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

Prasad Chebolu, Alan Frieze[*], Páll Melsted
Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh PA15213
U.S.A.

today

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.

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 [3] gave the first polynomial time algorithm for the problem. Micali and Vazirani [6] 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 [7] and Bast, Mehlhorn, Schäfer and Tamaki [2] 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 work 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 [5]. They describe a simple greedy algorithm and show that **whp** it will in linear time produce a matching that is within $o(n)$ of the maximum. Aronson, Frieze and Pittel [1] proved that **whp** the Karp-Sipser algorithm is off from the maximum by at most $\tilde{O}(n^{1/5})$. In this paper we show that **whp** we can take the output of the Karp-Sipser algorithm and augment it in $o(n)$ time to find a truly maximum matching. Our failure probability will be much smaller than $o(1/\log n)$ and so we get a linear expected time algorithm if we back it up with the algorithm from [2]. We will define an algorithm Match and prove

Theorem 1 *Let $m = 2cn$ 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.*

[*]Supported in part by NSF Grant CCF-0502793

1.1 The Karp-Sipser algorithm

This is a simple greedy algorithm. If the current graph G has a vertex of degree one, 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 and continues. If the current graph has minimum degree two then it picks a random edge (u, v), adds this to the matching and deletes u, v and continues. The algorithm stops when G has no edges. Algorithm 1 below is a formal description.

Algorithm 1 Karp-Sipser Algorithm

```
1: procedure KSGreedy($G$)
2:     $M \leftarrow \varnothing$
3:     while $G \neq \varnothing$ do
4:         if $G$ has vertices of degree 1 then
5:             Select a vertex $v$ uniformly at random from the set of vertices of degree 1
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:        $M \leftarrow M \cup (v, u)$
11:        $G \leftarrow G \setminus \{v, u\}$
12:    end while
13:    return M
14: end procedure
```

We identify two phases in the execution of the Karp-Sipser algorithm. Phase one starts at the beginning and ends when the current graph has minimum degree 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, i.e. no mistakes have been made so far.

Let the current graph at the beginning of Phase 2 be denoted by G'. As shown in [5], almost all vertices of G' are matched by the Karp-Sipser algorithm when G is a random graph. This result was improved in [1] to show in fact that **whp** all but $\tilde{O}(n^{1/5})$ vertices of G' are matched. When G is distributed as $G_{n,m}$ then G' is distributed as $G^{\delta\geq 2}_{\nu,\mu}$ i.e. G' has $\nu = \Omega(n)$ vertices and $\mu = \Omega(m)$ edges and $G^{\delta\geq 2}_{\nu,\mu}$ is uniformly chosen from simple graphs with ν vertices, μ edges and minimum degree ≥ 2. Here the values μ, ν are random variables which are concentrated around their (asymptotically) known means. It was further shown in Frieze and Pittel [4] that **whp** G' consists of a single giant component plus a collection of vertex disjoint cycles. The expected number of vertices on isolated cycles is $O(1)$. It is shown that **whp**, a maximum cardinality matching of G' matches every vertex except one for each isolated odd cycle and one vertex if the giant component of G' is odd. (This is an existence result, non-algorithmic). So, after running the Karp-Sipser algorithm and dealing with isolated odd cycles, our task will **whp** be to match together $\tilde{O}(n^{1/5})$ isolated vertices.

1.2 Outline Description of Match

We will take the output of the Karp-Sipser algorithm, remove small cyclic components and deal with them separately. We then take the isolated vertices in pairs and try to match them together using alternating paths. We will show that this can be done in $o(n)$ time **whp**. Our augmenting path phase will use all of the edges of the graph. The reader will be aware that the Karp-Sipser algorithm has conditioned the edges of the graph. We will show however that we can find a large set of edges A and show that they have an *understandable* conditional distribution. This distribution will be simple enough that we can make use of A to show that we succeed **whp**. 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. Oft 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 on the sub-graph will leave extra isolated vertices.

Algorithm 2 below is a formal description of Match.

Algorithm 2 Algorithm Match

```
1: procedure MAIN($G$)
2:     $(M, G') \leftarrow$ KS-Greedy($G$)                ▷ $G'$ is the graph $G$ after Phase 1 of KS-Greedy
3:     $G' \leftarrow$ Remove-small-components($G'$)
4:     $M' \leftarrow$ Augment($G', M \cap G'$)
5:     return $M \triangle M'$
6: end procedure
1: procedure REMOVE-SMALL-COMPONENTS($G$)
2:     Pick an arbitrary vertex $u \in G$ and run a breadth first search starting from $u$
3:     if the connected component containing $u$, $C_u$, has size less than $\log^2 n$ then
4:         $G \leftarrow G \setminus C_u$
5:     end if
6:     return $G$
7: end procedure
```

The rest of the paper is structured as follows: We describe our augmenting path algorithm in the next section. Then in Section 3 we discuss the conditioning imposed by the Karp-Sipser algorithm. Then in Section 4 we add the final touches to the proof.

2 Augmenting Path Algorithm

A vertex is said to be *unmatched* if it is not incident to a matching edge. Given an unmatched vertex u, an *augmenting tree* T_u will be a tree of even depth rooted at u such that an edge between vertices at depth $2k$ to $2k+1$ is not a matching edge and edges going from vertices at depth $2k+1$ to $2k+2$ are matching edges, for $k \geq 0$. We refer to the nodes at levels $2k$ as even nodes of the tree and nodes at level $2k+1$ as odd notes for $k \geq 1$. We refer to the leaves of T_u as the front of the tree. Our growth procedure ensures that the leaves are always even vertices.

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.

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).

Throughout the algorithm we will keep track of which vertices we have seen before, labeling some vertices as *exposed*. This is mainly to keep track of which vertices have "no randomness" left because we've seen all the vertices they are adjacent to.

The algorithm Augment(G, M) takes as input a graph G and a matching M. The algorithm runs in rounds, where in each round we try to find an augmenting path between two unmatched vertices. If such a path is found, the matching is augmented, if not the algorithm returns Failure and we resort to an alternate algorithm. This is repeated until there is at most one unmatched vertex left. Clearly, if the algorithm does not fail then it finds a maximum cardinality matching.

In each round of the algorithm two augmenting trees are maintained, T_u, T_v, which are rooted at two unmatched vertices u and v. The trees are grown one at a time 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 smaller of the two trees is grown, unless one tree has $\leq n^{.59}$ unexposed vertices at the front and the other has $> n^{.59}$ unexposed vertices at the front.

Suppose T_u is to be grown. Then for each vertex x on the front and each non-matching edge (x, y) we do one of the following:

1. If y belongs to neither of the trees and is matched, add it to T_u along with its matching edge (y, y')
2. If y is unmatched we have an augmenting path from u to y
3. 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
4. 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.
5. If y is an even vertex of T_u then (x, y) along with the paths from x, y to their most common ancestor in T_u form a blossom, append (x, y) to the list of blossom edges for u.
6. If y is an odd vertex of T_u we do nothing.

After examining all edges incident to x, label it as exposed.

If an augmenting path is found then the round is finished. If the tree doesn't grow we inspect first the list of hit edges and see if we can grow the tree using them. If there are no hit edges we inspect the list of blossom edges. For each blossom found contract the blossom into a supernode and add this supernode to the front of the tree and try to grow from there. If the tree still doesn't grow exit the round and fail. Examples of the 6 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.

2.1 Tree Expansion

We will show that there exist constants α_1, α_2 independent of c and a constant c_0 such that for $c \geq c_0$ and $m = 2cn$ the following Lemmas hold. The proofs of the first three lemmas are fairly standard and heavy on computation. We have moved them to an appendix. In the first three lemmas we implicitly condition on the values μ, ν and assume that these values are close to their expected values.

Lemma 2 *The following will hold with probability $1 - O(\frac{1}{n^2})$. For $G = G^{\delta \geq 2}_{\nu,\mu}$ and all matchings M of G and all augmenting trees T with $c^{-1}\alpha_1 \log n \leq |T| \leq n^{.99}$. T will expand to a new front of size $s \in \left[\frac{9c}{10}|T|, \frac{11c}{10}|T|\right]$*

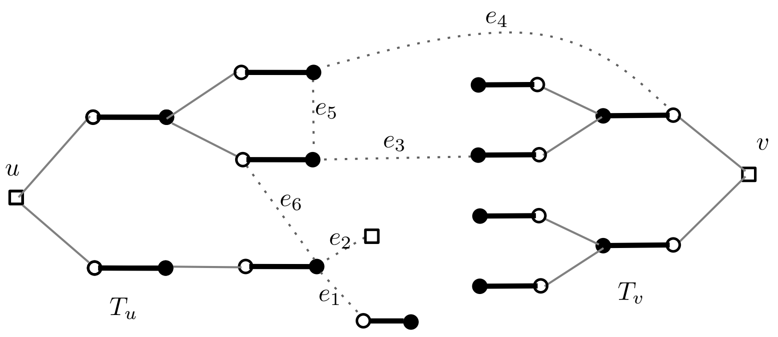

Figure 1: The trees T_u and T_v are shown with bold edges. The edges e_i correspond to cases i in the algorithm for $i = 1, \dots, 6$.

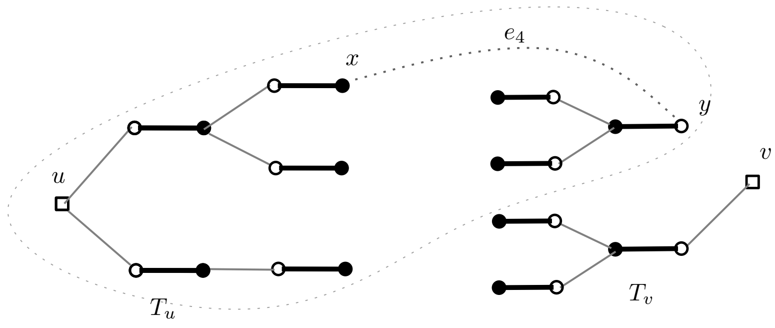

Figure 2: The trees T_u and T_v after using the hit edge e_4.

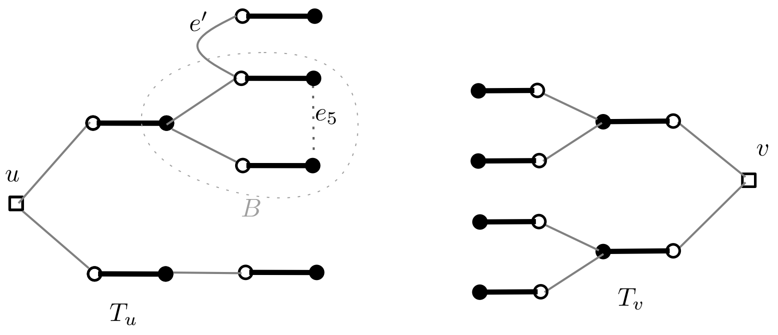

Figure 3: The trees T_u and T_v after using the blossom edge e_5. Note that the blossom B is contracted and the edge e' becomes a part of the new tree.

Lemma 3 *Let $G = G^{\delta \geq 2}_{\nu,\mu}$, then with probability $1 - \tilde{O}\left(\frac{1}{n^{1-\alpha_2}}\right)$ there do not exist 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 4 *Let $G = G^{\delta \geq 2}_{\nu,\mu}$, then with probability $1 - O(\frac{1}{n^2})$ there does not exist a set S with $\log n \leq |S| \leq n^{.99}$ that has more than $(1 + \epsilon)|S|$ edges inside S for all $\epsilon > 0$.*

Lemma 5 *Let G be a graph such that Lemmas 2, 3 and 4 hold. For all matchings M of G, if Augment(G, M) returns Failure, then the trees grown must be of size $\Omega(n^{.8})$.*

Proof of Lemma 5: Consider the two trees grown T_u and T_v where u, v are isolated vertices. If $Augment(G, M)$ fails then one of the trees must have an empty front, say T_u. We split the analysis into two cases depending on the size of $|T_u|$.

Case 1 $|T_u| \leq 4c^{-1}\alpha_1 \log_c n$

Note that this implies for large enough c that $|T_u| \leq \frac{1}{10}\alpha_2 \log_c n$. All of the edges of the front of T_u either go to within T_u or to odd vertices of T_v. If there are no hit edges then either we have violated the conditions of Lemma 3 or T_u is an isolated odd cycle (contradicting the fact that these are dealt with separately) or T_u is a blossom B. In the latter case, B will be shrunk to a supernode and T_u will be replaced by T_B, which will have to grow.

Now assume that we have at least one hit edge. Now either $V(T_u)$ contains a small cycle or there are at least two hit edges. But the latter case implies that $V(T_u \cup T_v)$ contains a small cycle. This is because we try to grow the smaller of the two trees. But then, after we have used the first hit edge, T_u must grow or we violate the condition of Lemma 3.

Case 2. $|T_u| \geq 4\alpha_2 \log_c n$

We now show that T_u will grow a new front of size at least $\frac{4c}{5}|T_u|$. Lemma 2 already shows that the size of the front is at most $\frac{11c}{10}|T_u|$. By Lemma 2 we know that the size of the front is at least $\frac{9c}{10}|T_u|$ when T_u is grown without considering T_v. But some of these vertices might be odd vertices of T_v. 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. Now T_A is contained within the tree T_v' which is T_v minus the front.

Consider the last time T_v' was grown and look at the rule used to decide which tree to grow.

Case a: If both or neither of the u-tree and T_v' had $\leq n^{.59}$ unexposed vertices, then the smaller one is grown and we have $|T_v'| \leq |T_u|$.

Case b: If exactly one of the u-tree and T_v' had $\leq n^{.59}$ vertices we know that it had to be T_v' since that was the tree grown. Then the u-tree contains a sub-tree that had $\leq n^{.59}$ unexposed vertices and was larger than T_v'. But then the previous level of T_v', call it T_v'' was smaller than the u-subtree and thus $|T_v''| \leq |T_u|$.

Now consider the set $S = T_u \cup A \cup T_A$, it has $|S| + |A|$ edges. In Case a we have $|T_A| \leq |T_v'| \leq |T_u| \leq |A| + |T_u|$. In Case b we have $|T_A| \leq |A| + |T_v''| \leq |A| + |T_u|$. This implies

$$|S| = |T_u| + |A| + |T_A| \leq 2(|T_u| + |A|) \leq 2\left(1 + \frac{10}{c}\right)|A|$$

so S is a set with $\frac{|A|}{|S|} \geq \frac{1}{2+\frac{20}{c}} \geq \frac{1}{3}$ fraction of extra edges, but this contradicts the result of Lemma 4. ■

3 Karp-Sipser conditioning

We now view G as an ordered set 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 original algorithm. We now state the modified Karp-Sipser algorithm. We assume the graph G' at the start of Phase 2 is given as $G = (e_1, \dots, e_\mu)$ an ordered set of μ edges.

We say that edge $e \in G$ has index i if it is the i-th edge in the list, i.e. $e = e_i$. 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

$G^{\delta\geq 2}_{\nu,\mu}$ as a family of ordered sets of edges. Furthermore, if c is large then μ/ν will be close to c, **whp**.

```
1: procedure KS*($G$)
2:     $M \leftarrow \varnothing$
3:     while $G \neq \varnothing$ 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:             Select an edge $e = (v, u)$ of lowest index in $G$
9:         end if
10:        $M \leftarrow M \cup (v, u)$
11:        $G \leftarrow G \setminus \{e\}$
12:    end while
13:    return M
14: end procedure
```

3.1 Witness edges

In addition to the edges of the matching we define edges based on the run 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 1 and 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, i.e. it was removed from the graph in line 8. For each of these vertices we define witness edges.

- For a regular vertex v, it 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.
- For a pendant vertex or an unmatched v. Find 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.
- For an unmatched vertex v, it 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 3 to pendant or from having degree 2 to degree 0 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 2ν.

We think of the graph 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. We wish to figure out what the unopened boxes could possibly contain. The following lemma provides necessary and sufficient conditions for a graph G to yield M and W as the output of KS*.

Lemma 6 *Let G be a graph such that the algorithm KS* will produce the matching set M and witness set W. Let $e' = (u, v)$ be an edge not in G that satisfies conditions 1,2,3 below. Then KS* will produce the same matching and witness set M and W when run on $G' = G \cup \{e'\}$. Furthermore if e is an edge of G that belongs neither to M nor W then KS* will produce the same matching and witness set when run on $G'' = G \setminus \{e\}$.*

1. *If both u and v are regular vertices and say u was removed from the graph before v then (u, v) can appear in any box that comes after the regular witness edge for u.*
2. *If u is a regular vertex and v is either a pendant or unmatched vertex . We need v to have degree at least 2 at the time when u is removed. Thus we need (u, v) to appear in a box that comes after the regular witness edge for u. Additionally if the pendant witness for v is incident to the matching edge of u we need (u, v) to appear in a box that comes after the pendant witness for v.*
3. *If neither u nor v are regular vertices, then edge (u, v) cannot appear in the graph.*

Proof of Lemma 6: To keep track of the algorithm KS* we let G_t denote the graph after the t-th iteration, so $G_0 = G$ and $G_T = \emptyset$ where T is the number of iterations. At timestep t let D_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' and G'' we define G'_t,G''_t etc. in the same manner.

We first deal with the case where the edge $e = (u, v)$ is added to G. Assume u is removed first from G at timestep $t_u + 1$, i.e. $u \in G_{t_u}$ and $u \notin G_{t_u+1}$. We first show that $M_t = M'_t$ for $t = 1, \dots, t_u$. 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 so 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 regular. Since e is not incident to a pendant vertex we have $D_t = D'_t$. If $D_t \neq \emptyset$ then we select the edge from D_t with minimum index, and add it to M_t, since $D_t = D'_t$ we have $M_{t+1} = M'_{t+1}$. If $D_t = \emptyset$ we select the edge in G_t of minimum index, which cannot be e since it comes after the regular witness edge for u. 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 regular vertex and v is either a pendant vertex or an unmatched vertex. Now u is removed from G before v and $\deg_{G_{t_u}}(v) \geq 2$. Neither u nor v are pendant vertices for $t \leq t_u$ so e is not incident with a pendant vertex, hence $D_t = D'_t$. Thus if $D_t \neq \emptyset$ we have $M_{t+1} = M'_{t+1}$ as before. If $D_t = \emptyset$ we have that e appears after the witness edge of u so it cannot be the edge of minimum index and thus we have $M_{t+1} = M'_{t+1}$ as before.

We have now shown that if e satisfies the given conditions then the same matching set M will be generated for both G and G'. We will now show that the same witness set is generated for both graphs. An edge can become a witness edge only when either one of its endpoints is removed, or when its degree drops below 2. Since G and G' differ only in the edge e, and e can only become a witness edge for u or v it is enough to show that e cannot become a witness edge given the conditions stated. We show this by looking at the same two cases.

Case 1: If u and v are regular and u is removed first then $e = (u, v)$ cannot be a witness edge for v. Now in G_{t_u} an edge (u, w) has the lowest index and becomes a matching edge, and the regular witness edge for u is taken to be the edge from $N_{G_{t_u}}(u) \setminus \{w\}$ of minimum index. In G'_{t_u-1} we have $N_{G'_{t_u}}(u) = N_{G_{t_u}}(u) \cup \{e\}$, but since e comes after the regular witness edge for u in G, e cannot be of minimum index in $N_{G'_{t_u}}(u) \setminus \{w\}$. Hence G and G' will have the same set of witness edges.

Case 2: If u is regular and v is either a pendant vertex or unmatched then as before e cannot be a regular witness for v, and since $\deg_{G_{t_u}}(v) \geq 2$ the only way for e to be a witness edge for v is if $\deg_{G_{t_u+1}}(v) = 1$, i.e. the edge (u, w) was chosen as a matching edge and (w, v) is in G. But then (w, v) is a pendant witness edge for v and in this particular case we have the extra restriction that e must come after (w, v) and so e cannot be chosen as the pendant witness edge of v.

Thus we have shown that e will not be a witness edge if it satisfies the conditions. Hence the graphs G and G' will generate the same witness and matching sets (in the same order, and for the same reasons). If e violates any of the conditions it will either not give the same set of matching edges or produce a different witness set.

Case 3: If u, v are both pendant vertices and u is matched before v then u has degree at least two at the time it is matched, contradiction. If u is pendant and v is unmatched then we draw the same conclusion. ■

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 interested in such cases.

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 and condition on the output matching edges M, as well as the witness edges W. Given the output M and W and Rules 1-3 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 list of edges E_i that could go into that box, from Rules 1-3 we see that this list depends only on M and W and is independent of the contents of other boxes. 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 $G^{\delta\geq 2}_{\nu,\mu}|M, W$, i.e. 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:      $G \leftarrow M \cup W$
6:      for unfilled boxes $i$ in increasing order do
7:          Select $e$ uniformly at random from $E_i$
8:          $G \leftarrow G \cup \{e\}$
9:          Remove $e$ from $E_j$ for all $j > i$
10:     end for
11:     return $G$
12: end procedure
```

Each G that outputs M and W can be generated with Generate-Random in exactly one way and that any graph G produced by Generate-Random will produce M and G when we run KS* on G. This shows that Generate-Random will gives a uniformly random graph from $G_{\nu,\mu}^{\delta\geq 2}|M,W$.

4 Final Proof

In Section 3.2 we gave a complete description of the probability space. However this is not enough to finish the proof of Theorem 1, we must dig deeper into the analysis of KSGreedy. We begin by listing some definitions and lemmas from the paper that we will need.

In [1] it is shown that $G(t)$ 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

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

where $f(z) = e^z - z - 1$. Conditional on $\vec{v}(t)$, the degrees of vertices of degree at least 2 is distributed as independent copies of a truncated Poisson random variable Z, where

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

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

As our input is taken from $G_{\nu,\mu}^{\delta\geq 2}$ we start in the state $\vec{v}(0) = (0, 0, \nu, \mu)$, i.e. 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 $t_1 < t < t_2$ we look at the edges and vertices removed from t_1 to t_2, i.e. the graph $G(t_1) \setminus G(t_2)$ and call it a *batch*. Note that each batch contains the regular matching edge removed at time t_1 and hence a batch is a connected set.

4.1 Good Matching edges

Let τ_0 be the last time such that the number of vertices removed from the graph is at most $n^{.99}$. We refer to vertices removed before τ_0 as *early vertices* and those removed after as *late*. We say that a matching edge e is a *good matching edge* if it is early, both of its endpoints are regular and the regular witness edges for both of its endpoints have index less than $\mu/2$.

Note that for $t = 0, \ldots, \tau_0$ we have only removed $n^{.99}$ vertices and $O(n^{.99})$ edges, thus

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

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

Lemma 7

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

and

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

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

This shows that for $t \leq \tau_0$ each batch corresponds to an interval of time of length at most $O(\log^3 n)$ and vertices of degree 1 (at the time of removal) and the total number of vertices is $O(\log^4 n)$. This also shows that during the first τ_0 time steps there will be many times when $v_1(t) = 0$ and thus regular edges are added to the matching set.

Lemma 8 *There are $\Omega\left(\frac{n^{.99}}{\log^4 n}\right)$ good matching edges in G.*

Proof of Lemma 8: First note that Lemma 7 shows that there are $\Omega\left(\frac{n^{.99}}{\log^4 n}\right)$ times $t \leq \tau_0$ when $v_1(t) = 0$. Now consider exposing the ordering of the edges of the graph as we remove edges from the graph. Thus at time t all edges in $G \setminus G(t)$ have been revealed. When $v_1(t) = 0$ an edge is picked as a matching edge and must be in the first 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 there are no restrictions on where the edges can go and at most $O(n^{.99})$ boxes have been revealed. Thus the edges are distributed uniformly at random over the available boxes. Since each endpoint has at least one edge incident to it the index of the witness edge is less than $\mu/2$ 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 $\frac{1}{4} - o(1)$. Thus the actual number of good matching edges dominates a binomial with expectation $\Omega\left(\frac{n^{.99}}{\log^4 n}\right)$. ∎

4.2 The Batch Graph

We split the edges removed up to time τ_0 into batches $B_1, \ldots, B_l$ and create a *Batch Graph* G_B, with vertices $B_1, \ldots, B_l$ and we put an edge between B_i and B_j if $\text{dist}(B_i, B_j) \leq 20 \log_c(\log n)$.

Lemma 9 *The probability that there exists a connected component in G_B of size at least 1000 is $O(n^{-4})$.*

Proof of Lemma 9: We claim that if k

$$\mathsf{P}(G_B \textit{ contains a component of size } \geq k) \leq \binom{l}{k} k^{k-2} \left(\frac{1}{n^{1-o(1)}}\right)^{k-1} \leq n^{1-.01k+o(1)}. \tag{1}$$

The lemma follows on taking $k = 1000$.

Explanation of (1)**:** We choose a tree T which spans a component in $\binom{l}{k} k^{k-2}$ ways. Order the vertices of T as $B_1, B_2, \ldots, B_k$ so that for each i, $B_1, B_2, \ldots, B_i$ spans a subtree T_i of T and B_i is of degree one in this tree. Then $n^{-1-o(1)}$ is the probability that random start vertex of batch B_i is close enough to the batch B_j where (B_i, B_j) is an edge of T_i. ∎

Lemma 10 *Let T be an augmenting tree with a front T of size $|T| = \Omega(n^{.02})$. Then,* **whp***, at there are at least $\frac{|T|}{\log n}$ late vertices on the front of the tree.*

Proof of Lemma 10: Given T, an augmenting tree with a front of size $|T| = t = \Omega(n^{.02})$ assume it has at most $\frac{t}{\log n}$ late vertices on the front. Let T' be the subtree of T, $10 \log_c(\log n)$ levels back, and let s be the size of the front of T'. From Lemma 2 we know that $s = \omega(\log n)$, otherwise the tree could not expand to a size of $\Omega(n^{.02})$ in only $O(\log \log n)$ steps. Thus we have

$$t \geq s \left(\frac{4c}{5}\right)^{5 \log_c(\log n)} \geq s \log^4 n$$

for large enough c and similarly

$$t \leq s\left(\frac{11c}{10}\right)^{5\log_c(\log n)} \leq s\log^6 n$$

so $s \in [\frac{t}{\log^6 n}, \frac{t}{\log^4 n}]$

For any vertex v at the front of T' consider the subtree of T rooted at v. **Whp**, it cannot contain early vertices from more than 1000 distinct batches since this would violate Lemma 9, and each batch is of size $O(\log^3 n)$. Thus each such subtree can contain only $O(\log^3 n)$ early vertices, there are s such subtrees and $\leq \frac{t}{\log n}$ early vertices which makes for a total of less than $O(s\log^3 n) + \frac{t}{\log n} = O\left(\frac{t}{\log n}\right) < t$ vertices at the front of T, a contradiction. ■

Lemma 11 *Let T be an augmenting tree with a front T of which $s = \tilde{\Omega}(n^{.03})$ are unexposed. Then there are $\tilde{\Omega}(\frac{s}{n^{.02}})$ good matching edges at the unexposed front, assuming the number of exposed vertices is $o(n^{.8})$.*

Proof of Lemma 11: We will assume that the algorithm runs for $\tilde{O}(n^{.2})$ rounds and prove that the condition holds with probability $1 - \exp^{-\Omega(n^{.01})}$ for every round.

In the first round, no vertices have been exposed. Given a set of $s = \Omega(n^{.02})$ unexposed vertices, we know that the previous level had at least $\frac{s}{11c/10}$ vertices, call that set of vertices S'. By Lemma 10, at least $\frac{|S'|}{\log n}$ vertices on the front are late vertices, call this set S''.

Consider any vertex $u \in S''$ and any good matching edge (x, y), such that x is unexposed. Since x is an early vertex and u is late the edge (u, x) is a potential edge for all currently open boxes that come after the regular witness edge for x. Except for at most $|S''|$ cases where a pendant witness for u is adjacent to y, we have $|S''|\tilde{\Omega}(n^{.99})$ potential edges going from S'' to unexposed good matching edges. We count the number of such edges that are between S'' and good matching edges and go into boxes with indices greater than $\mu/2$. Clearly for each such open box there are at most $\binom{\nu}{2}$ potential edges, so as we pick edges according to Algorithm 0 we choose one we're interested in with probability $\frac{|S''|\tilde{\Omega}(n^{.99})}{\binom{\nu}{2}}$. Therefore the number of such edges is lower bounded by a random variable $X \sim Bin\left(\Omega(n), \tilde{O}\left(\frac{|S''|}{n^{1.01}}\right)\right)$, since the number of filled boxes is $\leq 2\nu + o(n^{.8})$. The binomial X is at least $\tilde{\Omega}\left(\frac{|S''|}{n^{.02}}\right)$ with probability $1 - \exp^{-\Omega(n^{.01})}$ by standard large deviation bounds.

For later rounds, some unexposed vertices on the front might have exposed vertices as parents in the augmenting tree. Let S_1 be the unexposed vertices at the previous level (i.e. unexposed at the beginning of this round) and S_2 be the exposed vertices at the previous level. From Lemma 2 we know that $|S| \leq \frac{11c}{10}(|S_1| + |S_2|)$. If $|S_1| = \tilde{\Omega}(n^{.03})$ the above analysis show that we have $\tilde{\Omega}(\frac{|S_1|}{n^{.02}})$ good matching edges at the front. Suppose on the other hand that $|S_2| = \tilde{\Omega}(n^{.03})$. Since S_2 is at the front of the augmenting tree (one level back) we know that there must be at least $\frac{|S_2|}{\log n}$ late vertices in S_2, call this set S_2'.

Now each exposed vertex in S_2' was at some previous round an unexposed vertex at the front of it's augmenting tree, as shown before the number of edges going from late vertices in S_2' to good matching edges is $\tilde{\Omega}(\frac{|S_2'|}{n^{.02}})$ **whp** (the contribution from each round can be lower bounded by independent binomial random variables whose mean is $\tilde{\Omega}\left(\frac{|S_2'|}{n^{.01}}\right)$). Thus we have $\tilde{\Omega}\left(\frac{|S_2|}{n^{.02}}\right)$ good matching edges on the front.

Since $|S_1| + |S_2| = \Omega(|S|)$ and $|S| = \tilde{\Omega}(n^{.03})$ we have shown that there are $\tilde{\Omega}\left(\frac{|S_1|+|S_2|}{n^{.02}}\right) = \tilde{\Omega}\left(\frac{|S|}{n^{.02}}\right)$ good matching edges on the front. Since we only run for $\tilde{\Omega}(n^{.2})$ rounds the result holds for with probability $1 - n^{-a}$ for any constant $a > 0$. ■

4.3 Putting it all together

Proof of Theorem 1: We show that in each round the algorithm will always find an augmenting path and will find one by exposing at most $\tilde{O}(n^{.59})$ new vertices. This implies that the amount of work done in the i-th round is $\tilde{O}(i \cdot n^{.59})$, since we could in the worst case visit all previously exposed vertices. So the total work would be $\tilde{O}(l^2 n^{.59}) = \tilde{O}(n^{.99}) = o(n)$, where l is the total number of rounds and $l = \tilde{O}(n^{.2})$ since the number of unmatched vertices is $\tilde{O}(n^{.2})$.

Consider the i-th round and assume we've only exposed $\tilde{O}(i \cdot n^{.59}) = o(n^{.8})$ vertices. By Lemma 5 we know that **whp** the algorithm will be able to grow the trees and that we can assume that the trees T_u and T_v both have at least $n^{.59}$ unexposed vertices at the front. If the algorithm found an augmenting path before the first time this happened then we've exposed only $\tilde{O}(n^{.59})$ vertices, since there are at most $O(\log n)$ levels of the tree and each level has $\leq n^{.59}$ unexposed vertices. Assume therefore that we have two sets of unexposed vertices at the fronts S_u and S_v such that $|S_u|, |S_v| \geq n^{.59}$.

Since only $o(n^{.8})$ vertices have been exposed so far, Lemma 11 applies and there are at least $\tilde{\Omega}\left(\frac{|S_u|}{n^{.02}}\right)$ vertices in S_u that are endpoints of good matching edges, call this set S'_u. S'_v is defined similarly. Thus we have that there are at least $|S'_u||S'_v| = \tilde{\Omega}(n^{1.14})$ potential edges that could go in any of the $\mu/2 - 2\nu - o(n^{.8})$ currently open boxes with index $\geq \mu/2$. This follows from Lemma 6. (*The endpoints of a good matching edge are regular and they and their witnesses appear before* $\mu/2$.) The number of such edges dominates a binomially distributed random variable $X \sim Bin\left(\Omega(n), \tilde{\Omega}\left(\frac{n^{1.14}}{\binom{n}{2}}\right)\right)$, which has a mean of $\tilde{\Omega}(n^{.14})$ and thus is positive with probability $1 - \exp^{-\Omega(n^{.13})}$. This edge going between the fronts will guarantee that an augmenting path is found by inspecting either one of the augmenting trees. Thus the probability that we fail in the ith round is at most $\exp^{-\Omega(n^{.13})}$.

Since we repeat this for $\tilde{O}(n^{.2})$ rounds the probability of failure is $O(n^{-a})$ for any constant $a > 0$. ■

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 showed that if $c < e$ then **whp** Phase 1 leaves $o(n)$ vertices for Phase 2. In the paper [1], it was shown that for $c < e$, only a few vertex disjoint cycles are left, **whp**. So the problematical range is $e \leq c < c_0$.

References

[1] J. Aronson, A. M. Frieze and B. Pittel, Maximum matchings in sparse random graphs: Karp-Sipser revisited, *Random Structures and Algorithms 12 (1998) 111-177.*

[2] H. Bast, K. Mehlhorn, G. Schäfer and H. Tamaki, Matching Algorithms are Fast in Sparse Random Graphs, *Theory of Computing Systems* 39 (2006) 3-14.

[3] J. Edmonds, Paths, Trees and Flowers, *Canadian Journal of Mathematics* 17 (1965) 449-467.

[4] A. M. Frieze and B. Pittel, Perfect matchings in random graphs with prescribed minimal degree, *Trends in Mathematics*, Birkhauser Verlag, Basel (2004) 95-132.

[5] R.M. Karp and M. Sipser, Maximum Matchings in Sparse Random Graphs, *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science* (1981) 364-375.

[6] S. Micali and V.V. Vazirani, An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs. Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (1980) 17-27.

[7] R. Motwani, Average-case Analysis of Algorithms for Matchings and Related Problems, *Journal of the ACM* 41 (1994) 1329-1356.

Appendix

Proof of Lemma 2: We show that matching trees of size l, where $\frac{1}{\alpha_1 c}\log n \leq l \leq n^{0.9}$, expand at a steady rate close to c. In a matching tree, we denote by *Odd(T)* and *Even(T)* the set of vertices at odd and non-zero even depth, respectively. Every vertex in *Odd(T)* has exactly one child in *Even(T)*, in particular $|Odd(T)| = |Even(T)|$. We denote the neighborhood set of *Even(T)* by $\Gamma(Even(T)) \supset Odd(T)$. We also note that the matching tree can be represented using only *Even(T)* by contracting the matching edges.

We will first show that the event of a tree having $|Even(T)| = l$ and $|\Gamma(Even(T))| = r$, where $\frac{1}{\alpha_1 c}\log n \leq l \leq n^{0.9}$ and $l \leq r \leq 0.91cl$, 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 Γ(*Even(T)*) $\setminus$ *Odd(T)* and (iii) none of the $(l+1)(n-r-l-1)$ edges between *Even(T)* $\cup$ {root vertex} and $V \setminus \Gamma$ are present. The probability of the above event occurring in $G^{\delta\geq 2}_{n,m}$ is bounded above by

$$\sqrt{\nu}\left(\frac{1}{2\mu - 2(l+r)+1}\right)^{l+r} \times$$
$$\sum_{\substack{d_i\geq 2,\, i\in[r+l+1] \\ \sum_{i=1}^{l+1} d_i = r+l}} \left(\prod_{i=1}^{l+1} \frac{\lambda^{d_i} d_i!}{d_i!(e^\lambda - 1 - \lambda)} \prod_{i=l+2}^{2l+1} \frac{\lambda^{d_i}(d_i)_2}{d_i!(e^\lambda - 1 - \lambda)} \prod_{i=2l+2}^{r+l+1} \frac{\lambda^{d_i}(d_i)_1}{d_i!(e^\lambda - 1 - \lambda)}\right)$$

Explanation: In $G^{\delta\geq 2}_{\nu,\mu}$, the degrees of the vertices are distributed as truncated Poisson random variables with parameter λ where

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

(If c is large then Phase 1 removes relatively few edges).

The variables are truncated below two and are conditioned on the sum of the degrees of the vertices adding up to 2μ, see [1]. We will have to pay a factor of $\sqrt{\nu}$ for removing the conditioning. Given the degree sequence we make our computations in the configuration model. 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 $\frac{d_u d_v}{2\mu-2(l+r)+1}$. Hence, given the degree sequence, the probability that the matching tree exists is at most

$$\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}\prod_{i=1}^{l+1} d_i! \prod_{i=l+2}^{2l+1}(d_i)_2 \prod_{i=2l+2}^{r+l+1}(d_i)_1$$

where $(d_i)_k = d_i(d_i-1)...(d_i-k+1)$. We now simplify the expression obtained for the probability.

$$\begin{aligned}
&\sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}\times\\
&\sum_{\substack{d_i\ge 2,\, i\in[r+l+1]\\ \sum_{i=1}^{l+1} d_i=r+l}}\left(\prod_{i=1}^{l+1}\frac{\lambda^{d_i}d_i!}{d_i!(e^\lambda-1-\lambda)}\prod_{i=l+2}^{2l+1}\frac{\lambda^{d_i}(d_i)_2}{d_i!(e^\lambda-1-\lambda)}\prod_{i=2l+2}^{r+l+1}\frac{\lambda^{d_i}(d_i)_1}{d_i!(e^\lambda-1-\lambda)}\right)\\
&=\sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}\times\\
&\sum_{\substack{d_i\ge 2,\, i\in[r+l+1]\\ \sum_{i=1}^{l+1} d_i=r+l}}\left(\frac{\lambda^{r+l}}{(e^\lambda-1-\lambda)^{l+1}}\prod_{i=l+2}^{2l+1}\frac{\lambda^{d_i}(d_i)_2}{d_i!(e^\lambda-1-\lambda)}\prod_{i=2l+2}^{r+l+1}\frac{\lambda^{d_i}(d_i)_1}{d_i!(e^\lambda-1-\lambda)}\right)\\
&=\sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}\times\\
&\sum_{\substack{d_i\ge 2,\, i\in[r+l+1]\\ \sum_{i=1}^{l+1} d_i=r+l}}\left(\frac{\lambda^{2r+2l}}{(e^\lambda-1-\lambda)^{r+l+1}}\prod_{i=l+2}^{2l+1}\frac{\lambda^{d_i-2}}{(d_i-2)!}\prod_{i=2l+2}^{r+l+1}\frac{\lambda^{d_i-1}}{(d_i-1)!}\right)\\
&\le\sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}\binom{r-2}{l}\frac{\lambda^{2r+2l}}{(e^\lambda-1-\lambda)^{r+l+1}}\ e^{\lambda l}\ (e^\lambda-1)^{r-l}\\
&\le\sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}\left(\frac{er}{l}\right)^{l}\frac{\lambda^{2r+2l}}{(e^\lambda-1-\lambda)^{r+l}}e^{\lambda l}(e^\lambda-1)^{r-l}\\
&=\sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}\left(\frac{er}{l}\right)^{l}(\tilde{c}\lambda)^r\left(\frac{\tilde{c}\lambda e^\lambda}{e^\lambda-\lambda-1}\right)^{l}\left(\frac{1}{e^\lambda-1}\right)^{l}\\
&\qquad\text{using } \frac{\lambda(e^\lambda-1)}{e^\lambda-1-\lambda}=\tilde{c}\\
&\le\sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}(\tilde{c}\lambda)^r\left(\frac{0.92e\tilde{c}^2\lambda e^\lambda}{e^\lambda-\lambda-1}\right)^{l}\left(\frac{1}{e^\lambda-1}\right)^{l}\\
&\text{using } \frac{r}{l}\le 0.91c
\end{aligned}$$

$$
\begin{aligned}
&\le \sqrt{\nu}\left(\frac{1}{2\mu-2(l+r)+1}\right)^{l+r}(\tilde{c}\lambda)^r\left(\frac{0.92e\tilde{c}^2\lambda}{e^\lambda-\lambda-1}\right)^l \quad \text{using } \frac{e^\lambda}{e^\lambda-1}<1.01\\
&\le \sqrt{\nu}\left(\frac{1}{\tilde{c}\nu}\right)^{l+r}e^{4(l+r)^2/\tilde{c}\nu}(\tilde{c}\lambda)^r\left(\frac{0.92e\tilde{c}^2\lambda}{e^\lambda-\lambda-1}\right)^l \quad \text{using } 2\mu=\tilde{c}\nu\\
&= \sqrt{\nu}\left(\frac{1}{\nu}\right)^{l+r}e^{4(l+r)^2/\tilde{c}\nu}\,\lambda^r\left(\frac{0.92e\tilde{c}\lambda}{e^\lambda-\lambda-1}\right)^l \qquad (2)
\end{aligned}
$$

We now count the number of such configurations. We begin by choosing *Even(T)* and the root vertex of the tree in $\binom{n}{l+1}$ ways. We make the following observation about augmenting path trees with $|\mathit{Even(T)}|=l$. The removal of *Odd(T)* vertices from the tree, as illustrated in the diagram, would correspond to a unique combination of a tree on $l+1$ vertices and a sequence of l distinct vertices. We note, by Cayley's formula, that the number of trees that could be formed using $(l+1)$ vertices is $(l+1)^{l-1}$. We now choose the sequence of l vertices, *Odd(T)*, that connect up vertices in *Even(T)* in $(\nu-l-1)(\nu-l-1)...(\nu-2l)=(\nu-l-1)_l$ 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

$$
\binom{\nu}{l+1}(l+1)^{l-1}(\nu-l-1)_l\binom{\nu-2l-1}{r-l}l^{r-l}\le \nu^{r+l+1}\cdot e^{-(l+r)^2/4\nu}e^{r+1}\cdot\left(\frac{l}{r-l}\right)^{r-l}.
$$

Combining the bounds for probability and configurations, we get

$$
\begin{aligned}
&\nu^{r+l+1}\cdot e^{-(l+r)^2/4\nu}\,e^{r+1}\cdot\left(\frac{l}{r-l}\right)^{r-l}\sqrt{\nu}\left(\frac{1}{\nu}\right)^{l+r}e^{4(l+r)^2/\tilde{c}\nu}\,\lambda^r\left(\frac{0.92e\tilde{c}\lambda}{e^\lambda-\lambda-1}\right)^l\\
&= e\nu^{3/2}\cdot e^{-(l+r)^2/4\nu}\,e^r\cdot\left(\frac{l}{r-l}\right)^{r-l}e^{4(l+r)^2/\tilde{c}\nu}\,\lambda^r\left(\frac{0.92e\tilde{c}\lambda}{e^\lambda-\lambda-1}\right)^l\\
&= e\nu^{3/2}\cdot e^{-(\frac{1}{4}-\frac{4}{\tilde{c}})(l+r)^2/\nu}\,e^r\cdot\left(\frac{l}{r-l}\right)^{r-l}\lambda^r\left(\frac{0.92e\tilde{c}\lambda}{e^\lambda-\lambda-1}\right)^l\\
&\le e\nu^{3/2}\cdot\left(\frac{l}{r-l}\right)^{r-l}(e\lambda)^r\left(\frac{0.92e\tilde{c}\lambda}{e^\lambda-\lambda-1}\right)^l\\
&= e\nu^{3/2}\cdot\left(\frac{e\lambda l}{r-l}\right)^{r-l}\left(\frac{0.92e^2\tilde{c}\lambda^2}{e^\lambda-\lambda-1}\right)^l
\end{aligned}
$$

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

$$
\begin{aligned}
&e\nu^{3/2}\cdot\left(\frac{e\lambda l}{0.91cl}\right)^{0.91cl}\left(\frac{0.92e^2\tilde{c}\lambda^2}{e^\lambda-\lambda-1}\right)^l\\
&\le e\nu^{3/2}\cdot\left(\frac{e}{0.91}\right)^{0.91cl}\left(\frac{0.92e^2\tilde{c}\lambda^2}{e^\lambda-\lambda-1}\right)^l \quad \text{using } \lambda<c\\
&\le e\nu^{3/2}\cdot\left(\frac{e}{0.91}\right)^{0.91cl}\left(\frac{0.92e^2\tilde{c}\lambda^2}{e^{0.999c}}\right)^l \quad \text{using } e^\lambda-\lambda-1>e^{0.999c}\\
&\le e\nu^{3/2}\cdot\left(\frac{0.92e^2\tilde{c}\lambda^2}{e^{0.002c}}\right)^l \quad \text{using } \frac{e^{0.999}}{\left(\frac{e}{0.91}\right)^{0.91}}>e^{0.002}
\end{aligned}
$$

$$= e\nu^{3/2}q^l \qquad \text{where } q = \frac{0.92e^2\tilde{c}\lambda^2}{e^{0.002c}} \le e^{-.001c}.$$

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

$$2en^{5/2}q^{\frac{\alpha_1}{c}\log n} \le 2en^{5/2-\alpha_1/1000} = o(1)$$

for $\alpha_1 \ge 2501$.

We will now show that the event of a tree having $|Even(T)| = l$ and $|\Gamma(Even(T))| = r$, where $\frac{\alpha_1}{c}\log n \le l \le n^{0.9}$ and $r \ge 1.07cl$, is polynomially small.

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

The probability expression (2) remains the same with the 0.92 getting replaced by 1.08 and we get a bound of

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

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

$$2en^{3/2}q^{\frac{\alpha_1}{c}\log n} = 2en^{3/2-\alpha_1/1000} = o(1).$$

Hence, the probability that there exists a tree with $|Even(T)| = l$ and $|\Gamma(Even(T))| = r$, where $\frac{\alpha_1}{c}\log n \le l \le n^{0.9}$ and $r \notin [0.9cl, 1.1cl]$, is polynomially small. ■

Proof of Lemma 3: We show that this holds in $G_{n,m}$ and note that $G_{\nu,\mu}^{\delta\ge 2}$ is a vertex induced subgraph of $G_{n,m}$. Since the property is closed under edge deletion this will imply the Lemma. To upperbound the probability of failure in $G_{n,m}$ we switch to $G_{n,p}$ 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

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

The event in question is monotone and so we only have to inflate the probability by a constant to translate to $G_{n,m}$. ■

Proof of Lemma 4: We can work in $G_{n,p}$, as we did for Lemma 3.
We get the bound

$$\binom{n}{k}\binom{\binom{k}{2}}{(1+\epsilon)k}\left(\frac{c}{n}\right)^{(1+\epsilon)k} \leq \left(\frac{en}{k}\right)^k \left(\frac{ek^2/2}{(1+\epsilon)k}\right)^{(1+\epsilon)k} \left(\frac{c}{n}\right)^{(1+\epsilon)k} \leq \left(\frac{e^{3+\epsilon}(c)^{1+\epsilon}k^{\epsilon}}{n^{\epsilon}}\right)^k$$

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

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

The event in question is monotone and so we only have to inflate the probability by a constant to translate to $G_{n,m}$. ■