Hall-type theorems for fast almost dynamic matching and applications

Bruno Bauwens∗  Marius Zimand†

Abstract

A dynamic set of size up to $K$ is a set in which elements can be inserted and deleted and which at any moment in its history has at most $K$ elements. In dynamic matching in a bipartite graph, each element, when it is inserted in a dynamic subset of left nodes, makes a request to be matched with one of its neighbors, and the request has to be satisfied on-the-fly without knowing future insertions and deletions and without revoking past matchings. We consider a relaxation of dynamic matching in which each matching can survive at most $T$ insertions, and a right node can be assigned to more than one node of the dynamic set. We show that a bipartite graph satisfying the condition in Hall Marriage Theorem up to $K$ has fast $T$-surviving dynamic matching for dynamic sets of size up to $K$, in which every right node can be assigned to at most $O(\log(KT))$ left nodes. Fast matching means that each matching is done in time $\text{poly}(\log N, \log T, D)$, where $N$ is the number of left nodes, and $D$ is the left degree. We obtain a similar result for $\epsilon$-rich matching, in which a left node needs to be assigned $(1-\epsilon)$ fraction of its neighbors. By taking $O(\log(KT))$ clones of the right set, one obtains $T$-surviving dynamic standard matching (with no sharing of right nodes). We construct explicit bipartite graphs admitting $T$-surviving dynamic matching up to $K$ with small left degree $D$ and small right set $R$, and similarly for $\epsilon$-rich matching. Specifically, $D$ and $|R|/K$ are polynomial in $\log N$ and $\log T$, and for $\epsilon$-rich the dependency is quasipolynomial. Previous constructions, both non-explicit and explicit, did not require the $T$-surviving restriction, but had only slow matching algorithms running in time exponential in $K \log N$.

We give two applications. The first one is about non-blocking networks, i.e., directed graphs in which, roughly speaking, designated groups of nodes need to be connected via vertex-disjoint paths. A wide-sense non-blocking (wsnb) generalized connector is a graph that contains $N$ nodes called inputs, and other $N$ nodes called outputs. The objective is, for any sequence of pairs (input, output) in which an output can appear at most once, to connect the input and the output in every pair via paths that can only intersect in an initial segment. Moreover, the set of pairs is dynamic and the paths have to be found on-the-fly without knowing future arrivals and departures and without modifying existing paths. The depth is the length of the longest path that connects an input and an output. In a $T$-surviving network, a path can survive at most $T$ requests. For every constant $t$ and $T \leq 2^{N^{1/(t+2)}}$, we construct an explicit $T$-surviving wsnb generalized connector of depth $t$ with $N^{1+1/t} \cdot (\log(NT))^{O(t)}$ edges (value which almost matches a known lower bound) and which has a path finding algorithm that finds a connecting path in $t \cdot \text{poly}(\log N, \log T)$ time. This partially solves an open question in [FFP88], which has a network of similar size but with a very slow path-finding algorithm (on the other hand, it does not require the $T$-surviving restriction).

The second application concerns randomized storage schemes for representing a set $S$, so that a membership query “Is $x \in S$?” can be answered by reading a single bit. All the previous one-probe storage schemes from the literature were for static sets. We construct a one-probe

∗National Research University Higher School of Economics, Faculty of Computer Science, Moscow, Russia
†Department of Computer and Information Sciences, Towson University, Baltimore, MD. http://orion.towson.edu/~mzimand/; Partially supported by NSF grant CCF 1811729.
storage scheme for dynamic sets. We obtain a storage scheme which has smaller size than all previous explicit schemes from the literature, in spite of the fact that the previous schemes have the limitation of only handling static sets.

Contents

1 Introduction 3
   1.1 Dynamic and almost dynamic matching: formal definitions ................. 8

2 The slow dynamic matching algorithm of Feldman, Friedman, and Pippenger 9

3 From offline to fast online/almost dynamic matching 11
   3.1 From offline to fast online matching ........................................ 12
   3.2 From offline to fast almost dynamic matching ............................. 13
   3.3 Explicit graphs with fast online/almost dynamic matching .............. 14

4 Matching most neighbors 16
   4.1 From rich offline to rich online/almost dynamic matching ............. 16
   4.2 From $(\varepsilon, r)$-rich matching to $(\varepsilon, 1)$-rich matching ........ 17
   4.3 Explicit graphs with fast $\varepsilon$-rich almost dynamic matching .......... 19
      4.3.1 Proof of Theorem 20 ...................................................... 20

5 Application 1: Non-blocking connectors 21

6 Application 2: One-probe storage scheme for dynamic sets 23
1 Introduction

A bipartite graph has matching up to $K$ elements if for every set of $K$ left nodes, we can assign to each element in the set one of its neighbors so that the assigned neighbors are pairwise different. By Hall’s theorem, the existence of a matching is related to the expansion properties of the graph: There is a matching up to $K$ elements if and only if every subset $S$ of left nodes of size at most $K$ has at least $|S|$ right neighbors.\footnote{Unless specified otherwise, in this paper whenever we use the term graph, we mean a bipartite graph. As usual, $L$ and $R$ denote the left and, respectively, the right set of the bipartition. $\mathcal{N}(x)$ is the set of neighbors of the node $x$ and for a set of nodes $S$, $\mathcal{N}(S) = \bigcup_{x \in S} \mathcal{N}(x)$.} The following graphs have matching up to 2 elements.

In this paper we are interested in graphs that have stronger types of matching. Henceforth, we use the term offline matching in lieu of matching to distinguish the above concept from the stronger variants. A graph has online matching up to $K$ elements, if there exists a strategy that receives $K$ left nodes one by one, and assigns to the current one a right neighbor before the next node arrives. The left graph above has offline matching up to 2 elements, but it does not have online matching up to 2 elements. The right graph has the latter type, but it does not have an even stronger type of matching, called dynamic matching, in which left nodes not only arrive but can also depart releasing their assigned matches.

Example. Dynamic matching up to $K$ elements can be imagined as a game in which an Adversary adds and retracts matching requests so that no more than $K$ requests are active at any moment, and a Matcher has to satisfy the requests on the fly or else loses the game. The Adversary wins the game on the right graph with the following sequence of additions and retractions of requests. He first adds the middle left element, and the Matcher has to assign to it the top right neighbor (otherwise the Adversary wins at the next step by adding the bottom left node). Adversary next adds the top left node which can only be matched with the right bottom node, and at next step he retracts the left middle node and adds the bottom left node. At each moment at most 2 left nodes have active matching requests and we conclude that the right graph does not have dynamic matching up to 2 elements. A formal definition of dynamic matching is given in Section 1.1.\footnote{Dynamic matching is also used in the literature for a different type of algorithms, which are required to maintain a good (in various senses) matching in a dynamic graph in which edges are inserted and deleted (for example, see \cite{HKPS20}). Unlike these algorithms, in this paper a matching assignment cannot be revoked unless the left node departs.}

Feldman, Friedman, and Pippenger \cite[Proposition 1]{FFP88} have given a sufficient condition for dynamic matching (see Section 2 for details). Their result implies that any graph $G$ with offline matching up to $K$ elements has dynamic matching with overlap 4 up to $K$ elements. “Matching with overlap $r$” is a relaxation of matching in which a right node can be assigned to at most $r$ left nodes. When the overlap $r$ is 1, we get the standard definition of matching. It follows that $G$ can be converted into a graph $G'$ that has dynamic matching up to $K$ by making 4 clones of its right set. This means that $G'$ has the same left set as $G$ and 4 disjoint copies of the right set of $G$ connected to the left set in the same way as in $G$. Unfortunately, the dynamic matching procedures for the new graph is very slow, running in time exponential in $K \log N$, where $N$ is the number of left nodes. An important open question from \cite{FFP88} is to find a variant of their result giving graphs
with efficient matching algorithms. More generally it is of interest to obtain Hall-type theorems for fast online and dynamic matching, i.e., to show that graphs with certain expansion properties admit fast online or fast dynamic matching. A key part of the algorithm in [FFP88] is a co-NP hard task (see the discussion in Section 2) and, therefore, a different technique is needed to obtain fast matching.

In this paper we present a partial solution: We show that any graph with offline matching has fast online matching and also fast almost dynamic matching with logarithmic overlap (see Theorem 1). It follows that a graph with offline matching can be transformed by taking a few clones of the right set into a graph with fast online matching, or into a graph with fast almost dynamic matching. Let us define the two concepts that appear in the above statements.

- **Almost dynamic matching** is defined in terms of a parameter $T$, called the survival parameter, and it requires that left nodes can make and retract matching requests (like in standard dynamic matching), but any matching assignment has to be released or refreshed (i.e., the left node is rematched possibly with a different neighbor) after at most $T + 1$ matching requests. In other words, every assignment survives at most $T$ matching requests.

- **Fast matching** means that when the original graph is given as an oracle, the matching for the current request can be computed in time $\text{poly}(\log N, D)$ for online matching, and $\text{poly}(\log N, \log T, D)$ for $T$-surviving dynamic matching ($D$ is the left degree of the original graph; we assume for simplicity that all left nodes have the same degree).

Note that if a graph $G$ has almost dynamic matching up to $K$ elements and survival parameter $K$, then it also has online matching up to $K$ elements (because if there are no retractions, then there cannot be more than $K$ requests and therefore the $K$-survival property is automatically guaranteed).

**Theorem 1** (Hall-type theorem for fast almost dynamic matching). If $G$ has offline matching up to $K$, then it has fast almost dynamic matching up to $K$, with survival parameter $T$, and overlap $O(\log(KT))$. Therefore, the graph obtained from $G$ by taking $O(\log(KT))$ clones of its right set has fast almost dynamic matching up to $K$, with survival parameter $T$.

The problem of constructing graphs with fast dynamic matching (other than the trivial solution of taking $K/D$ clones), without any surviving restriction, remains open. Thus, currently we have the following picture.

| type of matching up to $K$ | overlap | runtime for current match | reference |
|---------------------------|---------|---------------------------|-----------|
| offline                   | 1       | N/A                       | Hall’s Theorem |
| slow dynamic              | 4       | $2^{O(\log N)}$          | [FFP88, Prop. 1] |
| fast online               | $O(\log K)$ | poly($\log N, D$)     | Theorem 1, with $T = K$ |
| fast $T$-surviving dynamic | $O(\log(KT))$ | poly($\log N, \log T, D$) | Theorem 1 |
| fast dynamic              | ?       | ?                         | ?         |

Table 1: The matching properties of a bipartite graph with $N$ left nodes and left degree $D$, satisfying Hall’s condition: any set $S$ of left nodes of size up to $K$, has $|\mathcal{N}(S)| \geq |S|$. A graph that has matching with overlap $r$ can be transformed into a graph that has matching with overlap 1 (i.e., standard matching), by taking $r$ clones of the right set.

In applications, one typically needs explicit graphs, with the size of the right set close to $K$ and with small $D$. Using a technique from [BMVZ18] based on dispersers, the following graph is obtained.

**Theorem 2** (Explicit graph with fast almost dynamic matching). There exist a family of graphs $\{G_{N,K,T}\}$ for all positive integers $N$, $K$ and $T$, and a matching algorithm such that:
• (small $|R|$ and $D$). In $G_{N,K,T}$, the left set has size $N$, the right set has size $K \cdot \text{poly} (\log N) \cdot \log T$, and the left degree $D = \text{poly}(\log N) \cdot \log T$.

• (explicitness). Given $N$, $K$, $T$, $x \leq N$, and $i \leq D$, one can compute in time $\text{poly}(\log N, \log D)$ the $i$-th neighbor of $x$ in $G_{N,K,T}$.

• (fast matching). Given a left node, the matching algorithm selects for every request the matching right neighbor in time $\text{poly}(\log N,D)$, provided the matching requests satisfy the condition of almost dynamic matching up to $K$ (the bound on active requests) and $T$ (the survival parameter).

One of our applications requires a stronger type of matching in which a left node making a request must be matched not to just one, but to most of its neighbors. More precisely, a left node has to be matched with $(1 - \epsilon)D$ of its neighbors. This is called $\epsilon$-rich matching. We observe that such graphs can be constructed from lossless expanders and we obtain the following graph.

Theorem 3 (Explicit graph with rich fast almost dynamic matching). There exist a family of graphs $\{G_{N,K,T,\epsilon}\}$ for all positive integers $N$, $K$ and $T$, and every $\epsilon > 0$, a matching algorithm, and a function $\Delta := \Delta(N,K,\epsilon) = O(\log \log N + (\log \log K + \log 1/\epsilon) \cdot \log \log K)$ such that

• (small $|R|$ and $D$). In $G_{N,K,T,\epsilon}$, the left set has size $N$, the right set has size $K \cdot 2^\Delta \cdot \log^2 T$, and the left degree $D = 2^\Delta \cdot \log T$.

• (explicitness). Given $N$, $K$, $T$, $x \leq N$, and $i \leq D$, one can compute in time $\text{poly}(\log N, \log D)$ the $i$-th neighbor of $x$ in $G_{N,K,T,\epsilon}$.

• (fast matching). Given a left node, the matching algorithm selects for every request $(1 - \epsilon)D$ matching right neighbors in time $\text{poly}(\log N,D)$, provided the matching requests satisfy the condition of almost dynamic matching up to $K$ (the bound on active requests) and $T$ (the survival parameter).

We next present two applications of fast almost dynamic matching.

Application 1: non-blocking networks

The first application is in the area of non-blocking network design, which is not surprising because this has been the motivation for introducing expanders fifty years ago [BP73, Mar73]. A network is a directed acyclic graph which, broadly speaking, allows good connections between designated nodes or groups of nodes. Our application is about a type of networks called connectors.

Definitions. An $N$-network is a graph in which $N$ nodes are called input nodes, and $N$ different nodes are called output nodes. The number of edges is called the size of the network. The number of edges on the longest path from input nodes to output nodes is called the depth of the network. A request is a pair consisting of an input node and an output node. A set of requests is valid if every input node and output node appears at most once. Note that such a set can contain at most $N$ requests. A rearrangeable network is a network such that for each valid set of requests, there exists a set of pairwise vertex disjoint paths that connect each input to the corresponding output in the set of requests. In many applications, the requests are not given at once, but arrive one by one and can later be retracted. A wide-sense non-blocking (wsnb) $N$-connector is an $N$-network for which there exists a strategy that can satisfy requests dynamically. This means that when a valid request arrives, the strategy has to immediately find a path that is vertex-disjoint from all existing paths. This path can not be changed until the request is retracted and the path is deleted. The aim is
Figure 1: A ssnb connector with 3 inputs (the nodes in the first column) and 3 outputs (the nodes in the last column), depth 3, and size 24.

to generate a network with a minimal number of edges and for which the path finding algorithm is as simple as possible. There exist also strict-sense non-blocking (ssnb) connectors which have a stronger property: Given any vertex-disjoint paths from inputs to outputs, and a valid request, there exists a path connecting the vertices in the request, which is vertex-disjoint with the given paths.

In a generalized connector, we consider inputs that broadcast to several outputs. More precisely, we consider now lists of requests in which each output appears at most once, but inputs can appear multiple times (thus, we want to connect an input to several outputs, with connections defining a one-to-many relation). Two paths are compatible if they only share an initial segment. This initial segment might be empty. Thus, after they separate two compatible paths do not have any common vertex. The requirement is to connect the nodes in the request via pairwise compatible paths. A wide-sense non-blocking generalized \( N \)-connector is defined in a similar way to wsnb connectors, and we simply refer to them as wsnb generalized \( N \)-connectors.

In [PY82] it was shown that rearrangeable \( N \)-networks of depth \( t \) must have at least \( tN^{1+1/t} \) edges. Obviously, this lower bound also holds for the more demanding wsnb \( N \)-connectors and ssnb \( N \)-connectors. In [ALM96] an explicit ssnb \( N \)-connector is constructed with almost optimal parameters and a very simple path finding algorithm. The number of edges is \( O(N \log N) \), all paths have length \( 2 \log N + 1 \) and all nodes have constant degree. Moreover, the path finding algorithm is local in the following strong sense: the path can be found by letting the nodes run a constant amount of computation. Only the nodes of the generated path and their neighbors participate in the computation. Thus to construct the path, the total amount of communicated bits is \( O(\log N) \) (to retract the path, more nodes might be involved and the complexity can be larger; the paper does not discuss this).

Our focus is on generalized connectors with constant depth \( t \). Some constant-depth ssnb \( N \)-connectors are known, with path-finding algorithms running in time poly(\( \log N \)), but their size is not optimal (see [Hwa98, Chapter 2]). For instance, the Clos network of depth \( t \) (for every odd \( t \)) has size \( \Theta(N^{1+2/(t+1)}) \) (the constant in \( \Theta() \) depends on \( t \)). Feldman, Friedman, and Pippenger [FFP88] give nonexplicit constructions of networks of size \( O(N^{1+1/t} \log^{1-1/t} N) \) with a slow path finding algorithm running in time exponential in \( N \). They ask, as an important question left open by their work, whether a wsnb generalized connector exists with small size and an effi-

\[\text{Note that the compatible paths define a collection of disjoint trees rooted at the input nodes making a request and with output nodes as leaves; each tree connects the input node in the root to the output nodes that it has requested.}\]
cient path finding algorithm. In [WZ99, Th. 5.4] an explicit construction was given achieving size $N^{1+1/t} \exp(\log \log N)^{O(1)}$, but the path finding algorithm is the same slow one from [FFP88].

Similarly to $T$-surviving dynamic matching, we define a weaker type of wsnb-connector (and also the generalized version), in which we require that any request is retracted or refreshed (i.e., the input and the output are connected by a new path) after at most $T + 1$ other requests. We obtain, for every $t$, and assuming the survival parameter $T$ is bounded by $2^{N^{1/(t+2)}}$, an explicit $T$-surviving wsnb generalized connector of depth $t$ with $N^{1+1/t} \cdot (\log(NT))^{O(t)}$ edges, with an efficient path-finding algorithm: it satisfies each (input, output) routing request in time $t \cdot \poly(\log N, \log T)$. Also note that for constant $t$ and $T = \poly(N)$, the size is smaller than the network in [WZ99, Th. 5.4]. This is a partial solution to the question of Feldman, Friedman, and Pippenger, however it does not completely solve the problem because of the $T$-surviving restriction.

**Application 2: one-probe storage schemes for dynamic sets**

This application is about one-probe storage schemes for the dictionary data structure. The goal is to store a subset $S$ of a large set $U$ (the “universe”). Let $N$ denote the size of $U$, and $K$ denote the size of $S$. A simple storage scheme is to keep in a table a sorted list of the $K$ elements of $S$. The table is stored on $K \cdot \log N$ bits, and, for $x \in U$, one can determine if $x$ is in $S$ or not, by reading $(\lceil \log K \rceil + 1) \cdot \log N$ bits from the table. An alternative is to have a table of $N$ bits indexed by the elements in $U$ and to set a bit to 1 if and only if its index is 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 (taking into account that typically $N \gg K$).

A one-probe storage scheme is a data structure like the second table above, that answers any membership query “Is $x \in S$?” by reading a single bit. It is a fundamental data structure that has been studied extensively.

Buhrman, Miltersen, Radhakrishnan, and Venkatesh [BMRV00] have used lossless expanders to build randomized one-probe storage schemes with short tables, in which every membership query has to be answered correctly with probability $1 - \epsilon$.\(^4\) They have a scheme based on a non-explicit expander that uses a table of size $O(K \cdot \log N \cdot (1/\epsilon)^2)$ bits. Note that $K \log N$ is essentially the information-theoretical lower bound for storing the set even without the one-probe restriction. They also have an explicit construction achieving table size $O(K^2 \cdot \log N \cdot (1/\epsilon^2))$. Using improved explicit lossless expanders, there are explicit one-probe storage schemes with smaller tables. Tsai-Shma [Ta-02] obtains table size $K \cdot 2^{O((\log \log N)^3)}$, and Guruswami, Umans, and Vadhan [GUV09, Theorem 7.4] obtain table size $K \cdot \poly((\log N)/\epsilon) \cdot \exp(\sqrt{\log((\log N)/\epsilon)} \log K)$. In all these schemes, making a membership query (i.e., finding the bit in the table that is probed) is fast, taking time $\poly(\log N, \log(1/\epsilon))$.

These one-probe schemes work for static sets, in the sense that any updating of $S$ requires the recomputation from scratch of the entire data structure, an operation that in the above schemes takes time $\poly(K, \log N, \log(1/\epsilon))$. Using graphs with $\epsilon$-rich almost dynamic matching we obtain one-probe storage schemes for dynamic sets. In our construction, the insertion or the deletion of an element $x$ can be done in time $\poly(D, \log N, \log(1/\epsilon))$ (where $D$ is the left degree of the graph), which, for typical regime of parameters, is much faster than reconstructing the whole table. A membership query is done by reading a single bit and has runtime $\poly(\log N, \log(1/\epsilon))$, the same as

---

\(^4\)Randomized one-probe storage schemes are different from Bloom filters which store an approximation of the set. More precisely, a Bloom filter stores a superset $S'$ of the intended set $S$. Thus for every $x$ in $S' - S$ (the false positives) the error probability of the query “Is $x$ in $S$?” is 1, and for $x$ in $S$ or in $U - S'$ 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 $S' - S$ is $\epsilon$).
in the previous schemes. Furthermore, during an insertion or deletion only the bits assigned to one or two elements need to be changed in the data structure, and consequently membership queries for other elements can be answered without locking the data structure during the update. The size of the data structure depends on the parameter $K$, which is the maximum size the dynamic set can have at any moment during its history. Our construction in which we plug the graph from Theorem 3 yields an explicit one-probe storage scheme for dynamic sets with a table of size $K \cdot \text{poly}(\log N) \cdot 2^{O((\log \log K + \log(1/\epsilon))\cdot \log \log K)}$. This improves all the previous explicit schemes mentioned above, in spite of the fact that they have the limitation of only handling static sets.

We remark another important feature of our construction. All the previous explicit one-probe storage schemes required a lossless expander with a special “list-decoding” property (see [GUV09, Th.7.2]), while our approach works with any lossless expander. Consequently, any future improvement in explicit lossless expanders will directly induce better one-probe storage schemes.

1.1 Dynamic and almost dynamic matching: formal definitions

We formally define dynamic matching via a game. The game is characterized by a bipartite graph $G = (L, R, E \subseteq L \times R)$ and two parameters $K \in \mathbb{N}$ (the match size) and $T \in \mathbb{N} \cup \{\infty\}$ (the survival parameter). There are 2 players, called Adversary and Matcher. The Adversary has $K$ blue tokens that he can place/remove on the left nodes, and the Matcher has $K$ red tokens that she can place/remove on right nodes. Initially there is no token placed on the nodes of the graph.

**Interpretation:** Making a matching request for a left node $u$ is done by placing a blue token on $u$. If the Matcher puts a red token on $v \in N(u)$, then $u$ is matched with $v$ (we also say the matching assignment $(u, v)$ is made). To retract a node $u$ (which is matched to some $v$), the tokens on $u$ and $v$ are removed, and $v$ becomes free to be matched to other nodes. At any moment there are at most $K$ matches. The game has survival parameter $T$, if every matching assignment $(u, v)$ has to be revoked after at most $T + 1$ steps (so a blue token can sit on a left node for at most $T$ moves of the game).

We continue with a detailed description of the game. A step of the game consists of an Adversary move followed by the Matcher response if she can find one. If the Matcher does not find a response at some step, then she loses the game. If the Matcher finds a response at every step, then she wins the game (in this case, the game may have an infinite number of steps).

**Adversary move:** He retracts a number (possibly zero) of blue tokens from some left nodes and adds a blue token on a single left node which does not have a token. If $u \in L$ is one of the nodes from which a token was retracted, we say that $u$ is retracted (from the list of requests), and if $u$ is the node on which a token was added, we say that $u$ is added (to the list of requests). For conciseness we will omit “(from/to the list of requests).”

**Matcher response:** She retracts the red tokens on the right nodes adjacent to the left nodes that have been retracted, and if $u$ is the left node that is added, she picks some $v \in N(u)$ that has no token and puts a red token on $v$. We say that $u$ has been matched with $v$, or that the matching assignment $(u, v)$ has been made.

**Matching requirement:** At each step in the game the edges having tokens on both endpoints define a 1-1 relation (i.e., if we restrict to nodes having tokens, the degree of every left and every right node is at most 1). We will also consider more general types of matching which induce many-to-many relations, and the game will be modified accordingly.

**Surviving requirement:** The game is $T$-surviving if every blue token is retracted after at most $T + 1$ moves of the game. Note that a blue token can be retracted and then added on the same node and at the same move, but the Matcher may find a different assignment.
Definition 4. 1. If the Matcher wins the above game, then the graph $G$ has $T$-surviving dynamic matching up to $K$ elements. Typically $T$ is a function of $N$ (the number of left nodes), and we refer to these games as dynamic matching with bounded surviving or, more concisely, as almost dynamic matching.

2. If $T = \infty$ (i.e., there is no restriction on how many steps a blue token can sit on a node) and the Matcher wins the game, then $G$ has dynamic matching up to $K$ elements.

3. If no retractions are allowed, and the Matcher wins the game, then $G$ has online matching up to $K$ elements. (Note that if there are no retractions then the game lasts at most $K$ steps and thus automatically $T \leq K$. For these games, there will be no surviving condition. Also, graphs that have $K$-surviving dynamic matching up to $K$ elements, automatically have online matching up to $K$ elements.)

Runtime: We are interested in fast algorithms for the Matcher response, and we analyze the runtime for doing one matching assignment.

2 The slow dynamic matching algorithm of Feldman, Friedman, and Pippenger

For the sake of completeness, we present the statement and the proof of the construction from [FFP88] of a graph admitting dynamic matching. There is no $T$-surviving restriction, however, the matching algorithm is slow. This section can be skipped, without affecting the understanding of the rest of the paper.

Definition 5. A graph has $e$-expansion up to $K$ elements if every set $S$ containing at most $K$ left nodes has at least $e|S|$ neighbors.

Hall’s theorem states that a graph has 1-expansion up to $K$ elements if and only if it has offline matching up to $K$ elements.

Theorem 6 ([FFP88], Proposition 1). Each graph with 2-expansion up to $2K$ has dynamic matching up to $K$.

Note that if a graph $G$ has 1-expansion up to $K$, by taking 4 clones of the right set we get a graph $G'$ with 2-expansion up to $2K$. Therefore, $G'$ has dynamic matching up to $K$. It follows that $G$ has dynamic matching with overlap 4 up to $K$.

The proof is related to a known proof of the following lemma, which is the special case of Hall’s theorem with $K$ equal to the left-hand size. The result is typically stated for offline matching, but it is valid for the stronger online matching.

Lemma 7. Let $N$ be the size of the left set of a graph. If the graph has 1-expansion up to $N$, then it has online matching up to $N$.

For a set of nodes $S$, let $|S|$ denote the size of $S$, $\mathcal{N}(S)$ denote the set of neighbors of $S$ and let $\mathcal{N}(x) = \mathcal{N}({x})$. We say that a set $A$ of left nodes is critical if it has at most $|A|$ neighbors. We first prove a technical lemma that we will also need later.

Technical Lemma. If $A$ and $B$ are critical sets of size at most $K$ in a graph with 1-expansion up to $K$, then $A \cup B$ is also critical.

\footnote{The notation is different and for simplicity, we consider the special case with $r = s = 1$ (in [FFP88] notation).}
Proof. We need an upper bound for

\[ |\mathcal{N}(A \cup B)| = |\mathcal{N}(A)| + |\mathcal{N}(B)| - |\mathcal{N}(A) \cap \mathcal{N}(B)| \]

The first 2 terms are at most \(|A|\) and \(|B|\), and the 3rd satisfies

\[ |\mathcal{N}(A) \cap \mathcal{N}(B)| \geq |\mathcal{N}(A \cap B)| \geq |A \cap B|, \]

where the last inequality follows from 1-expansion up to \(K\). Combined this implies

\[ |\mathcal{N}(A \cup B)| \leq |A| + |B| - |A \cap B| = |A \cup B|. \]

Proof of Lemma 7. We process the nodes in the left set one-by-one in the online manner (for an arbitrary ordering of \(L\)).

When the left node \(x\) arrives, we search for a neighbor \(y\) such that after deleting both \(x\) and \(y\) from the graph, the remaining graph has 1-expansion up to the size of the left set. We say that such a pair \((x, y)\) is safe. If a \(y\) is found so that \((x, y)\) is safe, we delete \(x\) and \(y\), and continue with the next node.

To prove that this works, we need to show that such a neighbor \(y\) always exists, provided the current graph has 1-expansion. Let \(G\) be the current graph. We show the contrapositive. Assume that for some \(x\) in \(G\) and each neighbor \(y\), after removing \(x\) and \(y\), there exists a set \(S_y\) of left nodes with less than \(|S_y|\) neighbors. We need to show that \(G\) does not have 1-expansion. If \(y\) is not a neighbor of \(S_y\) in \(G\), then we are finished, since \(S_y\) is a counter example to the 1-expansion of \(G\).

So we may assume that for each neighbor \(y\) of \(x\), we have \(y \in \mathcal{N}(S_y)\).

By the previous lemma, the union

\[ U = \bigcup_{y \in \mathcal{N}(x)} S_y \]  

(1)

is a critical set which means \(|\mathcal{N}(U)| \leq |U|\). By the definition of \(S_y\), the set \(U\) does not contain \(x\). So, \(|U \cup \{x\}| = |U| + 1\). On the other hand, each neighbor \(y\) of \(x\) is in \(\mathcal{N}(S_y)\) and hence in \(\mathcal{N}(U)\), which implies that \(\mathcal{N}(U \cup \{x\}) = \mathcal{N}(U)\). We conclude that the set \(U \cup \{x\}\) has size exactly \(|U| + 1\) and has at most \(|U|\) neighbors, which contradicts that the graph does not have 1-expansion.

The claim is proven and we conclude that the procedure always works and creates a matching. \(\square\)

We now want to prove a more general version for graphs that have 1-expansion up to some \(K\) (instead of up to \(N\) in Lemma 7). Can we use the above procedure to get online matching up to \(K\)? The natural idea would be to do the same thing: select a neighbor \(y\) so that after deleting \(x\) and \(y\), the remaining graph has 1-expansion up to \(K\). But the set \(U\) may have size larger than \(K\). For such large sets, 1-expansion is not covered by the assumption. To fix this problem, we assume that each set \(S\) with size in the interval \([K, 2K]\) has at least \(|S| + K\) neighbors. We will see that this implies that all critical sets in the above procedure have size less than \(K\). Since \(U\) is critical by the technical lemma, we conclude that \(|U| < K\) and the proof is fixed.

Lemma 8. If a graph has 1-expansion up to \(K\) and if every set \(S\) of left nodes with size \(|S| \in [K, 2K]\) has at least \(|S| + K\) neighbors, then it has online matching up to \(K\).

Proof. For any list of \(K\) left nodes, we run an iterative procedure similar to the one in Lemma 7: For each node \(x\) in the list, we select a neighbor \(y\) so that \((x, y)\) is safe, which now means that the graph obtained after removing \(x\) and \(y\) has 1-expansion up to \(K\). When such a \(y\) is found (we shall show that this happens for every \(x\) in the list), we match \(x\) with \(y\), we remove \(x\) and \(y\), and
we move to the next node in the list. Since we only remove $x$ and $y$ if $(x, y)$ is safe, the successive graphs in the iterative procedure preserve the 1-expansion up to $K$.

The original size of a set of left nodes, is its size in the initial graph (i.e., the graph before any node removal), and the current size refers to the size in the current graph. As before a set $S$ of left nodes is critical if $|S| \geq |\mathcal{N}(S)|$ (with sizes in the current graph; “critical” always refers to the current graph).

During the iterative procedure we only select neighbors $y$ in graphs in which at most $K - 1$ right nodes of the initial graph have been removed. In these graphs, the hypothesis implies that every set $S$ with original size in the interval $[K, 2K]$ has in the current graph at least $|S| + 1$ neighbors and therefore is not critical. So the key observation is that each critical set has original size either less than $K$ or greater than $2K$.

The existence of a safe $(x, y)$ at each step follows similarly to the argument in Lemma 7. Now, in the proof by contradiction, we define for each neighbor $y$ of $x$, a set of left nodes $S_y$ of current size at most $K$ that is critical. Since the sets $S_y$ are critical, they must have original size at most $K$ (the alternative would be to have original size larger than $2K$ but in that case the current size would be larger than $K$). Thus the union of two such sets has original size at most $2K$, but by the Technical Lemma the union is critical and so it has original size less than $K$. Inductively, we infer by this argument that any union of some sets $S_y$ is critical and has original size less than $K$. The set $U$ defined exactly as in Eq. (1) is such a union, and, therefore

$$(\text{the current size of } U) \leq (\text{original size of } U) < K.$$ 

As in Lemma 7, $U \cup \{x\}$ has current size $|U| + 1$ and has at most $|U|$ neighbors. Thus, $U$ contradicts that the current graph has 1-expansion up to $K$, which implies the existence of the desired safe $(x, y)$. 

Proof of Theorem 6. Note that in a graph with 2-expansion up to size $2K$, every set $S$ with $|S| \in [K, 2K]$ has at least $|S| + K$ neighbors. Thus, Lemma 7 implies online matching.

But we want the stronger dynamic matching. First, note that in the above proof, the procedure to find a neighbor $y$ so that $(x, y)$ is safe, does not depend on future arrivals.

Hence, to handle dynamic matching, we need to prove that the above argument remains valid in case a left node $x$ and some neighbor $y$ are restored. In particular, we need to prove that the 1-expansion up to $K$ property is preserved. Indeed, for all sets $X$ that do not contain $x$, the number of neighbors can only increase, and for sets $X$ that do contain $x$, we know that the extra neighbor $y$ is now present.

In conclusion, the graph always has 1-expansion up to $K$, and hence, the online algorithm will always succeed to select a neighbor $y$. The theorem is proven.

3 From offline to fast online/almost dynamic matching

Theorem 6 shows that a graph that has 2-expansion up to $2K$ has dynamic matching up to $K$. Such a graph can be obtained from a graph $G$ that has offline matching up to $K$ (by Hall’s Theorem, this is a graph with 1-expansion up to $K$) by taking 4 clones of the right set. The key part of the algorithm in Theorem 6 is finding a safe edge $(x, y)$. The only known way to do this is to check one-by-one for all $y \in \mathcal{N}(x)$ if $(x, y)$ is safe. Such a checking is co-NP hard [BKP+81] and, therefore, a fast matching procedure seems to require a different idea.
We show in Section 3.1 that if we convert a graph \( G \) as above by taking \( O(\log K) \) clones of the right set, we obtain a graph with fast online matching up to \( K \). Then in Section 3.2, we show that if we take \( O(\log K + \log T) \) clones, we obtain a graph with fast \( T \)-surviving dynamic matching up to \( K \). In Section 3.3, we show that in fact we can obtain graphs with fast online/almost dynamic matching by gluing disperser graphs with some appropriate parameters. Disperser graphs have a weaker type of offline matching in which only half of the matching requests are guaranteed to be satisfied. However this is enough for our technique, and the utilization of disperser graphs yields the best explicit graphs with fast online/ almost dynamic matching, as far as we are aware.

### 3.1 From offline to fast online matching

First a relaxed notion of matching is defined in which the right nodes can be shared among a few left nodes. Let \( r \) be a positive integer. We say that a graph has matching up to \( K \) elements with overlap \( r \) if for each set of \( K \) left elements, there exists a subgraph in which all left elements have degree 1 and all right elements have degree at most \( r \). This definition applies to offline, online and dynamic matching. In Definition 4 with games, we modify the requirements and ask that if we restrict to nodes with tokens, every left node has degree 1, and every right node has degree at most \( r \). Note that if \( r = 1 \), we re-obtain the original definitions of matching. The following lemma is based on an idea from [MRS11].

**Lemma 9.** Let \( G \) be a graph that has offline matching up to \( K \geq 1 \) elements. Then:

(a) \( G \) has online matching up to \( K \) elements with overlap \( \lfloor \log K \rfloor + 1 \).

(b) The graph \( G' \) obtained from \( G \) by taking \( \lfloor \log K \rfloor + 1 \) clones of the right set has online matching up to \( K \).

(c) If \( G \) is explicit, then the runtime of doing in \( G \) or in \( G' \) one matching assignment is \( D \cdot \text{poly}(\log N) \).

**Proof.** Statements (a) and (b) are clearly equivalent. Indeed, if in \( G \) the left node \( u \) is matched with the right node \( v \) with the given overlap, then in \( G' \) we assign to \( u \) an available clone of \( v \). And vice-versa, if \( G' \) has matching, then, by merging the clones, we obtain in \( G \) matching with the given overlap.

Therefore it suffices to prove (a). To understand the idea of the proof, let us first consider the following greedy online algorithm. Each new left node is matched to an unmatched right neighbor, provided such a neighbor exists. Otherwise, it does nothing, and we say that the left node is rejected.

Note that if \( K \) left nodes arrive, at most \( \lfloor K/2 \rfloor \) of them are rejected. Indeed, we know that a matching exists and when a left node is rejected, the matched node was already taken by another node. Thus, the number of rejected nodes is always bounded by the number of matched nodes.

The strategy that achieves the result of the lemma, assigns each left node to the right neighbor with the smallest number of previously matched elements. If there is a tie, we break it by choosing the first node in some canonical ordering.

We prove correctness by induction on \( K \). Clearly, if \( K = 1 \), then the bound of the lemma holds trivially. Consider a list of \( K > 1 \) elements and consider the set of elements that are matched to a right neighbor that was already used before, let's call this set \( S \). These are the nodes that are rejected in the above online matching algorithm. By the above reasoning, \(|S| \) is at most \( \lfloor K/2 \rfloor \). Now consider a run of the algorithm with a modified list of requests containing only the nodes in \( S \) (i.e., leaving out the nodes that are not rejected in the above procedure). By the induction
hypothesis, the nodes in the modified list are matched with overlap 

$$\lceil \log(K/2) \rceil + 1.$$  

The run of the algorithm with the original list of matching requests will match the left-out nodes to free right neighbors, and match the other nodes according to the run above with the modified list. Hence, the total overlap is one more than the bound above.

To implement the algorithm, each right node maintains a counter that keeps the number of left nodes to which it has been matched. Such a counter needs to count up to $K$ and thus reading and incrementing it takes $O(\log K)$ steps. To find the match when a left node $x$ makes a request, the procedure needs to access the counters of the $D$ neighbors of $x$, compare them, and increment the one with smallest value. If the graph is explicit, this takes $D \cdot (\text{poly}(\log N) + O(\log K)) = D \cdot \text{poly}(\log N)$ steps (we took into account that $K \leq N$). The lemma is proven.

3.2 From offline to fast almost dynamic matching

We now extend the results from fast online matching (Lemma 9) to fast $T$-surviving dynamic matching, and establish Theorem 1.

Note that for $T \leq K$, $T$-surviving dynamic matching up to $K$ nodes is the same as online matching up to $T$ nodes, because there can never exist more than $T$ active matching requests simultaneously. We aim at proving results for values of $T$ that are much larger than $K$, and actually much larger than the number of left nodes.

If a graph has online matching up to $K$, then $\lceil T/K \rceil$ copies of this graph has $T$-surviving dynamic matching. Indeed, one can use each copy for $K$ subsequent additions. By the time the last copy is filled, all nodes from the first copy are already retracted. Unfortunately, for large $T/K$, this construction leads to large left degrees and a large right set. The following result has an exponentially better dependency on the ratio $T/K$.

**Theorem 10** (restatement of Theorem 1). Let $G$ be graph that has offline matching up to $K$. Then:

(a) $G$ has $T$-surviving dynamic matching up to $K$ nodes with overlap $O(\log(KT))$.

(b) The graph $G'$ obtained from $G$ by taking $O(\log(KT))$ clones of the right set has $T$-surviving dynamic matching up to $K$ nodes.

(c) If $G$ is explicit and it has $N$ left nodes and left degree $D$, then the runtime of doing one matching assignment in $G'$ is $D \cdot \text{poly}(\log N) \cdot \log T$.

**Proof.** Statements (a) and (b) are equivalent as we have remarked in Lemma 9. We prove (b) and the runtime claimed in (c).

If we take 2 copies of a graph with offline matching up to $K$, we obtain a graph with offline matching up to $2K$. Hence, we may assume that the initial graph $G$ given in the hypothesis has offline matching up to $2K$. The extra factor 2 is absorbed in the $O(\cdot)$ notation of the theorem.

We construct a graph $G^*$ that has dynamic matching up to $K$ for all games with at most $T$ steps. More precisely, this means that the Matcher finds a response at steps $1, 2, \ldots, T$ in the game played on $G^*$ with $K$ blue tokens and $K$ red tokens. Then we merge 2 copies of $G^*$ and we are done, because the new graph has the desired $T$-surviving dynamic matching up to $K$ (since the Matcher uses the first copy for the first $T$ steps, the second copy for the next steps, then again the first copy for the next $T$ steps, and so on).

Consequently, it is enough to build $G^*$.  

13
For \( T \leq 2K \), we use Lemma 9 to obtain from the given \( G \), the graph \( G^* \) with online matching up to \( 2K \), which implies that \( G^* \) has dynamic matching up to \( 2K \) for games with \( 2K \) steps, which is even more than what we need.

Consider now the case \( T \geq 2K \). We can assume that \( T/(2K) \) is a power of two (if it is not, we prove the statement for the smallest value larger than \( T \) that has this property). So \( T = 2^i \cdot (2K) \).

By induction on \( i \) we show that if we merge \( 2i + c \) copies of \( G \), where \( c = 1 + \lceil \log(2K) \rceil \), we obtain \( G^* \) with the desired property.

The base case \( i = 0 \) has been handled above.

Assume the statement is proven up to \( T = 2^i \cdot (2K) \). To show the statement for \( i + 1 \), we need to explain how 2 extra copies can be used to obtain matching for sequences with \( T' = 2T \) steps.

We split the game in subsequent blocks that contain exactly \( 2K \) steps. Each block contains \( 2K \) additions (one in every step) and some retractions.

We now explain how the Matcher responds to the requests in the current block. She postpones releasing the nodes assigned to retractions till the end of the block and also skips handling re-additions (i.e., if a node makes several requests, she will match it only the first time). In this way, in each block, the Matcher handles additions by doing online matching. When processing the current block, the Matcher needs to find assignments to all the different nodes that are added and there are at most \( 2K \) such nodes (there may be less than \( 2K \) because multiple requests may come from the same left node). There are also at most \( K \) nodes with blue tokens which are already matched from the previous blocks and therefore some of the right nodes in the copies are assigned already. Each of the at most \( 2K \) different additions in the current block is matched in a greedy way on the 2 extra copies. If this greedy matching does not succeed, the node is rejected and passed to a recursive call of the algorithm that operates on the other copies. Let \( R \) be the number of nodes that are rejected and \( M \) be the number of nodes are matched (also counting the at most \( K \) nodes with blue tokens matched from the previous blocks). There are at most \( 2K \) new additions and at most \( K \) matches from previous blocks, and therefore \( R + M \leq 3K \). Recall that if they are clean (i.e., there are no assigned right nodes) each one of the extra 2 copies can do offline matching for \( 2K \) elements. Therefore, each time a node is rejected, we infer that two different nodes have already been matched, one in each copy. Thus, \( M \geq 2R \). The two inequalities imply \( R \leq 3K/3 = K \).

We conclude that while processing a block with \( 2K \) additions, at most \( K \) elements are rejected and passed to the recursive call. Summing over all the blocks, we infer that at most \( T'/2 = T \) are rejected on the two new copies and passed to the recursive call. By the inductive assumption these strings can be matched by the recursive algorithm on the other copies.

We now analyze the runtime. An attempt (that may be successful or rejected) to do the greedy matching of one left node in one copy of the graph has runtime \( D \cdot \text{poly}(\log N) \), because the procedure checks the \( D \) neighbors of the node to see if one is available. Since the total number of matching attempts is bounded by the number of copies which is \( O(\log(KT)) \), the total runtime to match one left node is \( D \cdot \text{poly}(\log N) \cdot \log T \) (taking into account that \( K \leq N \)). The theorem is proven.

3.3 Explicit graphs with fast online/almost dynamic matching

In [BMVZ18] it was noted that graphs with fast online matching can be obtained by gluing together dispersers, which are graphs with weaker expansion properties than those required for offline matching. This helps to obtain explicit constructions with better parameters. We present the construction from [BMVZ18].

**Definition 11.** A \((K, \delta)\)-disperser is a bipartite graph \( G = (L, R, E \subseteq L \times R) \) in which each set \( S \subseteq L \) containing at least \( K \) nodes has at least \((1 - \delta)|R|\) neighbors.
Let $N = 2^n$ and $K = 2^k$ with $K \leq N$. We use a family $G_k, \ldots, G_1$ of graphs where $G_i$ is an explicit $(2^{i-1},1/2)$-disperser with a left set $L(G_i)$ equal to $[N]$ (same set for all graphs), and pairwise disjoint right sets $R(G_i)$ of size at least $2^i$. By the definition of dispersers, $G_i$ has the following expander-type property: Each left set of size $2^{i-1}$ has at least $2^{i-1}$ right neighbors. We next build the bipartite graph $G_{N,K}$ by taking the union of $G_k, \ldots, G_1$.

This graph $G_{N,K}$ has online matching up to $K - 1$ elements using a greedy algorithm as follows: The Matcher assigns to a left node $x$ the first (in some canonical order) unused neighbor of $x$ among the right nodes in $R(G_i)$, and, if this fails, next she looks in $R(G_{k-1})$ and so forth till $R(G_1)$. We claim that if the Adversary adds up to $K - 1$ nodes, the Matcher will match in $G_k$ all of them except at most $K/2$. We prove the contrapositive. Assume there exist at least $K/2$ nodes that are not matched using $G_k$. This set of unmatched nodes is adjacent to $2^{k-1} = K/2$ right nodes in $G_k$, and these nodes must have been matched to other nodes. Thus at least $K/2$ nodes have been matched, and we have assumed that at least the same number are unmatched. Hence the Adversary has made at least $K/2 + K/2 = K$ additions, which contradicts that the number of additions is bounded by $K - 1$.

Thus, at most $K/2$ elements remain to be matched in the union $G_{k-1} \cup \ldots \cup G_0$, and continuing the argument in the same way, we conclude that eventually all elements are matched.

If the graphs $G_i$ are all explicit and have left degree bounded by $D$, then one matching is done in time $D \cdot \text{poly}(\log N)$.

It remains to build the graphs $G_i$, for $i = k, \ldots, 1$ with the above properties. They are obtained from the disperser graphs of Ta-Shma, Umans, and Zuckerman.

**Theorem 12.** [Ta-Shma, Umans, Zuckerman [TSUZ07]] For every $K, n$ and constant $\delta$, there exists an explicit $(K, \delta)$-disperser $G = (L = \{0,1\}^n, R = \{0,1\}^m, E \subseteq L \times R)$ in which every node in $L$ has degree $D = \text{poly}(n)$ and $|R| = \frac{\alpha K D}{n\delta}$, for some constant $\alpha$.

For $K = 2^{i-1}$ and $\delta = 1/2$, Theorem 12 gives an explicit $(2^{i-1},1/2)$-disperser with degree $D = \text{poly}(n)$, $L = \{0,1\}^n$ and $|R| = \frac{\alpha 2^{i-1} D}{n\delta}$. This is almost what we need, except that the right set is too small. Consider $t = \max\{1, \left\lceil \frac{2^{i-1}}{\alpha \delta} \right\rceil\}$ disjoint copies of this graph and identify left nodes of the resulting graphs (keeping their sets of right nodes disjoint). We get an explicit $(2^{i-1},1/2)$-disperser $G_i$ with left degree $\text{poly}(n) \cdot t = \text{poly}(n)$, $2^n$ left nodes and $2^{i-1}\text{poly}(n)$ right nodes. Now the right set has size at least $2^i$ as desired.

Plugging these graphs $G_i$ in the above construction of $G_{N,K}$, we obtain the following result.

**Theorem 13** (Explicit graph with fast online matching). Let $N$ and $K$ be powers of two and $K \leq N$. There exists an explicit family $\{G_{N,K}\}$, where each graph $G_{N,K}$ has left size $N$ and online matching up to $K$ nodes. Moreover, $G_{N,K}$ has left degree at most $\text{poly}(\log N)$ and right set of size at most $K \cdot \text{poly}(\log N)$.

The runtime for doing one matching assignment is $\text{poly}(\log N)$.

Next we transform the graph $G_{N,K}$ from Theorem 13 using the method from Section 3.2 that converts graphs with offline matching into graphs with $T$-surviving dynamic matching, and obtain the following result.

**Theorem 14** (Explicit graph with fast almost dynamic matching). Let $N$ and $K$ be powers of 2 and $K \leq N$ and let $T$ be an arbitrary integer. There exists an explicit family $\{G_{N,K,T}\}$ where each graph $G_{N,K,T}$ has left size $N$ and $T$-surviving dynamic matching up to $K$ nodes. Moreover, the left degree is at most $\log T \cdot \text{poly}(\log N)$ and the right size is at most $K \cdot \log T \cdot \text{poly}(\log N)$.

The runtime for doing one matching assignment is $\log T \cdot \text{poly}(\log N)$.
The above theorem is essentially a restatement of Theorem 2, where $N$ and $K$ are not required to be powers of 2. (Clearly if they are not powers of 2, we can do the construction for the smallest powers of 2 larger than them).

4 Matching most neighbors

Some applications require a stronger type of matching in which to a left node we assign not only one, but most of its neighbors. We give a general definition in which a right node can be assigned to $r$ left nodes. In Section 3.1, this was called matching with overlap $r$. Typically we are interested in the case $r = 1$, but sometimes it is convenient to first obtain slightly larger overlap, and then reduce it to 1. For convenience, we consider graphs that are left-regular.

**Definition 15.** A graph with left degree $D$ has $(\epsilon, r)$-rich offline matching up to $K$ elements if for every set $S$ of $K$ left nodes, there exists a subgraph with $S$ as left set, in which the degree of each left node is at least $(1 - \epsilon)D$ and the degree of each right node is at most $r$.

$(\epsilon, r)$-rich online matching is obtained as usual by requiring that the nodes in $S$ arrive one-by-one and that, for the current left node, the assignment of $(1 - \epsilon)D$ of its right neighbors defining the subgraph is made before the next arrival.

$(\epsilon, r)$-rich (almost) dynamic matching is also defined in the straightforward manner. Formally, we have a similar game as in Definition 4. The modifications are as follows: the Adversary has $K$ blue tokens (as before) and the Matcher has an unlimited number of red tokens; when a node $u$ is added by the Adversary, the Matcher has to respond by putting one red token on at least $(1 - \epsilon)D$ neighbors of $u$ (these neighbors are assigned to $u$); when a node $u$ is retracted by the Adversary, the Matcher removes one red token from the neighbors that were assigned to $u$; the matching requirement is that at every step in the game, if we restrict to nodes with tokens, the degree of every left node is at least $(1 - \epsilon)D$ and the degree of every right node is at most $r$; the $T$-surviving property is defined the same way, i.e., every blue token has to be retracted after at most $T$ moves of the game.

$\epsilon$-rich matching is short for $(\epsilon, 1)$-rich matching.

4.1 From rich offline to rich online/almost dynamic matching

We have the same situation as in the case of 1-to-1 matching that we have discussed in Section 3. The result of Feldman, Friedman, and Pippenger [FFP88, Proposition 1] shows that a graph with $\epsilon$-rich offline matching can be converted into one with $\epsilon$-rich dynamic matching. But again, the fastest algorithm we know for the graph, which is obtained via the [FFP88, Proposition 1] conversion, runs in time exponential in $K \log N$. We show conversions into graphs, for which the online and the almost dynamic matching algorithms have runtime $\text{poly}(\log N, D)$ for online matching, and $\text{poly}(\log N, \log T, D)$ for $T$-surviving dynamic matching.

**Proposition 16.** If a graph with left degree $D$ has $\epsilon$-rich offline matching up to $K$, then it has $(2\epsilon, 1 + \lfloor \log K \rfloor)$-rich online matching up to $K$ elements. Moreover, if the graph is explicit the runtime for doing one matching assignment is $D \cdot \text{poly}(\log N)$.

**Remark.** With the same proof, one obtains that $(\epsilon, r)$-rich offline matching up to $K$ implies $(2\epsilon, r(1 + \lfloor \log K \rfloor))$-online matching up to $K$. We will later see that by appending hash codes, it is easy to improve the parameter $r$ to 1, thus for the sake of simplicity we use the above result.
Proof. The proof has a similar structure as the conversion of standard offline to online matching. Let us first consider a Matcher that processes the left nodes added by the Adversary and assigns to the current node a set of \((1 - 2\varepsilon)D\) right neighbors that have not been previously matched. If no such set exists, then we say that the node is rejected. We claim that at most half of the added nodes are rejected.

Let \(M\) and \(R\) be the number of nodes that are matched, respectively rejected. We show that \(M \geq R\). For this, let us give an upper bound and a lower bound for the total number of neighbors of both rejected and matched nodes. On one hand, since the graph admits \(\varepsilon\)-rich offline matching up to \(K\), this number is at least \((1 - \varepsilon)D(M + R)\). On the other hand, it is at most \(DM + (1 - 2\varepsilon)DR\).

Indeed suppose we count the neighbors of left nodes in the order in which the left nodes are processed without double counting the neighbors already seen. Then each matched node contributes with at most \(D\) and each rejected node contributes with at most \((1 - 2\varepsilon)D\). The two bounds imply \(M \geq R\), and the claim is proved.

We now move to the actual proof and describe a Matcher that achieves \((2\varepsilon, 1 + \lceil \log K \rceil)\)-rich online matching up to \(K\) elements.

Due to the possibility of rejection, the Matcher repeatedly attempts to do the above operation for each node added by the Adversary. There are at most \(s := 1 + \lceil \log K \rceil\) attempts and they are done as follows. Each right node keeps during the entire game \(s\) flags, where the \(i\)-th flag indicates if the node has been assigned (flag is on) or not (flag is off) at attempt \(i \in [s]\) of some left node. Let \(x\) be the left node that is added by the Adversary and for which the Matcher has to respond by finding matching assignments. The Matcher checks the flags for attempt 1 of all \(x\)’s neighbors to see if there is a fraction of \((1 - 2\varepsilon)\) fraction of them that have not been assigned at this attempt. If the answer is yes, it assigns all these neighbors to \(x\), sets on their flags for attempt 1, and we say that \(x\) succeeds at attempt 1. Otherwise, \(x\) is rejected at attempt 1, and the Matcher does the same process using the flags for attempt 2, and so on, till \(x\) succeeds at some attempt and has a fraction of \((1 - 2\varepsilon)\) of its neighbors assigned to it. By the above analysis, at least half of the strings that are added by the Adversary succeed at attempt 1. At most half are rejected at attempt 1, but at least half of them succeed at attempt 2, and so on. Since the Adversary adds at most \(K\) nodes, eventually \(x\) succeeds in one of the \(s\) attempts. The flags guarantee that a right node is assigned to at most one left node at each attempt, and thus overall a right node can be assigned to at most \(s\) left nodes.

We now analyze the runtime. One attempt requires the inspection of the flags of the \(D\) neighbors, and, if the graph is explicit, this takes \(D \cdot \text{poly} \log N\) steps. The number of attempts is bounded by \(s = O(\log N)\), and so the total runtime is also bounded by \(D \cdot \text{poly} \log N\).

By an argument similar to the one in the proof of Theorem 10, we can extend online matching to dynamic matching with bounded surviving and we obtain the following.

**Theorem 17.** If a graph has \(\varepsilon\)-rich offline matching up to \(2K\), then it has \(T\)-surviving dynamic \((2\varepsilon, O(\log(KT)))\)-rich matching up to \(K\). Moreover, if the graph is explicit and has left degree \(D\), the runtime for doing a matching assignment is \(D \cdot \text{poly} \log N \cdot \log T\).

### 4.2 From \((\varepsilon, r)\)-rich matching to \((\varepsilon, 1)\)-rich matching

We convert any graph \(G\) that admits \((\varepsilon, r)\) online matching to a graph \(G'\) that admits \((2\varepsilon, 1)\) online matching. We dub this the “\(G \mapsto G + \text{hash transformation}\)” The idea is to use hashing to
Lemma 18

They have degree at most \( r \) that have been assigned to other desired property.

Proof. \( \frac{1}{2} \) node with a right side is \( \mid \). Proof.

We introduce the runtime of \( F \) by the available. This can be done in poly(\( G \)) consists in the computation of \( R \) at least \( (1 + 1) \) points in \( F \) available (for all \( G \)). We label all the left nodes by vectors in \( F \). We view each \( x \in L \) as a polynomial of degree at most \( n - 1 \) over \( F \) in the natural way, by considering each bit in the label of \( x \) as a coefficient of the polynomial (the polynomial \( x \) has only 0, 1 coefficients; \( F \) and \( F \) are the finite fields with 2, respectively \( s \) elements).

We construct the bipartite graph \( G' = (L' \cup R', E') \). The left side \( L' \) is \( L \) (so, the same left side as \( G \)). The right side is \( R' = R \times (F_s)^2 \). The multiset of edges \( E' \) of \( G' \) is defined as follows:

For each edge \( (x, p) \) of \( G \) (where \( x \) is a left node, and \( p \) is a right node in \( G \)), we introduce \( s \) edges in \( G' \), namely \( \{ (x, (p, x(a), a)) \mid a \in F_s \} \), where \( x(a) \) is the value of the polynomial \( x \) at \( a \).

Note that \( s = O(r \log |L|(1/\epsilon)) \), the graph \( G' \) has left degree \( D' = D \cdot s \), and the size of the right side is \( |R'| = |R| \cdot s^2 \). Also note that if \( G \) is explicit, \( G' \) is explicit as well.\(^6\)

Lemma 18 (\( G \mapsto G + \text{hash} \) transformation). If \( G \) admits \((\epsilon, r)\) online matching up to size \( K \), then \( G' \) admits \((2\epsilon, 1)\) online matching up to size \( K \). An algorithm \( f \) that matches a node with a \((1 - \epsilon)\) fraction of his neighbors in \( G \) with overlap \( r \), can be modified into an algorithm \( f' \) that matches a node with a \((1 - 2\epsilon)\) fraction of its neighbors in \( G' \) with no overlap. The runtime of \( f \) is bounded by the runtime of \( f + \text{poly}(|L|, 1/\epsilon, r) \).

Proof. We modify a matching algorithm \( f \) for \( G \) into a matching algorithm \( f' \) for \( G' \) with the desired property.

On input a left node \( x \), we first run \( f \) on \( x \). Each right node \( p \) that \( f \) assigns to \( x \) may also have been assigned to other \( r' \leq r - 1 \) elements that have arrived before \( x \), say, to \( \{x_1, \ldots, x_{r'}\} \). For every \( i \leq r' \), the polynomials \( x \) and \( x_i \) can be equal in at most \( (n - 1) \) points in \( F \), because they have degree at most \( n - 1 \). Thus there exists at least \( s - (n - 1)(r - 1) \geq (1 - \epsilon)s \) points \( a \) in \( F \) such that \( x(a) \notin \{x_1(a), \ldots, x_{r'}(a)\} \). Then \( f' \) assigns to \( x \) the first \((1 - \epsilon)s \) elements \((p, x(a), a)\) that are available (for all \( p \) assigned to \( x \) by \( f \)). By the above estimations, \( f' \) assigns to each left \( x \) at least \((1 - \epsilon)^2 > (1 - 2\epsilon) \) fraction of its neighbors, and the assignment procedure ensures that no right node is assigned to more than one left node. Besides running \( f \), the additional computation consists in the computation of \( x(a) \) for all \( a \) in \( F \), and checking which of the nodes \((p, x(a), a)\) are available. This can be done in poly(s) time.

\(^6\)We remind that \( G \) is explicit if it belongs to a family of graphs indexed by \( |L| \) and there exists an algorithm running in time \( \text{poly}(\log |L|) \) that, on input \( x \in L \) and \( i \in [D] \), outputs the \( i \)-th neighbor of \( x \).
4.3 Explicit graphs with fast $\epsilon$-rich almost dynamic matching

We have seen that if we start with a graph that has $\epsilon$-rich offline matching, we can obtain a graph that has fast $\epsilon$-rich almost dynamic matching. The next diagram summarizes the steps.

$$G \text{ has } \epsilon\text{-rich offline matching up to } 2K$$

Theorem 17

$$G \text{ has } T\text{-surviving dynamic (}2\epsilon,O(\log(KT))\text{-rich matching up to } K$$

$G \mapsto G + \text{hash transformation (Lemma 18)}$

$$G' \text{ has fast } T\text{-surviving dynamic } 4\epsilon\text{-rich matching up to } K$$

We now build an explicit graph with fast almost dynamic matching by following these steps. Naturally, we start by indicating a graph $G$ satisfying the condition at the top of the diagram. For this, we use the following variant of Hall’s theorem for rich matching.

**Proposition 19.** For a bipartite graph $G = (L,R,E \subseteq L \times R)$ with left degree $D$, the following are equivalent.

(a) $G$ has $\epsilon$-rich offline matching up to $K$.

(b) Every set $S \subseteq L$ with size $|S| \leq K$ has at least $(1-\epsilon)D|S|$ neighbors.

**Proof.** (a) $\Rightarrow$ (b) is obvious. We show (b) $\Rightarrow$ (a). Let $\ell = \lceil (1-\epsilon)D \rceil$. We construct the bipartite graph $G'$ which has the same right side as $G$ and left side equal to $\ell$ clones of the left side of $G$ (i.e., every clone node has the same neighbors as the original node).

If $S$ is a subset of the left nodes in $G'$, we denote by $\text{merge}(S)$ the set of left nodes in $G$ obtained by merging the clone nodes in $S$ to the original version. If $|S| \leq K$, we have:

$$|N_{G'}(S)| = |N_G(\text{merge}(S))| \geq \ell \cdot |\text{merge}(S)| \geq |S|.$$  

The first equality holds because cloning implies that $N_{G'}(S) = N_G(\text{merge}(S))$, the second inequality holds by (b), and the third inequality holds because at most $\ell$ clone nodes can be merged into an original node.

By Hall’s Theorem, $G'$ restricted to $S$ has a $(1,1)$ matching, i.e., there exists a subset of the edges of $G'$ such that in $G'$ restricted to $S$ and to this subset, every node has degree 1. By merging back the clones into the original node in $S$, we assign to every node $x$ in $S$, $\ell$ neighbors in $G$ (namely the matches of the clones of $x$), so that a right node is assigned to at most one node in $S$. \qed

The graphs satisfying the conditions in Proposition 19 are known as **lossless bipartite expanders**. According to the standard definition, a $(K,(1-\epsilon)D)$ lossless expander is a bipartite graph satisfying condition (b). The moniker “lossless” is justified by the fact that the expansion factor is close to the degree. These graphs have been studied extensively and have numerous applications (for a concise descriptions of the most important applications, see the introduction section of Capalbo et al. [CRVW02] and also [HLW06]). In applications it is desirable that the degree $D$ is small and that the **overhead** $\Delta$ of the right side, defined by $\Delta := \log |R| - \log K$, is also small. By “small,” we mean polynomial or quasipolynomial in $\log |L|$.

The probabilistic method yields nonexplicit lossless bipartite expander graphs with degree $D = O(n/\epsilon)$ and overhead $\Delta = O(\log(n/\epsilon))$. We use the following explicit lossless expander, whose construction is presented in Section 4.3.1.
Theorem 20. For every \( n \in \mathbb{N}, \ k_{\text{max}} \leq n, \) and \( \epsilon > 0, \) there exists an explicit \((2^{k_{\text{max}}}, \epsilon)\) lossless expander \( G = (L, R, E \subseteq L \times R), \) with

- \( L = \{0, 1\}^n, \)
- \( R = \{0, 1\}^{k_{\text{max}} + \Delta}, \)
- \( \text{degree} \ D = 2^\Delta, \)

where \( \Delta = \log n + O(\log(k_{\text{max}}/\epsilon) \cdot \log(k_{\text{max}})). \)

Remark. The lossless expander in Theorem 20 is obtained by combining the expanders from [GUV09, Th. 4.4] and [BZ19]. We note that the main lossless expander constructed by Guruswami, Umans, and Vadhan [GUV09] (different from the one we mentioned above) has a smaller degree, namely \( D = 2^d, \) for \( d = (1 + 1/\alpha)(\log n + \log k_{\text{max}} + \log(1/\epsilon)) + O(1), \) for any \( \alpha \in (0, 1), \) but a larger overhead \( \Delta = \alpha k_{\text{max}} + 2d. \) The lossless expander from [BZ19] has overhead \( \Delta = O(\log(n/\epsilon) \cdot \log(k_{\text{max}})) \) and \( D = 2^\Delta. \)

Next, we apply the steps in the above diagram to the graph \( G \) in Theorem 20, and we obtain the following family of explicit graphs with almost dynamic matching.

Theorem 21 (Explicit graph with fast \( \epsilon \)-rich almost dynamic matching). Let \( N = 2^n \) and \( K = 2^k \) be powers of two and \( K \leq N \) and let \( T \) be an arbitrary integer, and \( \epsilon > 0. \)

There exists an explicit family of graphs \( \{G_{N,K,T}\} \) such that every \( G_{N,K,T} \) has the following properties:

- Left size \( |L| = N, \) right size \( |R| = K \cdot \text{poly}(n) \cdot 2^{O(\log(k/\epsilon) \cdot \log k)} \cdot (\log T)^2, \) and the left degree \( D = \text{poly}(n) \cdot 2^{O(\log(k/\epsilon) \cdot \log k)} \cdot \log T. \)
- It has \( \epsilon \)-rich, \( T \)-surviving dynamic matching up to \( K \) nodes, and
- The runtime for doing one matching assignment is \( \text{poly}(n, D). \)

The above theorem is essentially a restatement of Theorem 3, where \( N \) and \( K \) are not required to be powers of 2. (Clearly if they are not powers of 2, we can do the construction for the smallest powers of 2 larger than them).

4.3.1 Proof of Theorem 20

It is known from [TSUZ07] that lossless expanders are equivalent to a certain type of lossless condensers, and therefore we need to construct a condenser of the appropriate type.

We start with some standard concepts from the theory of pseudo-randomness (see [Vad12]). A source is a random variable whose realizations are binary strings. A source has min-entropy \( t \) if each value has probability at most \( 2^{-t}. \) The statistical distance between two measures \( P \) and \( Q \) with the same domain is \( \sup |P(S) - Q(S)| \) (supremum is over all subsets \( S \) of the common domain of \( P \) and \( Q \)). Given a set \( B, \) we denote \( U_B \) to be a random variable that is uniformly distributed on \( B. \) A condenser uses an additional random string (called the seed), which is uniformly distributed over the set of \( d \)-bit strings, for some small \( d. \)

Definition 22. A function \( C : \{0, 1\}^n \times \{0, 1\}^d \to \{0, 1\}^m \) is a \( t \to \epsilon, t' \) condenser, if for every \( S \subseteq \{0, 1\}^n \) of size at least \( 2^t, \) the random variable \( X = C(U_S, U_{\{0,1\}}) \) is \( \epsilon \)-close to a random variable \( \tilde{X} \) that has min-entropy at least \( t'. \)
We use functions that are condensers for an entire range of min-entropies, and preserve all the input min-entropy (so called lossless condensers). More precisely, we use families of functions \( \{ C_n \} \) indexed by \( n \) with parameters \( d, \epsilon, m, t_{max} \) functions of \( n \) (all, except \( \epsilon \) being positive integers; we assume \( \epsilon < 1/2 \)) and satisfying

\[
C_n \text{ has type } C_n : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m, \quad \text{and}
\]

\[
C_n \text{ is an explicit } t \to \epsilon + d \text{ condenser for all } t \leq t_{max} \text{ such that } 2^t \in \mathbb{N}.
\] (2)

Functions satisfying Eq. (2) are very similar to lossless conductors, the only difference being that conductors do not have the restriction that \( 2^t \in \mathbb{N} \). As usual we drop the subscript \( n \) in the notation. Note that \( C \) has entropy loss bounded by \( e \) for inputs with min-entropy \( t \) for all \( t \leq t_{max} \) such that \( 2^t \in \mathbb{N} \). Functions \( C_n \) having the type in Eq. (2) can be viewed as bipartite graphs \( G_n = (L, R, E \subseteq L \times R) \) in the standard way: \( L = \{0,1\}^n \), \( R = \{0,1\}^m \) and \( (x, z) \in E \) if and only if there is \( y \in \{0,1\}^d \) such that \( C_n(x,y) = z \). As mentioned above, Tal-Shma, Umans, and Zuckerman have shown that functions satisfying Eq. (2), when viewed as graphs, are equivalent to lossless expanders.

**Theorem 23.** A family of functions \( \{ C_n \}_{n \in \mathbb{N}} \) satisfies the conditions in Eq. (2) if and only if the corresponding family of graphs \( \{ G_n \}_{n \in \mathbb{N}} \) is a \( (2^{t_{max}},(1-\epsilon)D) \) bipartite expander (and therefore, by Proposition 19, has \( \epsilon \)-rich offline matching up to size \( 2^{t_{max}} \)).

Consequently, to prove Theorem 20 we need to construct a condenser \( C \) satisfying Eq. (2) with \( d = O(\log n + \log(t_{max}/\epsilon) \cdot \log(t_{max})) \) and \( m = t_{max} + O(\log n + \log(t_{max}/\epsilon) \cdot \log(t_{max})) \). This is obtained by composing the following two condensers from the literature.

**Theorem 24 (\cite{GUV09}, Theorem 4.4).** For every \( n \), \( t_{max} \leq n, \epsilon > 0 \), there exists an explicit function \( C_{GUV} : \{0,1\}^n \times \{0,1\}^{d_{GUV}} \to \{0,1\}^{m_{GUV}} \) satisfying the conditions in Eq. (2) with \( d_{GUV} \leq \log n + \log(t_{max}) + \log(\frac{1}{\epsilon}) + 1 \) and \( m_{GUV} \leq d_{GUV} \cdot (t_{max} + 2) \).

**Theorem 25 (\cite{BZ19}, implicit in Theorem 2.1).** For every \( n \), \( t_{max} \leq n, \epsilon > 0 \), there exists an explicit function \( C_{BZ} : \{0,1\}^n \times \{0,1\}^{d_{BZ}} \to \{0,1\}^{m_{BZ}} \) satisfying the conditions in Eq. (2) with \( d_{BZ} = O(\log(n/\epsilon) \cdot \log(t_{max})) \) and \( m_{BZ} \leq t_{max} + O(\log(n/\epsilon) \cdot \log(t_{max})) \).

Now, as announced, we obtain the condenser \( C \) with the parameters from Theorem 20, by composing \( C_{GUV} \) and \( C_{BZ} \). Namely, \( C : \{0,1\}^n \times \{0,1\}^{d_{GUV}+d_{BZ}} \to \{0,1\}^{m_{BZ}} \) is defined by

\[
C(x, (y_1, y_2)) = C_{BZ}(C_{GUV}(x, y_1), y_2).
\]

We now specify the parameters. We first condense with \( C_{GUV} \) for sources \( X \) with min-entropy \( t \leq t_{max} \) using a seed of length \( d_{GUV} \leq \log n + \log(t_{max}) + \log(1/\epsilon) + 1 \) and obtain an output \( X_1 \) of length \( m_{GUV} \leq d_{GUV} \cdot (t_{max} + 2) \) and min-entropy at least \( t + d_{GUV} \). Then we further condense \( X_1 \) with \( C_{BZ} \) with parameters \( d_{BZ} \) and \( m_{BZ} \) set-up for \( \epsilon \)-lossless condensing and for sources with input length \( m_{GUV} \) and min-entropy bounded by \( t'_{max} = t_{max} + d_{GUV} \). In this way, we obtain a \( t \to 2t \) condenser for all \( t \leq t_{max} \) such that \( 2^t \in \mathbb{N} \), with the parameters required by Theorem 20, except for the case \( t_{max} = o(\log n) \). But for such small \( t_{max} \), we can take the trivial expander in which every left node is connected to all right nodes.

## 5 Application 1: Non-blocking connectors

We construct a wsnb generalized \( N \)-connector with surviving parameter \( T \) and depth \( t \). The relevant definitions are given in the Introduction.
Theorem 26. Assume $T \leq 2^{N^{1/(t+2)}}$. There exists an explicit family $\{C_{N,T,t}\}$ in which each graph $C_{N,T,t}$ is a $T$-surviving wsnb generalized $N$-connector of depth $t$. Moreover, this graph has $N^{1+1/t} \cdot (\log(NT))^{O(t)}$ edges and a path finding algorithm running in time $t \cdot \text{poly}(\log N, \log T)$.

A path can be found locally, and it requires in each node of the found path at most a $\text{poly}(\log N, \log T)$ amount of computation with a constant amount of communication to each neighbor.

Proof. We introduce the function $\alpha(N,T) = \log(\log^{c}N \cdot \log T)$ for a constant $c$ chosen so that in the graph $G_{N,K,T}$ in Theorem 14, the left degree $D$ is bounded by $2^{\alpha(N,T)}$ and $|R|$ (i.e., the size of the right set) is bounded by $K \cdot 2^{\alpha(N,T)}$.

We prove by induction on $t$ that there is a $T$-surviving wsnb generalized connector with $N$ inputs and $N$ outputs, having depth $t$, and with the number of edges bounded by $N^{1+1/t} \cdot (2^{t} \cdot 2^{\alpha(N,T)})$.

If $t = 1$, then the wsnb generalized connector is the complete bipartite graph with $N$ nodes on each side (the left nodes are the inputs, and the right nodes are the outputs).

We move to the induction step $t \to t + 1$. We construct the network $G$ of depth $t + 1$ that satisfies the conditions for $t + 1$. We take

$$K := \left\lfloor N^{t/(t+1)} \cdot 2^{-((t^{2}+1)/(t+1)) \cdot \alpha(N,T)} \right\rfloor. \quad (3)$$

$G$ consists of $\left\lfloor N/K \right\rfloor$ copies of the following network $H$. $H$ is build by connecting a bipartite graph $G'$ with the wsnb generalized connector $G_{t}$ of depth $t$ implied by the inductive hypothesis, so that the right side of $G'$ is identified with the inputs of $G_{t}$ (see Fig. 2, left graph). $G'$ is the graph $G_{N,K,T}$ from Theorem 14 $G' = (L' \cup R', E')$, where $L' = \{0,1\}^{n}$ for $n = \log N$, having left degree $D'$, that admits $T$-surviving dynamic matching up to size $K$. By the definition of the function $\alpha$, the right side of $G'$ has size $|R'| = K \cdot 2^{\alpha(N,T)}$ and $D'$ is bounded by $2^{\alpha(N,T)}$. Taking into account the assumption that $T \leq 2^{N^{1/(t+2)}}$, it follows that $|R'| \leq N$. The graph $G_{t}$ is the wsnb generalized connector with $|R'|$ inputs and $K$ outputs of depth $t$, obtained from the induction hypothesis, but in which we keep only the first $K$ outputs (out of $|R'|$ many ones). Clearly, deleting some output nodes preserves the wsnb property.
As announced, we build the generalized connector $G$ of depth $t + 1$ by taking $\lceil N/K \rceil$ copies of the network $H$, which all share the same set of $N$ inputs (see Fig. 2, right graph). By deleting some outputs, $G$ has $N$ inputs and $N$ outputs.

To simplify the notation, we denote $\alpha(N,T)$ by just $\alpha$. We need to show that $|\mathcal{E}(G)|$ (the number of edges of $G$) is bounded by $N^{1+1/(t+1)} \cdot 2^{3(t+1)} \cdot 2^{t\alpha}$. This holds because

$$|\mathcal{E}(G)| = \lceil \frac{N}{K} \rceil |\mathcal{E}(H)| \leq \frac{2N}{K} |\mathcal{E}(H)| \leq \frac{2N}{K} (ND' + |R'|^{1+1/t} \cdot 2^{3t} \cdot 2^{t\alpha}(|R'|,T))$$

$$\leq \frac{2N}{K} \left( N \cdot 2^\alpha + |R'|^{1+1/t} \cdot 2^{3t} \cdot 2^{t\alpha} \right) \leq \frac{2N}{K} \left( N \cdot 2^\alpha + K^{1+1/t} \cdot 2^{(1+1/t)\cdot \alpha} \cdot 2^{3t} \cdot 2^{t\alpha} \right)$$

$$= \left( 2 \cdot \frac{N^2 \cdot 2^\alpha}{N^{1/(t+1)}} + 2^{3t+2} \cdot N \cdot N^{1/(t+1)} \cdot 2^{-\alpha \cdot (t^2+1)/(t^2+t)} \cdot 2^{(t+1+1/t)\cdot \alpha} \right)$$

$$\leq \left( 2 \cdot N^{1+1/(t+1)} \cdot 2^{(t+1)\cdot \alpha} + 2^{3t+2} \cdot N^{1+1/(t+1)} \cdot 2^{(t+1)\cdot \alpha} \right)$$

$$\leq 2^{3(t+1)} \cdot N^{1+1/(t+1)} \cdot 2^{(t+1)\cdot \alpha}.$$

We next describe the path-finding algorithm, which is rather straightforward. For clarity, let us consider first the case of a non-generalized connector. Suppose the request is to connect the input $u$ to the output $v$, and the request is valid (i.e., $u$ and $v$ do not appear in any path existing at this moment). The algorithm identifies the copy of $H$ containing $v$, and attempts to find a match for $u$ in the graph $G'$ that makes the first layer of this copy of $H$ (see Fig. 2 right graph). Since $G'$ has $T$-surviving matching up to the $K$ from Eq. (3) and there are at most $K$ outputs in the copy of $H$, the algorithm successfully matches $u$ with some node $u'$, and $u'$ does not appear on any existing path (because otherwise it would have been assigned to the input on that path). Next, inductively, the algorithm finds a path from $u'$ to $v$ in the remaining layers of the copy of $H$, which is vertex-disjoint with the existing paths. In this way the path from $u$ to $v$ is obtained.

In the case the connector is generalized, the node $u$ may be the input of an existing path to some other output $v'$ (if it is not, we proceed as above). Let $(u,u')$ be the first edge on this path. Then the algorithm finds inductively a path from $u'$ to $v$ that satisfies the requirements for a generalized connector.

The path-finding algorithm consists essentially of at most $t$ matchings (one in each layer in the non-generalized connector case; in the generalized case, some matchings are re-used from previous paths with which the current path shares an initial segment). The claim regarding the runtime follows immediately from the complexity of the fast $T$-surviving dynamic matching algorithm. 

6 Application 2: One-probe storage scheme for dynamic sets

Recall from the Introduction that the goal is to store a subset $S$ of size $K$ of a large set $U$ (the “universe”) of size $N$ and that a one-probe storage scheme is a randomized data structure that answers any membership query “Is $x$ in $S$?” by reading a single bit, with error probability at most $\epsilon$.

We show that graphs that admit almost dynamic matching can be used to obtain one-probe storage schemes for dynamic sets, i.e., with fast insert and delete operations. Moreover, when an element $x$ is inserted, the table is changed only locally in a few positions associated with $x$. When $x$ is deleted, only the positions associated with $x$ and with the oldest element in the dynamic set
are modified. The dynamic set is characterized by the following parameters: \( N \) is the size of the universe, \( \epsilon \) is the error probability, \( K \) is a bound on the size of the dynamic set at every moment in its history. Every element can be deleted and inserted infinitely many times.

We first give an informal description of the data structure. For the implementation, we use a graph \( G = (L \cup R, E) \) with left degree \( D \), where \( L = U \) (the “universe”), and which admits \( ((1 - \epsilon)D, 1) \) \( K \)-surviving dynamic matching up to \( K + 1 \) nodes.

The idea is simple. The data structure contains a table \( A \) of \( |R| \) bits, whose entries are indexed by elements in \( R \). When some \( x \) is inserted, we run the dynamic matching algorithm for \( x \) and set to 1 the bits at the locations assigned to \( x \). When \( x \) is deleted, the bits at these locations are set to 0, and we also refresh the oldest element \( x_{old} \) in the dynamic set, which means that \( x_{old} \) is deleted and immediately re-inserted and consequently is matched with new locations (this is needed to ensure that no matching assignment survives more than \( K \) steps). The key claim is that the bits in \( A \) are set so that for all \( x \in U \), the bits in the positions assigned to \( x \) by the dynamic matching procedure (for real if \( x \) has been inserted or refreshed in the last \( K \) steps, and virtually, otherwise, as we explain shortly) are 1 if \( x \in S \), and 0 if \( x \not\in S \). Since all but at most an \( \epsilon \) fraction of right neighbors are assigned to \( x \), by probing the table at the bit indexed by a random neighbor, we determine if \( x \) is in \( S \) or not, with probability \( 1 - \epsilon \). Now let’s analyze the key claim. First, taking into account the matching assignments, the claim holds true for the elements inserted or refreshed in the last \( K \) steps, because the graph has dynamic matching up to size \( K \) and survival parameter \( K \). What about the other elements? On one hand, such an element cannot be in \( S \) (because elements that are inserted early, are refreshed within the last \( K \) insertions), and on the other hand it can be viewed as a dummy element for which a virtual request can be made (recalling that the game allows \( K + 1 \) matching requests and the size of \( S \) is at most \( K \)). Therefore, the Matcher will find assignments for the dummy element. By definition the sets of indices assigned to actual elements and virtually to the dummy element are pairwise disjoint which implies that, as desired, the entries in the table are 0 in all positions virtually assigned to the dummy element.

**Formal definition of one-probe storage scheme.** A dynamic data set with elements from a set \( U \) is defined by a *history*, which is an infinite sequence \( H = (\mathsf{op}_1(x_1), \mathsf{op}_2(x_2), \ldots) \), where every \( \mathsf{op}_i \) is either \textbf{insert} or \textbf{delete}, and every \( x_i \) is an element in \( U \). The dynamic data set with history \( H \) is an infinite sequence of subsets of \( U \), \( S = (S_0, S_1, \ldots) \), where \( S_0 = \emptyset \), and \( S_i = S_{i-1} \cup \{x_i\} \) if \( \mathsf{op}_i = \textbf{insert}(x_i) \) and \( S_i = S_{i-1} \setminus \{x_i\} \) if \( \mathsf{op}_i = \textbf{delete}(x_i) \).

We finally define one-probe storage schemes for dynamic sets of size at most \( K \), table size \( M \), and error probability \( \epsilon \) (in short a \( (K, M, \epsilon) \)-one probe storage scheme). Such a storage scheme consists of deterministic algorithms \textbf{insert} and \textbf{delete} and probabilistic algorithm \textbf{query} which have the properties that we present next. The algorithms \textbf{insert} and \textbf{delete} share a *state* chosen from a set of states \( S \) and all three algorithms share a binary string of length \( M \).\(^7\) The algorithms compute functions of type

\[
\begin{align*}
\text{\textbf{insert}} & : S \times \{0, 1\}^M \times U \to S \times \{0, 1\}^M \\
\text{\textbf{delete}} & : S \times \{0, 1\}^M \times U \to S \times \{0, 1\}^M \\
\text{\textbf{query}} & : \{0, 1\}^M \times U \to \{0, 1\}
\end{align*}
\]

The operational view of the history \( H \) is the sequence \((\mathsf{op}_1(\sigma_0, A_0, x_1), \mathsf{op}_2(\sigma_1, A_1, x_2), \ldots) \) where \( \mathsf{op}_i \) and \( x_i \) are as above, \( \sigma_0, \sigma_1, \ldots \) is a sequence of states from \( S \), and \( A_0, A_1, \ldots \) is a sequence of bitstrings of length \( M \) defined as follows: \( \sigma_0 \) is a designated state called the initial state, \( A_0 = 0^M \), and \( (\sigma_i, A_i) = \mathsf{op}_i(\sigma_{i-1}, A_{i-1}, x_i) \) for \( i = 1, 2, \ldots \).

\(^7\)We can think that \textbf{insert} and \textbf{delete} are operated by an administrator, and \textbf{query} is operated by clients. The state is used by the administrator to maintain the dynamic set, and the binary string is the interface that is accessible to the clients.
A history $H$ is $K$-legal if for every $i$, $|S_i| \leq K$. The algorithm query has to satisfy the following conditions: For any $K$-legal history $H$

(a) For all $x \in U$ and all $i$, $\text{query}(A_i, x) = [x \in S_i]$ with probability $1 - \epsilon$ (correctness condition; $[x \in S_i]$ is the predicate which is 1 if $x \in S_i$, and 0 otherwise),

(b) In the computation of query on input $A$ and $x$, a single bit of $A$ is read with probability 1 (one-probe condition: the result of query depends on a single bit of $A$).

The one-probe storage scheme. We follow the plan described informally above. Let $G = (L \cup R, E)$ be a bipartite graph with left degree $D$, where $L = U$ (the “universe”), and which admits dynamic $\epsilon$-rich matching up to $K + 1$ nodes for survival parameter equal to $K$. $A$ is a binary string of length $|R|$ indexed by the elements of $R$, i.e., for every $p \in R$, $A(p)$ denotes the bit of $A$ at location $p$. We describe the three operations: insert, delete, query. We think that a matching game is played during the history of the set. When an element $x$ is inserted, the Adversary adds $x$ in the game (by putting a blue token on it), and the Matcher responds if she can by assigning a fraction of $(1 - \epsilon)$ of $N(x)$ to $x$ (by putting red tokens on them). We denote by $f(x)$ the set of nodes assigned to $x$.

The operations insert and delete share a state $\sigma$ which is a set of at most $K$ tuples of the form $(x, f(x))$, with one such tuple for every $x$ that currently is in the dynamic set, where $f(x)$ is equal to the set of strings assigned to $x$ when $x$ was inserted the last time. The initial state is the empty set. The state $\sigma$ is implemented as a queue so that if $(x, f(x))$ is the first (respectively, last) node in the queue then $x$ is the currently oldest (respectively, most recent) element inserted in the dynamic set. The queue is implemented in a way that allows searching and deleting of arbitrary nodes (i.e., not necessarily the first node) in $\log K$ steps (for example, by using a Red-Black search tree).

**insert($\sigma, A, x$)**

*Let $S$ be the current instance of the dynamic set. The elements which belong to $S$ can be determined from the state $\sigma$.]*

Lock writing in $A$ and $\sigma$.

If $x \notin S$, run the matching algorithm and let $f(x)$ be the elements assigned to $x$; set $A(p) \leftarrow 1$ for all $p \in f(x)$.

Enqueue $(x, f(x))$ in $\sigma$ and release the lock for $A$ and $\sigma$.

If the history is legal, when insert is executed, $S$ has at most $K$ elements and no assignment has survived more than $K$ steps (this latter claim also takes into account the implementation of delete). Consequently, the matching algorithm succeeds, and the $(1 - \epsilon)D$ positions assigned to $x$ are set to 1. Assuming oracle access to $G$ or if $G$ is explicit, the runtime for insert is $\text{poly}(\log N, D)$ (this is the runtime for doing the matching; the other operations take less time and are absorbed in the $\text{poly}()$ factor).
To obtain the following one-probe storage scheme.

Let \( S \) be the current instance of the dynamic set. The elements which belong to \( S \) can be determined from the state \( \sigma \).

Lock writing in \( A \) and \( \sigma \).

If \( x \in S \) get \( f(x) \) from \( \sigma \) and set \( A(p) \leftarrow 0 \) for all \( p \in f(x) \).

Delete \( (x, f(x)) \) from \( \sigma \).

Let \( (x_{old}, f(x_{old})) \) be the first element in the queue \( \sigma \). Delete it from the queue and run \( \text{insert}(\sigma, A, x_{old}) \) (i.e., the oldest element in the dynamic set is refreshed).

Release the lock for \( A \) and \( \sigma \).

Assuming oracle access to \( G \) or if \( G \) is explicit, the runtime for \( \text{delete} \) is poly(\( \log N, D \)).

\!
\begin{tabular}{|c|}
\hline
\text{query}(A, x) \\
\hline
\end{tabular}

Pick \( p \) uniformly at random in \( \mathcal{N}(x) \). If \( A(p) = 1 \), return 1 otherwise return 0.

Assuming oracle access to \( G \) or if \( G \) is explicit, the running time for \( \text{query} \) is poly(\( \log N, \log D \)).

We check that \( \text{query}(A, x) \) satisfies the conditions (a) and (b).

Property (a): If \( x \) has been inserted (we include here the re-insertion case of the oldest element) or deleted in the last \( K \) steps, the last insert/delete operation with argument \( x \) has set the bits of \( A \) at positions assigned to \( x \) to 1 or 0 in the correct way. The positions cannot be modified by the insertion or deletion of other elements (which only affect the positions assigned to them).

If \( x \) has not been inserted or deleted in the last \( K \) steps, then \( x \) is not in the current instance of the set (because older elements are re-inserted within the last \( K \) steps). We can think that we run a step of the game in which the Adversary adds \( x \) and the Matcher assigns \((1 - \epsilon)D \) of its neighbors to \( x \) (\( x \) is the dummy element from the informal argument given above). The Matcher succeeds because the graph admits matching up to \( K + 1 \). The bits in \( A \) at positions assigned to \( x \) are all 0 because previous \( \text{insert} \) set to 1 only bits at positions assigned to their arguments.

In both cases, since \( \text{query}(A, x) \) reads with probability \( 1 - \epsilon \) a position in \( A \) assigned to \( x \) (for real or virtually), property (a) is satisfied.

Property (b): \( \text{query}(A, x) \) reads only the bit \( A(p) \) for a random \( p \).

**Generic one-probe storage scheme from any lossless expander.** It follows from Section 4 that any lossless expander can be used to obtain a one-probe storage scheme. Indeed, let \( G = (L, R, E \subseteq L \times R) \), with left degree \( D \), be a \((1 - 2\epsilon)D, 2K'\) expander, and suppose that \( L = \{0, 1\}^n \) is the "universe" set. Then \( G \) has \( 2\epsilon \)-rich offline matching up to size \( 2K \) (by Proposition 19). This means that \( G \) has fast \((2\epsilon, O(\log K))\)-rich almost dynamic matching up to size \( K' \) and survival parameter \( K + 1 \) (by Theorem 17). If we apply the \( G \mapsto G + \text{hash} \) transformation, we obtain a graph \( G' = (L, R', E' \subseteq L \times R') \), left degree \( D' \), with fast \( 4\epsilon \)-rich dynamic matching up to size \( K \) and survival parameter \( K + 1 \), where \( |R'| = |R| \cdot s^2 \) and \( D' = D \cdot s \), for \( s = O(n \cdot \log K \cdot (1/\epsilon)) \) (by Lemma 18). By our discussion above, this yields a one-probe storage scheme for dynamic subsets of \( L \) of size up to \( K \), with the following parameters:

- table size = \( |R| \cdot s^2 \),
- \( \text{query} \) has runtime bounded by \( \text{poly}(n, \log D) \),
- \( \text{insert} \) and \( \text{delete} \) have runtimes bounded by \( \text{poly}(n, D) \).

In particular, if we use the graph \( G \) with \( \epsilon \)-rich almost dynamic matching from Theorem 3, we obtain the following one-probe storage scheme.
**Theorem 27** (one-probe storage scheme for dynamic sets). For every functions $k := k(n) \leq n$, $\epsilon := \epsilon(n) > 0$, there exists a one-probe storage scheme for dynamic subsets of $U = [N]$, of size at most $K := 2^k$, that achieves the following values:

- **table size** $= K \cdot \text{poly}(\log N) \cdot 2^{O(\log(k/\epsilon) \cdot \log k)}$.
- **query** has runtime bounded by $\text{poly}(\log N, \log 1/\epsilon)$.
- **insert** and **delete** have runtimes bounded by $\text{poly}(\log N) \cdot 2^{O(\log(k/\epsilon) \cdot \log k)}$.

**Remark 1.** The one-probe storage scheme in Theorem 27 has smaller table size than the one-probe storage schemes in [BMRV00, Ta-02, GUV09], even though these schemes have the limitation of handling only static sets, which means that insertion and deletion can only be handled by reconstructing the whole table. Plugging in the generic construction the lossless expanders used in [BMRV00, Ta-02, GUV09], one obtains one-probe storage schemes with table sizes equal to (table sizes from these papers) $\times O((\log N \cdot \log K \cdot (1/\epsilon))^2)$, in which query, insert, and delete have runtime $\text{poly}(\log N, \log 1/\epsilon)$.

**References**

[ALM96] 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.

[BKP+81] M. Blum, R. Karp, C. Papadimitriou, O. Vornberge, and M. Yannakakis. The complexity of superconcentrators. *Info. Proc. Lett.*, 13:164–167, 1981.

[BMRV00] 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.

[BMVZ18] 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.

[BP73] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1973.

[BZ19] Bruno Bauwens and Marius Zimand. Universal almost optimal compression and Slepian-Wolf coding in probabilistic polynomial time. *CoRR/arxiv*, abs/1911.04268, 2019.

[CRVW02] M. R. Capalbo, O. Reingold, S. P. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In John H. Reif, editor, *STOC*, pages 659–668. ACM, 2002.

[FFP88] Paul Feldman, Joel Friedman, and Nicholas Pippenger. Wide-sense nonblocking networks. *SIAM J. Discret. Math.*, 1(2):158–173, 1988.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *J. ACM*, 56(4), 2009.

[HKPS20] 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 (Virtual Conference), volume 173 of LIPIcs, pages 58:1–58:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[HLW06] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. Bull. Amer. Math. Soc., 43:439–561, 2006.

[Hwa98] Frank K. Hwang. The Mathematical Theory of Nonblocking Switching Networks. World Scientific, 1998.

[Mar73] G. A. Margulis. Explicit constructions of concentrators. Problems of Information Transmission, 9:325–332, 1973.

[MRS11] D. Musatov, A. E. Romashchenko, and A. Shen. Variations on Muchnik’s conditional complexity theorem. Theory Comput. Syst., 49(2):227–245, 2011.

[PY82] N. Pippenger and A.C.-C. Yao. Rearrangeable networks with limited depth. SIAM J. Algebraic Discrete Methods, 2:411–417, 1982.

[Ta-02] Amnon Ta-Shma. Storing information with extractors. Inf. Process. Lett., 83(5):267–274, 2002.

[TSUZ07] A. Ta-Shma, C. Umans, and D. Zuckerman. Lossless condensers, unbalanced expanders, and extractors. Combinatorica, 27(2):213–240, 2007.

[Vad12] Salil P. Vadhan. Pseudorandomness. Foundations and Trends in Theoretical Computer Science, 7(1-3):1–336, 2012.

[WZ99] Avi Wigderson and David Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. Comb., 19(1):125–138, 1999.