

Edge-disjoint paths in expanders: online with removals

Nemanja Draganić*

Rajko Nenadov†

Abstract

We consider the problem of finding edge-disjoint paths between given pairs of vertices in a sufficiently strong d -regular expander graph G with n vertices. In particular, we describe a deterministic, polynomial time algorithm which maintains an initially empty collection of edge-disjoint paths \mathcal{P} in G and fulfills any series of two types of requests:

1. Given two vertices a and b such that each appears as an endpoint in $O(d)$ paths in \mathcal{P} and, additionally, $|\mathcal{P}| = O(nd/\log n)$, the algorithm finds a path of length at most $\log n$ connecting a and b which is edge-disjoint from all other paths in \mathcal{P} , and adds it to \mathcal{P} .
2. Remove a given path $P \in \mathcal{P}$ from \mathcal{P} .

Importantly, each request is processed before seeing the next one. The upper bound on the length of found paths and the constraints are the best possible up to a constant factor. This establishes the first online algorithm for finding edge-disjoint paths in expanders which also allows removals, significantly strengthening a long list of previous results on the topic.

1 Introduction

Finding a collection of pairwise edge-disjoint paths which connect prescribed pairs of vertices $(a_i, b_i)_{i \in [r]}$ in a graph G is a classical and extensively studied problem in computer science. It is an NP-complete problem which becomes tractable when r is fixed [22]. In the case of directed graphs, the problem remains NP-complete even for $r = 2$ [11].

In this paper we focus on the extensively studied instance of this problem when G is a sufficiently strong d -regular *expander* on n vertices. Peleg and Upfal [21] showed that the problem on such graphs becomes tractable for significantly larger values of r , under an assumption that all the given pairs (a_i, b_i) are pairwise disjoint. In particular, they showed that any set of at most $r = O(n^c)$ disjoint pairs of vertices can be connected by edge-disjoint paths, for some $c < 1/2$ which depends on the expansion properties of G , and such paths can be found in polynomial time. Let us briefly discuss why this is not surprising and establish a (theoretical) upper bound of r that can be attained.

Routing in expanders. Without going into detail of the definition of d -regular expanders (Definition 1.1), let us note that one of their main features is that the diameter is $O(\log n)$ even if we remove, say, up to $d/3$ edges touching each vertex. Therefore, if we can ‘nicely’ distribute paths (i.e. no vertex belongs to too many) between $(a_1, b_1), \dots, (a_{r-1}, b_{r-1})$, then this observation immediately tells us that we can connect a_r and b_r using a path of length $O(\log n)$. As there

*Department of Mathematics, ETH, Zürich, Switzerland. Research supported in part by SNSF grant 200021_196965. *email*: nemanja.draganic@math.ethz.ch.

†School of Computer Science, University of Auckland, New Zealand. *email*: rajko.nenadov@auckland.ac.nz.

exist expanders in which most pairs of vertices are actually at distance $\Theta(\log n)$, this is clearly a length one cannot avoid if the pairs are chosen adversarially. Moreover, there also exist expanders in which the second shortest path between *every* two vertices is of length $\Omega(\log n)$ (i.e. expanders with high girth), which provides further evidence that shorter paths, even if they exist, are not feasible. Therefore, if we restrict to sets of disjoint pairs, the largest r one can hope for is of order $n/\log n$. In a more general case where we do not impose such a constraint, as G has $nd/2$ edges, the largest r is (or could be) of order $nd/\log n$.

Previous results. The result of Peleg and Upfal falls short of the theoretically best possible bound on r , and it was an important open problem to determine whether such a bound is (algorithmically) attainable. Starting with Broder, Frieze and Upfal [6], who improved r to $\Theta(n/\log^C n)$, for some $C \geq 7$ (which, again, depends on the expansion properties), this problem has attracted significant attention. The bound on r was further improved by Leighton and Rao [17] (see [18, Section 3.18]), Broder, Frieze and Upfal [7], and Leighton, Lu, Rao, and Srinivasan [16, 19]. Finally, Frieze [13] gave a randomised polynomial time algorithm which connects any pair of $\Theta(nd/\log n)$ pairs of vertices, as long as each vertex appears as an endpoint at most εd times, for some constant $\varepsilon > 0$. In the case of directed expanders, the same result was established later by Bohman and Frieze [4]. We note that in the case where G is a random graph with average degree d , an even better bound on r of order $nd/\log_d n$ was obtained by Broder, Frieze, Suen, and Upfal [5] and Frieze and Zhao [14] (note that the diameter of a random graph can be somewhat smaller than the diameter of an expander graph, which results in slightly larger r). As the latest result on this topic, Alon and Capalbo [2] have further improved the undirected case by designing a deterministic polynomial time algorithm which not only can deal with the optimal number of requested pairs, but their algorithm is also *online* in the sense that the pairs are given one by one, and the algorithm has to find a path between the current pair of vertices before seeing the next one (once a path is established, it cannot be altered).

A related problem of finding vertex-disjoint paths in random graphs was considered by Shamir and Upfal [23], Hochbaum [15], Broder, Frieze, Suen, and Upfal [8] and, quite recently, Draganić, Krivelevich, and Nenadov [9].

Our contribution. We generalise all these results by presenting an online deterministic algorithm with removals: The client can request a new pair of vertices to be connected, or a previously established path to be removed. The client can continue with requests indefinitely as long as the total number of (active) paths is at most $O(nd/\log n)$ in case of expanders, that is, at most $O(nd/\log_d n)$ in the case of the so-called (n, d, λ) -graphs with $\lambda < d^{1-\varepsilon}$ (which covers the case of random d -regular graphs). In addition, we always guarantee that the length of the found paths is at most of order $\log n$, which resolves a question of Alon and Capalbo [2]. Our approach significantly deviates from the methods used in the aforementioned work. It is conceptually simple and exploits the previously discussed, fundamental reason why finding many edge-disjoint paths is possible – even if we only have a subset of edges at our disposal, the diameter is still $O(\log n)$.

The paper is organised as follows. In the next section we formally state our results. Section 2 describes a data structure that we call EDGE-ORACLE, which is the heart of the algorithm. Instead of working with all available edges to find a path between two given vertices, this data structure carefully chooses, in an online fashion, a subset of edges which makes sure that the found paths are well distributed. Finally, in Section 3 we describe the algorithm for establishing new paths and

prove its correctness. The algorithm is largely based on Breadth-First Search, with the edges used to explore the graph obtained via EDGE-ORACLE.

1.1 Online routing with removals

We say that a directed graph (*digraph* for short) is d -regular if the in- and out-degree of every vertex is d . Given a d -regular graph G (directed or undirected) and $r \in \mathbb{N}$, we define the r -Routing game on G played by two players, *Adversary* and *Router*, as follows. Throughout the game, Router maintains a family of pairwise edge-disjoint paths \mathcal{P} , which is initially empty. In each turn, Adversary has two types of requests:

- **FIND-PATH $a b$:** Given vertices a and b in $V(G)$ such that both a and b appear as endpoints of less than $d/200$ paths in \mathcal{P} and, additionally, $|\mathcal{P}| < r$, Router is required to find a path from a to b (this path has to be directed from a to b if G is a directed graph) which is edge-disjoint from all the other paths in \mathcal{P} . The found path is then added to \mathcal{P} . If such a path does not exist, Router loses the game.
- **REMOVE-PATH P :** A path $P \in \mathcal{P}$ is removed from \mathcal{P} .

It is important to note that each request has to be fulfilled by Router before seeing the next one. We say that Router wins the r -Routing game if she can satisfy any (infinite) series of requests.

Definition 1.1 (Expander graphs). We say that a n -vertex d -regular graph G is an (β, γ) -expander, for some $\beta, \gamma > 0$, if for every $S \subseteq V(G)$, $|S| \leq n/2$, we have

$$e_G(S) \leq \begin{cases} \gamma d |S| & \text{if } |S| \leq \beta n \\ d |S| / 3 & \text{if } \beta n < |S| \leq n/2. \end{cases}$$

A d -regular digraph D is a (β, γ) -expander if the $2d$ -regular (multi-)graph D_\emptyset obtained from D by ignoring the directions of the edges is a (β, γ) -expander.

The bound on the number of edges within large subsets is somewhat arbitrary and chosen for convenience, akin to the one used by Alon and Capalbo [2]. The following is our main theorem.

Theorem 1.2. *Let G be a d -regular (β, γ) -expander graph (directed or undirected) with n vertices, for $200 < d < n$, positive constants β and $\gamma < 1/1000$, and n large enough. Then Router has a strategy to win the r -Routing game for $r = \varepsilon nd / \log n$, for a constant $\varepsilon = \varepsilon(\beta, \gamma) > 0$. Moreover, Router can respond to each request in $O(n^3 d^3)$ time and each path in \mathcal{P} is of length $O(\log n)$.*

Let us remark that our main goal was to: (a) show that a strategy for winning an r -Routing game, for $r = \Theta(nd / \log n)$, exists, and (b) that it can be implemented in polynomial time. We leave it for future work to improve the time complexity. Moreover, the chosen constants could be improved with a more meticulous analysis; we made a conscious decision not to pursue this in order to maintain clarity and simplicity in our presentation.

When applied on a graph with stronger guarantees on its edge distribution, the algorithm underpinning Theorem 1.2 allows for an improved bound on r and the length of found paths. This is summarised in the following result.

Definition 1.3. We say a graph G is an (n, d, λ) -graph if it is a d -regular graph with n vertices and the second largest absolute eigenvalue of its adjacency matrix is at most λ .

It is well-known that, when λ is bounded away from d , (n, d, λ) -graphs have stronger expansion properties than the ones given in Definition 1.1 (see [3, Section 9.2]). A random d -regular graph with n vertices, for example, is with high probability an $(n, d, \Theta(\sqrt{d}))$ -graph (see [12]). Moreover, there exist explicit constructions of $(n, d, \Theta(\sqrt{d}))$ -graphs, which include the famous Ramanujan graphs [20].

Theorem 1.4. *Let G be an (n, d, λ) -graph for $200 < d < n$, where $\lambda < \varepsilon d$ for a small enough constant $\varepsilon > 0$. Then Router has a strategy to win the r -Routing game for $r = \alpha n d \frac{\log(d/\lambda)}{\log n}$, for some absolute constant $\alpha > 0$. Moreover, Router can respond to each request in $O(n^3 d^3)$ time and each path in \mathcal{P} is of length $O\left(\frac{\log n}{\log(d/\lambda)}\right)$.*

Finally, let us note that for our results we do not use the fact that a (di)graph G is d -regular in an essential way, and the same algorithm would work assuming that each degree is within $(1 \pm \varepsilon)d$. The choice to work with regular graphs is, again, purely for simplicity.

1.2 Notation

If G is a (directed) graph, we denote with $e_G(S)$ the number of edges with both endpoints in S , for a given $S \subseteq V(G)$. Let D be a directed graph, and let $S_1, S_2 \subseteq V(D)$ be subsets of its vertices. By $\text{out}_D(S_1, S_2)$ we denote the number of edges (u, v) with $u \in S_1$ and $v \in S_2$, and by $\text{in}_D(S_1, S_2)$ the number of edges (u, v) with $u \in S_2$ and $v \in S_1$. We write $\text{out}_D(S_1)$ to denote $\text{out}_D(S_1, V(D))$. We denote by $\text{Out}_D(S_1)$ the set of vertices $v \notin S_1$ such that there exists a vertex $u \in S_1$ with $(u, v) \in D$. Similarly, $\text{In}_D(S_1)$ denotes the set of vertices $v \notin S_1$ such that there exists an edge $(v, u) \in D$ for some $u \in S_1$. We write $\text{out}_D(v)$ and $\text{in}_D(v)$ for the out- and in- degree of v . A digraph is d -regular if it both the out- and in-degree of every vertex is d . By \overleftarrow{D} we denote the digraph obtained from D by reversing the direction of each of its edges.

2 Edge-Oracle: a data structure for accessing edges

The following lemma shows that one can choose, in an online manner, a subset of edges in a digraph D with a significant imbalance between in- and out-degrees of some vertices. Why this is useful will become apparent in the next section, where we prove Theorem 1.2. For now, let us remark that the lemma can be viewed as a generalization of a result of Aggarwal et al. [1, Theorem 2.2.7] which, in turn, is an algorithmic version of an earlier result by Feldman, Friedman and Pippenger [10].

Lemma 2.1. *Suppose D is a d -regular digraph with n vertices, for some $d \geq 10$, such that for every $S \subseteq V(D)$ of size $|S| \leq \beta n$ we have*

$$e_D(S) \leq \gamma |S| d,$$

for $\gamma \leq 1/50$. Then there exists a data structure, dubbed $\text{EDGE-ORACLE}(D)$, which maintains an initially empty set of edges $H \subseteq D$ and supports the following two requests:

- **ADD-EDGE v :** Given a vertex $v \in V(D)$ with $\text{out}_H(v) < \lfloor d/2 \rfloor$, the data structure returns an out-edge $e = (v, w) \in D \setminus H$ of v such that $\text{in}_H(w) < \lfloor d/5 \rfloor$, and adds it to H .

Note: this request is only allowed if $|H| < \beta d n / 120$.

- REMOVE-EDGE e : Remove a given edge $e \in H$ from H .

Both requests are handled using a deterministic algorithm. The time complexity of ADD-EDGE is $O(n^2d^2)$, and of REMOVE-EDGE is $O(nd)$.

It is worth noting that the proposed implementation of the data structure does not depend on γ and β , and it is up to the user of the data structure to respect the constraints under which ADD-EDGE can be invoked.

Proof of Lemma 2.1. The implementation of ADD-EDGE is given in Algorithm 1, and the implementation of REMOVE-EDGE in Algorithm 2. In Algorithm 1, we use the following definition of an *alternating walk*.

Definition 2.2 (Alternating walk). Given a subsets of edges $E_1, E_2 \subseteq E(D)$, we say that a sequence of vertices (v_1, v_2, \dots, v_k) , which are not necessarily distinct, forms an (E_1, E_2) -*alternating walk* W from v_1 to v_k if $(v_{2i+1}, v_{2i+2}) \in E_1$ for every $0 \leq i \leq k/2 - 1$, and $(v_{2i+1}, v_{2i}) \in E_2$ for every $1 \leq i < k/2$.

Note: the direction of the edges along W does not form a directed walk. In particular, they form what is known as an *anti-directed* walk.

Other than the subset of edges H , the data structure maintains a subset $B \subseteq D \setminus H$ of *buffer* edges and two sets of vertices – the *saturated* ones denoted by Sat which are sinks of many edges in $H \cup B$, and those with a low number of out-neighbours which are not saturated, denoted by Low . These sets are initially empty. In the description of the algorithm we use F to denote the set of edges $H \cup B$, always with respect to the current H and B .

Invariants. Let us, for now, assume that lines 2, 7, and 13 in ADD-EDGE are well-defined, that is, a desired edge or a walk exist whenever these lines are executed. More formally, we could say that whenever we reach one of these lines and a desired edge/walk does not exist, we abandon the current request and reset the internal state to how it was right before the request. Then, before handling each request (either ADD-EDGE or REMOVE-EDGE), the following properties hold:

- (P1) $\text{Sat} = \{v \in V(D) : \text{in}_F(v) \geq d/10\}$,
- (P2) $\text{Low} = \{v \in V(D) : \text{out}_D(v, \text{Sat}) \geq d/4\}$, and
- (P3) if $v \in \text{Low}$ then $\text{out}_F(v) = \lfloor d/2 \rfloor$.

Moreover, the following holds throughout each execution of ADD-EDGE:

- (C1) if $v \in \text{Sat}$ then $\text{in}_F(v) \geq d/10$,
- (C2) if $v \in \text{Low}$ then $\text{out}_D(v, \text{Sat}) \geq d/4$,
- (C3) if $v \notin \text{Low}$ then $\text{out}_B(v) = 0$, and
- (C4) $\text{out}_F(v) \leq \lfloor d/2 \rfloor$ and $\text{in}_F(v) \leq \lfloor d/5 \rfloor$ for every $v \in V(D)$.

These properties follow from the description of the algorithm and the fact that they are trivially satisfied before the first request. We now show a useful corollary of (C1)–(C4).

Claim A. *Throughout the execution of ADD-EDGE we have $|\text{Low}| < \beta n/12$.*

<p>Input : $v \in V(D)$ with $\text{out}_H(v) < \lfloor d/2 \rfloor$</p> <pre style="margin: 0; padding-left: 0;"> 1 if $v \in \text{Low}$ then 2 Choose any out-edge $e = (v, w) \in B$ of v 3 Add e to H 4 Remove e from B 5 return e 6 else 7 Choose any out-edge $e = (v, w) \in D \setminus F$ of v such that $w \notin \text{Sat}$ 8 Add e to H 9 if $\text{in}_F(w) \geq d/10$ then add w to Sat 10 while <i>there exists</i> $x \in V(D) \setminus \text{Low}$ <i>such that</i> $\text{out}_D(x, \text{Sat}) \geq d/4$ do 11 Add x to Low 12 while $\text{out}_F(x) < \lfloor d/2 \rfloor$ do 13 Find a $(D \setminus F, B)$-alternating walk W (see Definition 2.2) from x to some y with 14 $\text{in}_F(y) < \lfloor d/5 \rfloor$ 15 for $e \in P$ do 16 if $e \in D \setminus F$ then add e to B 17 else remove e from B 18 if $\text{in}_F(y) \geq d/10$ then add y to Sat 19 return e </pre>

Algorithm 1: ADD-EDGE(v)

Proof. Suppose, towards a contradiction, that there exists a series of requests which results in $|\text{Low}| \geq \lfloor \beta n / 12 \rfloor$ at some point. Consider the first moment when this happens, in which case we have that equality holds, and note that this has to be during an execution of ADD-EDGE, as during the execution of REMOVE-EDGE no vertices are added to Low. Then

$$\begin{aligned}
|\text{Sat}|d/10 &\stackrel{(C1)}{\leq} \text{in}_F(\text{Sat}) \leq e(F) = e(H) + e(B) < \beta dn/120 + e(B) \\
&\stackrel{(C3)}{\leq} \beta dn/120 + \text{out}_B(\text{Low}) \leq \beta dn/120 + |\text{Low}|d \leq (1 + 1/10)|\text{Low}|d,
\end{aligned}$$

where the last inequality follows from the assumption on the size of Low. Therefore,

$$|\text{Sat} \cup \text{Low}| \leq 12|\text{Low}| \leq \beta n.$$

On the one hand, by the assumption on D we have

$$\text{out}_D(\text{Low}, \text{Sat}) \leq e_D(\text{Low} \cup \text{Sat}) \leq |\text{Low} \cup \text{Sat}|\gamma d < 12\gamma|\text{Low}|d,$$

and on the other hand, by (C2),

$$\text{out}_D(\text{Low}, \text{Sat}) > d|\text{Low}|/4,$$

which leads to a contradiction for $\gamma < 1/48$. □

Input : edge $e = (v, w) \in H$

```

1 Remove  $e$  from  $H$ 
2 if  $v \in \text{Low}$  then add  $e$  to  $B$ 
3 else if  $w \in \text{Sat}$  and  $\text{in}_H(w) < d/10$  then
4   | Remove  $w$  from  $\text{Sat}$ 
5   | while there exists  $x \in \text{Low}$  with  $\text{out}_D(x, \text{Sat}) < d/4$  do
6   |   | Remove from  $B$  all the out-edges of  $x$ 
7   |   | Remove  $x$  from  $\text{Low}$ 
8   |   | Remove every  $y \in \text{Sat}$  with  $\text{in}_F(y) < d/10$  from  $\text{Sat}$ 

```

Algorithm 2: REMOVE-EDGE(e)

Correctness. We need to check that a returned edge $e = (v, w)$ has the required properties, that is, $\text{in}_H(w) < \lfloor d/5 \rfloor$. If $v \in \text{Low}$ (line 1), then this follows from (C4), $F = H \cup B$, and $e \in B$. Otherwise, it follows from $w \notin \text{Sat}$ (line 7) and (P1).

The algorithm ADD-EDGE is well-defined. As lines 2 and 7 happen before any changes to the internal state of the data structure, we can use (P1)–(P3) to prove these steps are well-defined. The existence of an edge $e = (v, w) \in B$ in line 2 follows from $\text{out}_H(v) < \lfloor d/2 \rfloor$ (the assumption of **add-edge**), (P3), and $F = H \cup B$. The existence of a required edge in line 7 follows from (C4), which implies $\text{out}_{D \setminus F}(v) \geq d - \lfloor d/2 \rfloor \geq d/2$ (recall D is d -out-regular), and (P2) together with $v \notin \text{Low}$. Showing that line 13 is well-defined is more complicated, and this is what we do next.

Claim B. *Line 13 is well-defined.*

Proof. Throughout the argument we use the subscript 0, as in Sat_0 for example, to denote the state of a set as it was just before the current invocation of ADD-EDGE. Consider some point during the execution of the algorithm when it has reached line 13. We need to show that, at that point, a desired alternating walk exists. Suppose, towards a contradiction, that this is not the case.

Let $\Delta = B \setminus B_0$ and note that $F \setminus F_0 = \Delta \cup \{e\}$. Let $Y \subseteq V(D) \setminus \text{Sat}_0$ be the set of all $y \in V(D) \setminus \text{Sat}_0$ such that there exists an $(D \setminus F, B)$ -alternating walk from x to y . As x was added to Low in line 11, we know $x \notin \text{Low}_0$, and hence by (P2) we have $\text{out}(v, \text{Sat}_0) < d/4$. Together with $\text{out}_F(x) < d/2$ (condition in line 12), this implies Y is non-empty because $\text{out}_{D \setminus F}(x, V(D) \setminus \text{Sat}_0) \geq d - d/2 - d/4$. Set $X = \text{In}_\Delta(Y)$ (as $x \in X$, it is non-empty as well). By (C3) and the definition of Δ , since Δ does not contain edges emanating from Low_0 , we have $X \subseteq \text{Low} \setminus \text{Low}_0$.

Observe that $\text{in}_F(y) \geq \lfloor d/5 \rfloor$ for every $y \in Y$, as otherwise a desired path exists. From

$$\text{in}_F(y) = \text{in}_{F_0}(y) + \text{in}_\Delta(y) + \mathbb{1}_{y=w}$$

and $y \notin \text{Sat}_0$, we conclude

$$\text{in}_\Delta(y) \geq \lfloor d/5 \rfloor - 1 - d/10 > d/11.$$

This implies

$$|Y|d/11 \leq \text{in}_\Delta(Y, X) \leq d|X|,$$

thus $|Y| \leq 11|X|$.

Next, note that for every $y \in \text{Out}_{D \setminus F}(X)$ there exists an $(D \setminus F, B)$ -alternating walk from x to

y , thus $\text{Out}_{D \setminus F}(X) \subseteq \text{Sat}_0 \cup Y$ as otherwise we get a contradiction with the choice of Y . On the one hand, from (C4), $X \cap \text{Low}_0 = \emptyset$, and (P2), we have

$$e_D(X \cup Y) \geq \text{out}_{D \setminus F}(X, Y) \geq \sum_{x \in X} \text{out}_{D \setminus F}(x) - \text{out}_D(x, \text{Sat}_0) \geq |X|d/4.$$

On the other hand, since $|Y| \leq 11|X|$ and $X \subseteq \text{Low}$, by Claim A we conclude $|X \cup Y| < \beta n$, thus by the assumption on D we have

$$e_D(X \cup Y) < 12\gamma|X|d,$$

which is a contradiction for $\gamma < 1/48$. \square

Complexity. Finding an alternating walk in line 14 can be done using BFS, which takes $O(nd)$ time. Within one call of ADD-EDGE this line is executed at most $O(nd)$ times, which gives $O(n^2d^2)$ and this dominates complexity coming from any other step of ADD-EDGE. With careful bookkeeping, REMOVE-EDGE can be implemented in $O(nd)$. \square

3 Disjoint paths via Edge-Oracle

In this section we prove Theorem 1.2.

Pre-processing G . Suppose that G is an undirected graph. If d is even, we find an Eulerian trail in G and orient the edges along this trail to obtain a $(d/2)$ -regular digraph D . Note that if G is a (β, γ) -expander, then so is D , by definition. If d is odd, we first remove a perfect matching (the existence follows from Tutte's theorem and the fact that G is d -edge-connected) and then repeat the previous procedure. Note that in this case we end up with a digraph which is not quite a (β, γ) -expander, but the upper bounds on $e_D(S)$ (see Definition 1.1) hold up to an additive factor of $O(d)$, which is negligible thus we will not concern ourselves with it.

By the previous discussion, we can assume now that G is an (β, γ) -expanding d -regular digraph. Next, we split G into three disjoint (spanning) subgraphs $G = G_1 \cup G_2 \cup G_3$, where G_1 and G_2 are d' -regular for $d' = \lfloor d/10 \rfloor$, and G_3 is consequently $(d - 2d')$ -regular. This can be done as follows: Create an auxiliary bipartite graph $B = (V_1 \cup V_2, E_B)$, where each V_i is a copy of $V(D)$ and there is an edge between $v \in V_1$ and $w \in V_2$ if $(v, w) \in D$. Then B is d -regular, hence its edges can be decomposed into d perfect matchings (in polynomial time). Take any d' perfect matchings and assign the corresponding edges to G_1 , and another d' (different) perfect matchings and assign the corresponding edges to G_2 .

Internal state. We maintain an initially empty sets of edges $H_1 \subseteq G_1, H_2 \subseteq \overleftarrow{G_2}$, and $H_3 \subseteq G_3$, and make use of two instances of EDGE-ORACLE data structure, $\text{out-oracle} = \text{EDGE-ORACLE}(G_1)$ and $\text{in-oracle} = \text{EDGE-ORACLE}(\overleftarrow{G_2})$, noting that both G_1 and $\overleftarrow{G_2}$ are d' -regular and for every $S \subseteq V(G_i)$ of size $|S| \leq \beta n, i \in \{1, 2\}$, we have

$$e_{G_i}(S) \leq \gamma|S|2d = 20\gamma|S|d' \leq \frac{1}{50}|S|d',$$

as $\gamma < \frac{1}{1000}$, so the conditions of Lemma 2.1 are satisfied for G_1 and $\overleftarrow{G_2}$. Recall that we can add and remove edges from H_i using the respective oracle as long as $|H_i| \leq cnd$ where $c = \beta/1200$.

Algorithm. We process $\text{FIND-PATH}(a, b)$ as follows:

1. Let $(H_a, V_a) = \text{ORACLE-BFS}(a, \text{out-oracle})$ and $(H_b, V_b) = \text{ORACLE-BFS}(b, \text{in-oracle})$.
2. Find a shortest path P' from V_a to V_b in $G_3 \setminus H_3$. Let $a' \in V_a$ be the starting point of P' , and $b' \in V_b$ the ending point.
3. Take a shortest path P_a from a to a' in H_a and a shortest path P_b from b to b' in H_b . The desired path P is then the concatenation of P_a , P' , and $\overleftarrow{P_b}$.
4. (**Internal update**) For every $e \in H_a \setminus E(P_a)$ call $\text{REMOVE-EDGE}(e)$ on out-oracle . Similarly, for every $e \in H_b \setminus E(P_b)$ call $\text{REMOVE-EDGE}(e)$ on in-oracle . Set $H_1 := H_1 \cup E(P_a)$, $H_2 := H_2 \cup E(P_b)$, and $H_3 := H_3 \cup E(P')$.

Input : a vertex $v \in V$
an instance **oracle** of **EDGE-ORACLE** data structure

```

1 q = QUEUE(v)
2  $V' = \{v\}$ 
3  $H' = \emptyset$ 
4 while q is not empty,  $|V'| \leq \beta n/5$ , and  $e(H') < cnd/2$  do
5    $u = \mathbf{q}.\text{DEQUEUE}()$ 
6   for  $i = 1, \dots, \lfloor d'/4 \rfloor$  do
7      $(u, w) = \mathbf{oracle}.\text{ADD-EDGE}(u)$ 
8     if  $w \notin V'$  then
9        $\mathbf{q}.\text{ENQUEUE}(w)$ 
10      Add  $w$  to  $V'$ 
11     Add the edge  $(u, w)$  to  $H'$ 
12 return  $(H', V')$ 

```

Algorithm 3: $\text{ORACLE-BFS}(v, \text{oracle})$

Processing $\text{REMOVE-PATH}(P)$ is significantly simpler:

1. For each $e \in H_1 \cap E(P)$ call $\text{REMOVE-EDGE}(e)$ on out-oracle . For each $e \in H_2 \cap E(\overleftarrow{P})$ call $\text{REMOVE-EDGE}(e)$ on in-oracle .
2. Set $H_1 := H_1 \setminus E(P)$, $H_2 := H_2 \setminus E(\overleftarrow{P})$, and $H_3 := H_3 \setminus E(P)$.

Note that ADD-EDGE and REMOVE-EDGE are called $O(nd)$ times, which results in $O(n^3d^3)$ and this dominates all other steps of the algorithm.

In the rest of the proof we show that the algorithm for finding new paths is well-defined. We denote by $\mathcal{P}_s(v)$ the number of paths in \mathcal{P} which start at v , and by $\mathcal{P}_e(v)$ the number of those which end at v . Recall that, by the rules of the game, we always have $|\mathcal{P}_s(v)|, |\mathcal{P}_e(v)| \leq d/200$, and that $|\mathcal{P}| \leq r = \varepsilon nd / \log n$, for a small enough $\varepsilon = \varepsilon(\gamma, \beta)$.

Invariants. Before each request the following properties hold:

- (P1) $\bigcup_{P \in \mathcal{P}} E(P) = H_1 \cup \overleftarrow{H_2} \cup H_3$,
- (P2) $|H_1|, |H_2| \leq |\mathcal{P}| \log n \leq cnd/2$ and $|H_3| \leq 300|\mathcal{P}|/\beta$,
- (P3) $\text{out}_{H_1}(v) \leq \text{in}_{H_1}(v) + \mathcal{P}_s(v)$ and $\text{out}_{H_2}(v) \leq \text{in}_{H_2}(v) + \mathcal{P}_e(v)$ for every $v \in V$,
- (P4) $\text{in}_{H_i}(v) \leq \lfloor d'/5 \rfloor$ for every $v \in V$ and $i \in \{1, 2\}$.

The property (P4) is responsible for bounding the number of times each vertex is used as an inner vertex of paths in \mathcal{P} , and is the heart of the proof!

The property (P1) follows from the description of the algorithm. To see that the property (P3) holds, note that every out-edge of a vertex in a path which belongs to H_1 is preceded by an in-edge of v in H_1 , unless it is the first vertex which is accounted by $\mathcal{P}_s(v)$. Note that we do not have equality as the out-edge and the in-edge might not belong to the same H_i . The second inequality in (P1) is obtained analogously, taking into account that the edges in H_2 are reversed compared to G . Since H_i for $i \in \{1, 2\}$ is obtained through ADD-EDGE requests to an oracle, property (P4) holds by the definition. It remains to show that (P2) holds. This is done in the following two claims.

Claim C. *Assuming that every call of ADD-EDGE in line 7 of Algorithm 3 is valid (that is, assumptions of ADD-EDGE are satisfied), after ORACLE-BFS finishes, every vertex in V' is at distance at most $\log n$ from v , using only edges in H' . Furthermore, $|V'| \geq \beta n/5$.*

Proof. Let T be the tree rooted at v consisting of edges (u, w) added to H' , such that w was not in V' when the edge (u, w) was returned by the oracle. Let ℓ be the depth of T , and note that vertices in \mathbf{q} are all contained in levels $\ell - 1$ and ℓ . Importantly, any vertex in V' which is not in \mathbf{q} has exactly $\lfloor d'/4 \rfloor > d'/5$ out-edges in H' . Denote by S_i the i -th level, and let us show that $S_{\ell-1}$ is at distance at most $\log n - 1$ from the root v , which will give that every vertex in $V' = V(T)$ is at distance at most $\log n$ from v . Let $S_{<i}$ be the union of the vertices in the first $i - 1$ levels. It suffices to show that $|S_i| \geq 2|S_{<i}|$ for all $i < \ell$.

Suppose that is not the case for some i , that is, $|S_i| < 2|S_{<i}|$. From $\text{Out}_{H'}(S_{<i}) \subseteq S_{<i} \cup S_i$ we get

$$e_G(S_{<i} \cup S_i) \geq e_{H'}(S_{<i} \cup S_i) \geq \text{out}_{H'}(S_{<i}) \geq |S_{<i}|d'/5. \quad (1)$$

As $|S_{<i} \cup S_i| < 3|S_{<i}|$ and $|V'| \leq \beta n$, by the assumption that G is an expander we have

$$e_G(S_{<i} \cup S_i) \leq \gamma \cdot 3|S_{<i}| \cdot 2d,$$

which contradicts (1) as $\gamma < 1/300$ and $d' = d/10$.

To finish, we have to show that the algorithm terminates when $|V'| \geq \beta n/5$. If that was not the case, then either H' contains at least $cnd/2$ edges, or \mathbf{q} is empty. If the former is true, then $H' \subseteq G$ is a digraph whose vertex set V' is of size at most $|V'| \leq \beta n/5$ and which has at least

$$e(H') \geq cnd/2 = \frac{\beta}{2400}nd > \gamma|V'| \cdot 2d$$

edges, again contradicting the assumption that G is an expander. Finally, it is easy to see that \mathbf{q} cannot be empty, as then again H' is a graph on at most $\beta n/5$ vertices with every vertex of degree at least $d'/5$ – thus a contradiction as in the previous case. \square

Claim D. Let $V_a, V_b \subseteq V$ be sets of size at least βn . Then for any subset of edges $R \subseteq G_3$ of size $|R| \leq \beta nd/50$, $G_3 \setminus R$ contains a path P from V_a to V_b of length at most $300/\beta$.

Proof. Let $G' = G_3 \setminus R$. Note that for every $S \subseteq V(G)$ of size $\beta n/5 \leq |S| \leq n/2$ we have

$$\text{out}_{G'}(S, V \setminus S) \geq (d - 2d')|S| - e_G(S) - |R| \geq 4d|S|/5 - 2d|S|/3 - |S|d/10 \geq |S|d/30,$$

thus

$$|\text{Out}_{G'}(S)| \geq |S|/30 \geq \beta n/150.$$

Therefore, there are more than $n/2$ vertices which can be reached from V_a within $150/\beta$ steps in G' . Note that this estimate can be improved, but for us this simple bound suffices.

The same argument shows that there are more than $n/2$ vertices from which a vertex in V_b can be reached within $150/\beta$ steps. Therefore, there exists a vertex which can be both reached from V_a in $150/\beta$ steps, and which can reach V_b in $150/\beta$ steps, implying a desired path from V_a to V_b exists. \square

With Claim C and Claim D at hand, we clearly see that in Step 3 of the main algorithm, the found paths P_a, P_b are always of length at most $\log n$, while P' is of length at most $300/\beta$ (and is disjoint of H_3 as we let $R = H_3$, and inductively we have $|R| = |H_3| \leq 300|\mathcal{P}|/\beta \leq \beta nd/50$). Since we only add the edges of those paths to H_1, H_2 and H_3 respectively, (P2) follows.

To complete the proof, we show that line 7 of Algorithm 3 is valid. Indeed, before each execution of Algorithm 3, by (P2) we have $|H_1|, |H_2| \leq cnd/2$, hence by the definition of the oracle, we can make at least $cnd/2$ new requests of the form $\text{ADD-EDGE}(v)$, as long as $\text{out}_{H_i}(v) \leq d'/2$. By (P3) and (P4), we further know that before we call Algorithm 3, we always have $\text{out}_{H_1}(v) \leq \text{in}_{H_1}(v) + \mathcal{P}_s(v) \leq d'/5 + d'/20 \leq d'/4$. Therefore, since by the description of the algorithm we request a vertex v at most $\lceil d'/4 \rceil$ times, we have that during the execution of the algorithm we always have $\text{out}_{H_i}(v) \leq d'/2$, which finishes the proof.

3.1 Sketch of proof for Theorem 1.4

Recall that by the Expander mixing lemma, in every (n, d, λ) -graph G it holds that every set of vertices of size αn has average degree at most $\alpha d + \lambda$ (see Section 9.2 in [3]). Hence G is a (β, γ) -expander for $\beta = 1/10^4$ and $\gamma = 1/5000$, for say $\lambda < d/10^4$.

The proof of Theorem 1.4 is essentially the same as the proof of Theorem 1.2 presented in this section, up to Claim C, where we can use the stronger expansion properties of (n, d, λ) -graphs to show that every vertex in V' is in fact at distance at most $O(\frac{\log n}{\log(d/\lambda)})$ from v . Indeed, even after deleting a constant fraction of edges at each vertex in an (n, d, λ) -graph, each set up to a certain linear size expands by a factor of $\Theta(\frac{d^2}{\lambda^2})$ (which again follows from the Expander mixing lemma, see Lemma 3.6 in [9]). Consequently, we have $|S_i| \geq \Theta(\frac{d^2}{\lambda^2})|S_{<i}|$, which evidently gives the required distance between v and every vertex in V' .

References

- [1] A. Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan. Efficient routing in optical networks. *Journal of the ACM (JACM)*, 43(6):973–1001, 1996.
- [2] N. Alon and M. Capalbo. Finding disjoint paths in expanders deterministically and online. In *Proc. of the Symposium on Foundations of Computer Science (FOCS)*, pages 518–524, 2007.
- [3] N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 2016.
- [4] T. Bohman and A. Frieze. Arc-disjoint paths in expander digraphs. *SIAM Journal on Computing (SICOMP)*, 32(2):326–344, 2003.
- [5] A. Broder, A. Frieze, S. Suen, and E. Upfal. Optimal construction of edge-disjoint paths in random graphs. *SIAM Journal on Computing (SICOMP)*, 28(2):541–573, 1998.
- [6] A. Broder, A. Frieze, and E. Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM Journal on Computing (SICOMP)*, 23(5):976–989, 1994.
- [7] A. Broder, A. Frieze, and E. Upfal. Static and dynamic path selection on expander graphs (preliminary version): A random walk approach. In *Proc. of the Symposium on Theory of Computing (STOC)*, page 531–539, 1997.
- [8] A. Z. Broder, A. Frieze, S. Suen, and E. Upfal. An efficient algorithm for the vertex-disjoint paths problem in random graphs. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 261–268. 1996.
- [9] N. Draganić, M. Krivelevich, and R. Nenadov. Rolling backwards can move you forward: on embedding problems in sparse expanders. *Transactions of the American Mathematical Society*, 375(07):5195–5216, 2022.
- [10] P. Feldman, J. Friedman, and N. Pippenger. Wide-sense nonblocking networks. *SIAM Journal on Discrete Mathematics*, 1(2):158–173, 1988.
- [11] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.
- [12] J. Friedman. On the second eigenvalue and random walks in random d-regular graphs. *Combinatorica*, 11(4):331–362, 1991.
- [13] A. Frieze. Edge-disjoint paths in expander graphs. *SIAM Journal on Computing (SICOMP)*, 30(6):1790–1801, 2001.
- [14] A. Frieze and L. Zhao. Optimal construction of edge-disjoint paths in random regular graphs. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 346–355. 1999.
- [15] D. S. Hochbaum. An exact sublinear algorithm for the max-flow, vertex disjoint paths and communication problems on random graphs. *Operations Research*, 40(5):923–935, 1992.
- [16] T. Leighton, C.-J. Lu, S. Rao, and A. Srinivasan. New algorithmic aspects of the Local Lemma with applications to routing and partitioning. *SIAM Journal on Computing (SICOMP)*, 31(2):626–641, 2001.
- [17] T. Leighton and S. Rao. Circuit switching: a multicommodity flow based approach. In *Proc. of a Workshop on Randomized Parallel Computing*, 1996.
- [18] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999.
- [19] T. Leighton, S. Rao, and A. Srinivasan. Multicommodity flow and circuit switching. In *Proc. of the Hawaii International Conference on System Sciences*, volume 7, pages 459–465, 1998.

- [20] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
- [21] D. Peleg and E. Upfal. Constructing disjoint paths on expander graphs. In *Proc. of Symposium on Theory of Computing (STOC)*, pages 264–273, 1987.
- [22] N. Robertson and P. Seymour. Graph Minors. XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.
- [23] E. Shamir and E. Upfal. A fast parallel construction of disjoint paths in networks. In *Selected Papers of the International Conference on "Foundations of Computation Theory" on Topics in the Theory of Computation*, page 141–153, 1985.