- a probabilistic algorithm $\text{pos} : [N] \to [s]$,

such that for all $(N, K)$-legal histories $h$ and all $x \in [N]

$$\Pr[\text{upd}(h, 0^s)_{\text{pos}}(x) = [x \in \text{set}(h)]] \geq 1 - \varepsilon,$$

where $\text{upd}(h_1 \ldots h_k, v) = \text{upd}(h_k, \text{upd}(h_2, \ldots, \text{upd}(h_1, v) \ldots))$.

Our main result shows the existence of dynamic one bitprobes that are small and have efficient implementations for queries and updates.

**Theorem B.1.** There exists a family of one bitprobes with parameters $(N, K, \varepsilon)$ with

- size $K(\log N)^{O(1)}(\frac{1}{\varepsilon} \log K)^{O(\log \log K)}$,
- query time poly($\log N$),
- update time $(\log N)^{O(1)}(\frac{1}{\varepsilon} \log K)^{O(\log \log K)}$.

We start with the simpler case of static one bitprobes, which follows directly from graphs with $\varepsilon$-rich incremental matching.

---

4 This requirement is stronger than in the standard definition of bitprobes, where the bit $v_{\text{pos}}(x)$ may correspond both in a positive and a negative way to the membership condition.
Lemma. If a left regular graph with left and right sets \([N]\) and \([s]\) has incremental \(\varepsilon\)-rich matching up to \(K + 1\), then the mapping \(\text{pos}\) that maps a left node to a random neighbor defines a static one bitprobe of size \(s\) with parameters \((N, K, \varepsilon)\).

Proof. Given a \(K\)-element \(S \subseteq [N]\), run the matching algorithm for all elements of \(S\) in an arbitrary order and let \(v \in \{0, 1\}^s\) be the string that has 1's in precisely those indices in \([s]\) that are covered by the matching.

We prove that \(v \in \{0, 1\}^s\) satisfies the one bitprobe condition. Indeed, if \(x \in S\), then at least \(1 - \varepsilon\) of \(x\)'s neighbors are covered, and hence \(\Pr[v_{\text{pos}(x)} = 1] \geq 1 - \varepsilon\). Assume \(x \notin S\). If the incremental matching algorithm would be given \(x\), then, it would find \((1 - \varepsilon)D\) right neighbors that are not matched, since the incremental matching is up to \(K + 1 > \#S\). By choice of \(v\), the corresponding indices are 0, and hence \(\Pr[v_{\text{pos}(x)} = 0] \geq 1 - \varepsilon\).

In fact, by this proof we obtain one bitprobes in which elements can be dynamically inserted but not removed. If there exist graphs with dynamic \(\varepsilon\)-rich matching, then we could apply a similar argument and we are done, but we do not know whether such graphs with small right sizes exist.

Fortunately, it is enough to have graphs with \(\varepsilon\)-rich matching with \((2K)\)-expiration. The idea is to refresh old elements. More precisely, if an element \(x\) was inserted, and it was not removed during the \(K\) next insertions, then we delete \(x\) and reinset \(x\). After this modification, each insertion in the probe's history corresponds to 2 rounds of the matching game, and hence, \((2K)\)-expiration is required.

Lemma. If there exists a graph with degree \(D\), left set \([N]\) and right set \([s]\) that has \((2K)\)-expiring \(\varepsilon\)-rich matching up to \(K + 1\), then there exists a dynamic one bitprobe of size \(s + O(KD \log N)\) with parameters \((N, K, \varepsilon)\). Moreover, if the graph is explicit, then query time is \(\text{poly}(\log N, \log D)\).

Proof. The \(\text{pos}\) function is defined as in the previous lemma and satisfies the conditions on the runtime.

The update function requires the storage of the matching and the last \(K\) insertions of the history. This increases the size \(s\) by \(KD[\log N]\) for storing the matching and by \(K[\log N]\) for the queue.

For removals, the update function first checks the presence of the element in the stored set. If not present, it is finished. Otherwise, it runs the retraction algorithm of the matching, and sets the bits of \(v\) to zero for the right nodes that are no longer covered.

For inserting a node \(x\), the update function first checks whether \(x\) is already present in the stored set. If so, it finishes. Otherwise, it refreshes the \(K\)-th oldest insertion, (it runs the retraction and then the matching algorithm for it), runs the matching algorithm for \(x\), and sets the assigned bits to 1.

To see that this works, we need to verify that every match is retracted after at most \(2K\) requests of the matching algorithm. Indeed, \(K\) insertions in the probe’s history now correspond to at most \(2K\) requests for the matching algorithm. If an element is removed after at most \(K\) other insertions, then we are done. Otherwise, the update algorithm will retract it at the \(K\)-th insertion.

\(^5\) In fact, this could be avoided, since we do not consider running times now, but this require us to explain something else, and eventually, we need a version of this result with runtimes.
Lemma. Suppose that the retraction and matching algorithms in the previous lemma run in time \( \text{poly}(D, \log N) \). Then the update algorithm of the one bitprobe runs in time \( \text{poly}(D, \log N) \) as well.

Proof. The only technical issue here is that one needs to efficiently store and access the matching and the \( K \) last insertions of the history. For the latter, we simply use a single linked list. For the matching, one may use for example a Red-Black search tree.

Proof of Theorem B.1. Apply the above lemma to the graph from Corollary 3.3.

Remark. The explicit lossless expander \( G \) from Theorem 3.2 is based on the construction in [8] and is not practical. Other than this, the algorithms of our one bitprobe storage scheme in Theorem B.1 are very efficient. It is conceivable that replacing the lossless expander with empirical hash functions (for instance CityHash, murmur, SHA-3, etc.) may lead to implementations that are attractive in practice (even though the proven guarantees would be lost).

C The (slow) dynamic matching algorithm of Feldman, Friedman, and Pippenger

For completeness, we present a special case of [7, Proposition 1]. Our proof is based on the original one. The result implies that if a graph has offline matching up to \( K \), then it has dynamic matching up to \( K \) elements with load 3.

Theorem C.1. If a graph has 1-expansion up to \( K \) and each left set \( S \) with \( K < \#S \leq 2K \) has at least \( \#S + K \) neighbors, then the graph has dynamic matching up to \( K \).

Corollary. If a graph \( G \) has 1-expansion up to \( K \), then it has dynamic matching up to \( K \) with load 3.

Proof. We modify \( G \) by taking 3 clones of each right node. The new graph \( G' \) satisfies the hypothesis of Theorem C.1. Indeed, let \( S \) be subset of left nodes with \( K < \#S \leq 2K \). We partition \( S \) into a set \( S_1 \) of size \( K \) and a set \( S_2 \) of size \( \#S - K \leq K \). \( S_1 \) has at least \( 2K \) neighbors in the right subset made with the first 2 clones, and \( S_2 \) has at least \( \#S - K \) neighbors in the set made with the third clones. Thus, \( S \) has at least \( 2K + \#S - K = \#S + K \) neighbors. Theorem C.1 implies that \( G' \) has dynamic matching up to \( K \). By merging the 3 clones into the original nodes, it follows that \( G \) has dynamic matching with load 3.

We continue with the proof of Theorem C.1. We start with 2 technical lemmas.

Definition. For a set of nodes \( S \), let \( \mathcal{N}(S) \) be the set of all neighbors of elements in \( S \). A left set \( S \) is critical if \( \#\mathcal{N}(S) \leq \#S \).

Lemma. If \( A \) and \( B \) are critical and \( \#\mathcal{N}(A \cap B) \geq \#A \cap B \), then \( A \cup B \) is also critical.

The one bitprobe storage scheme in Theorem B.1 has smaller size than the one bitprobe storage schemes in [6, 17, 8] (provided \( \varepsilon \geq 1/K^{1/\log \log N} \), see Table 1), even though these schemes have the limitation of handling only static sets. Plugging in the above generic construction the lossless expander used in [8], one obtains a one bitprobe storage scheme for dynamic sets with data structure size equal to (storage size from [8] \( \times O((\log N \cdot \log K \cdot 1/\varepsilon)^2) \), in which \( \text{pos} \) and \( \text{upd} \) have runtime \( \text{poly}(\log N \cdot \log 1/\varepsilon) \). Compared to Theorem B.1, \( \text{upd} \) is faster. The reason is that the lossless expander in [8] has \( D = \text{poly}(\log N, \log(1/\varepsilon)) \), smaller than the left degree of the graph in Theorem B.1 (but the right set is larger).
Proof. We need to bound the quantity $\#N(A \cup B)$ which equals $\#N(A) \cup N(B)$. By the inclusion-exclusion principle this equals

$$= \#N(A) + \#N(B) - \#N(A) \cap N(B).$$

Since $N(A \cap B) \subseteq N(A) \cap N(B)$ and the assumption of the lemma, this is at most

$$\leq \#N(A) + \#N(B) - #A \cap B.$$  \hfill \Box

Lemma. Assume a graph has 1-expansion up to $K$ and has no critical set $S$ with $K < \#S \leq 2K$. Then, for every left node $x$ there exists a right node $y$ such that after deleting $x$ and $y$, the remaining graph has 1-expansion up to $K$.

Proof. A right neighbor $y$ of $x$ is called bad if after deleting $y$, there exists a left set $S_y$ of size at most $K$ such that $\#N(S_y) < \#S_y$. Note that $S_y$ is critical, and by the 1-expansion of the original graph, $N(S_y)$ contains $y$. We show that by iterated application of the above lemma, the set

$$U = \bigcup_{y \text{ is bad}} S_y$$

is critical. Indeed, for each critical set $C$ of size at most $K$, the set $C \cup S_y$ is critical by 1-expansion and the previous lemma. Also this set has cardinality at most $2K$, thus by the assumption this union must have cardinality at most $K$.

Note that if all neighbors $y$ of $x$ were bad, then $N(U \cup \{x\}) = N(U)$ because $y \in N(S_y) \subseteq N(U)$. Thus

$$\#N(U \cup \{x\}) \leq \#U \leq \#U \cup \{x\}.$$  \hfill \Box

If $\#U < K$, then this violates 1-expansion, and if $\#U = K$, this violates the assumption about the sizes of critical sets. Hence, at least 1 neighbor of $x$ is not bad and satisfies the conditions of the lemma.

Proof of Theorem C.1. The dynamic matching strategy maintains a copy of the graph. If Requester makes a matching request for a left node $x$, Matcher replies by searching for a right node $y$ that satisfies the condition of the above lemma for the copy graph and adds the edge $(x, y)$ to the matching $M$. In the copy she deletes the nodes $x$ and $y$. When Requester removes an edge $(x, y)$ from $M$, Matcher restores the nodes $x$ and $y$ in the copy graph.

It remains to show that in each application of the above lemma, the conditions are satisfied. Note that if Matcher restores the endpoints $x$ and $y$ of an edge, the conditions always remain true, because if $x \notin S$, then $\#S$ and $\#N(S)$ do not change, and otherwise both values increase by 1.

It remains to show that before any matching request, the copy graph has no critical set $S$ with $K < \#S \leq 2K$ (and thus the Matcher can apply the lemma and satisfy the request). Assume to the contrary that there is such an $S$. In the original graph, $S$ has at least $\#S + K$ neighbors. When a right neighbor is assigned, Matcher deletes it from the copy graph. Therefore before any request, the Matcher has deleted from $S$ at most $K - 1$ right nodes (since there can be at most $K - 1$ active requests), hence, $S$ has at least $\#S + K - (K - 1) = \#S + 1$ neighbors, thus it is not critical.

Therefore, the conditions of the lemma are always satisfied and the strategy can always proceed by selecting a neighbor $y$. The theorem is proven.  \hfill \Box
Remark. In the matching algorithm from [7], the condition on the 1-expansion up to \( K \) elements is checked using a brute force check over all left sets of size at most \( K \). This can be done in \( O\left(\binom{\#L}{K}\right) \) time. In general, checking whether a graph has 1-expansion up to \( K \) elements is coNP-complete, see [5]. However, this hardness result does not exclude algorithms that run in time \( \text{poly}(\log \#L) \) for specially chosen graphs.

\textbf{D} \hspace{1cm} \textit{Prime hashing implies \( \epsilon \)-rich matching}

\textbf{Lemma.} For all \( \epsilon, N, \) and \( K \), there exists an explicit graph with left size \( N \), right size \( K^2 \cdot \text{poly}\left(\frac{1}{\epsilon} \log N\right) \), and \( \epsilon \)-rich matching up to \( K \).

\textbf{Proof.} Let \( D = \frac{1}{\epsilon}K \log N \). Let \( p_i \) denote the \( i \)-th prime number. Left nodes are \( \{1, \ldots, N\} \), and right nodes are pairs \( \{0, \ldots, p_D\}^2 \). Note that \( p_D \leq O(D \log D) \), and the condition on the right size is satisfied for \( K \leq N \). For \( K > N \) the lemma is trivial.

A left node \( x \) is connected to all pairs \((p_i, x \mod p_i)\) with \( i \leq D \). The matching strategy is the greedy strategy that matches a node \( x \) to all unmatched right neighbors.

We prove that this provides \( \epsilon \)-rich matchings. Assume that there are matches for \( x_1, \ldots, x_{K-1} \), and let \( \tilde{x} \) be an element that is not in this set. For each \( x_i \), there are at most \( \log N \) common neighbors of \( \tilde{x} \) and \( x_i \). Hence, at most a fraction \( (K \log N)/D \) of \( \tilde{x} \)'s neighbors have already been matched. Thus the greedy matching is \( \epsilon \)-rich.

\textbf{CVIT 2016}