Finding a Maximum Matching in a Sparse Random Graph in $O(n)$ Expected Time

PRASAD CHEBOLU, ALAN FRIEZE, AND PÁLL MELSTED

Carnegie Mellon University, Pittsburgh, Pennsylvania

Abstract. We present a linear expected time algorithm for finding maximum cardinality matchings in sparse random graphs. This is optimal and improves on previous results by a logarithmic factor.

Categories and Subject Descriptors: G.2.1 [**Discrete Mathematics**]: Combinatorics; G.3 [**Probability and Statistics**]

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Random graphs, matching

ACM Reference Format:

Chebolu, P., Frieze, A., and Melsted, P. 2010. Finding a maximum matching in a sparse random graph in $O(n)$ expected time. J. ACM 57, 4, Article 24 (April 2010), 27 pages.
DOI = 10.1145/1734213.1734218 http://doi.acm.org/10.1145/1734213.1734218

1. *Introduction*

A matching M in a graph $G = (V, E)$ is a set of vertex disjoint edges. The problem of computing a matching of maximum size is central to the theory of algorithms and has been subject to intense study. Edmond's landmark paper [Edmonds 1965] gave the first polynomial time algorithm for the problem. Micali and Vazirani [1980] reduced the running time to $O(mn^{1/2})$ where $n = |V|$ and $m = |E|$. These are worst-case results. In terms of average case results, we have Motwani [1994] and Bast et al. [2006] who have algorithms that run in $O(m \log n)$ expected time on the random graph $G_{n,m}$, in which each graph with vertex set $[n]$ and m edges is equally likely.

One natural approach to finding a maximum matching is to use a simple algorithm to find an initial matching and then augment it. This will not improve execution time

A. Frieze was supported by National Science Foundation (NSF) Grant CCF-0502793.

Author's address: Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213, Contact authors's e-mail: alan@random.math.cmu.edu.

DOI 10.1145/1734213.1734218 http://doi.acm.org/10.1145/1734213.1734218

in the worst-case, but as we will show, it can be used to obtain an $O(n)$ expected time algorithm for graphs with constant average degree ($O(m)$ in general). For a simple algorithm we go to the seminal paper of Karp and Sipser [1981]. They describe a simple greedy algorithm and show that, with high probability, it will in linear time produce a matching that is within $o(n)$ of the maximum. Here "with high probability" is short hand for "with probability $1 - o(n^{-1/2})$". Aronson et al. [1998] proved that with high probability, the Karp-Sipser algorithm is off from the maximum by at most $\tilde{O}(n^{1/5})$ (this notation hides poly-logaritmic factors). In this article, we show that with high probability we can take the output of the Karp-Sipser algorithm and augment it in $o(n)$ time to find a maximum cardinality matching. Our failure probability will therefore be $o(1/\log n)$ and so we get a linear expected time algorithm if we back it up with the algorithm from Bast et al. [2006]. We will define a randomized algorithm Match and prove

THEOREM 1.1. *Let $m = cn/2$ where c is a sufficiently large constant. Let $G = G_{n,m}$. Then, the algorithm Match finds a maximum matching in G in $O(n)$ expected time.*

The expectation here is over the choice of input and the random choices made by the algorithm.

1.1. THE KARP-SIPSER ALGORITHM. This is a simple greedy algorithm. It adds a matching edge as follows: If the graph G has a vertex of degree 1, then it chooses one such vertex v at random and adds the unique edge (u, v) to the matching it has found so far and deletes the vertices u, v from G and continues. If the graph has minimum degree at least 2, then it picks a random edge (u, v), adds this to the matching and deletes u, v from G and continues. Vertices of degree zero are also considered to be deleted from G. The algorithm stops when G has no vertices.

We identify two phases in the execution of the Karp-Sipser algorithm. Phase one starts at the beginning and finishes when the current graph has minimum degree at least two. We note that, if M_1 is the set of edges chosen in Phase 1, then there is some maximum cardinality matching that contains M_1, that is no mistakes have been made so far.

Algorithm 1 below is a pseudo-code description.

A vertex is said to be *matched* by a matching M if it is incident to an edge of M, otherwise it is *unmatched*.

Let M_1 denote the matching produced by Phase 1 of the Karp-Sipser algorithm. Let the graph at the end of Phase 1, beginning of Phase 2 be denoted by Γ. Γ will comprise (i) a collection $C_1, C_2, \ldots, C_k$ of vertex disjoint cycles and (ii) a graph Γ_1 where each component has minimum degree two, but is not a cycle. At the end of the Karp-Sipser algorithm there will be some unmatched vertices. We partition them into $I_1 \cup I_2$. Here, I_1 consists of (i) vertices that are not matched in Phase 1 and are already isolated by the end of Phase 1 and so are not in Γ, (ii) one vertex from each odd cycle in the collection $C_1, C_2, \ldots, C_k$. Thus, I_2 consists of the vertices of Γ_1 that are not matched by M_{KS}. This partition is significant in that to obtain a maximum matching, it is enough to augment the matching M_1 so that as many vertices in I_2 as possible are matched.

We now consider the output of the Karp-Sipser algorithm to be amended to include I_2.

Algorithm 1. Karp-Sipser Algorithm

```
1: Procedure KSGreedy(G)
2:    M_KS ← ∅
3:    while G ≠ ∅ do
4:       if G has vertices of degree one then
5:          Select a vertex v uniformly at random from the set of vertices of degree one.
6:          Let (v, u) be the edge incident to v.
7:       else
8:          Select an edge (v, u) uniformly at random.
9:       end if
10:      M_KS ← M_KS ∪ (v, u).
11:      G ← G \ {v, u}.
12:      Remove any isolated vertices of G.
13:   end while
14:    return M_KS
15: end procedure
```

As shown in Karp and Sipser [1981], all but $o(n)$ vertices of Γ are matched by the Karp-Sipser algorithm when G is the random graph $G_{n,m}$. This result was improved in Aronson et al. [1998] to show in fact that with high probability all but $\tilde{O}(n^{1/5})$ vertices of Γ are matched. It was shown in Frieze and Pittel [2004] that with high probability a maximum cardinality matching of Γ covers every vertex except one for each isolated odd cycle and one further vertex if after removing odd cycles, the number of vertices remaining is odd. (This is an existence result, nonalgorithmic). The number of vertices on isolated cycles is $O(\log n)$ with high probability but this fact will not be needed and therefore not proven. After running the Karp-Sipser algorithm and dealing with the isolated cycles, our task will with high probability be to match together $\tilde{O}(n^{1/5})$ unmatched vertices.

1.2. Outline Description of Match. We will take the output of the Karp-Sipser algorithm and then take the unmatched vertices $I_2 = \{v_1, v_2, \ldots, v_l\}$ in pairs $v_{2i-1}, v_{2i}, i = 1, 2, \ldots, \lfloor l/2 \rfloor$. We then search for an augmenting path from v_{2i-1} to v_{2i} for $i = 1, 2, \ldots, \lfloor l/2 \rfloor$. We will refer to this as Phase 3 of our algorithm. We will show that this can be done in $o(n)$ time with high probability. Phase 3 will make use of all of the edges of the graph Γ. The reader will be aware that the Karp-Sipser algorithm has conditioned the edges of this graph. We will show however that we can find a large set of edges A that have an easily described conditional distribution. This distribution will be simple enough that we can make use of A to show that we succeed with high probability. Intuitively, we can do this because the Karp-Sipser algorithm only "looks" at a small number of edges and discards most of the edges incident with the pair u, v chosen at each step. Dealing with conditioning is a central problem in Probabilistic Analysis. Often times it is achieved by the use of concentration. Here, the problem is more subtle. Note that one cannot simply run the Karp-Sipser algorithm on a random subgraph $G_{n,m_1} \subseteq G_{n,m}$ and then use the $m - m_1$ random edges. This is because in this case, Phase 1, when run on the subgraph will leave extra unmatched vertices.

Algorithm 2 below is a pseudo-code description of Match. Before giving this, let us describe match in words. The input is a graph G.

Algorithm 2. Algorithm Match

```
1: Procedure MAIN(G)
2:   (M_KS, I_2) ← KSGreedy(G); M* ← M_KS
3:   Let I_2 = {v_1, v_2, ..., v_l}.
4:   for j = 1 to ⌊l/2⌋ do
5:     M* ←AugmentPath(M*, v_{2j-1}, v_{2j})
6:     if AugmentPath(M*, v_{2j-1}, v_{2j}) returns failed then
7:       Abort and execute the algorithm of Bast et al. [2006].
8:     end if
9:   end for
10:  return M*
11: end procedure
```

(1) Run Phase 1 of the Karp-Sipser algorithm on G.

(2) As above let Γ denote G at the end of Phase 1. Γ is comprised of (i) a collection $C_1, C_2, \ldots, C_k$ of vertex disjoint cycles and (ii) a graph Γ_1 where each component has minimum degree two, but is not a cycle.

(3) Do a breadth first search of Γ and use it to identify and mark the vertices of $C_1, C_2, \ldots, C_k$.

(4) Run Phase 2 of the Karp-Sipser algorithm on Γ. Note that the Karp-Sipser algorithm deals correctly with $C_1, C_2, \ldots, C_k$. Let $I_2 = \{v_1, v_2, \ldots, v_l\}$ be the vertices of Γ_1 that remain unmatched at the end of Phase 2. Unmatched vertices in I_1 will remain unmatched and the marking done in 3. will allow us to distinguish I_2.

(5) For $i = 1, 2, \ldots, \lfloor l/2 \rfloor$, look for an augmenting path from v_{2i-1} to v_{2i} and use it to augment the current matching. Formally, this entails running a procedure AugmentPath.

(6) If there is a $i \leq \lfloor l/2 \rfloor$ such that we fail to find an augmenting path, declare failure and run the algorithm of Bast et al. [2006] from scratch.

Here is an alternative pseudo-code description of the above.

Procedure AugmentPath(M^*, u, v) is described in Section 2. Its analysis constitutes the main subject matter of this article.

Given an unmatched vertex u, an *augmenting tree* T_u will be a tree of even depth rooted at u such that for $k \geq 0$, edges between vertices at depth $2k$ and $2k+1$ are not matching edges and edges between vertices at depth $2k+1$ and $2k+2$ are matching edges. We refer to the vertices at levels $2k$ as even vertices of the tree and vertices at level $2k+1$ as odd vertices for $k \geq 1$. We let *Odd(T)* and *Even(T)* denote the set of even and odd vertices respectively. In an augmenting tree, every vertex in *Odd(T)* has exactly one child in *Even(T)*, and so in particular $|Odd(T)| = |Even(T)|$. We define the (disjoint) neighborhood $N(S)$ of a set of vertices S by

$$N(S) = \{w \notin S : \exists v \in S \text{ such that } (v, w) \in E(G)\}.$$

In which case, we have by $N(Even(T)) \supset Odd(T)$.

We refer to the leaves of T_u as the *front* of the tree and denote them by F_u or $\Phi(T_u)$.

Given a pair of unmatched vertices u, v, AugmentPath will grow trees T_u, T_v until they are both "large" and then with high probability we will be able to connect a vertex x at the *front* (leaves) of T_u, to a vertex y at the front of T_v by a path x, a, b, y where (a, b) is a matching edge. This produces an augmenting path from u to v. (There are some less likely ways resulting in an augmenting path and these are also taken account of below).

Analyzing the growth of these trees is the subject of Section 2. It requires detailed knowledge of the distribution of the random graph Γ. This is discussed in Section 2.3.1.

To prove that we can with high probability find a quadruple x, a, b, y for T_u, T_v requires knowledge of the conditioning imposed by Karp-Sipser algorithm on the edges not in M_{KS}. This is the subject of Section 3.

In Section 4, we use what we prove in Section 3 to show that for a given pair of unmatched vertices u, v and corresponding *augmenting trees* T_u, T_v, we will with high probability be able to complete an augmenting path.

2. *Augmenting Path Algorithm*

As already explained, an execution of AugmentPath searches for an augmenting path between two unmatched vertices of I_2. If such a path is found, the matching is augmented, if not the algorithm returns Failure and we resort to the algorithm of Bast et al. [2006]. Match executes AugmentPath until either there is at most one unmatched vertex of I_2 or there is a failure. Clearly, if the algorithm does not fail then it finds a maximum cardinality matching.

We now define some more terms associated with the tree growing process.

A *blossom* rooted at v is an cycle of odd length where the edges on the path starting and ending at v alternate between matching and non-matching edges. An edge (x, y) that creates a blossom when added to T_u is called a *blossom edge* of T_u.

Given two augmenting trees T_u, T_v, rooted at u, v respectively, a *hit edge* is an edge (x, y) such that x is an even node in T_u and y is an odd node in T_v. Note that given a hit edge (x, y) the subtree of T_v rooted at y can be taken from T_v and added to T_u, by removing the edge from y to its parent node in T_v (see Figure 2).

The trees T_u, T_v are grown until we either find an augmenting path or the trees cannot be grown further. For each of u, v, we maintain a list of blossom edges and hit edges encountered so far.

The basic operation of our algorithm is to *explore* a vertex on the front. Vertices are explored in the order in which they are added to the tree. Throughout the algorithm, we will keep track of which vertices we have explored in Phase 3, labeling them as *exposed*. This is mainly to keep track of which vertices have "no randomness" left because we have seen all the vertices they are adjacent to.

To explore x on the front F_u of T_u, we look at each y incident to x such that (x, y) is not a matching edge:

T1. If y belongs to neither of the trees and is matched, add it to T_u along with its matching edge (y, y')

T2. If y is unmatched, we have an augmenting path from u to y

T3. If y is an even vertex of T_v, then the path from u to x in T_u, with the edge (x, y) and the path from y to v in T_v forms an augmenting path

T4. If y is an odd vertex of T_v, then (x, y) is a hit edge, append it to the list of hit edges for u.

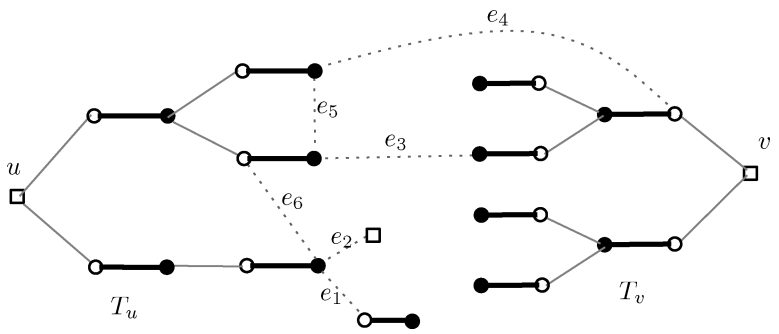

FIG. 1. The trees T_u and T_v are shown with bold edges. The edges e_i correspond to cases i in the algorithm for $i = 1, \ldots, 6$.

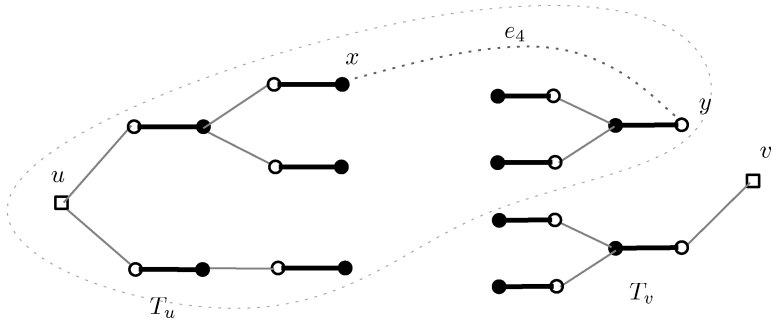

FIG. 2. The trees T_u and T_v after using the hit edge e_4.

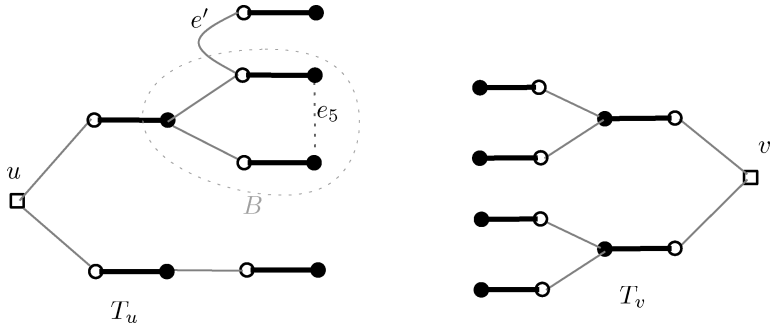

FIG. 3. The trees T_u and T_v after using the blossom edge e_5. Note that the blossom B is contracted and the edge e' becomes a part of the new tree.

T5. If y is an even vertex of T_u, then (x, y) along with the paths from x, y to their lowest common ancestor in T_u form a blossom, append (x, y) to the list of blossom edges for u.

T6. If y is an odd vertex of T_u, we do nothing.

After examining all edges incident to x, we label it as exposed. Examples of the six cases for the edge (x, y) are shown in Figure 1 and examples for hit edges and blossoms are shown in Figures 2 and 3.

The description of AugmentPath uses a small constant ε, any ε small enough will work, but one concrete value is given in (1). In each execution of AugmentPath,

the growth of both trees is split into three stages. Within a stage, the growth of each tree is divided into *growth spurts*. In a spurt, one of the trees T_u, T_v will be selected and all of the *eligble* vertices of its front will be explored. A tree starts in *Stage 1* until it reaches a size of $n^{.6-\varepsilon}$, after that *Stage 2* starts. If a tree is in Stage 1 then all vertices on its front are eligible. Although such a tree is "large enough", we cannot be sure that it has enough unexposed vertices on the front. This is why we need Stage 2. During this stage only already exposed vertices on the front are eligible. Thus, we do not explore the unexposed vertices at the front in this stage and it lasts until an $n^{-\varepsilon}$ fraction of the vertices on its front are unexposed. This means that we create at most $n^{.6-\varepsilon}$ new exposed vertices per tree in Phases 1 and 2. When the tree, T_u say, has the required fraction of unexposed vertices on the front, it is placed in *Stage 3* and we select a set S_u of size $n^{.6-2\varepsilon}$ at random from the set of unexposed vertices from the front. If both trees reach Stage 3, then we try to connect S_u, S_v by a path (x, a, b, y) of length three whose middle edge is a matching edge. If we succeed, then we will have created an augmenting path from u to v. Otherwise, we declare failure. Note that we expose at most $n^{.6-\varepsilon} + n^{.6-2\varepsilon}$ new currently unexposed vertices per tree per execution of AugmentPath.

During a spurt, we explore the front of the selected tree and first deal with cases of **T1,T2**. If exploring a vertex finds an augmenting path (**T2**), then the current execution of AugmentPath is finished. If this does not happen and there are no instances of **T1**, then we inspect first the list of blossom edges and see if we can grow the tree using them. For each blossom found, contract the blossom into a supernode and consider this supernode to be at the front of the tree and try to grow by exploring edges available to the supernode. If there is no growth through blossom edges, then we inspect the list of hit edges. If (x, y) is a hit edge in **T4** with $x \in F_u$, then we can grow T_u by adding to it the edge (x, y) plus the subtree of T_v below y. If the tree still doesn't grow then we exit the execution of AugmentPath and *fail*.

The restriction in Stage 2 to only exploring already exposed vertices is a little *un-natural* and one would hope to be able to avoid it. We do it here so that we can properly bound the amount of work done over-all. Without the restriction, we could conceivably do a super-linear amount of work by unnecessarily exposing too many vertices.

To determine which of the trees to grow, we look at the stages and sizes of the trees. If both trees are in Stages 1 or 2, we grow the smaller of the two. When only one tree is in Stage 1, we grow that tree. When one tree is in Stage 2 and the other is in Stage 3, we grow the one in Stage 2. When both are in Stage 3, we finish via a simple search for an augmenting path.

As part of our argument, we will show that with high probability we only expose $o(n^{.8})$ vertices altogether. This will ensure that we reach Stage 3, unless **T2** or **T3** occurs. Otherwise we will have run out of exposed vertices. Typically, we expect to find an augmenting path through **T3** or we find that both trees reach Stage 3. Then at the first time both are in Stage 3, there will be many opportunities for the unexposed vertices to find a path x, a, b, y where $x \in S_u,\ y \in S_v, (a, b) \in M$.

2.1. DATA STRUCTURES. Aiming for a linear running time for Match means that we have to be careful in our use of data structures. We describe what data structures are needed for Karp-Sipser and AugmentPath to ensure a linear running time.

For Karp-Sipser, given in Algorithm 1, we need to efficiently (i) access a list of neighbors of a vertex, (ii) delete a vertex from the graph, (iii) select an edge at random, and (iv) select a vertex of degree one at random.

This can be done by storing the vertices of G as array of vertices V. Each vertex $V[i]$, stores its degree, a linked list, L, of edge-structures, and a pointer to its position in the array of vertices of degree one if it has one. Each edge-structure stores a pair of vertex indices and a pointer to the edge-representative-structure for (u, v) described below.

We also store a list E of edge-representative-structures. Each edge-representative-structure corresponding to an edge (u, v) stores the vertex indices u and v and one pointer per vertex pointing to the position of the (u, v) in the edge-structure list of $V[u]$ and $V[v]$, respectively.

Finally, we keep track of which vertices are of degree one as an array $V1$ of vertex indices.

Accessing the neighborhood of any vertex u can be done in time $O(\deg_G(u))$. Selecting an edge at random can be done by selecting an index i at random from 1 to $|E|$ and accessing $E[i]$, similarly for selecting a vertex of degree one at random. Deleting a vertex u involves deleting all edges (u, v) for all neighbors of v, this can also be achieved in time $O(\deg_G(u))$. This shows that the total running time of Karp-Sipser is $O(\sum_{u\in G} \deg_G(u)) = O(m) = O(n)$.

When deleting a vertex u we repeat the following for all neighbours v of u. First, swap the edge-representative-structure with the last element of E, repairing the pointers as necessary and decreasing the size of E by one. Then, the two edge nodes of $V[u]$ and $V[v]$ are removed from their lists. After deleting all edges incident with u, we swap $V[i]$ with the last element of V, repairing all pointers of the elements swapped and decreasing the index of V by one. Finally, if u is in $V1$, we delete it from $V1$ by swapping the element it points to out with the last element of $V1$ and decreasing its size by one.

The matching computed by Karp-Sipser is stored as an array M indexed by vertex indicies containing the pair of vertices in the edge, unmatched vertices have an empty pair.

For the extension to Karp-Sipser, we note that we can detect if a vertex u is on an isolated cycle by running a breadth-first search algorithm and counting the number of vertices and edges seen. If the two are equal, then the connected component containing u is an isolated cycle.

Given a list of all vertices on isolated cycles allows us to find I_2 in time $O(n)$.

For AugmentPath, we allow ourselves more room in keeping track of things. We need to keep track of the trees T_u and T_v and determine whether vertex x belongs to either of them. We also need to keep track of blossom and hit edges and supernodes formed by contracting blossoms. Each of these tasks can be done with a suitable sophisticated data structure, such as a balanced tree, with a running time of $O(\log n)$. Our analysis will show that the number of operations executed in Phase 3 is less than $n^{1-\varepsilon}$ and this saving of $n^{-\varepsilon}$ will easily account for any $\log n$ factors.

2.2. TREE EXPANSION. In this section, we will show that with high probability both of our augmenting trees will reach Stage 3, unless their growth stops prematurely because an augmenting path has been found. Later, we will argue that when both of the trees are in Stage 3, then their fronts contain many unexposed vertices

and then with high probability an augmenting path can be found via a short path connecting these fronts.

In particular, we will show that if the absolute constant α_1 is sufficiently large and $0 < \alpha_2 < 1/2$ and $c = 2m/n \geq c_0$ and c_0 is sufficiently large then the following four lemmas hold. The proofs of the first three lemmas are heavy on computation. We have moved them to a separate section.

The algorithm uses the constant ε and in addition the proof uses a constant γ:

$$\varepsilon = \frac{1}{200} \quad \text{and} \quad \gamma = \frac{1}{20}. \tag{1}$$

LEMMA 2.1. *The following will hold with probability* $1 - O(\frac{1}{n^2})$. *For all matchings* M *of* Γ *and all augmenting trees* T *with* $c^{-1}\alpha_1 \log n \leq |T| \leq n^{.99}$, T *can be grown to a new tree* T' *for which* $\Phi(T') \in [\frac{9c}{10}|T|, \frac{11c}{10}|T|]$.

LEMMA 2.2. *With probability* $1 - \tilde{O}(\frac{1}{n^{1-\alpha_2}})$, Γ *does not contain two cycles of length* a *and* b, *at distance* d *apart for any* a, b, d *such that* $a + b + d \leq \alpha_2 \log_c n$

LEMMA 2.3. *With probability* $1 - O(\frac{1}{n^2})$ Γ, *does not contain a set* $S \subseteq V(\Gamma)$ *with* $\log \log n \leq |S| \leq n^{.99}$ *that has more than* $(1 + \epsilon)|S|$ *edges.*

LEMMA 2.4. *Let* Γ *be such that the low probability events described in Lemmas* 2.1, 2.2, and 2.3 *do not hold. Let* u, v *be a pair of isolated vertices for which an augmenting path is sought. Let* M *be an arbitrary matching of* Γ. *Suppose that we do not find an augmenting path as in Steps T2,T3. Then, the following hold:*

(a) Both T_u *and* T_v *will reach Stage* 3.

(b) The number of growth spurts undergone by either tree in Stage 2 *is at most* $\log_{3c/4} n$.

PROOF OF LEMMA 2.4. Suppose that (a) does not hold and that T_u is selected for a spurt and the execution of AugmentPath fails. Neither tree can grow in size via T1–T6. We split the analysis into two cases depending on the size of T_u.

Case 1. $|T_u| \leq \frac{1}{10}\alpha_2 \log_c n$.

Note that, when we explore vertices on the front, we have seen one matching edge incident to them and that they have degree at least two. Furthermore, u has degree at least two, so F_u starts out with at least two vertices. The only way the tree cannot grow is if we are in cases **T4, T5** or **T6**.

All of the nonmatching edges incident with the front F_u of T_u go to within T_u. Observe that there are no hit edges leaving F_u, for otherwise T_u would grow and AugmentPath would not fail. Observe next that $|F_u|$ can only be reduced either

R1. When a nonmatching edge (x, y) with $x \in F_u$ has $y \in T_u$, thus creating a cycle inside T_u, or

R2. A piece of T_u is removed by a hit edge from T_v.

Remark 2.5. We will see in the analysis in Case 2 that no hit edges are needed by T_v when $|T_v| > \frac{1}{10}\alpha_2 \log_c n$.

If $|F_u| \geq 2$, then either we have two edges going from F_u to within itself that form two cycles, a violation of Lemma 2.2, or $|F_u| = 2$ and we have one edge that forms a blossom. If the algorithm has failed after the contraction of the blossom,

then T_u plus the blossom edge then there are several cases: (i) T_u plus the blossom edge is an isolated odd cycle (contradicting the fact that $u \in I_2$); (ii) The vertices of T_u already contain a short cycle through a case of **T6**, which with the blossom contradicts Lemma 2.2; (iii) The tree T_u had a branch which was grafted to T_v via a hit edge. But, by Remark 2. 5, we see that at this time $|T_v| \leq \frac{1}{10}\alpha_2 \log_c n$. So either the vertices of T_v contain a small cycle, which with the blossom contradicts Lemma 2.2 or there are at least two hit edges, which again contradicts Lemma 2.2.

If $F_u = \{x\}$ is of size one, then either x has degree at least two or it is a supernode from a previously contracted blossom. If u has degree at least two, we must already have encountered **R1** or **R2** above. In case **R1**, we have found one small cycle C with vertex set contained in $V(T_u) \setminus F_u$. Since there are no hit edges and T_u cannot grow, the nonmatching edge incident with F_u has both endpoints in T_u and we therefore have two cycles in $V(T_u)$, which violates the conditions of Lemma 2.2, a contradiction. (The reader might be concerned that we could have found an isolated odd cycle. However, $u, v \in I_2$ and so neither of them are on such a cycle.)

Now consider the case where $F_u = \{x\}$ is of size one and the reduction in front size was caused by a hit edge (x, y) from T_v. By the argument above, we know that when T_v used this hit edge we must have had either (i) $|F_v| = 1$ or (ii) $|T_v| > \frac{1}{10}\alpha_2 \log_c n$. We see from Remark 2.5 that when T_v used (x, y) to grow its size was at most $\frac{1}{10}\alpha_2 \log_c n$ and either its vertices contained a cycle C or there were at least two hit edges from T_v to T_u which again leads to a small cycle. Now, consider the nonmatching edge (x, y) incident with F_u where $x \in T_u$. y cannot be in T_v since there are no hit edges from T_u at this time and it cannot be in T_u because of Lemma 2.2, contradiction.

If the only node on the front F_u is a supernode with degree one, then either **R1** or **R2** have happened previously since $|F_u| \geq 2$ initially. If **R1** occurred, this would imply that $V(T_u)$ has two cycles. If **R2** occurred, then by the same argument as above both trees must have been of size $\frac{1}{10}\alpha_2 \log_c n$ at the time. Therefore, there exists a subgraph contained in $V(T_u \cup T_v)$, of size at most $\frac{2}{10}\alpha_2$, with two cycles, contradicting Lemma 2.2.

Case 2. $|T_u| \geq \frac{1}{10}\alpha_2 \log_c n$.

We now show if T_u is selected for a spurt then it will grow a new front of size at least $\frac{4c}{5}(1 - n^{-\varepsilon})|T_u|$. We can assume that $c \geq 10\alpha_1/\alpha_2$. By Lemma 2.1, we know that the size of the new front is at least $\frac{9c}{10}(1 - n^{-\varepsilon})|T_u|$ when T_u is grown without considering T_v. The factor $(1 - n^{-\varepsilon})$ accounts for only growing from exposed vertices. Some of these vertices might be odd vertices of T_v and we must account for this. It is enough to show that the front of T_u cannot be adjacent to $\frac{c}{10}|T_u|$ odd vertices of T_v. Suppose this is the case and call this set A. Let T_A be the tree obtained by taking the union all the paths from A to v within T_v. A is not considered to be part of T_A. Now T_A is contained within the tree T'_v which is T_v minus the matching edges on the front.

Consider the last time T'_v was grown and look at the rule used to decide which tree to grow. At this point T_u is a subtree T'_u of the T_u at failure.

Case 2a. If both trees were in Stage 1 or 2, then the smaller tree was grown, so T'_v was smaller than a subtree of T_u. This implies that $|T'_v| \leq |T_u|$.

Case 2b. If one tree was in Stage 1 and the other in Stage 2 or 3, then since we grew T'_v, a subtree of T_u must have been in Stage 2 or 3 and T'_v in Stage 1. This implies that $|T'_v| \le n^{.6-\varepsilon} \le |T_u|$.

Case 2c. If one tree is in Stage 3 and the other in Stage 2, then T'_v must be in Stage 2 and T_u must be in Stage 3. T_v must also be in Stage 3, since we are trying to grow T_u. This contingency, T_u and T_v both in Stage 3 is not dealt with in the lemma. It will be dealt with in Section 4.3.

Now consider the set $S = T_u \cup A \cup T_A$, it has at least $|S| + |A| - 2$ edges. We have $|T_A| \le |T'_v| \le |T_u|$. This implies that

$$|S| = |T_u| + |A| + |T_A| \le 2|T_u| + |A| \le \left(\frac{20}{c} + 1\right)|A|.$$

So S is a set with $1 + \frac{|A|-2}{|S|} \ge 1 + \frac{1-o(1)}{1+\frac{20}{c}} \ge \frac{3}{2}$ edges, and this contradicts the result of Lemma 2.3.

We have seen that while in Stage 2, a tree grows in size by a factor of at least $3c/5$, unless it reaches size $n^{.99}$, see Lemma 2.1. This is impossible as it would require the exposure of at least $n^{.99-\varepsilon}$ vertices to this point. But we only expose at most $2n^{.6-\varepsilon}$ new vertices during an execution of AugmentPath and there are only $\tilde{O}(n^{1/5})$ executions of AugmentPath. Thus, both trees reach Stage 3.

Part (b) follows because we have shown that T_u grows by a factor $3c/5$ during Stage 2. □

2.3. PROOFS OF LEMMAS 2.1, 2.2, AND 2.3. This requires some knowledge of the distribution of the random graph Γ.

2.3.1. *Structure of* Γ. Suppose that it has ν vertices and μ edges. By construction it has minimum degree at least two. It is shown in Aronson et al. [1998] (Lemma 2) that if G is distributed as $G_{n,m}$, $m = cn/2$, $c > e$ (the base of natural logarithms), then Γ_1 is distributed as $G^{\delta\ge 2}_{\nu,\mu}$ that is, Γ has ν vertices and μ edges and $G^{\delta\ge 2}_{\nu,\mu}$ is uniformly chosen from simple graphs with ν vertices, μ edges and minimum degree at least 2. The precise values of μ, ν are not essential but we will describe how they are related to c: Let z, β be defined by

$$\beta e^{c\beta} = e^z \text{ and } z - c\beta(1 - e^{-z}) = 0. \tag{2}$$

Then with high probability

$$\nu \sim \beta(1 - (1 + z)e^{-z})n \text{ and } \mu \sim \frac{z^2 n}{2c} \tag{3}$$

where $\sim$ denotes $= (1 + o(1))$. (Equation (3) follows from Lemma 8 of Aronson et al. [1998]).

Note that as $c \to \infty$ we have $z \nearrow c$. ($z < c\beta \Rightarrow ze^z < c\beta e^{c\beta} = ce^z \Rightarrow z < c$. Also $c \to \infty \Rightarrow ce^z \to \infty \Rightarrow c\beta \to \infty \Rightarrow z \to \infty \Rightarrow z/c\beta \to 1$. Now use $\beta = e^{-e^{-z}c\beta}$ and $c\beta = ce^{-o(c\beta)}$ to deduce that $\beta \to 1$).

The degrees of the vertices in $\Gamma_1 = G^{\delta\ge 2}_{\nu,\mu}$ are distributed as truncated Poisson random variables $Po(\lambda; \ge 2)$. More precisely, we can generate the degree sequence

by taking random variables $Z_1, Z_2, \ldots, Z_\nu$ where

$$\mathsf{P}(Z_i = k) = \frac{\lambda^k e^{-\lambda}}{1-(1+\lambda)e^{-\lambda}} \qquad \textit{for } i = 1, 2, \ldots, \nu \textit{ and } k \geq 2. \tag{4}$$

Then, we condition on $Z_1 + Z_2 + \cdots + Z_\nu = 2\mu$. The resulting $Z_1, Z_2, \ldots, Z_\nu$ have the same distribution as the degrees of Γ_1. This follows from Lemma 4 of Aronson et al. [1998]. If we choose λ so that $\mathsf{E}(Po(\lambda; \geq 2) = \frac{2\mu}{\nu}$ or $\frac{\lambda(e^\lambda-1)}{e^\lambda-1-\lambda} = \frac{2\mu}{\nu}$, then the conditional probability, $\mathsf{P}(Z_1 + Z_2 + \cdots + Z_\nu = 2\mu) = \Omega(1/\sqrt{\nu})$ and so we will have to pay a factor of $O(\sqrt{\nu})$ for removing the conditioning, that is, to use the simple inequality $\mathsf{P}(A \mid B) \leq \mathsf{P}(A)/\mathsf{P}(B)$.

Hence, we can take

$$\frac{\lambda(e^\lambda - 1)}{e^\lambda - 1 - \lambda} = \tilde{c} = \frac{2\mu}{\nu} \in [.999c, 1.001c].$$

It follows from (2) and (3) that when c is large with high probability $\frac{z(e^z-1)}{e^z-1-z} \sim \frac{2\mu}{\nu} \sim c$ and so $\lambda \sim z \sim c$. This explains why we can assume that $\tilde{c} \in [.999c, 1.001c]$.

The maximum degree in G is less than $\log n$ with high probability and Γ inherits this property. Equation (7) of Aronson et al. [1998] enables us to claim that that if ν_k, $2 \leq k \leq \log n$ is the number of vertices of degree k, then with high probability

$$\left|\nu_k - \frac{\nu\lambda^k e^{-\lambda}}{k!(1-(1+\lambda)e^{-\lambda})}\right| \leq K_1\left(1 + \sqrt{\nu\lambda^k e^{-\lambda}/(k!(1-(1+\lambda)e^{-\lambda}))}\right)\log n \tag{5}$$

for $2 \leq k \leq \log n$, and for some constant $K_1 > 0$.

In particular, this implies that if the degrees of the vertices in Γ are $d_1, d_2, \ldots, d_\nu$, then with high probability,

$$\sum_{i=1}^{\nu} d_i(d_i - 1) = O(n). \tag{6}$$

Given the degree sequence we make our computations in the configuration model, see Bollobás [1980]. Let $\mathbf{d} = (d_1, d_2, \ldots, d_n)$ be a sequence of non-negative integers with $m = nd$ even. Let $W = [nd]$ be our set of *points* and let $W_i = [d_1 + \cdots + d_{i-1} + 1, d_1 + \cdots + d_i]$, $i \in [n]$, partition W. The function $\phi : W \to [n]$ is defined by $w \in W_{\phi(w)}$. Given a pairing F (i.e., a partition of W into $m = dn/2$ pairs) we obtain a (multi-)graph G_F with vertex set $[n]$ and an edge $(\phi(u), \phi(v))$ for each $\{u, v\} \in F$. Choosing a pairing F uniformly at random from among all possible pairings of the points of W produces a random (multi-)graph G_F.

This model is valuable because of the following easily proven fact: Suppose $G \in \mathcal{G}_{n,\mathbf{d}}$. Then

$$\mathsf{P}(G_F = G \mid G_F \text{ is simple}) = \frac{1}{|\mathcal{G}_{n,\mathbf{d}}|}.$$

It follows that if G is chosen randomly from $\mathcal{G}_{n,\mathbf{d}}$, then for any graph property $\mathcal{P}$

$$\mathsf{P}(G \in \mathcal{P}) \leq \frac{\mathsf{P}(G_F \in \mathcal{P})}{\mathsf{P}(G_F \text{ is simple})}. \tag{7}$$

Furthermore, applying Lemmas 4.4 and 4.5 of McKay [1985], we see that if the degree sequence of Γ satisfies (6) then $\mathsf{P}(G_F \text{ is simple}) = \Omega(1)$. In which case, the configuration model can substitute for $\mathcal{G}_{n,\mathbf{d}}$ in dealing events of probability $o(1)$.

2.3.2. *Proof of Lemma* 2.1. We show that with high probability augmenting trees of size $\lambda = 2l + 1$, where $\frac{\alpha_1}{c} \log n \le \lambda \le n^{0.9}$, expand at a steady rate close to c.

We will first show that the probability of the event that there exists a tree T with $|Even(T)| = l$ and $|N(Even(T))| = r$, $\frac{\alpha_1}{c} \log n \le l \le n^{0.9}$ and $l \le r \le 0.91cl$, is polynomially small.

If the above event occurs, then the following configuration appears (i) $2l$ edges of the tree connect *Even(T)* to *Odd(T)* (ii) $r - l$ edges connect *Even(T)* to $U = N(Even(T)) \setminus Odd(T)$ and (iii) none of the $(l + 1)(n - r - l - 1)$ edges between *Even(T)* $\cup$ {root vertex} and $V \setminus U$ are present. Given the vertices of T and $N(Even(T))$, the probability of the above event occurring in $G^{\delta\ge 2}_{n,m}$ is bounded above by

$$O(\sqrt{\nu})\left(\frac{1}{2\mu - 2(l+r)+1}\right)^{l+r} \times \sum_{\substack{d_i \ge 2,\, i\in[r+l+1] \\ \sum_{i=1}^{l+1} d_i = r+l}} \left(\prod_{i=1}^{l+1} \frac{\lambda^{d_i} d_i!}{d_i!(e^\lambda - 1 - \lambda)} \prod_{i=l+2}^{2l+1} \frac{\lambda^{d_i} d_i(d_i-1)}{d_i!(e^\lambda - 1 - \lambda)} \prod_{i=2l+2}^{r+l+1} \frac{\lambda^{d_i} d_i}{d_i!(e^\lambda - 1 - \lambda)}\right) \quad (8)$$

Explanation. The probability that an edge exists between vertices u and v of degrees d_u and d_v, given the existence of other edges in the tree, is at most $\frac{d'_u d'_v}{2\mu - 2(l+r)+1}$ where $d'_u = d_u$ less the number of edges already assumed to be incident with u. Hence, given the degree sequence, the probability that the augmenting tree exists is at most

$$\left(\frac{1}{2\mu - 2(l+r)+1}\right)^{l+r} \prod_{i=1}^{l+1} d_i! \prod_{i=l+2}^{2l+1} d_i(d_i-1) \prod_{i=2l+2}^{r+l+1} d_i.$$

Here, the first product corresponds to $Even(T)$, the second product corresponds to $ODD(T)$ and the final product corresponds to neighbors of T (not in T).

We now simplify the expression (8) obtained for the probability to

$$O(\sqrt{\nu})\left(\frac{1}{2\mu - 2(l+r)+1}\right)^{l+r} \times \frac{\lambda^{2r+2l}}{(e^\lambda - 1 - \lambda)^{r+l+1}} \sum_{\sum_{i=1}^{l+1} d_i = r+l} \left(\sum_{d_i\ge 2,\, i\in[r+l+1]} \prod_{i=l+2}^{2l+1} \frac{\lambda^{d_i-2}}{(d_i-2)!} \prod_{i=2l+2}^{r+l+1} \frac{\lambda^{d_i-1}}{(d_i-1)!}\right)$$
$$\le O(\sqrt{\nu})\left(\frac{1}{2\mu - 2(l+r)+1}\right)^{l+r} \times$$

$$\frac{\lambda^{2r+2l}}{(e^\lambda-1-\lambda)^{r+l+1}} \sum_{\substack{\sum_{i=1}^{l+1} d_i = r+l \\ d_i \geq 2, i \in [l+1]}} \left(\left(\prod_{i=l+2}^{2l+1} \sum_{d_i \geq 2} \frac{\lambda^{d_i-2}}{(d_i-2)!} \right) \left(\prod_{i=2l+2}^{r+l+1} \sum_{d_i \geq 2} \frac{\lambda^{d_i-1}}{(d_i-1)!} \right) \right)$$

$$\leq O(\sqrt{\nu}) \left(\frac{1}{2\mu - 2(l+r)+1} \right)^{l+r} \frac{\lambda^{2r+2l}}{(e^\lambda-1-\lambda)^{r+l+1}} \binom{r}{l} e^{\lambda l} \ (e^\lambda-1)^{r-l}$$

$$\leq O(\sqrt{\nu}) \left(\frac{1}{2\mu - 2(l+r)+1} \right)^{l+r} \frac{\lambda^{2r+2l}}{(e^\lambda-1-\lambda)^{r+l}} \left(\frac{er}{l} \right)^l e^{\lambda l} (e^\lambda-1)^{r-l}$$

$$= O(\sqrt{\nu}) \left(\frac{1}{2\mu - 2(l+r)+1} \right)^{l+r} \left(\frac{er}{l} \right)^l (\tilde{c}\lambda)^r \left(\frac{\tilde{c}\lambda e^\lambda}{(e^\lambda-1)^2} \right)^l$$

using $\dfrac{\lambda(e^\lambda-1)}{e^\lambda-1-\lambda} = \tilde{c}$

$$\leq O(\sqrt{\nu}) \left(\frac{1}{2\mu - 2(l+r)+1} \right)^{l+r} (\tilde{c}\lambda)^r \left(\frac{0.91ec\tilde{c}\lambda e^\lambda}{(e^\lambda-1)^2} \right)^l \quad \text{using } r/l \leq 0.91c$$

$$\leq O(\sqrt{\nu}) \left(\frac{1}{2\mu - 2(l+r)+1} \right)^{l+r} (\tilde{c}\lambda)^r \left(\frac{0.92ec\tilde{c}\lambda}{e^\lambda-1} \right)^l \quad \text{using } \frac{e^\lambda}{e^\lambda-1} < 1.01$$

$$\leq O(\sqrt{\nu}) \left(\frac{1}{\tilde{c}\nu} \right)^{l+r} e^{3(l+r)^2/\tilde{c}\nu} (\tilde{c}\lambda)^r \left(\frac{0.92ec\tilde{c}\lambda}{e^\lambda-1} \right)^l \quad \text{using } 2\mu = \tilde{c}\nu$$

$$= O(\sqrt{\nu}) \left(\frac{1}{\nu} \right)^{l+r} e^{o(l)} \ \lambda^r \left(\frac{0.92ec\lambda}{e^\lambda-1} \right)^l \tag{9}$$

since $r = O(l)$.

We now count the number of such configurations. We begin by choosing *Even(T)* and the root vertex of the tree in at most $\nu\binom{\nu}{}l)$ ways. We make the following observation about augmenting path trees with $|Even(T)| = l$. The contraction of the matching edges of the tree yields a unique tree on $l+1$ vertices. We note, by Cayley's formula, that the number of trees that could be formed using $(l+1)$ vertices is $(l+1)^{l-1}$. Reversing this contraction, we now choose the sequence of l vertices, *Odd(T)*, that connect up vertices in *Even(T)* in $(\nu-l-1)(\nu-l-1)\cdots(\nu-2l) = (\nu-l-1)_l$ ways. We pick the remaining $r-l$ vertices from the remaining $\nu-2l-1$ vertices in $\binom{\nu-2l-1}{r-l}$ ways. These $r-l$ vertices can connect to any of *Even(T)* in l^{r-l} ways. Hence, the total number of configurations is at most

$$\nu\binom{\nu}{l}(l+1)^{l-1}(\nu-l-1)_l\binom{\nu-2l-1}{r-l}l^{r-l} \leq \nu^{r+l+1}e^r\left(\frac{l}{r-l}\right)^{r-l}.$$

Combining the bounds for probability and configurations, we get an upper bound of

$$\nu^{r+l+1}e^r\left(\frac{l}{r-l}\right)^{r-l} \quad O(\sqrt{\nu})\left(\frac{1}{\nu}\right)^{l+r} e^{o(l)}\ \lambda^r\left(\frac{0.92ec\lambda}{e^\lambda-1}\right)^l$$

$$\leq O(\nu^{3/2}) \cdot \left(\frac{e\lambda l}{r-l}\right)^{r-l}\left(\frac{0.93e^2c\lambda^2}{e^\lambda-1}\right)^l$$

The expression $(\frac{e\lambda l}{x})^x$ is maximized at $x = \lambda l > 0.9999\tilde{c}l$. But $r - l \leq 0.91cl < \lambda l$. Hence, we have the bound

$$
\begin{aligned}
& O(\nu^{3/2}) \cdot \left(\frac{e\lambda l}{0.91cl}\right)^{0.91cl} \left(\frac{0.93e^2c\lambda^2}{e^\lambda - 1}\right)^l \\
& \quad \leq O(\nu^{3/2}) \cdot \left(\frac{e}{0.91}\right)^{0.91cl} \left(\frac{0.93e^2c\lambda^2}{e^\lambda - 1}\right)^l \qquad \text{using } \lambda < c \\
& \quad \leq O(\nu^{3/2}) \cdot \left(\frac{e}{0.91}\right)^{0.91cl} \left(\frac{0.93e^2c\lambda^2}{e^{0.999c}}\right)^l \qquad \text{using } e^\lambda - 1 > e^{0.999c} \\
& \quad \leq O(\nu^{3/2}) \cdot \left(\frac{0.93e^2c\lambda^2}{e^{0.002c}}\right)^l \qquad \text{using } \frac{e^{0.999}}{\left(\frac{e}{0.91}\right)^{0.91}} > e^{0.002} \\
& \quad = O(\nu^{3/2})q^l \qquad \text{where } q = \frac{0.93e^2c\lambda^2}{e^{0.002c}} \leq e^{-.001c}.
\end{aligned}
$$

We sum the above expression over all r and l with $\frac{\alpha_1}{c}\log n \leq l \leq n^{0.9}$ and $l \leq r \leq 0.91cl$ and we get the probability to be at most

$$O(n^{7/2})q^{\frac{\alpha_1}{c}\log n} = O(n^{7/2-\alpha_1/1000}) = o(n^{-2})$$

for $\alpha_1 \geq 5501$.

We will now show that the probability of the event that there exists a tree having $|Even(T)| = l$ and $|N(Even(T))| \geq 1.07cl$, where $\frac{\alpha_1}{c}\log n \leq l \leq n^{0.9}$.

It is enough to show that the probability that there exists a tree with $|Even(T)| = l$ and $|N(Even(T))| \geq r$, where $\frac{\alpha_1}{c}\log n \leq l \leq n^{0.9}$ and $r = 1.07cl$, is polynomially small since a tree with $|N(Even(T))| > 1.09cl$ also contains a tree with $|N(Even(T))| \geq 1.07cl$.

The probability bound (9) remains the same with the 0.92 replaced by 1.08 and we get a bound of

$$
\begin{aligned}
& O(\nu^{3/2}) \cdot \left(\frac{e\lambda l}{1.07cl}\right)^{1.07cl} \left(\frac{1.08e^2c\lambda^2}{e^\lambda - 1}\right)^l \\
& \quad \leq O(\nu^{3/2}) \cdot \left(\frac{e}{1.07}\right)^{1.07cl} \left(\frac{1.08e^2c\lambda^2}{e^\lambda - 1}\right)^l \qquad \text{using } \lambda < c \\
& \quad \leq O(\nu^{3/2}) \cdot \left(\frac{e}{1.07}\right)^{1.07cl} \left(\frac{1.08e^2c\lambda^2}{e^{0.999c}}\right)^l \qquad \text{using } e^\lambda - 1 > e^{0.999c} \\
& \quad \leq O(\nu^{3/2}) \cdot \left(\frac{1.08e^2c\lambda^2}{e^{0.002c}}\right)^l \qquad \text{using } \frac{e^{0.999}}{\left(\frac{e}{1.07}\right)^{1.07}} > e^{0.002} \\
& \quad = O(\nu^{3/2})q^l \qquad \text{where } q = \frac{1.08e^2c\lambda^2}{e^{0.002c}} \leq e^{-.001c}.
\end{aligned}
$$

We sum the above expression over all l with $\frac{\alpha_1}{c}\log n \le l \le n^{0.9}$ and we get the probability to be at most

$$2en^{5/2}q^{\frac{\alpha_1}{c}\log n} = 2en^{5/2-\alpha_1/1000} = o(n^{-2})$$

for $\alpha_1 \ge 4501$.

Hence, with high probability, there does not exist a tree with $|Even(T)| = l$ and $|N(Even(T))| = r$, where $\frac{\alpha_1}{c}\log n \le l \le n^{0.9}$ and $r \notin [0.9cl, 1.1cl]$. □

2.3.3. *Proof of Lemma* 2.2. We show that this holds in $G_{n,m}$ and note that $G^{\delta\ge 2}_{\nu,\mu}$ is a vertex induced subgraph of $G_{n,m}$. Since the property is closed under edge deletion this will imply the lemma. Because the property in question is monotone increasing we can estimate the events probability in $G_{n,p}$ where $p = \frac{c}{n}$.

If there are two small cycles close together, then there will be a path P of length at most $k = a + b + d$ plus two extra edges joining the endpoints of P to internal vertices of P. The probability of this can be bounded by

$$n^k k^2 p^{k+1} \le \frac{k^2 c^{k+1}}{n} = \tilde{O}\left(\frac{1}{n^{1-\alpha_2}}\right). \qquad \square$$

2.3.4. *Proof of Lemma* 2.3. We can work in $G_{n,p}$, for exactly the same reasons as in Lemma 2.2.

We get the bound

$$\begin{aligned}\binom{n}{k}\binom{\binom{k}{2}}{(1+\epsilon)k}\left(\frac{c}{n}\right)^{(1+\epsilon)k} &\le \left(\frac{en}{k}\right)^k \left(\frac{ek^2/2}{(1+\epsilon)k}\right)^{(1+\epsilon)k}\left(\frac{c}{n}\right)^{(1+\epsilon)k}\\ &\le \left(\frac{e^{2+\epsilon}c^{1+\epsilon}k^{\epsilon}}{n^{\epsilon}}\right)^k\end{aligned}$$

and since $k = O(n^{.99})$ the summand can be upper bounded by 2^{-k} for $k \ge \sqrt{n}$ and by $n^{-\epsilon k/200}$ for $k \le \sqrt{n}$. The union bound then gives an upper bound of

$$\sum_{k=\log\log n}^{\sqrt{n}} n^{-\epsilon k/200} + \sum_{k=\sqrt{n}}^{n^{.99}} 2^{-k} = o(n^{-3}) \qquad \square$$

3. *Karp-Sipser Conditioning*

We now study the conditioning introduced on unused edges by the Karp-Sipser algorithm.

We now view Γ as an ordered sequence of edges and look at an equivalent version of the Karp-Sipser algorithm. In the analysis of Karp-Sipser on random graphs, we have two sources of randomness. One is the random graph itself and the other one is the random choices made by the algorithm. In order to simplify the analysis, we change the choices into deterministic ones and simply randomize the order in which the edges are stored and take them in this (random) order. This is equivalent to the original algorithm. We now state the modified Karp-Sipser algorithm. We stress that we do not propose to implement the Karp-Sipser algorithm in this way, it is merely a vehicle used in our analysis. It does however produce the same output if the ordering is matched to the choices made by the original Karp-Sipser algorithm.

We consider its effect on Phase 2. We assume the graph Γ at the start of Phase 2 is given as $\Gamma = (\xi_1, \ldots, \xi_\mu)$, an ordered set of μ edges.

We say that edge $e \in \Gamma$ has index i if it is the ith edge in the list, that is, $e = \xi_i$. Note that every graph in the support of $G_{\nu,\mu}^{\delta\geq 2}$ will yield $\mu!$ ordered sets of edges, so from now on we will think of $G_{\nu,\mu}^{\delta\geq 2}$ as a family of ordered sets of edges. Furthermore, if c is large then μ/ν will be close to c, with high probability.

```
 1: procedure KS*(G)
 2:    M ← ∅
 3:    while G ≠ ∅ do
 4:       if G has vertices of degree 1 then
 5:          Of all edges incident to vertices of degree 1, let e have the lowest index
 6:          Let e = (v, u) where v has degree 1.
 7:       else
 8:          Let e = (v, u) be the edge of lowest index in G
 9:       end if
10:       M ← M ∪ (v, u)
11:       G ← G \ {v, u}
12:    end while
13:    return M
14: end procedure
```

The output of Karp-Sipser algorithm will consist of an *ordered* matching $M = \{e_1, e_2, \ldots, e_\ell\}$ plus some extra *witness* edges W. Here, $e_1, e_2, \ldots, e_\ell$ is the order in which the matching edges are produced by the Karp-Sipser algorithm. More precisely, the output will be a sequence $\sigma_1, \sigma_2, \ldots, \sigma_\mu$ where σ_i is either (i) an edge of G, marked as being a matching edge or a witness, or (ii) $*$. The choice $\sigma_i = *$ signifies that the ith edge is random with a distribution described in Section 3.2. In addition, the output includes the ordering of M by the time of choice by the Karp-Sipser algorithm.

There will therefore be a set Ψ of possible edge replacements for the positions with $*$. Given the output, each member of Ψ will be equally likely. This follows from the fact that G is sampled uniformly from a set of instances. We need however to know more about the structure of Ψ and this is the content of Lemma 3.1 below.

3.1. WITNESS EDGES. In addition to the edges of the matching, we define edges based on the execution of the algorithm. We split the vertices of the graph into three classes, *regular, pendant* and *unmatched*. A vertex is regular if when it was removed from the graph, it had degree 2 or more. A vertex is said to be a pendant vertex if when it was removed it had degree exactly one and it is the endpoint of a matching edge in M. Unmatched vertices are those vertices that are not incident to matching edges. We say that an edge e is regular if both of its endpoints are regular, that is, it was removed from the graph in line 8. For each of these vertices we define witness edges.

—*Regular Vertices*. v is removed from the graph when the edge e is picked as a matching edge. Since it has degree at least 2, there are other edges incident to it at the time it is removed. Pick the one with the lowest index and define it to be the *regular witness edge* for v.

—*Pendant or Unmatched Vertices*. Consider the last point of time when v has degree at least 2, an edge $e = (x, y)$ is removed from the graph and v is incident to at least one of them (perhaps both), say x. We then define (v, x) to be the *pendant witness edge* for v.

—*Unmatched Vertices*. v has a pendant witness edge, and since it is never picked for a matching its last edge is incident to some matching edge $e = (x, y)$, say x, we then define (v, x) to be the *removal witness edge* for v.

—In case of any ambiguities, define pendant witness edges first and then removal witness edges. Use the lowest index of edges to break all ties. This can happen if a vertex goes from having degree three to pendant or from having degree 2 to degree zero if it is incident to both endpoints of a matched edge.

—Note that an edge e can be a regular witness edge for one vertex and a pendant or removal witness edge for another vertex.

Let W be the set of witness edges. Regular and pendant vertices are incident to matching edges and their witness edges. Unmatched vertices are incident to two witness edges. Hence, the graph defined by M and W has minimum degree 2 and size at most $2v$.

We think of the graph G as an ordered set of μ boxes filled with edges. Suppose we know the output of KS*, M, W and also the order in which the matching and witness edges were added to M and W, but the underlying graph is unknown to us. This corresponds to μ ordered boxes, of which the ones corresponding to M and W have been opened. The following lemma provides necessary and sufficient conditions for a graph G to yield M and W as the output of KS*.

LEMMA 3.1. *In the following, $e = (u, v)$ is not an edge of graph G' and $G = G' \cup \{e\}$. Think of G' as an ordered set of μ boxes where one box is unopened. To obtain G we open this box and find e. Suppose that when algorithm KS* is run on G it produces the ordered matching M and the witness set W. Suppose further that when algorithm KS* is run on G' it produces the ordered matching M' and the witness set W'.*

(*a*) *If $e \notin M \cup W$, then $M' = M$ and $W' = W$ and e satisfies the preservation conditions 1,2, and 3 below where the type of a vertex is determined by the execution of the Karp-Sipser algorithm on G.*

(*b*) *If e satisfies the preservation conditions 1,2, and 3 below, where the type of a vertex is determined by the execution of the Karp-Sipser algorithm on G' then $M = M'$ and $W = W'$.*

(*c*) *In both cases (a), (b) above, the type of u or v is the same with respect to the execution of the Karp-Sipser algorithm on G or G'.*

Preservation Conditions.

(1) *If both u and v are both regular vertices and u was removed from the graph before v, then e has a higher index than the regular witness edge for u.*

(2) *If u is a regular vertex and v is either a pendant or unmatched vertex, then (i) e has a higher index than the regular witness edge for u, (ii) u is removed before v by the Karp-Sipser algorithm, and (iii) when u is removed from the graph v has degree at least 2. Additionally, if the pendant witness for v is incident*

to the matching edge of u then e has a higher index than the pendant witness for v.

(3) *At least one of u, v is a regular vertex.*

PROOF OF LEMMA 3.1. To keep track of the algorithm KS* we let G_t denote the graph after the tth iteration, so $G_0 = G$ and $G_T = \emptyset$ where T is the number of iterations. At timestep t let Π_t be the set of edges incident with pendant vertices of G_t. Let M_t denote the set of matching edges and W_t denote the set of witness edges at time t. For G' we define G'_t etc. in the same manner.

(a) Assume that u is removed first from G at time step $t_u + 1$, that is, $u \in G_{t_u}$ and $u \notin G_{t_u+1}$. If u, v are both G-regular, then this can be assumed. (We use the term G-regular to stress the fact that we are considering the execution of the Karp-Sipser algorithm on G.) If u is a G-regular vertex and v is either a G-pendant vertex or a G-unmatched vertex then u must be removed first. Suppose not and v is removed first. Suppose that v is G-pendant. Then, when v is removed, edge e is still in G_t and it has a higher index than the G-witness edge for v. But this implies that v is a G-regular vertex, contradiction. A similar argument handles the case when v is unmatched.

We show next that $M_t = M'_t$ for $t = 1, \ldots, t_u$. If this holds for all t up to t_u, then after that we will have $G_t = G'_t$ for $t > t_u$ since u has been removed and e is gone from the graph. This is proved by induction. The base case is easy since $M_0 = M'_0 = \emptyset$. Assume that $M_t = M'_t$ so $G_t = G'_t \cup \{e\}$. We now show that $M_{t+1} = M'_{t+1}$

Case 1. Both u and v are G-regular. Since e is not incident to a G-pendant vertex and $e \notin W$ we have $\Pi_t = \Pi'_t$. If $\Pi_t \neq \emptyset$, then we select the edge from Π_t with minimum index, and add it to M_t, since $\Pi_t = \Pi'_t$ we have $M_{t+1} = M'_{t+1}$. If $\Pi_t = \emptyset$ we select the edge in G_t of minimum index. This cannot be e since it comes after the regular G-witness edge f for u. If e came before f then f would not be the G-witness edge for u. Hence, the preservation conditions hold and the same edge is chosen in both G_t and G'_t and $M_{t+1} = M'_{t+1}$.

Case 2. u is a G-regular vertex and v is either a G-pendant vertex or a G-unmatched vertex. Now u is removed from G before v and $\deg_{G_{t_u}}(v) \geq 2$. This is clear since e is not a pendant witness for v and the pendant witness edge is the second last edge incident to v to be removed from the graph.

Neither u nor v are G-pendant vertices for $t \leq t_u$ and so e is not incident with a G-pendant vertex, hence, $\Pi_t = \Pi'_t$. Thus, if $\Pi_t \neq \emptyset$ we have $M_{t+1} = M'_{t+1}$ as before. If $\Pi_t = \emptyset$, we use the argument for Case 1 with the additional comment. If the G-pendant witness f for v is incident to the matching edge of u, then e must have a higher index than f; otherwise, e would be the G-pendant witness for v.

Case 3. If u, v are both G-pendant vertices and u is matched before v, then u is incident to e and its matching edge at the time it is matched, contradicting the fact that $\deg_{G_{t_u}}(u) = 1$. If u is G-pendant and v is unmatched, then we draw the same conclusion. If both u, v are G-unmatched and u becomes isolated first, then e would have to be the G-removal witness for u, a contradiction since v is still in the graph.

We have now shown that e satisfies the preservation conditions with respect to G and the same ordered matching set M will be generated for G and G'. In particular,

we have shown

$$G_t = \begin{cases} G'_t \cup \{e\} & t \leq t_u \\ G'_t & t > t_u. \end{cases} \tag{10}$$

This together with the fact that e has a higher index than the G-regular witness edge f for u immediately implies that $W' = W$. This is because the Karp-Sipser algorithm will basically have the same choices for witnesses at each point.

This completes the proof of Part (a).

(b) Assume now that the preservation conditions hold with respect to the execution of the Karp-Sipser algorithm on G'. It is enough to show that (10) holds under these assumptions. This will imply that $W' = W$. Consider the execution of the Karp-Sipser algorithm on G. Assume as in (a) that $M_t = M'_t$ so $G_t = G'_t \cup \{e\}$. We show that $M_{t+1} = M'_{t+1}$.

Case 1. Both u and v are G'-regular. Since e is not incident to a G'-pendant vertex, we have $\Pi_t = \Pi'_t$. If $\Pi_t \neq \emptyset$, then we select the edge from Π_t with minimum index and add it to M_t. Since $\Pi_t = \Pi'_t$, we have $M_{t+1} = M'_{t+1}$. If $\Pi_t = \emptyset$ we select the edge in G_t of minimum index. This cannot be e since it comes after the regular G'-witness edge f for u. Note that $f \in G'_t$ implies that $f \in G_t$ at this point. Hence, the same edge is chosen in both G_t and G'_t and $M_{t+1} = M'_{t+1}$.

Case 2. u is a G'-regular vertex and v is either a G'-pendant vertex or an G'-unmatched vertex. By the preservation conditions, u is removed from G' before v and $\deg_{G'_{t_u}}(v) \geq 2$. Neither u nor v are G_t-pendant vertices for $t \leq t_u$ so e is not incident with a G_t-pendant vertex, hence $\Pi_t = \Pi'_t$. Thus, if $\Pi_t \neq \emptyset$, we have $M_{t+1} = M'_{t+1}$ as before. If $\Pi_t = \emptyset$, we use the argument for Case 1.

Hence, the graphs G and G' will generate the same ordered matchings and (10) holds.

This completes the proof of Part (b).

Part (c) follows from (10). It immediately shows that if x is G'-regular then it is G-regular. If x is G'-pendant, then it is either G-regular or G-pendant. Also, x is G-unmatched iff it is G'-unmatched. We only have to check now that if x is G'-pendant, then it is G-pendant. Here, we must have $x \in \{u, v\}$ and then $x = v$ since we have assumed that u is deleted first and is G'-regular. But the fact that u is deleted before v means that adding the edge e cannot change the type of vertex v when going from G' to G. □

Remark 3.2. Note that it is possible to add an edge to the graph that will produce the same set of matching edges, but a different witness set. Since we want to condition on both sets, and the exact order in which they were produced we are not concerned with such cases.

Remark 3.3. Part (a) of Lemma 3.1 shows that we can remove all edges except matching and witness edges and KS* will produce the same output on this "skeleton" graph. Thus, we can partition the set of all graphs where each part corresponds to the same skeleton graph. Part (b) of Lemma 3.1 allows us to construct each graph corresponding to the same skeleton graph.

Remark 3.4. We stress the following point about condition 2(iii). If (u, x) is a pendant edge at the time of u's removal and v was pendant at this time, then adding (u, v) would mean that G_{t_u} and G'_{t_u} have different sets of pendant vertices and this contradicts our argument for Part (a).

3.2. PROBABILITY SPACE. We describe the probability space after we sample a random graph from $G^{\delta\geq 2}_{\nu,\mu}$ and run KS* on the graph. We condition on the output matching edges M and also on the witness edges W. For the purpose of proof, we assume that KS* outputs W as well as M. Given the output M and W and Lemma 3.1 we can find all graphs that would give M and W as the output of KS* and generate one uniformly at random.

First, note that for each box i that is not in M or W we can create a set of edges E_i that could go into that box, from Lemma 3.1 we see that this set depends only on M and W. Also note that all the rules state that an edge can go into any box that comes after some specified box, thus we have $E_i \subseteq E_j$ when $i < j$. This leads us to the following algorithm for generating a random graph from the distribution $G^{\delta\geq 2}_{\nu,\mu}|M, W$, that is, conditioned on the output of KS*.

1: **Procedure** GENERATE-RANDOM(M,W)
2: **for** unfilled boxes i **do**
3: $E_i \leftarrow$ {all edges e that can go into box i}
4: **end for**
5: $\Gamma \leftarrow M \cup W$
6: **for** unfilled boxes i in increasing order **do**
7: Select e uniformly at random from E_i
8: $\Gamma \leftarrow \Gamma \cup \{e\}$
9: Remove e from E_j for all $j > i$
10: **end for**
11: **return** Γ
12: **end procedure**

Each Γ that outputs M and W can be generated with Generate-Random in exactly one way (since the graph is an ordered set of edges) and that any graph Γ produced by Generate-Random will produce M and W when we run KS* on Γ. This shows that Generate-Random will gives a uniformly random graph from $G^{\delta\geq 2}_{\nu,\mu}|M, W$.

4. *Final Proof*

In Section 3.2, we gave a description of the probability space. However, this is not enough for us to finish the proof of Theorem 1.1. We need to dig a little deeper into the analysis of the Karp-Sipser algorithm. We begin by listing some definitions and lemmas from this article that we will need. The constants ε and γ are given in (1).

Let τ_0 be the last time when the number of vertices removed from the graph is at most $n^{.8+10\varepsilon}$. We refer to vertices removed before τ_0 as *early vertices* and those removed afterwards as *late*. We say that a matching edge e is a *good matching edge* if both of its endpoints are early and regular and if the regular witness edges for both of its endpoints have indices in $[\mu/2]$.

We will show that with high probability when we reach Stage 3, T_u and T_v will both have many late vertices at the front (Lemma 4.3). There will also be many good matching edges outside the two trees (Lemma 4.4).

The good matching edges are useful since they allow us to identify a large set of potential edges in the graph. Suppose that $e = (x, a)$ where a is an endpoint of a good matching edge and x is a late vertex. Then, a is a regular vertex removed before x and when a is removed from the graph all vertices have degree at least two. So unless (i) x is a pendant or unmatched vertex and (ii) the pendant witness edge for x is incident to the other endpoint of the matching edge for b, then all of the preservation conditions are satisfied and (x, a) can be part of the graph. We say that such an edge is *available*. But for each a, Lemma 2.2 implies that there is at most one vertex x such that (i) and (ii) hold. Because a is an endpoint of a good matching edge the edge $e = (x, a)$ can go into any open box after the regular witness edge for a, which is guaranteed to have index less than $\mu/2$. In summary, if e is available, then

$$\mathsf{P}(\textit{Box } i \geq \mu/2 \textit{ contains } e \mid \textit{contents of boxes } j \neq i) \geq \frac{1}{\binom{n}{2}}. \tag{11}$$

Using this, we will be able to show that if T_u, T_v contain many unexposed late vertices at their front and there are many good matching edges, then we will be able to find x, a, b, y such that x, y are unexposed late vertices at the fronts of T_u, T_v repsectively and (a, b) is a good matching edge and x, a, b, y is a path, producing the required augmenting path from u to v.

4.1. THE BATCH GRAPH. Let $\Gamma(t)$ denote the current graph after t steps of Phase 2. It is shown in Aronson et al. [1998] that $\Gamma(t)$ is distributed uniformly at random from the set of all graphs with $v_0(t)$ vertices of degree 0, $v_1(t)$ vertices of degree 1, $v(t)$ vertices of degree at least 2 and $m(t)$ edges, we denote this sequence by $\vec{v}(t) = (v_0(t), v_1(t), v(t), m(t))$. Furthermore, the sequence $\vec{v}(t)$ is a Markov chain. Thus, the analysis of the algorithm is done by tracking the sequence $\vec{v}(t)$. Additionally we define $z(t)$ by

$$\frac{2m(t) - v_1(t)}{v(t)} = \frac{z(t)(e^{z(t)} - 1)}{e^{z(t)} - 1 - z(t)}.$$

Thus, $z(0) = \lambda \sim z$ of Section 2.3.1. Notice that $v_1(0) = 0$.

Conditional on $\vec{v}(t)$, the degrees of the vertices of degree at least 2 are distributed as independent copies of a truncated Poisson random variable Z, where

$$\mathsf{P}(Z = k) = \frac{z^k}{k!(e^z - 1 - z)} \quad k = 2, 3, \ldots$$

conditional on $\sum_{v:\deg(v)\geq 2} Z_v = 2m(t) - v_1(t)$.

As our input is taken from $G^{\delta\geq 2}_{\nu,\mu}$ we start in the state $\vec{v}(0) = (0, 0, \nu, \mu)$, that is, with $v_1(0) = 0$. For $t_1 < t_2$, such that $v_1(t_1) = v_1(t_2) = 0$ and $v_1(t) > 0$ for $t_1 < t < t_2 \leq \tau_0$ we look at the set of edges and vertices removed from t_1 to t_2, that is, the graph $\Gamma(t_1) \setminus \Gamma(t_2)$ and call it a *batch*. Note that each batch contains the regular matching edge removed at time t_1 and is a connected graph.

We note next that for $t = 0, \ldots, \tau_0$ we have removed at most $o(n)$ vertices and $o(n)$ edges, thus

$$\frac{2m(t) - v_1(t)}{v(t)} = (1 + o(1))\frac{2m(0) - v_1(0)}{v(0)} = (1 + o(1))\tilde{c}$$

and $z(t) = (1+o(1))z(0)$ and $z(t)$ is bounded away from 0 by a constant. Corollary 3 of Aronson et al. [1998], then gives that $E[v_1(t+1)-v(t)] \le -\alpha$ for some positive constant α. Using this α in Lemmas 13 and 14 in Aronson et al. [1998] gives the following lemma:

LEMMA 4.1.

$$\mathsf{P}\left(\exists t \le \tau_0 : v_1(t) > \frac{4\log^3 n}{\alpha}\right) = O(n^{-4})$$

and

$$\mathsf{P}\left(\exists t \le \tau_0 - T : v_1(t) = 0, v_1(t') > 0 \text{ for } t < t' \le t + T\right) = O(n^{-4})$$

for $T = \frac{16\log^3 n}{\alpha^3}$.

This shows that for $t \le \tau_0$ each batch corresponds to an interval of time of length at most $O(\log^3 n)$. Since the maximum degree in Γ is $o(\log n)$ with high probability, we see that the total number of pendant vertices (at the time of removal) is $O(\log^4 n)$, which implies that the number of vertices in a batch is also $O(\log^4 n)$.

This also shows that during the first τ_0 time steps there will be at least $\Omega(\frac{n^{.8+10\varepsilon}}{\log^4 n})$ times when $v_1(t) = 0$ and thus at least that many regular edges are added to the matching.

LEMMA 4.2. *The probability ρ that there exists a vertex $v \in G$ that is within distance $\gamma \log_c n$ of* 100 *batches is at most* n^{-5}

PROOF OF LEMMA 4.2. Letting *dist* denote distance in Γ we bound this probability by

$$\rho \le n \sum_{v=1}^{n} \binom{n^{.8+10\varepsilon}}{100} \prod_{i=1}^{100} \mathsf{P}(dist(v, B_i) \le \gamma \log_c n \mid dist(v, B_j) \le \gamma \log_c n,\ 1 \le j < i).$$

Explanation. Here $\binom{n^{.8+10\varepsilon}}{k}$ is the number of choices for the start times of the batches $B_1, B_2, \ldots, B_k$.

We claim that for each i, v,

$$\mathsf{P}(dist(v, B_i) \le \gamma \log_c n \mid dist(v, B_j) \le \gamma \log_c n,\ 1 \le j < i) \le n^{\gamma+\varepsilon+o(1)-1}. \tag{12}$$

Using the values of ε, γ from (1), this gives

$$\rho \le n(n^{.8+11\varepsilon+\gamma+o(1)-1})^{100} \le n^{-5}.$$

PROOF OF (12). Suppose that B_i is constructed at time t_i. It is a subgraph of $\Gamma(t_i)$ and depends only on this graph. Using Lemma 4.1, we argue that

$$\mathsf{P}(\exists w \in B_i : dist(v, w) \le \gamma \log_c n \mid dist(v, B_j) \le \gamma \log_c n,\ 1 \le j < i) \le O(n^{-4}) + \frac{O(\log^4 n)}{n - o(n)}(n^{\gamma+\varepsilon} + \mathsf{P}(\exists w : |N_{\gamma \log_c n}(w)| \ge n^{\gamma+\varepsilon}), \quad (13)$$

where $N_k(w)$ is the set of vertices within distance k of w in $G_{n,p}$.

Explanation. The $O(n^{-4})$ term is the probability the batch B_i is large. The term $\frac{O(\log^4 n)}{n-o(n)}$ in (13) arises as follows. We can assume that $|N_{\gamma \log_c n}(w)| \le n^{\gamma+\varepsilon}$, see (14) below. Suppose, as in Aronson et al. [1998], we expose the graph Γ at the same time that we run the Karp-Sipser algorithm. For us, it is convenient to work within the configuration model of Bollobás [1980]. (Note that in this case the multi-graph produced by the configuration model has an $\Omega(1)$ probability of being simple, see Section 2.3.1). Assume that we have exposed $N_{\gamma \log_c n}(w)$. At the start of the construction of a batch, we choose a random edge of the current graph. The probability this edge lies in $N_{\gamma \log_c n}(w)$ is $\le n^{\gamma+\varepsilon}/(n - o(n))$. In the middle of the construction of a batch, one endpoint of a pendant edge is known and then the other endpoint is chosen randomly from the set of configuration points associated with $\Gamma(t)$. The probability this new endpoint lies in $N_{\gamma \log_c n}(w)$ is also $\le n^{\gamma+\varepsilon}/(n-o(n))$ and there are only $O(\log^4 n)$ steps in the creation of a batch. This explains the term $\frac{O(\log^4 n)}{n-o(n)} n^{\gamma+\varepsilon}$.

It only remains to verify

$$\mathsf{P}(\exists w : |N_{\gamma \log_c n}(w)| \ge n^{\gamma+\varepsilon}) = O(n^{-10}). \quad (14)$$

To prove this, we can resort to proving the same inequality for $G_{n,c/n}$. This is an easy application of Chernoff bounds. We do a BFS from w until the first time that the breadth first tree has $\ge \log n$ leaves. Given a maximum degree of $o(\log n)$ we see that there will at this time be $o(\log^2 n)$ leaves. If there are never $\ge \log n$ leaves, then, $|N_{\gamma \log_c n}(w)| = o(\log^3 n)$ and so assume it does. If there are ℓ_s leaves at depth s, then the number of leaves at depth $s + 1$ in the BFS tree is bounded by $Bin(\ell_s n, p)$ where $p = \frac{c}{n}$ and so

$$\mathsf{P}(\ell_{s+1} \ge 2c\ell_s) \le e^{-c\ell_s/3} \le n^{-c/3}$$

and so with probability $1 - O(n^{-10})$ say, the tree will grow multliplicatively at a rate of less than $2c$ per round. In which case $|N_{\gamma \log_c n}(w)| \le (2c)^{\gamma \log_c n} \le n^{\gamma+\varepsilon}$. □

Early pendant vertices can be a nuisance at the front of a tree. It follows from Lemma 3.1, Rule 2(ii), that if v is early, then there could be relatively few choices for u so that we can find edge (u, v) in an unopened box. The next lemma puts a bound on the number of early vertices at the front.

LEMMA 4.3. *Let T be an augmenting tree of size $|T| = \Omega(n^{.5})$. Then, with high probability, there are at most $|F|/n^{2\varepsilon}$ early vertices at the front F of the tree.*

PROOF OF LEMMA 4.3. Suppose first that T is in Stage 1 and that there are $s = \frac{|T|}{n^{3\varepsilon}}$ early vertices at the front of the augmenting tree T. Let F' be the set of nodes in the tree at distance $\gamma \log_c n$ from the front. Consider the subtree T' formed

by F' and the paths from nodes in F' to the root. By Lemma 2.1, if T' underwent $\frac{\gamma}{2}\log_c n$ spurts, its front would increase by a factor of at least

$$\left(\frac{9c}{10}\right)^{\frac{\gamma}{2}\log_c n} \geq n^{\gamma/3} \tag{15}$$

for c large enough. This shows that $|F'| \leq |T|/n^{\gamma/3}$. Since there are at least $|T|/n^{3\varepsilon}$ early vertices on the front, there exists a node v in F' that is an ancestor to at least $n^{\gamma/3-3\varepsilon}$ early vertices. Each batch contains at most $\log^4 n$ vertices so there are at least $n^{\gamma/3-3\varepsilon-o(1)}$ early vertices from distinct batches. By the definition of F', the batches are at a distance of at most $\gamma\log_c n$ from v and so by Lemma 4.2 the probability of this event can be bounded above by n^{-5}.

Now assume that T is in Stage 2 and that r levels have been added since the start of Stage 2. This is the interesting case, but the argument for Stage 1 will come in handy. The problem is that (15) need not be true, as we do not grow from unexposed vertices. Let $T^* \supseteq T$ denote the tree we would grow if we had also grown from unexposed vertices in Stage 2. Let $f_0, f_1, \ldots, f_r$ be the front sizes in T from the start of Stage 2 and let $f_0^*, f_1^*, \ldots, f_r^*$ denote the sizes of the corresponding fronts F_i^* in T^*. Here, $f_0 = f_0^*$ is the size of the front at the end of Stage 1. The argument already given shows that there are at most $f_i^*/n^{3\varepsilon}$ early vertices at the front of the ith such level of T^*. Now, under the worst-case assumption that the early vertices of F_i^* are all unexposed, we see that

$$\text{there at most } n^{-3\varepsilon}\textstyle\sum_{i=1}^{r} f_i^* \text{ early vertices in the front } F \text{ of } T. \tag{16}$$

Now we have

$$\sum_{i=1}^{r} f_i^* \leq \sum_{i=1}^{r}\left(\frac{11c}{10}\right)^i f_0 \text{ and } f_r \geq \left(\frac{3c}{4}\right)^r f_0.$$

Using Lemma 2.4(d), this implies that

$$\sum_{i=1}^{r} f_i^* \leq \left(\frac{44}{30}\right)^r f_r \leq \left(\frac{44}{30}\right)^{\log_{3c/4} n} f_r \leq n^{\varepsilon} f_r.$$

The lemma follows from this and (16). □

4.2. GOOD MATCHING EDGES.

LEMMA 4.4. *There are with high probability* $\Omega(\frac{n^{.8+10\varepsilon}}{\log^4 n})$ *good matching edges in* Γ.

PROOF OF LEMMA 4.4. As already observed, Lemma 4.1 shows that there are $\Omega(\frac{n^{.8+10\varepsilon}}{\log^4 n})$ times $t \leq \tau_0$ when $v_1(t) = 0$. Consider exposing the ordering of the edges of the graph as we remove edges from the graph. Thus, at time t, all edges in $\Gamma \setminus \Gamma(t)$ have been revealed. When $v_1(t) = 0$ an edge is picked as a matching edge and must be in the available box of lowest index. Then, for both endpoints, we reveal the indices of edges incident to the endpoints. The edges of lowest index for each vertex become the witness edges. Note that at this point in time the contents of at most $O(n^{.8+10\varepsilon})$ boxes have been revealed. Since each endpoint has at least one edge incident to it the index of the witness edge is in $[\mu/2]$ with probability at least

$\frac{1}{2} - o(1)$. This shows that the regular edge created at this time is a good matching edge with probability at least $\frac{1}{4} - o(1)$, independently of the previous history. Thus, the actual number of good matching edges dominates a binomial with expectation $\Omega(\frac{n^{.8+10\varepsilon}}{\log^4 n})$. □

4.3. PUTTING IT ALL TOGETHER. We show that with high probability in each execution of Augmentpath the algorithm will find an augmenting path and will find one by exposing at most $O(n^{.6-\varepsilon})$ new vertices.

4.3.1. *Running Time.* The size of the front of an augmenting tree in the ith execution of Augmentpath will be at most $O(in^{.6-\varepsilon})$ since we claim that with high probability we only explore $O(n^{.6-\varepsilon})$ unexposed vertices in an execution of Augmentpath. Thus all fronts can easily be fitted into an array of size n. This also implies that the amount of work done in the ith execution of Augmentpath is $O(((i-1)n^{.6-\varepsilon} + n^{.6-\varepsilon}\log n)\log n)$, since we could in the worst case visit all previously exposed vertices and the maximum degree is $o(\log n)$ with high probability. The final $\log n$ accounts for the overhead from using more sophisticated data structures as discussed in Section 2.1.

So the total work would be $\tilde{O}(l^2 n^{.6-\varepsilon}) = \tilde{O}(n^{1-\varepsilon}) = o(n)$, where l is the total number of executions of AugmentPath and $l = \tilde{O}(n^{.2})$ since the size of I_2 is $\tilde{O}(n^{.2})$ with high probability.

4.3.2. *Proof of Theorem* 1.1. Consider the ith execution of Augmentpath and assume we have only exposed $O(i \cdot n^{.6-\varepsilon}) = o(n^{.8})$ vertices. By Lemma 2.4, we know that with high probability the algorithm will be able to grow the trees by a factor $\geq 3c/4$ in each execution of Augmentpath. If we have found an augmenting path before both trees are in Stage 2, then at most $O(n^{.6-\varepsilon})$ new vertices have been exposed. We will now show that during Stage 3, the algorithm will find an augmenting path and expose at most $O(n^{.6-\varepsilon})$ new vertices in the process.

We know from Lemma 2.4 that T_u, T_v will enter Stage 3. Let F_u and F_v be the unexposed vertices on the fronts of T_u and T_v respectively, and let S_u and S_v be two sets of size $n^{.6-2\varepsilon}$ selected randomly from the fronts. Now F_u and F_v represent at least an $n^{-\varepsilon}$ fraction of the size of the front, and with high probability at most an $n^{-2\varepsilon}$ fraction of the front can be early vertices. So in the worst case at most an $n^{-\varepsilon}$ fraction of F_u and F_v are early vertices. Since S_u and S_v are selected at random from F_u and F_v at least half of S_u and S_v are late vertices with probability at least $1 - o(n^{-1})$.

We now show that AugmentPath can find an augmenting path by a simple search. Let A be the set of good matching edges whose endpoints are unexposed. Lemma 4.4 implies that initially we have at least $n^{.8+9\varepsilon}$ good matching edges and at most $o(n^{.8})$ vertices have been exposed, we know that $|A| = \Omega(n^{.8+9\varepsilon})$.

If there exists an $x \in S_u$, $y \in S_v$ and $(a, b) \in A$ such that $(x, a), (b, y) \in E$ then $(x, a), (a, b), (b, y)$ forms an augmenting path with the paths from x and y to their roots. AugmentPath will do a simple search for such a quadruple x, a, b, y.

Let A_u be the set of edges in A whose vertices are adjacent to S_u. It follows from Lemma 3.1 that an available edge (x, a), $x \in S_u$ and a incident to A can go into any one of the unopened boxes after $\mu/2$. At most $2n + o(n)$ edges have been revealed. Revealing the contents of the boxes one by one we have a lower bound of $\frac{1}{\binom{n}{2}}$ for the

probability of seeing any one fixed edge e, see (11). This is under the assumption that we have not already seen e in an earlier box, regardless of previously opened boxes. Suppose now that we open the boxes $\mu/2$ to $3\mu/4$ and stop when we have either exhausted all such boxes or have found $n^{.4+7\varepsilon}$ members of A_u. When we open a box, there is a probability of at least $(n^{.8+9\varepsilon}-n^{.4+7\varepsilon})n^{.6-2\varepsilon}/\binom{n}{2} \geq n^{.9\varepsilon-.6}$ of finding a new member of A_u. Thus, $|A_u|$ dominates $\{n^{.4+7\varepsilon}, Bin(\mu/4-2n-o(n), n^{.9\varepsilon-.6}\}$ and so with high probability $n^{.4+7\varepsilon}/2 \leq |A_u| \leq n^{.4+7\varepsilon}$. For each edge $a \in A_u$, let ξ_a be a vertex of a that is adjacent to S_u and let η_a be the other endpoint.

We now consider the edges in unopened boxes $3\mu/4$ to μ. For any unopened box, the probability it contains an edge from $\{\eta_a : a \in A_u\}$ to S_v is $\Omega(\frac{n^{.6-2\varepsilon}n^{.4+7\varepsilon}/2}{\binom{n}{2}})$. Thus, the number of such edges dominates $Bin(\mu/4-2n-o(n), n^{-1+5\varepsilon})$ and this is non-zero with probability $1-o(n^{-1})$. Because the maximum degree of a vertex is $o(\log n)$, we see that the search for x, a, b, y can be done in $O(n^{.6-2\varepsilon}\log^2 n) = O(n^{.6-\varepsilon})$ time.

This completes the proof of Theorem 1.1. □

5. *Conclusion*

We have shown that a maximum matching can be found in $O(n)$ expected time if the average degree is a sufficiently large constant. It is easy to extend this to the case where the average degree grows with n. It is much more challenging to try to extend the result to any constant c. Karp and Sipser [1981] showed that, if $c < e$, then with high probability Phase 1 leaves $o(n)$ vertices for Phase 2. In Aronson et al. [1998], it was shown that for $c < e$, only a few vertex disjoint cycles are left, with high probability. So the problematical range is $e \leq c < c_0$.

REFERENCES

ARONSON, J., FRIEZE, A., AND PITTEL, B. 1998. Maximum matchings in sparse random graphs: Karp-Sipser revisited. *Rand. Struct. Algor. 12*, 111–177.

BAST, H., MEHLHORN, K., SCHÄFER, G., AND TAMAKI, H. 2006. Matching algorithms are fast in sparse random graphs. *Theory Comput. Syst. 39*, 3–14.

BOLLOBÁS, B. 1980. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *Europ. J. Combinat. 1*, 311–316.

EDMONDS, J. 1965. Paths, trees and flowers. *Canad. J. Math. 17*, 449–467.

FRIEZE, A. M., AND PITTEL, B. 2004. Perfect matchings in random graphs with prescribed minimal degree. *Trends Math. 17*, 95–132.

KARP, R. M., AND SIPSER, M. 1981. Maximum matchings in sparse random graphs. In *Proceedings of the 22nd IEEE Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 364–375.

MCKAY, B. D. 1985. Asymptotics for symmetric 0-1 matrices with prescribed row sums. *Ars Combinatoria 19*, 15–25.

MICALI, S., AND VAZIRANI, V. 1980. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st IEEE Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 17–27.

MOTWANI, R. 1994. Average-case analysis of algorithms for matchings and related problems. *J. ACM 41*, 1329–1356.

RECEIVED AUGUST 2008; REVISED JANUARY 2010; ACCEPTED FEBRUARY 2010