Matching Algorithms in the Sparse Stochastic Block Model

Anna Brandenberger[1], Byron Chin[1], Nathan Sheffield[1], and Divya Shyamal[1]

abrande@mit.edu, byronc@mit.edu, shefna@mit.edu, dshyamal@mit.edu
[1]Massachusetts Institute of Technology

August 2023

Abstract

The **stochastic block model** is a generalization of the Erdős–Rényi model of random graphs that describes the interaction of a finite number of distinct communities. In Erdős–Rényi graphs, it is known that a linear-time algorithm of Karp and Sipser achieves near-optimal matching sizes almost surely, giving a law-of-large numbers for the matching sizes of such graphs in terms of solutions to an ODE [10]. We provide an extension of this analysis, identifying broader ranges of stochastic block model parameters for which the Karp–Sipser algorithm achieves near-optimal matching sizes, but demonstrating that it cannot perform optimally on general SBM instances.

We also consider the problem of constructing a matching *online*, in which the vertices of one half of a bipartite stochastic block model arrive one-at-a-time, and must be matched as they arrive. We show that the competitive ratio lower bound of 0.837 found by Mastin and Jaillet for the Erdős–Rényi case [16] is tight, and that it can be achieved in the stochastic block model whenever the expected degrees in all label classes are equal. We propose several reasonable linear-time algorithms for online matching in the general stochastic block model, but prove that despite very good experimental performance, none of these achieve the online asymptotic optimal.

1 Introduction

1.1 A Motivating Fable

Suppose you're running a summer research program for undergrads – perhaps similar to MIT's SPUR. You have n students participating; each of them has checked a couple boxes indicating the sorts of background they have and the sorts of research areas they're interested in. You also have n grad students, each of whom has proposed a project to work on. These projects are divided by subject area (maybe you have some projects in statistics, some in combinatorics, etc.), but you expect that within a given subject area all projects are pretty comparable, so that the probability of any given student being interested in Algebraic Topology Project A is the same as the probability that they're interested in Algebraic Topology Project B. There's a couple questions you might ask yourself:

i) If all of the undergrads are really picky, how many undergrads do you expect to be able to give projects they're interested in? Clearly this should depend somehow on the sizes and probabilities of the populations; for instance, if all of the undergrads have indicated that they're die-hard set theorists, and you know that die-hard set theorists are very rarely interested in non-set theory problems, you might expect to not be able to please as many people as if the students had more varied interests.

ii) What if you have to interview the students to find out what they're interested in, and are expected to assign them to a project at the end of the interview? Here, you might make mistakes, because you could assign one person to a project before you discover later that it would be better to assign someone else. Are there reasonable heuristics you can use to avoid making too many mistakes? (For example, maybe if you have many more analysis projects available than you expect to have students willing to do them, you should try to always give students analysis projects when you can.)

This paper will examine both of these questions. Of course, there are many other similar-sounding stories that could be told about this problem – the key element is that you have enough information about your population to divide it up into a finite number of differently-behaving groups, but that you expect each of those groups to be reasonably homogeneous.

1.2 Problem Statement

The idea of a population divided into a fixed number of distinct but internally-homogeneous groups is captured by the **stochastic block model**, first proposed by Holland et. al. to model social networks [7].

Definition 1 (stochastic block model)**.** Let our vertices be divided into k disjoint sets S_1,...,S_k, which we think of as the "label classes". We also specify a probability matrix, associating a value p_{ij} to each pair i,j of labels. Given these parameters, the **stochastic block model** is the distribution over graphs obtained by adding each edge (u,v) with independent probability $p_{l_u l_v}$, where l_u and l_v are the labels of u and v.

We are interested in the size of a maximum-cardinality matching on a graph drawn from this distribution.

Definition 2. A **matching** of a graph is a subset M of the graph's edges such that no two edges in M involve the same vertex. (In other words, every vertex is matched with at most one partner.) The **matching number** of a graph G is defined as the maximum cardinality among matchings of G. A matching is called "perfect" if it involves every vertex of G.

We are also interested in the problem of constructing a matching **online**.

Definition 3 (Online bipartite matching problem)**.** Suppose we have a bipartite instance of the stochastic block model, where there are n "left" vertices and n "right" vertices, the vertex classes on the left have zero connection probability to the other left classes, and likewise the right classes have no connection probability to each other. An "online" algorithm is given the labels of all the right vertices, and knows a distribution over the left vertices. For each of n time-steps, a new left-node is revealed (a left label is picked from the distribution, and coins are flipped to determine which edges it has to the right vertices) – the algorithm then decides which if any of these edges to add to M. Once it has made this decision and moved on, it is never allowed to revisit that vertex.

When it comes to the questions we're asking, the interesting range of stochastic block model parameters turns out to be when all probabilities p_{ij} are equal to c_{ij}/n for some constant c_{ij}. The reason for this is because, when p_{ij} grows faster than $1/n$, in the limit of n the graph becomes dense enough that it's easy to find a perfect matching between S_i and S_j. On the other hand, when p_{ij} grows slower than $1/n$, the graph becomes so sparse that we can include almost every edge in M. So, we'll assume that all p_{ij} are c_{ij}/n.

1.3 Background and Previous Work

In 1981, Karp and Sipser demonstrated that a simple linear-time heuristic achieved found matchings within $o(n)$ of the true matching number of Erdős–Rényi graphs (i.e. stochastic block model with only a single block). By associating the performance of that algorithm with a deterministic procedure, they were able to prove a law of large numbers on the matching number of such graphs [10]. That paper has prompted continued investigation into their algorithm. In 1998, Frieze, Pittel, and Aronson proved a central limit theorem on how far the Karp–Sipser algorithm gets from the matching number in Erdős–Rényi graphs. In 2011, Bohman and Frieze extended analysis of the Karp–Sipser algorithm to the model of graphs drawn uniformly over a fixed degree sequence, showing that a log concavity condition is sufficient for the algorithm to find near-perfect matchings in such graphs [2]. Because of its simplicity, the Karp–Sipser algorithm has also received attention as a practical method for data reduction; some recent work investigates efficient implementations [12, 14].

The online bipartite matching problem was first introduced by Karp, Vazirani and Vazirani in 1990; they showed that a tight $1-1/e$ competitive ratio was possible on worst-case inputs [11]. In 2009, Feldman et. al. showed that in the model where the left vertices are instead drawn from an arbitrary known distribution, with integral expected arrival rates, it is possible to get a competitive ratio strictly better than $1-1/e$ [6]. The problem of online matching in the known distribution model has since seen considerable attention because of applications in internet ad allocation; for arbitrary distributions (with arbitrary arrival rates), the best known algorithm achieves competitive ratio 0.716, and there is a known upper bound of 0.823 [9, 15]. There has also been work considering algorithms for specific left vertex distributions. Mastin and Jaillet found that in $G(n,n,p)$ (random bipartite graph where all edges are independent and equally likely to exist) all greedy algorithms achieve competitive ratio at least 0.837 [16]. Sentenac et. al. studied the problem in the 1-dimensional geometric model, where they found expressions for both the true matching size and the performance of a particular online heuristic [18]. To the best of our knowledge, the only previous work that considers the stochastic block model is by Soprano-Loto et. al., who consider the regime where the graph is dense (i.e. all probabilities are constants not depending on n), and characterize when it is possible to achieve an asymptotically near-perfect matching [19].

1.4 Summary of Our Contributions

In the first section of the paper, we consider behaviour of the Karp–Sipser algorithm on stochastic block model graphs. We show that it does not achieve near-optimal matchings in general, but that it does for probability matrices satisfying any of the following conditions:

- **Equitable**: When $\sum_j c_{ij}|S_j|$ is the same for all i, we are in the "equitable" case – we show that the asymptotic matching number here is the same as that of a sparse Erdős–Rényi graph $G(n,c/n)$ with parameter $c=\frac{\sum_j c_{ij}|S_j|}{n}$. In particular, this captures the standard symmetric p_{in}-p_{out} model, in which the population is divided into equal-sized groups, and there are distinct probabilities for connecting within-group as opposed to across-group.
- **Sub-Critical**: When $\sum_j c_{ij} < e$ for all i, we are in a sub-critical case similar to the one identified for Erdős–Rényi graphs, and we can identify the asymptotic matching number with the solution of an ODE.
- **Bipartite Erdős–Rényi**: We also determine in terms of the solution of an ODE the asymptotic matching number of $G(kn,n,c/n)$, the bipartite graph with part sizes kn and n and independent edge probability c/n.

As a particular special case, this implies that Mastin and Jaillet's upper bound on the matching number of $G(n,n,p)$ is tight, and so their 0.837 competitive ratio lower bound is also tight.

In the second section of the paper, we consider the online version of the problem. We identify the following algorithms:

- **DUMB-GREEDY**: When a match is possible, match to a uniform random neighbour.
- **DEGREEDY**: Match to the available class with the lowest expected degree.
- **SHORTSIGHTED**: Match to the available class that maximizes the probability of being able to match on the next step.
- **BRUTE-FORCE**: Precompute the winning probability of every possible state, and then always match to the available class that maximizes winning probability (this gives optimal performance among all online algorithms, but is very costly)

In the equitable case, we find that all of these algorithms achieve the same tight 0.837 competitive ratio as in $G(n,n,p)$. In the general setting, we show that only BRUTE-FORCE achieves optimal performance. We conjecture based on experimental results that SHORTSIGHTED achieves competitive ratio very close to that of BRUTE-FORCE, but are unable to give good lower bounds outside of special cases.

2 Offline Setting

In 1981, Karp and Sipser proposed an algorithm for approximating the matching number of graphs, and proved that it achieves near-optimal matching sizes on Erdős–Rényi graphs [10]. By extending the analysis of their algorithm to the stochastic block model case, we hope to show a law of large numbers on the matching size of a more general class of graph.

2.1 Karp–Sipser Algorithm

The form of the Karp–Sipser algorithm we will study is as follows:

Algorithm 1 Karp–Sipser

$M \leftarrow \emptyset$
$V \leftarrow V(G)$
while $E(G \cap V) \neq \emptyset$ **do**
    if exists $v \in V$ of degree 1 – that is, such that exactly one $u \in V$ has $(uv) \in E(G)$ **then**
        Choose such a v uniformly at random over all degree 1 vertices
        $M \leftarrow M \cup \{(uv)\}$
        $V \leftarrow V \setminus \{u,v\}$
    else
        Choose an edge (uv) uniformly randomly over all $u,v \in V$, $(uv) \in E(G)$
        $M \leftarrow M \cup \{(uv)\}$
        $V \leftarrow V \setminus \{u,v\}$
Return M

In other words, whenever there exists a vertex of degree 1 in the graph, add its edge to the matching and remove both endpoints. When you can't do that, instead just add an edge at random. When there exist degree 1 vertices it's always "safe" to add their edges in the sense that there always exists an optimal matching which does so, and so any "mistakes" the algorithm makes can only happen after the first time you've reached 0 degree 1 vertices (we call the steps before this "Phase 1", and the steps after this "Phase 2"). The analysis of this algorithm in the Erdős–Rényi setting proceeds as follows:

i) Argue that, conditioning on the state of a Markov process on small tuples of integers, the graph maintains a simple distribution law even after several steps of the algorithm.
ii) Use estimates of the degree distribution of the graph to determine transition probabilities for that Markov process.
iii) Appeal to theorems from analysis to argue that the Markov process stays close to the solution of a corresponding ODE with high probability in the limit of the size of the graph.
iv) Observe that, in Phase 2 of the algorithm, it's very likely that the algorithm finds a matching within $o(n)$ vertices of perfect on the remaining graph. Since the algorithm makes only optimal decisions in Phase 1, this means that overall it finds within $o(n)$ vertices of the true optimal, and so the true matching number of the graph is described by solutions to the ODE from step 3.

We will show how to apply steps analogous to 1-3 in the general stochastic block model. Our methods will largely follow the framework of Aronson, Frieze, and Pittel, with appropriate modifications [1]. Step 4 of the analysis will turn out not to be true in general, although we will find a couple interesting cases in which it is.

2.2 Passing to a Configuration Model

For analytical purposes, it will be convenient to consider the Karp–Sipser algorithm acting on random *multigraphs* as opposed to random graphs. This will allow us to simplify our discussion of degree distributions, at the cost of having to put "multi" in front of some words (we will often be careless about actually including those "multi" prefixes – please forgive the inconsistency).

Definition 4 (Blocked configuration model)**.** Let G be a graph drawn from a stochastic block model. For each pair i,j of block model labels let m_{ij} denote the number of edges between vertices of label i and vertices of label j in G. Construct a random multigraph as follows: if $i \neq j$, distribute m_{ij} half-edges among the i vertices and m_{ij} half-edges among the j vertices uniformly at random (à la balls-in-bins), then define edges by a uniform random pairing between the half edges on the i and j sides. If $i = j$, instead distribute $2m_{ii}$ half-edges within class i, then choose a uniform random pairing among those $2m_{ii}$ half-edges. This process defines a distribution over multigraphs, because in the process of "reshuffling" the half-edges of G we introduce the possibility of multiple edges and self-loops.

Equivalently, once the m_{ij} are determined, we could consider this process as choosing uniformly at random for each i,j pair of classes an ordered list of $m_{i,j}$ edges from i to j (with possible duplicates).

Lemma 1. Conditional on the output being simple (no multiple edges or self-loops), this process gives the same distribution as the original stochastic block model.

Proof. If the output is simple, then for each pair of classes the size m_{ij} of the edge set is distributed the same as the stochastic block model, and all edge sets of that size are equally likely (since there are exactly $m_{ij}!$ possible ordered lists of edges that produce that edge set, and all of those are equally likely). As in the stochastic block model, the values of m_{ij} and the distribution of edges within those are independent for all pairs i and j. So, this is the same distribution. ■

Lemma 2. If all label classes are of size $\Theta(n)$ and each pair of labels has $O(n)$ edges between them, then with probability bounded away from zero the blocked configuration model produces a simple graph.

Proof. It suffices to show that with probability bounded away from zero the graph between any given pair of labels is simple, because the pairs are independent and there are only constantly many of them. Denote the set of vertices with label i as S_i. Within a given label class i, the probability of being simple is noted in [Aronson, Frieze, Pittel 1997] as

$$\left(1-\frac{1}{|S_i|}\right)^{m_{ii}} \cdot \prod_{j=0}^{m_{ii}-1}\left(1-\frac{j}{\binom{|S_i|}{2}}\right) \geq e^{-\frac{m_{ii}}{2|S_i|}-\frac{m_{ii}^2}{4|S_i|^2}} = \Theta(1).$$

Between two different label classes i and j, the probability of being simple is

$$\prod_{k=0}^{m_{ij}-1}\left(1-\frac{k}{|S_i|\cdot|S_j|}\right) \geq e^{-\frac{m_{ij}}{|S_i||S_j|}} = \Theta(1).$$

So, the entire multigraph is simple with probability bounded away from 0. ■

Thus, any result which holds with high probability in the blocked configuration model also holds with high probability for the stochastic block model.

2.3 Markov Property

As we run the Karp–Sipser algorithm on a random multigraph, though, the distribution of our multigraph changes, since the earlier steps of the algorithm have produced some effects conditional on the previous states of the graph. Fortunately, it turns out that these effects have a simple description.

Lemma 3. If we generate a graph from the blocked configuration model and run the Karp–Sipser algorithm for an arbitrary number of steps, then the resulting graph still follows the blocked configuration model once we condition on:

- For each pair of (i,k) of block model labels, the number E_{ij} of edges between label class i and j (when $i=j$, we'll actually let E_{ij} denote *twice* the number of edges lying within class $i=j$, so that E_{ij} always refers to the number of j-type half-edges attached to label-i vertices – this is just a notational choice to make things cleaner later)
- For each label class i, the number T_i of label-i vertices of degree exactly 1 (the "thin" vertices)
- For each label class i, the number F_i of label-i vertices of degree at least 2 (the "fat" vertices)

Thus, if we collect all of those values into a tuple $Y=(E_{ij},T_i,F_i)$, the algorithm's progress can be described by a Markov chain on Y.

A proof of this statement is given in Appendix A.

2.4 Degree Distributions

In order to prove things about the asymptotic behaviour of the algorithm, we would like to understand what the transition probabilities of this Markov process look like in the limit of large n. Towards this goal, we need to understand how the degrees of random vertices are distributed when we condition an output of the blocked configuration model on the number of thin and fat vertices in each class.

Lemma 4. Fixing a description tuple Y, let nY be the tuple obtained by scaling every element of Y by n, and let G_n be a multigraph drawn from a blocked configuration model conditional on tuple nY. Choose a uniform half-edge incident to class i, and let v be its incident vertex. In the limit $n\to\infty$, the degree of v converges in distribution to

$$\mathbb{P}[d(v)=k]=\begin{cases}\frac{T_i}{\sum_l E_{il}} & \text{for } k=1\\ \frac{(\sum_l E_{il})-T_i}{\sum_l E_{il}}\cdot\frac{\lambda^{k-1}}{(k-1)!e^{\lambda}(1-e^{-\lambda})} & \text{for } k>1,\end{cases}$$

where λ is a solution to

$$\frac{\lambda e^{\lambda}-\lambda}{e^{\lambda}-1-\lambda}=\frac{(\sum_l E_{il})-T_i}{F_i}.$$

Proof. First of all, we are conditioning on exactly how many degree 1 vertices there are in G_n, and so we know that the probability of a random half-edge being incident to one of them is exactly the number of degree-1 vertices divided by the total number of half-edges. Among the remaining vertices, note that in our model all half-edges are treated indistinguishably regardless of the label of the other half. So, the question of the degree of the vertex attached to a random half-edge, conditional on that vertex having degree at least 2, is equivalent to asking "Suppose I throw $((\sum_l E_{il})-T_i)/F_i$ balls (the total number of half-edges associated to the fat vertices) into F_i bins. If I condition on all bins having at least 2 balls, how many balls are in the bin the first ball landed in?". Note that the bin the first ball ends up in is a uniform random bin independent of the other balls, so the probability that it ends up with k balls in it converges to the probability that a randomly selected bin has $k-1$ balls. We claim that this distribution converges to the above truncated Poisson distribution; a proof of this is given by Aronson, Frieze, and Pittel by showing that a sum of truncated Poissons is highly concentrated around its mean, which allows us to ignore the conditioning on all the other bins [1]. ■

Corollary 1. In particular, as $n\to\infty$, the probability that a random edge into class i connects to a vertex of degree exactly 2 tends to $\frac{(\sum_l E_{il})-T_i}{(\sum_l E_{il})}\cdot\frac{\lambda}{e^{\lambda}-1}$. This value is important for understanding the evolution of the algorithm.

Corollary 2. The other important quantity this tells us is, for a random edge (uv) where u has label i and v has label j, how many of u's other edges go to vertices of class k in expectation. The above degree estimates tell us that there should be $\frac{(\sum_l E_{il})-T_i}{\sum_l E_{il}}\cdot\frac{\lambda}{1-e^{-\lambda}}$ other edges out of u in expectation, and because half-edges of all types are distributed interchangeably, the fraction of them that go to k should be $\frac{E_{ik}}{(\sum_l E_{il})}$. So, as $n\to\infty$, this value tends to $\frac{(\sum_l E_{il})-T_i}{\sum_l E_{il}}\cdot\frac{\lambda}{1-e^{-\lambda}}\cdot\frac{E_{ik}}{(\sum_l E_{il})}$.

Remark 1. Another point we note here is that the probability that a given vertex is attached to any self-loops or multiple edges is negligible, so long as the average degrees in each class remain bounded by a constant. In this case, we know that with high probability the maximum degree in the graph is $O(\frac{\log n}{\log\log n})$. If a vertex v has $O(\frac{\log n}{\log\log n})$ edges to class j, and class j has at least $n^{0.01}$ vertices (otherwise we can safely ignore all effects of class j), then the probability that two of those edges are to the same vertex tends to 0 in n. Since average degree will indeed stay bounded by a constant, this justifies us in not worrying about multiple edges (if we had more multiple edges, we might be concerned that we're frequently estimating that we'll remove two vertices from the graph, but actually both of them are the same vertex).

2.5 Transition Probabilities

From these degree distribution estimates, we can produce estimates on the transition probabilities of the Markov process. On a step of the algorithm, if there exist degree-1 vertices remaining, then we will choose one of them and remove it and its neighbour. So, we expect to lose one edge from between class i and j whenever that degree-1 vertex has its edge between i and j. Also, whenever the neighbour vertex is in class i, we lose edges equal to however many neighbours *it* had in class j (and vice versa when the neighbour vertex is in class j). Similar accounting can be made for the number of fat or thin vertices in a class: we lose a fat vertex either by having it as the neighbour of the degree-1 vertex we removed, or by having it initially have degree 2 and appear as the neighbour of the neighbour, so that it's then reduced to degree 1; we lose a thin vertex whenever its the degree-1 vertex, whenever its the neighbour, or whenever its a neighbour-of-the-neighbour, but gain one whenever a degree 2 vertex is a neighbour-of-the-neighbour. To make these expressions explicit, we can define some notation:

- Let h_i denote the total number of half-edges in class i,
$$h_i = \sum_l E_{il}.$$
- Let ω_{ij} denote the probability that a randomly selected degree-1 vertex is in class i and has its neighbour in class j,
$$\omega_{ij} = \frac{T_i}{\sum_l T_l} \cdot \frac{E_{ij}}{h_i}.$$
- Let δ_{ij} denote expected number of other j-type half-edges attached to the vertex a random half-edge in class i is attached to. If λ is a solution to $\frac{\lambda(e^\lambda - 1)}{e^\lambda - 1 - \lambda} = \frac{h_i - T_i}{F_i}$, then, by our degree estimates,
$$\delta_{ij} = \frac{h_i - T_i}{h_i} \cdot \frac{\lambda}{1 - e^{-\lambda}} \cdot \frac{E_{ij}}{h_i}.$$
- Let θ_i denote the probability that a random half-edge attached to class i is attached to a degree 2 vertex. Again, letting $\frac{\lambda(e^\lambda - 1)}{e^\lambda - 1 - \lambda} = \frac{h_i - T_i}{F_i}$, we have
$$\theta_i = \frac{h_i - T_i}{h_i} \cdot \frac{\lambda}{e^\lambda - 1}.$$

Now, while there are degree-1 vertices remaining in the graph, we can write the expected change in our tuple after one step of the algorithm as

$$\begin{aligned}
\mathbb{E}[\Delta E_{ij}] &= -\omega_{ij} - \omega_{ji} - \sum_l \omega_{li}\delta_{ij} - \sum_l \omega_{lj}\delta_{ji} \\
\mathbb{E}[\Delta F_i] &= -\left(\sum_l \omega_{li}\right)\left(\frac{h_i - T_i}{h_i}\right) - \sum_j \sum_l \omega_{jl}\delta_{li}\theta_i \\
\mathbb{E}[\Delta T_i] &= -\left(\sum_l \omega_{il}\right) - \left(\sum_l \omega_{li}\right)\left(\frac{T_i}{h_i}\right) - \sum_j \sum_l \omega_{jl}\delta_{li}\left(\frac{T_i}{h_i} - \theta_i\right).
\end{aligned}$$

On the other hand, when there are no degree-1 vertices remaining, we choose an edge uniformly at random, so, by similar reasoning,

$$\begin{aligned}
\mathbb{E}[\Delta E_{ij}] &= -\left(\frac{2E_{ij}}{\sum_k h_k}\right) - \left(\frac{2h_i}{\sum_k h_k}\right)\delta_{ij} - \left(\frac{2h_j}{\sum_k h_k}\right)\delta_{ji} \\
\mathbb{E}[\Delta F_i] &= -\left(\frac{2h_i}{\sum_k h_k}\right) - \sum_l \left(\frac{2h_l}{\sum_k h_k}\right)\delta_{li}\theta_i \\
\mathbb{E}[\Delta T_i] &= \sum_l \left(\frac{2h_l}{\sum_k h_k}\right)\delta_{li}\theta_i.
\end{aligned}$$

We will now argue that the evolution of Phase 1 of the algorithm stays close to the solution of an ODE.

2.6 Convergence to Continuous Approximation

Associating Markov processes on graphs to differential equations is a very useful tool, and there have been a variety of versions of this argument with varying degrees of generality and probability bound guarantees [13, 5]. Here, we will justify the passage to differential equations by Wormald's theorem [21], although we will relegate the verification of the technical conditions of the theorem to Appendix B. The important thing to note is that the expected transitions above are "scale-invariant", meaning that they remain the same upon re-scaling all entries in Y by the same amount. So, letting $\bar{E}_{ij} = \frac{E_{ij}}{n}$, $\bar{T}_i = \frac{T_i}{n}$, $\bar{F}_i = \frac{F_i}{n}$, we can write (for Phase 1):

$$\mathbb{E}[\Delta \bar{E}_{ij}] = \frac{-\omega_{ij} - \omega_{ji} - \sum_l \omega_{li}\delta_{ij} - \sum_l \omega_{lj}\delta_{ji}}{n}$$

$$\mathbb{E}[\Delta\bar{F}_i] = -\frac{(\sum_l \omega_{li})\left(\frac{\bar{h}_i - \bar{T}_i}{\bar{h}_i}\right) - \sum_j \sum_l \omega_{jl}\delta_{li}\theta_i}{n}$$

$$\mathbb{E}[\Delta\bar{T}_i] = -\frac{(\sum_l \omega_{il}) - (\sum_l \omega_{li})\left(\frac{\bar{T}_i}{\bar{h}_i}\right) - \sum_j \sum_l \omega_{jl}\delta_{li}\left(\frac{\bar{T}_i}{\bar{h}_i} - \theta_i\right)}{n}$$

This is a process that takes order n time steps, and where the expected change at each time-step is scaling like $\frac{1}{n}$. Informally, we can observe that in the limit of n, the many small steps should average out and produce a process evolving according to their expectations; this suggests looking at the following system of equations:

$$\frac{d}{dt}\bar{E}_{ij}(t) = -\omega_{ij}(t) - \omega_{ji}(t) - \sum_l \omega_{li}(t)\delta_{ij}(t) - \sum_l \omega_{lj}(t)\delta_{ji}(t)$$

$$\frac{d}{dt}\bar{F}_i(t) = -\left(\sum_l \omega_{li}(t)\right)\left(\frac{\bar{h}_i(t) - \bar{T}_i(t)}{\bar{h}_i(t)}\right) - \sum_j \sum_l \omega_{jl}(t)\delta_{li}(t)\theta_i(t)$$

$$\frac{d}{dt}\bar{T}_i(t) = -\left(\sum_l \omega_{il}(t)\right) - \left(\sum_l \omega_{li}(t)\right)\left(\frac{\bar{T}_i(t)}{\bar{h}_i(t)}\right) - \sum_j \sum_l \omega_{jl}(t)\delta_{li}(t)\left(\frac{\bar{T}_i(t)}{\bar{h}_i(t)} - \theta_i(t)\right)$$

with initial conditions

$$\bar{E}_{ij}(0) = c_{ij}\bar{S}_i\bar{S}_j, \quad \bar{F}_i(0) = 1 - \left(1 + \sum_j c_{ij}\bar{S}_j\right)e^{-\sum_j c_{ij}\bar{S}_j} \quad \text{and} \quad \bar{T}_i(0) = \left(\sum_j c_{ij}\bar{S}_j\right)e^{-\sum_j c_{ij}\bar{S}_j},$$

where c_{ij} and $\bar{S}_i = \frac{|S_i|}{n}$ are the connection probabilities and label class sizes, respectively, of the stochastic block model instance. Wormald's theorem guarantees that, in the limit of n, the evolution of Phase 1 stays close to the unique solution of this ODE with probability approaching 1 (this is formally justified in Appendix B). That implies the following:

Theorem 1. If $\mathcal{Y}(t) = \{\bar{\mathcal{E}}_{ij}, \bar{\mathcal{F}}_i, \bar{\mathcal{T}}_i\}$ is a solution to the above ODE, with high probability the total number of unmatched isolated vertices created in Phase 1 of the Karp–Sipser algorithm is $n\big(1 - \tau - \bar{\mathcal{F}}_i(\tau)\big) + o(n)$, where τ is the first time such that $\bar{\mathcal{T}}_i(\tau) = 0$ for all i.

These equations don't seem to have a simple analytical solution in general, but they can be effectively numerically evaluated for any specific stochastic block model instance. In the Erdős–Rényi case, studying Phase 2 of the Karp–Sipser algorithm reveals that with high probability at most $o(n)$ unmatched isolated vertices are created, meaning that the algorithm is asymptotically optimal, and that the true matching size is $n\big(\tau + \bar{\mathcal{F}}_i(\tau)\big) + o(n)$. However, this analysis turns out not to work for general stochastic block model instances. We'll first illustrate a few examples (namely the equitable and sub-critical cases) where, with a little bit of work, we can prove similar results; then, we will examine where the algorithm fails.

2.7 Equitable Case

The first, and most important success story we will show for the Karp–Sipser algorithm happens in a case we'll call "equitable":

Definition 5. We call stochastic block model parameters **equitable** if there's some constant c such that for all classes i,

$$\sum_j \frac{c_{ij}|S_j|}{n} = c.$$

In other words, although the edge density in some parts of the graph may be higher than other parts, the expected degree of every vertex is c regardless of what label class it belongs to. In these cases, we claim that not only does the Karp–Sipser algorithm construct an asymptotically-optimal matching, but that the matching size it constructs is asymptotically **exactly the same as the matching number of the Erdős–Rényi graph** $G(n, c/n)$.

The intuition behind this claim is that, even though there's nontrivial correlation between the edges of the graph, we still expect the degree distributions to look the same everywhere. So, since our estimates of transition

probabilities were functions of the degree distributions, they should evolve similarly to the Erdős–Rényi case. The crucial point we will need to justify to make this intuition precise is that the degree distributions necessarily *remain* close to equal across classes given that they start that way.

Lemma 5. On the output of an equitable stochastic block model, after the first phase of the algorithm with high probability all of the values of F_i (number of vertices) and h_i (total number of incident edges) for each class differ by at most $o(n)$ from the values they would have had in $G(n,c/n)$.

Proof. This follows directly from our application of Wormald's theorem. We know that initially all $\frac{\bar{T}_i}{\bar{S}_i}$'s, $\frac{\bar{F}_i}{\bar{S}_i}$'s and $\frac{\bar{h}_i}{\bar{S}_i}$'s are equal; now we observe that if that equality holds at some time t, our ODE looks like

$$\begin{aligned}
\frac{d}{dt}\bar{h}_i(t) &= \sum_j \frac{d}{dt}\bar{E}_{ij}(t) = \sum_j \left(-\omega_{ij}(t) - \omega_{ji}(t) - \sum_l \omega_{li}(t)\delta_{ij}(t) - \sum_l \omega_{lj}(t)\delta_{ji}(t) \right) \\
&= \bar{S}_i(-2-2\delta) \\
\frac{d}{dt}\bar{F}_i(t) &= -\left(\sum_l \omega_{li}(t)\right)\left(\frac{\bar{h}_i(t)-\bar{T}_i(t)}{\bar{h}_i(t)}\right) - \sum_j\sum_l \omega_{jl}(t)\delta_{li}(t)\theta_i(t) \\
&= \bar{S}_i\left(\frac{\bar{h}-\bar{T}}{\bar{h}} - \delta\theta\right) \\
\frac{d}{dt}\bar{T}_i(t) &= -\left(\sum_l \omega_{il}(t)\right) - \left(\sum_l \omega_{li}(t)\right)\left(\frac{\bar{T}_i(t)}{\bar{h}_i(t)}\right) - \sum_j\sum_l \omega_{jl}(t)\delta_{li}(t)\left(\frac{\bar{T}_i(t)}{\bar{h}_i(t)} - \theta_i(t)\right) \\
&= \bar{S}_i\left(\frac{\bar{T}}{\bar{h}} - \delta\left(\frac{\bar{T}}{\bar{h}} - \theta\right)\right),
\end{aligned}$$

where $\bar{h} = \frac{\bar{h}_i}{\bar{S}_i}$, $\bar{T} = \frac{\bar{T}_i}{\bar{S}_i}$, δ is our estimate on the expected degree - 1 of the vertex attached to a random half-edge, and θ is our estimate on the probability the degree of the vertex attached to a random half-edge is exactly 2 (it can be seen from our estimates that these values are the same across classes). Note that these values evolve the same way in every class, just scaled by the size of the class.

So, in a solution to these equations, the values of $\frac{\bar{T}_i}{\bar{S}_i}$'s, $\frac{\bar{F}_i}{\bar{S}_i}$'s and $\frac{\bar{h}_i}{\bar{S}_i}$ remain equal across classes for all time, and the evolution of the total number of edges / thin vertices / fat vertices follows the same trajectory as it would in $G(n,c/n)$. Since Wormald's theorem guarantees that with high probability Phase 1 differs by at most $o(n)$ from a solution to this equation, we have the desired statement. ■

Lemma 6. If, at the start of Phase 2, the values of $\frac{F_i}{\bar{S}_i}$ and $\frac{h_i}{\bar{S}_i}$ each differ by at most $o(n)$ between classes, then, with high probability, at most $o(n)$ isolated vertices are produced in Phase 2.

Proof. Fix some constant $\epsilon > 0$. We want to show that for any such ϵ, the probability of isolating more than ϵn vertices over the course of Phase 2 tends to 0 in the limit of n. We divide our analysis into two parts:

i) First, we consider the case where the average degree $\frac{h_i}{F_i} > 2+\epsilon$ for all classes i. In this regime, we argue that the number of steps between the times when the graph is free of thin vertices is small, so we can again control the evolution by an ODE. The following is a rough sketch of the argument (a more detailed justification of these points involving Wormald's theorem is given in Appendix B):
When we first remove a random edge, this may create some vertices of degree 1. In removing those, we may create more vertices of degree 1. In general, the expected number of new degree-1 vertices created when a degree-1 vertex of class i is removed is $\sum_j \frac{E_{ij}}{h_i}\delta_i\theta_j \leq (\max_i \delta_i)(\max_j \theta_j)$. Now, when we have that the average degree in each class is at least $2+\epsilon$, and we know that the difference in average degree between any pair of classes i and j is very small (say, less than γ), then we know $\delta_i\theta_j = \frac{\lambda_i}{1-e^{-\lambda_i}} \cdot \frac{\lambda_j}{e^{\lambda_j}-1} < 1-\eta$ for any pair i, j of classes, where η depends on γ and ϵ (the existence of some $\gamma > 0$, $\eta > 0$ in terms of ϵ with this property is guaranteed by a continuity argument). So, the expected number of degree-1 vertices created for each degree-1 vertex removed is at most a constant, $1-\eta$, that's less than 1. The size of a subcritical Galton–Watson tree with $1-\eta$ expected offspring is very unlikely to exceed a bound in terms of η. Thus, while we are within the region where average degrees are

γ-close and greater than $2+\epsilon$, the duration of a "run" of degree-1 stripping is unlikely to be much more than a constant independent of n. Since the length of each run is small, we can appeal to the law of large numbers and claim the process evolves like its expectation. Whenever the average degree is the same in all classes, the expected change in the number of edges of a given type is proportional to the number of edges currently of that type (all edges in the graph are equally likely to be chosen as the first edge removed, equally likely to be the edge chosen one step into the run, etc). So, since average degrees start out $o(1)$ away from each other, we expect them to remain that way up until one of them drops below $2+\epsilon$. Also, for a run of constant length we expect to create $o(1)$ isolated vertices, so the total number created while we're in this regime is $o(n)$ with high probability.

ii) Once we've left that region and the average degree in some class has dropped below $2+\epsilon$, by the above analysis the average degree in *every* class is $o(1)$ away from $2+\epsilon$. If we perform another run of the algorithm, we're no longer guaranteed that the expected change in number of degree-1 vertices is bounded away from 1 – we do know, however, that the only way to remove a thin vertex and create more than one in its place is to have its neighbour have degree greater than 2. Since no class has average degree more than $2+2\epsilon$, we know that the entire graph has at most $2n\epsilon$ edges associated with vertices of degree greater than 2. Those edges are the only places we can branch out and create more degree-1 things than we consume, so, throughout the course of the rest of the algorithm, there can never be more than $2n\epsilon$ thin vertices in the graph at once. Now, note that the rate of creation of isolated vertices on a given step is always proportional to the fraction of thin vertices in the graph: when we remove a vertex (the neighbour of a thin vertex), it has $\delta_i < 2$ other neighbours in expectation, and by our Markov property we know that those neighbour edges are equally likely to be any of the edges leaving the class. So, the number of vertices isolated at each step is in expectation at most $\frac{2}{v}$ times the total number of degree-1 vertices, where v is the total number of vertices in the graph. By another law-of-large-numbers argument, when we perform $\Theta(n)$ such steps, the probability of being a total of $\Theta(n)$ above this expected bound is $o(1)$. So, we can upper bound the number of isolated vertices by

$$2\epsilon n + \int_{2\epsilon n}^{n} \frac{2}{v} 2\epsilon n dv = 2\epsilon n(1 - 2\log(2\epsilon))$$

We can make this arbitrarily small by choice of ϵ; so, with high probability the total number of vertices isolated in Phase 2 is $o(n)$.

■

Theorem 2. With high probability, the matching number of an equitable stochastic block model is

$$\left(1 - \frac{x + ce^{-x} + xce^{-x}}{2c}\right)n + o(n)$$

where x is the smallest solution to $x = ce^{-ce^{-x}}$, and the Karp–Sipser algorithm achieves within $o(n)$ of this value.

Proof. This follows directly from the previous lemmas. In Phase 1, the algorithm is guaranteed to perform optimally, and with high probability it isolates only $o(n)$ vertices in Phase 2, so the Karp–Sipser algorithm is within $o(n)$ of optimal. Since the ODE determining the evolution of Phase 1 evolves the same as in an Erdős–Rényi graph with parameter c/n, the total number of lost vertices must be within $o(n)$ of the number lost in the Erdős–Rényi case. Since the above expression is known to be the matching number in the Erdős–Rényi case [10, 1], it must therefore also be here. ■

2.8 Criticality

In the Erdős–Rényi case, Karp and Sipser proved that the number of unmatched non-isolated vertices remaining in the graph after Phase 1 (which we will follow recent literature in calling the "Karp–Sipser core" [3]) is $o(n)$ with high probability if $c < e$, and $\Theta(n)$ with high probability if $c > e$ [10]. We've seen that, even when we allow some correlation into the graph in terms of different classes, so long as the expected degree into all vertices is the same across classes, the progress of the algorithm is the same as if the graph was Erdős–Rényi. This implies that any equitable stochastic block model also follows this critical transition at $c = e$. In this section, we will examine criticality in the non-equitable case – can we describe a similar phase transition in terms of the expected degrees of a general block model? We might initially expect a statement of the following form:

Claim 1 (False)**.** The Karp–Sipser core of a graph drawn from a stochastic block model is $o(n)$ with high probability (that is, Phase 1 strips away all but a sublinear number of vertices) **if and only if** the expected degree $\sum_j c_{ij}\bar{S}_j$ is less than e for all classes i.

We will prove the "if" direction of this claim. However, the "only if" direction turns out not to be true – in fact, it appears to be possible for the model to be subcritical even when $\sum_j c_{ij}\bar{S}_j > e$ for all i! A complete characterization of the critical boundary is left as an open question.

In principle, analysis of the criticality of the Karp–Sipser algorithm could be done by analysis of the ODE from 2.6 – however, instead of trying to understand that system in general, we will choose to follow the methodology of Karp and Sipser's original paper in associating the probability of removing a vertex in the graph with the probability of removing the root of a random tree [10]. We note the following facts:

Fact 1 (Karp and Sipser [10])**.** The set of vertices removed by Phase 1 is fixed, regardless of the order in which degree-1 vertices are stripped. So, if there exists some valid sequence of degree-1 strippings that removes a given vertex v from the graph (either matching or isolating it), that vertex is not in the Karp–Sipser core.

Fact 2 (Implied by a result of Mossel, Neeman and Sly [17] for 2 classes; Sly and Chin [4] for more than 2)**.** For any constant d, in the limit of n the d-neighbourhood of any vertex of G (i.e. the subgraph obtained by a BFS of depth d from the vertex) converges in distribution to the first d levels of a multitype branching process, where nodes of type i have independently Pois($c_{ij}\bar{S}_j$) children of type j, and the root class corresponds to the class of the vertex in G.

If we can show that, under certain conditions, with probability going to 1 in d, there exists sequence of valid Karp–Sipser vertex removals in that tree all of which are at depth at most d, and which result in the root being removed, this will then imply that the Karp–Sipser algorithm is subcritical. That follows because we know that, in the limit of n, any structure that appears in the first d levels of the tree is equally likely to appear in the d-neighbourhood of a given vertex in G; so, if with probability at least 1 - ϵ there's a way to remove the root of such a tree for any root class, the expected number of vertices remaining in G after Phase 1 is at most ϵn. The following characterizes these conditions:

Lemma 7. The probability of removing the root of this branching process tends to 1 in the limit of d whenever the following system has no more than one solution $(x_1,...,x_q)$ in $[0,1]^q$:

$$x_i = e^{-\left(\sum_j c_{ij}\bar{S}_j e^{-\left(\sum_k c_{jk}\bar{S}_k x_k\right)}\right)}$$

The proof of Lemma 7 is given in Appendix C, where it is derived as a corollary of results about winning probabilities of games on multitype branching processes (those results are an extension of work of Holroyd and Martin on Galton–Watson trees [8], and may be of independent interest). We can now prove the forward (true) direction of our claim:

Theorem 3. If $\sum_j c_{ij}\bar{S}_j < e$ for all classes i, then the size of the Karp–Sipser core is $o(n)$.

Proof. By the above analysis, it suffices to show that whenever $\sum_j c_{ij}\bar{S}_j < e$ for all classes i, the function

$$\begin{bmatrix} x_1 \\ ... \\ x_q \end{bmatrix} \mapsto \begin{bmatrix} e^{-\left(\sum_j c_{1j}\bar{S}_j e^{-\left(\sum_k c_{jk}\bar{S}_k x_k\right)}\right)} \\ ... \\ e^{-\left(\sum_j c_{qj}\bar{S}_j e^{-\left(\sum_k c_{jk}\bar{S}_k x_k\right)}\right)} \end{bmatrix} - \begin{bmatrix} x_1 \\ ... \\ x_q \end{bmatrix}$$

has only one root on $[0,1]^q$. Denoting $e^{-\left(\sum_j c_{ij}\bar{S}_j e^{-\left(\sum_k c_{jk}\bar{S}_k x_k\right)}\right)}$ as f_i, the Jacobian of this function looks like

$$\begin{bmatrix} f_1\left(\sum_j (c_{1j}\bar{S}_j)(c_{j1}\bar{S}_1)e^{-(\sum_k c_{jk}\bar{S}_k x_k)}\right) - 1 & ... & f_1\left(\sum_j (c_{1j}\bar{S}_j)(c_{jq}\bar{S}_q)e^{-(\sum_k c_{jk}\bar{S}_k x_k)}\right) \\ ... & ... & ... \\ f_q\left(\sum_j (c_{qj}\bar{S}_j)(c_{j1}\bar{S}_1)e^{-(\sum_k c_{jk}\bar{S}_k x_k)}\right) & ... & f_q\left(\sum_j (c_{qj}\bar{S}_j)(c_{jq}\bar{S}_q)e^{-(\sum_k c_{jk}\bar{S}_k x_k)}\right) - 1 \end{bmatrix}.$$

The sum of the entries in the ith row of this matrix is

$$f_i\left(\sum_j\left(c_{ij}\bar{S}_j e^{-\left(\sum_k c_{jk}\bar{S}_k x_k\right)}\cdot\sum_l\left(c_{jl}\bar{S}_l\right)\right)\right)-1.$$

By assumption, we know $\sum_l\left(c_{jl}\bar{S}_l\right)<e$. So, the above expression is strictly less than

$$ef_i\log f_i-1.$$

For any value of f_i, $ef_i\log f_i$ is at most 1 (taking the derivative, we find a unique maximum at $f_i=\frac{1}{e}$). So, we have shown that the sum of every row of the Jacobian is negative everywhere. Now, suppose that this function has two distinct roots, $x=(x_1,...,x_q)$ and $y=(y_1,...,y_q)$. Let i be the index where y_i-x_i is maximal. We have increased x_i by (y_i-x_i), and increased all the other coordinates of x by at most (y_i-x_i). We know that the directional derivative of the ith coordinate in the $[1,...,1]^T$ direction is negative, and that the partial derivative with respect to every $j\neq i$ is positive; this implies that the ith coordinate of the function at y must be smaller than the ith coordinate of the function at x, so they cannot both be roots. ■

As a consequence of this, since the Karp–Sipser algorithm is guaranteed to be optimal in Phase 1, we know the Karp–Sipser algorithm gives a near-optimal matching whenever $\sum_j c_{ij}\bar{S}_j<e$ for all i. As we stated, though, this condition turns out not to be sufficient but not necessary for subcriticality. In fact, we find that it is possible to achieve subcriticality even when every class has expected degree more than e! In Figure 1, we give a plot demonstrating this for the two-class model where one class has internal edge probability $\frac{5.6}{n}$ and external edge probability $\frac{5.6}{n}$, and the other class has no internal edge probability. Note that in this case, the expected degree of a vertex in the first class is $5.6>2e$, and the expected degree of a vertex in the second class is $2.8>e$. If we removed either the across edges or the internal edges, the model would be supercritical; however, somehow both edge sets together cancel out and become subcritical again.

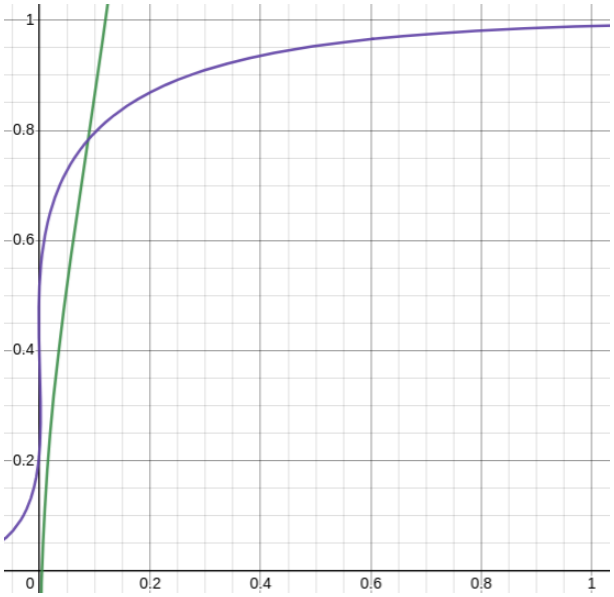


Figure 1. A plot showing that for $c_{11}=5.6$, $c_{12}=5.6$, $c_{22}=0$, the system has only 1 fixed point, and so we are subcritical. Here the horizontal axis is x_1 and the vertical axis is x_2; the green line is the set of points fixing x_1 and the purple line is the set of points fixing x_2. Note that they have only a single intersection.

The values of 5.6, 5.6, 0 were chosen above to make the single fixed point easily visible in the plot; it is possible to make these parameters larger. When there are 2 classes, we find that it's possible to make c_{11} as high as 33.5 while still maintaining criticality (by setting $c_{12}=2e+0.0514$, $c_{22}=0$); we conjecture from plots that it is not possible to make it higher than 34.

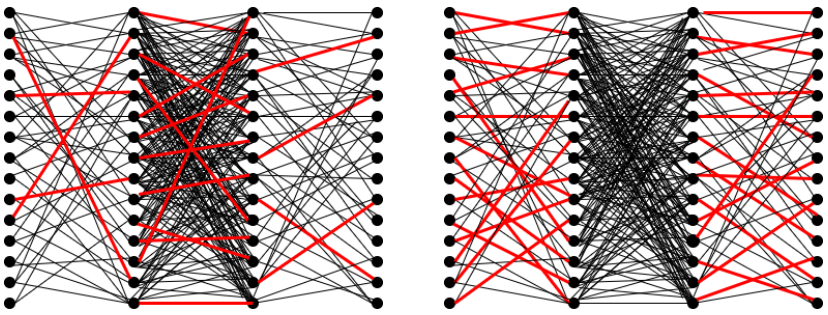
Qualitatively, the reason for this behaviour seems to be that, when there are no internal edges in the second class, there's a sort of back-and-forth thinning that can happen. That is, once we strip off all the degree-1 vertices lying entirely within the first class, the number of vertices in the first class is lower, so the average degree of vertices in the second class has now been lowered. Removing all the vertices of degree 1 in the second class in turn thins the first class, and it becomes possible to remove almost all the vertices in the graph, even though either of the two edge sets on their own wouldn't have been possible to remove. Further work in quantitatively understanding the fixed points of the system of equations in question would give insight into exactly where and why this is possible; we leave this as an open direction.

2.9 When Karp–Sipser Fails

We've seen a couple instances where we can guarantee that the Karp–Sipser algorithm achieves a near-optimal matching, using essentially the same framework as the Erdős–Rényi case (i.e., showing that it can achieve within $o(n)$ of a perfect matching during Phase 2, either because all degrees are close to 2 in the equitable case, or because the entire remaining graph has $o(n)$ vertices in the subcritical case). However, the algorithm does **not** return a near-optimal matching in general, and even when it does we will show that this analytical framework doesn't always quite work. For instance, consider a stochastic block model with 4 classes, all of size $\frac{n}{4}$, and the following probability matrix:

$$\begin{bmatrix} 0 & \frac{100}{n} & 0 & 0 \\ \frac{100}{n} & 0 & \frac{10000}{n} & 0 \\ 0 & \frac{10000}{n} & 0 & \frac{100}{n} \\ 0 & 0 & \frac{100}{n} & 0 \end{bmatrix}$$

Here, we expect the true matching size to be very close to perfect. We know that even if we ignored all of the edges between classes 2 and 3 entirely, we could match asymptotically more than 99.99% of the vertices (we would just be left with two copies of $G(n,n,p)$, which is equitable so we know the matching number). However, when we analyze the performance of the Karp–Sipser algorithm, we find a different story. Phase 1 finishes very quickly, because the graph is dense enough that very few degree 1 vertices are created. Then, in Phase 2, for a long time we are in the regime of short runs as described in our analysis of the equitable case; in this regime, the algorithm chooses many of its edges uniformly at random, and so will likely choose many of them from between classes 2 and 3. Every edge we choose between classes 2 and 3, however, effectively decreases the matching number by 1. When we carefully formalize this argument, we find that the Karp–Sipser algorithm on this graph finds a matching containing only slightly more than 50% of the vertices.



(a) Karp–Sipser **(b)** Optimal

Figure 2. An example in which the Karp–Sipser algorithm performs suboptimally

This suggests the need for a modified version of the Karp–Sipser algorithm that's given the stochastic block model parameters, and takes into account the label classes of the vertices. We propose one such algorithm here:

Algorithm 2 Label-Aware Karp–Sipser

$M \leftarrow \emptyset$
$V \leftarrow V(G)$
while $E(G \cap V) \neq \emptyset$ **do**
  if exists $v \in V$ of degree 1 – that is, such that exactly one $u \in V$ has $(uv) \in E(G)$ **then**
    Choose such a v uniformly at random over all degree 1 vertices
    $M \leftarrow M \cup \{(uv)\}$
    $V \leftarrow V \setminus \{u,v\}$
  else
    Badness$(i) \leftarrow \sum_j \theta_j \delta_{ij}$ for each class i
    $i,j \leftarrow \min_{i,j \text{ such that } \exists u \in S_i, v \in S_j, (uv) \in E(G \cap V)}$ Badness(i)+Badness(j)
    Choose an edge (uv) uniformly randomly over all $u \in S_i \cap V$, $v \in S_j \cap V$, $(uv) \in E(G)$
    $M \leftarrow M \cup \{(uv)\}$
    $V \leftarrow V \setminus \{u,v\}$
Return M

Among all edge types that exist in the current graph, when this algorithm has to make a random choice it chooses from the edge type that we estimate will create the fewest degree-1 vertices on this step. This algorithm has performed well in our experimental simulations for a range of block model parameters; we are unsure whether it performs near-optimally asymptotically. Part of the reason for the difficulty of analysis lies in the fact that we no longer expect a perfect matching to be possible in Phase 2 – the following section will illustrate this with a simple example.

2.10 Bipartite Erdős–Rényi

The bipartite graph $G(n,n,c/n)$ with equal part sizes and iid edges corresponds to an equitable stochastic block model: each vertex on the left and each vertex on the right has c expected neighbours. So, we know the Karp–Sipser algorithm performs optimally and returns a matching the same size as that of $G(2n,c/2n)$. However, the asymmetric case $G(kn,n,c/n)$ where $k \neq 1$ is not equitable, because left vertices have c expected neighbours while right vertices have kc. It is this case that we'll analyze in this section. Note that, as before, the algorithm is guaranteed to be optimal for Phase 1; we need to show that it's also near optimal for Phase 2. However, now even the optimal algorithm on Phase 2 is not guaranteed to find a near-perfect matching. Consider a graph with very unequal part sizes, and very high c; for example, $G(n,10n,1000/n)$. This graph is dense enough that we expect at least $5n$ vertices to remain on the right after Phase 1, however we know that the matching number of this graph is at most n – so, we can't actually expect Phase 2 to find a near-perfect matching. In this particular example, though, that fact doesn't especially obstruct the analysis – we just need a slightly different objective. We will sketch the argument.

Lemma 8. Let F_l, F_r and E denote the number of non-isolated unmatched vertices on the left, non-isolated unmatched vertices on the right, and edges in the graph at the start of Phase 2. With high probability the Karp–Sipser algorithm finds a matching of size $\min(F_l,F_r) - o(n)$ on this graph.

Proof. Assume without loss of generality that $F_l < KF_r$ for some $K > 0$ (if $F_l = F_r + o(n)$, we know the algorithm performs optimally by our equitable case analysis). First, we argue that with probability going to 1 in n, over the course of Phase 2 the number of non-isolated unmatched vertices on the right is always higher than the number on the left. To show this, note that at a given step of the algorithm, the action with the highest expected loss on the right compared to the left is removing a degree-1 vertex on the right. This removes 1 vertex on the left and 1 on the right, but creates $\delta_L \theta_R$ vertices of degree 1 on the right that might later be lost. Clearly, the expected loss would be bigger if instead of reducing those vertices to degree 1, we just isolated them immediately, so the right size can overall be decreasing at a rate of at most $\delta_L \theta_R$ faster than the left side. But now, note that whenever almost all vertices are of degree 2 and there are K times as many vertices on the right, δ_L can be at most K times as large as δ_R; since we know $\delta_R \theta_R < 1$, this means $\delta_L \theta_R < K$. Continuing an argument along these lines by bounding with an ODE would allow us to show that with probability going to 1, the right side can never get meaningfully smaller than the left side.

From that point, we simply note that $\delta_R \theta_L \leq \delta_L \theta_L < 1$, so the rate of creation of degree 1 vertices on the left is always less than 1, meaning that with high probability there are never more than $o(n)$ degree 1 vertices on the

left. That in turn means that with high probability we do not isolate more than $o(n)$ degree 1 vertices on the left; so, we find a matching of size $F_l - o(n)$. ■

We know the first phase of the algorithm is optimal, and then once we reach the second phase the matching number is clearly bounded by $\min(F_l, F_r)$, so this implies that the Karp–Sipser algorithm is near-optimal on $G(kn,n,c/n)$. Here, even though we couldn't find a perfect matching, we showed a way to upper bound the true matching of the Karp–Sipser core, and show that we achieved that bound. A natural next step might be to try to find similar bounds in other settings to show that Label-Aware Karp–Sipser can perform optimally there. (One simple case we note to be open is the bipartite setting where there is only one class on the left – i.e., parameters where the probability matrix graph is "star shaped". On such graphs, Label-Aware Karp–Sipser simplifies to "prefer edges to the available right class with minimum average degree".) For now, however, we move on to analyze a more restrictive type of matching algorithm.

3 Online Setting

In this section of the paper, we consider the problem of online matching on a bipartite stochastic block model, where each of the left and right halves have q classes of vertices. We analyze this model under the assumption that the class of the each left vertex is uniformly and independently drawn from $[q]$. This assumption is roughly without loss of generality; variations from this model could be approximated by further subdividing the classes.

We propose four different algorithms for this problem, presented in increasing order of complexity. The first, called DUMB-GREEDY, simply chooses an available edge uniformly at random at each step. The second, DEGREEDY, prefers matching to right vertices of lowest average degree. The third, named SHORTSIGHTED, maximizes the probability of there being an available vertex on the arrival of the next left vertex. Finally, BRUTE-FORCE is the optimal online matching algorithm and follows a dynamic-programming approach - at each step, it prefers the class which maximizes the expected size of the matching of the remaining graph. While BRUTE-FORCE is optimal, it's runtime is $\Theta(n^{q+1})$ while the first three algorithms require only $\Theta(n)$ time.

We provide examples of models where the subsequent algorithm performs better, as well as analyze DUMB-GREEDY and SHORTSIGHTED in specific cases where we take $n \to \infty$. We also prove the optimality of BRUTE-FORCE.

We will use the following notation in the description of the algorithms.

- $R^{(t)}$: set of remaining (i.e. unmatched) right vertices at time t
- $R_i^{(t)}$: set of remaining right vertices of class i at time t, $i = 1,2...q$
- $deg(\cdot)$: average degree of a vertex in given class
- $c(\cdot)$: class of a given vertex
- $v^{(t)}$: the left vertex revealed at time t
- $A^{(t)} = \{w \in R^{(t)} | (v^{(t)}, w) \in E\}$: available vertices at time t
- $C^{(t)} = \{l \in [q] | A^{(t)} \cap R_l^{(t)} \neq \emptyset\}$: available classes at time t

Before we delve into these algorithms, we consider the simpler setting of $G(n,n,c/n)$. First note that any online algorithm that chooses not to match a left vertex when it is adjacent to at least one available right vertex is sub-optimal. It immediately follows that in this setting, at each step choosing an available edge at random is optimal. We have the following result:

Fact 3 (Mastin and Jaillet [16])**.** In the online setting, the expected size of a matching in $G(n,n,c/n)$ produced by an optimal online algorithm is given by

$$\left(1 - \frac{\ln(2 - e^{-c})}{c}\right) n$$

In that case, all online algorithms that always match a left vertex when possible are equivalent, and so achieve this same bound. The stochastic block model setting allows for more nuance; we will see that although a similarly arbitrary algorithm works in some cases, designing optimal algorithms is nontrivial in general.

3.1 DUMB-GREEDY

For each left vertex that arrives, DUMB-GREEDY chooses at uniform one of the available edges adjacent to that vertex to add to the matching.

Algorithm 3 DUMB-GREEDY

$M \leftarrow \emptyset$
for t in $\{1,2...n\}$ **do** ▷ vertex $v^{(t)}$ revealed
  if $|A^{(t)}|>0$ **then**
    Choose an element uniformly at random from $A^{(t)}$, denoted by u
    $M \leftarrow M \cup \{(u,v)\}$
Return M

Lemma 9. In the equitable case, i.e., when each of the right classes have the same average degree and each of the left classes have the same average degree, DUMB-GREEDY returns an expected matching size equal to that in the bipartite Erdős-Rényi case ($G(n,n,c/n)$) with $c=\frac{1}{n}\sum_j c_{ij}|R_j| = \frac{1}{q}\sum_i c_{ij}$.

Proof. Let $X_j^{(t)}$ denote the number of unmatched vertices of class R_j at time t. We have that $X_j^{(0)} = |R_j|$ and

$$\mathbb{P}\left(X_j^{(t+1)} = x_j - 1 | X_j^{(t)} = x_j\right) = \sum_{i=1}^{q} \frac{1}{k}\left(1 - \prod_{l=1}^{q}(1-p_{il})^{X_l^{(t)}}\right)\frac{p_{ij}x_j}{\sum_{l=1}^{q} p_{il}X_l^{(t)}}$$

where we approximate the probability of choosing a vertex of R_j as $\frac{p_{ij}x_j}{\sum_{l=1}^{q} p_{il}x_l}$ given that there is at least one available edge. Therefore, if we let u_j denote the evolution of the number of unmatched vertices of R_j divided by $|R_j|n$, we have the following system of differential equations (from Wormald's) describing the evolution of unmatched vertices in each class:

$$u_j' = -\frac{1}{|R_j|}\sum_{i=1}^{q}\frac{1}{q}\left(1 - e^{-\sum_{l=1}^{q} c_{il}|R_l|u_l}\right)\frac{p_{ij}|R_j|u_j}{\sum_{l=1}^{q} p_{il}|R_l|u_l} \qquad j \in \{1,2,...q\} \tag{1}$$

Note that $u_j(0) = \frac{1}{n}$ for all j, i.e. the u_j's start out equal. Assume $u_j(t) = u(t)$ for some function u, for all j. Then we have

$$u' = -\sum_{i=1}^{q}\frac{1}{q}\left(1 - e^{-\sum_{l=1}^{q} c_{il}|R_l|u}\right)\frac{p_{ij}}{\sum_{l=1}^{q} p_{il}|R_l|} = -\sum_{i=1}^{q}\frac{1}{q}\left(1 - e^{-ncu}\right)\frac{c_{ij}}{nc} = -\frac{1}{n}(1 - e^{-ncu}) \tag{2}$$

Recall that the evolution of the fraction of unmatched vertices x in the bipartite Erdős-Rényi case is the solution to by $x' = e^{-cx} - 1$. Note that $u = \frac{x}{n}$ is a solution to (2) (so $u_j = u$ is a solution to (1)), therefore the total number of unmatched vertices at time t is

$$\sum_{j=1}^{n} |R_j| n \cdot u(t) = n^2 u(t) = n x(t)$$

as desired. ■

We also note that this is tight; i.e. that no algorithm can do asymptotically better than DUMB-GREEDY on an equitable block model. The justification for this comes from the following fact:

Lemma 10. In an equitable stochastic block model, if there are a total of $|R^{(t)}|$ unmatched vertices on the right, the probability of matching on the next step is maximized when those $|R^{(t)}|$ vertices are equally distributed among all classes (that is, each right type r has $|R_r^{(t)}| = \frac{|R^{(t)}|}{q}$ unmatched vertices).

Proof. Let $\rho_r = \frac{|R_r^{(t)}|}{|R^{(t)}|}$ denote the fraction of unmatched right vertices belonging to type r. The probability of being able to match to something on the next step of the algorithm is (as $n \to \infty$)

$$\frac{1}{q}\sum_{l} e^{-\sum_r c_{lr}\rho_r}$$

By AM-GM inequality, this is at most

$$\left(\prod_l e^{-\sum_r c_{lr}\rho_r}\right)^{1/q} = e^{-\sum_l \frac{1}{q}\sum_r c_{lr}\rho_r} = e^{-\sum_r \frac{1}{q}(\sum_l c_{lr})\rho_r} = e^{-\left(\frac{c|R^{(t)}|}{q}\right)}$$

which is precisely the value obtained by setting all $\rho_r = \frac{|R^{(t)}|}{q}$ ■

From this fact, we see that no algorithm can do asymptotically better than DUMB-GREEDY.

Theorem 4. The optimal competitive ratio for any online algorithm in an equitable stochastic block model is

$$\frac{c - \ln(2 - e^{-c})}{(2c - x + ce^{-x} + xce^{-x})}$$

where x is the smallest solution of $x = ce^{-ce^{-x}}$.

Proof. First, note that this value is precisely what we've shown for the competitive ratio of DUMB-GREEDY; this can be found by dividing the asymptotic matching size we proved in that case by the offline matching size we proved in Theorem 2. So, we just need to argue that no algorithm can do better than DUMB-GREEDY. To do this, simply take such a supposedly better algorithm and consider the expected values of ρ_i as in Lemma 10. If the ρ_i's stay within $o(n)$ of equal over all time, then the algorithm looks asymptotically identical to DUMB-GREEDY. If at some point they become unequal, then at that time the algorithm must fall behind DUMB-GREEDY and never catch up, by Lemma 10. ■

Note that this value is exactly the same as the competitive ratio lower bound conjectured by Mastin and Jaillet to be tight for Erdős–Rényi graphs; as a special case, we have shown that conjecture.

Throughout the rest of the paper, we examine non-equitable block models, providing specific instances where one algorithm beats another. Our pictorial notation includes nodes, which represent classes, and edges with a number c, which represents an edge probability between the corresponding classes of $\frac{c}{n}$. The absence of an edge represents 0 probability of edges between classes.

DUMB-GREEDY does not perform optimally in some cases. Consider the following graph with one left class and two right classes:

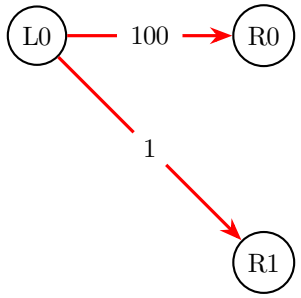


Figure 3. DEGREEDY beats DUMB-GREEDY. $|L_0| = n$, $|R_0| = |R_1| = \frac{n}{2}$

In such a scenario, it seems sensible to match R_1 vertices as they are available, because it is much less likely they will be available again as opposed to R_0 vertices (due to the average degree of R_1 vertices being lower than that of R_0 vertices). However, DUMB-GREEDY will naturally miss out on available R_1 vertices as simply choosing available edges at random heavily favors R_0 (as the edge probability is 100 times higher). This leads us to considering the following algorithm: DEGREEDY, which prioritizes matching right vertices with the lowest average degrees. We introduce the algorithm below – DEGREEDY and all following algorithms choose which class to match to and then randomly choose an available edge from that class at each step.

3.2 DEGREEDY

At each step, DEGREEDY identifies the available class (i.e. a right class with at least one available edge) with the lowest average degree - if there is a tie, choose at random one of the lowest-average-degree classes. It then uniformly chooses an available edge at random from this class.

Algorithm 4 DEGREEDY

$M \leftarrow \emptyset$
for t in $\{1,2...n\}$ **do** ▷ vertex $v^{(t)}$ revealed
  if $|A^{(t)}|>0$ **then**
    $C = \underset{j \in C^{(t)}}{\text{argmin}} \deg(R_j)$
    Choose an element uniformly at random from C, denoted by c.
    Choose a vertex uniformly at random from $R_c^{(t)}$, denoted by u
    $M \leftarrow M \cup \{(u,v)\}$
Return M

While DEGREEDY fills in some of the holes of DUMB-GREEDY, it is easy to break by having a very slight difference in average degree between right classes. Consider the following graph:

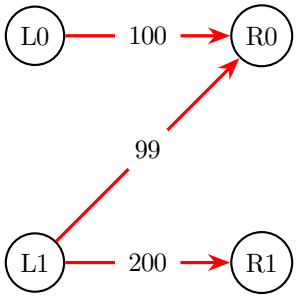


Figure 4. SHORTSIGHTED beats DEGREEDY. All classes of size $\frac{n}{2}$

In this graph, DEGREEDY will always match to R_0 while it is available. Simply matching L_0 vertices to R_0 vertices and L_1 vertices to R_1 vertices gives an expected matching size of

$$\frac{n}{2}\left(1-\frac{\log(2-e^{-50})}{50}+1-\frac{\log(2-e^{-100})}{100}\right)\approx .9896n.$$

However, DEGREEDY will match L_1 vertices to an R_0 vertex if one is available - for each occurrence of this, we are "blocking off" an L_0 vertex, and reducing the size of our potential matching by 1. The number of times an L_1 vertex arrives and has an available edge to R_0 is much greater than $.02n$, and therefore DEGREEDY is suboptimal to the simple approach of matching within class.

Therefore, we consider an algorithm that minimizes the probability of not matching the next vertex. We call this algorithm SHORTSIGHTED.

3.3 SHORTSIGHTED

The idea of SHORTSIGHTED is to, at each step, minimize the probability of not matching the next vertex (i.e. the probability that the next vertex has no available edges). Let $u_i^{(t)}$ denote the number of unmatched vertices of R_i at time t. The class(es) we choose to take an edge from at time t is given by

$$S^{(t)} = \underset{l \in C^{(t)}}{\text{argmin}} \sum_{i=1}^{q} \frac{1}{q} \prod_{j=1}^{q} (1-p_{i,j})^{u_j^{(t)} - \mathbb{1}\{l=j\}}$$

where the expression in the argmin is the probability that the next vertex has no available edges.

Algorithm 5 Shortsighted

$M \leftarrow \emptyset$
$u_i \leftarrow |R_i|, i \in [q]$
for t in $\{1,2...n\}$ **do** ▷ vertex $v^{(t)}$ revealed
  if $|A^{(t)}| > 0$ **then**
    Compute $S^{(t)}$ per the formula above, using the current values of u
    Choose an element uniformly at random from $S^{(t)}$, denoted by c
    Choose a vertex uniformly at random from $R_c^{(t)}$, denoted by w
    $M \leftarrow M \cup \{(w,v)\}$
    $u_c \leftarrow u_c - 1$
Return M

We restrict our attention to the case where $q=2$, where the classes are indexed by 0 and 1. In this case, $S^{(t)}$ can be computed generally, facilitating the analysis of SHORTSIGHTED. We consider states of the form (i,j,r), analogous to $(u_0^{(t)}, u_1^{(t)}, n-t)$. The first coordinate represents the number of unmatched vertices in R_0, the second is the number of unmatched vertices in R_1, and r indicates the number of vertices left to arrive. The initial state is $(|R_0|, |R_1|, n)$.

3.3.1 Analysis of SHORTSIGHTED in the $q=2$ case

Below we compute for which states (i,j,r) SHORTSIGHTED prefers class R_0 over R_1. If either i or j is 0, there is no choice to be made, so we consider cases where $i,j>0$. SHORTSIGHTED prefers R_0 when

$$\frac{1}{2}(1-p_{0,0})^{i-1}(1-p_{0,1})^{j} + \frac{1}{2}(1-p_{1,0})^{i-1}(1-p_{1,1})^{j} \leq \frac{1}{2}(1-p_{0,0})^{i}(1-p_{0,1})^{j-1} + \frac{1}{2}(1-p_{1,0})^{i}(1-p_{1,1})^{j-1}$$

Rearranging, we have

$$(1-p_{0,0})^{i-1}(1-p_{0,1})^{j-1}(p_{0,0}-p_{0,1}) \leq (p_{1,1}-p_{1,0})(1-p_{1,0})^{i-1}(1-p_{1,1})^{j-1}$$

Now we take the following cases:

i) $p_{0,0}-p_{0,1} \leq 0$, $p_{1,1}-p_{1,0} \geq 0$: the inequality is always true, so SHORTSIGHTED prefers class 0 where $i,j>0$

ii) $p_{0,0}-p_{0,1} \geq 0$, $p_{1,1}-p_{1,0} \leq 0$: the inequality is always false, so SHORTSIGHTED prefers class 1 where $i,j>0$

iii) $p_{0,0}-p_{0,1}, p_{1,1}-p_{1,0} > 0$: the states where SHORTSIGHTED will prefer class 0 are such that

$$(i-1)\ln\left(\frac{1-p_{0,0}}{1-p_{1,0}}\right) + (j-1)\ln\left(\frac{1-p_{0,1}}{1-p_{1,1}}\right) \leq \ln\left(\frac{p_{1,1}-p_{1,0}}{p_{0,0}-p_{0,1}}\right)$$

iv) $p_{0,0}-p_{0,1}, p_{1,1}-p_{1,0} < 0$: the states where SHORTSIGHTED will prefer class 0 are such that

$$(i-1)\ln\left(\frac{1-p_{0,0}}{1-p_{1,0}}\right) + (j-1)\ln\left(\frac{1-p_{0,1}}{1-p_{1,1}}\right) \geq \ln\left(\frac{p_{1,1}-p_{1,0}}{p_{0,0}-p_{0,1}}\right)$$

We define the boundary as the line on the $x-y$ plane that separates states where class R_0 is preferred and states where class R_1 is preferred. Taking $n \to \infty$, the boundary is given by

$$y = \frac{c_{1,0}-c_{0,0}}{c_{0,1}-c_{1,1}}x + \frac{\ln\left(\frac{c_{0,0}-c_{0,1}}{c_{1,1}-c_{1,0}}\right)}{c_{0,1}-c_{1,1}}n$$

In our analysis below, it proves useful to rescale x and y by a factor of $\frac{1}{n}$. Our new boundary is given by

$$y = \frac{c_{1,0}-c_{0,0}}{c_{0,1}-c_{1,1}}x + \frac{\ln\left(\frac{c_{0,0}-c_{0,1}}{c_{1,1}-c_{1,0}}\right)}{c_{0,1}-c_{1,1}} \tag{3}$$

Note that the condition for preferring class R_0 is given by a linear inequality and is independent of the number of remaining vertices. This independence is time is crucial in being able to analyze SHORTSIGHTED in some cases. We refer to P_c the set of (x,y) where class c is preferred.

Lemma 11. If the boundary does not intersect the rectangle defined by the possible range of x and y ($x \in [0,|R_0|/n], y \in [0,|R_1|/n]$) or if the slope of the boundary is non-positive, then SHORTSIGHTED is of the following form for some c and T (where T may be n): we prefer class R_c up until there are T vertices left to arrive; thereafter we prefer the opposite class, R_{1-c}. In these cases, we can numerically compute the expected size of matching that SHORTSIGHTED produces by the following process:

i) Determine which class (i.e. c) is preferred at the point $(|R_0|,|R_1|)$, the starting state.

ii) If $c=0$, solve the following system of differential equations with initial conditions $(a(0),b(0)) = \left(\frac{|R_0|}{n}, \frac{|R_1|}{n}\right)$:

$$a'(t) = -\frac{1}{2}\left(2 - e^{-c_{0,0}a(t)} - e^{-c_{1,0}a(t)}\right)$$

$$b'(t) = -\frac{1}{2}\left(e^{-c_{0,0}a(t)}(1 - e^{-c_{0,1}b(t)}) + e^{-c_{1,0}a(t)}(1 - e^{-c_{1,1}b(t)})\right)$$

Otherwise, if $c=1$, solve the system below:

$$a'(t) = -\frac{1}{2}\left(e^{-c_{0,1}b(t)}(1 - e^{-c_{0,0}a(t)}) + e^{-c_{1,1}b(t)}(1 - e^{-c_{1,0}a(t)})\right)$$

$$b'(t) = -\frac{1}{2}\left(2 - e^{-c_{0,1}b(t)} - e^{-c_{1,1}b(t)}\right)$$

Let $(a(t),b(t))$ be the solution to this system. If $(a(1),b(1)) \in P_c$, then the expected size of the matching is simply $n(1-a(1)-b(1))$.
Otherwise, if $(a(1),b(1)) \notin P_c$, solve for T such that $(a(T),b(T))$ lies on the boundary. Then solve the other system of equations above (i.e. solve the system corresponding to $1-c$), with initial conditions at $t=0$ being $(a(T),b(T))$. Denote the solution (f,g). The expected matching size is given by $n(1-f(1-T)-g(1-T))$

Proof. If the boundary does not intersect the feasible region for (x,y) (i.e. $x \in [0,|R_0|/n]$, $y \in [0,|R_1|/n]$), then SHORTSIGHTED will simply prefer one class at all times. Note that any possible "path" of the algorithm may only move down and left (as the number of unmatched vertices in each class can only decreasing over time). Therefore, if the boundary intersects the feasible region but has negative slope, any possible "path" can only intersect the boundary at most once at some time, where before this time the path will live in P_c and after it will live in P_{1-c} for some c. In both cases, the number of unmatched vertices can be represented by a Markov Chain (or two), and we can use Wormald's theorem to describe its behavior through the systems of differential equations given above. We discuss the specifics of our use of Wormald's theorem in Appendix B.

The case where the slope of the boundary is positive is more tricky because we cannot guarantee that the path of the algorithm only crosses the boundary at most once (though in practice, SHORTSIGHTED seems well-behaved). Nevertheless, the above lemma covers $\frac{2}{3}$ of the simplex of values of $p_{i,j}$.

Next, we introduce BRUTE-FORCE, and subsequently prove optimality. We then compare SHORTSIGHTED and BRUTE-FORCE.

3.4 BRUTE-FORCE

BRUTE-FORCE is optimal over online algorithms, as in the expected size of the matching it returns is optimal over any bipartite stochastic block model. The idea of BRUTE-FORCE is to precompute the expected size of the matching in the remaining graph at each state: $a[i_1,i_2,...i_q,r,c]$, where i_j denotes the number of remaining vertices of R_j, r denotes the number of remaining left vertices to arrive, and c denotes the class of the arrived vertex. Also define $av[i_1,i_2,...i_k,r] = \sum_{c=1}^{q} \frac{1}{q} a[i_1,i_2,...i_q,r,c]$. Then at each step of the algorithm, we move to the state where av is maximal. Computing this array of values takes $O(n^{q+1})$ time, because each entry can be computed in constant time. We present the algorithm for 2 classes on each side ($q=2$); the generalization is clear. We compute a and av from the bottom up:

Algorithm 6 BRUTE-FORCE PRE-COMPUTATION

$a[i,j,0,c] \leftarrow n-i-j$ for $i \in \{0,1,...|R_0|\}$, $j \in \{0,1,...|R_1|\}$, $c \in \{0,1\}$
for r in $\{1,2...n\}$ **do**
  for $i \in \{0,...|R_0|\}$, $j \in \{0,...|R_1|\}$, $c \in \{0,1\}$ **do**
    if $i=0$ and $j=0$ **then**
      $a[i,j,r,c] \leftarrow n$
    else if $i=0$ **then**
      $a[i,j,r,c] \leftarrow \left(1-(1-p_{c,1})^j\right) av[i,j-1,r-1] + (1-p_{c,1})^j av[i,j,r-1]$
    else if $j=0$ **then**
      $a[i,j,r,c] \leftarrow \left(1-(1-p_{c,0})^i\right) av[i-1,j,r-1] + (1-p_{c,0})^i av[i,j,r-1]$
    else if $av[i-1,j,r-1] \geq av[i,j-1,r-1]$ **then**
      $a[i,j,r,c] \leftarrow \left(1-(1-p_{c,0})^i\right) av[i-1,j,r-1] + (1-p_{c,0})^i \left(1-(1-p_{c,1})^j\right) av[i,j-1,r-1] + (1-p_{c,0})^i (1-p_{c,1})^j av[i,j,r-1]$
    else
      $a[i,j,r,c] \leftarrow \left(1-(1-p_{c,1})^j\right) av[i,j-1,r-1] + (1-p_{c,1})^j \left(1-(1-p_{c,0})^i\right) av[i-1,j,r-1] + (1-(1-p_{c,0})^i (1-p_{c,1})^j) av[i,j,r-1]$
Return av

Now we are ready to state BRUTE-FORCE.

Algorithm 7 BRUTE-FORCE

$M \leftarrow \emptyset$
`state` $\leftarrow (|R_0|,|R_1|,n)$
Compute values of av per above algorithm
for r in $\{1,2...n\}$ **do** ▷ vertex $v^{(t)}$ revealed, $c(v^{(t)})=c$
  if $|A_t|>0$ **then**
    $(i,j,r) \leftarrow$ `state`
    if $av[i-1,j,r-1,c] >= av[i,j-1,r-1,c]$ **then**
      Choose an element uniformly at random from R_0, denoted by u
      `state` $\leftarrow (i-1,j,r-1)$
    else
      Choose an element uniformly at random from R_1, denoted by u
      `state` $\leftarrow (i,j-1,r-1)$
    $M \leftarrow M \cup \{(u,v)\}$
  else
    `state` $\leftarrow (i,j,t-1)$
Return M

Lemma 12. BRUTE-FORCE is optimal in expectation.

This follows because BRUTE-FORCE effectively searches over all possible strategies and chooses the one with highest expectation. A careful proof of optimality is given in Appendix D. While BRUTE-FORCE is optimal, it has proved difficult to analyze its performance because, unlike SHORTSIGHTED, its boundary of class preferences may change over time.

3.5 SHORTSIGHTED in comparison to BRUTE-FORCE

SHORTSIGHTED performs very well in practice compared to BRUTE-FORCE. Below is a graph where BRUTE-FORCE very slightly outperforms SHORTSIGHTED. To show that BRUTE-FORCE outperforms SHORTSIGHTED, we show that there exists an algorithm for this particular graph that outperforms SHORTSIGHTED. As BRUTE-FORCE is an optimal algorithm, BRUTE-FORCE must also outperform SHORTSIGHTED. We take this indirect approach to showing BRUTE-FORCE is better as it has proved difficult to directly analyze BRUTE-FORCE.

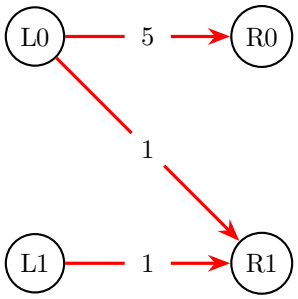


Figure 5. BRUTE-FORCE beats SHORTSIGHTED. All classes of size $\frac{n}{2}$

In the above model, SHORTSIGHTED prefers class 0 when the number of unmatched vertices of class R_0 is at least $\frac{2\ln 2}{5}n$. Carrying out the analysis as described in Lemma 11, we obtain that the expected size of the matching is $\approx 0.574946n$.

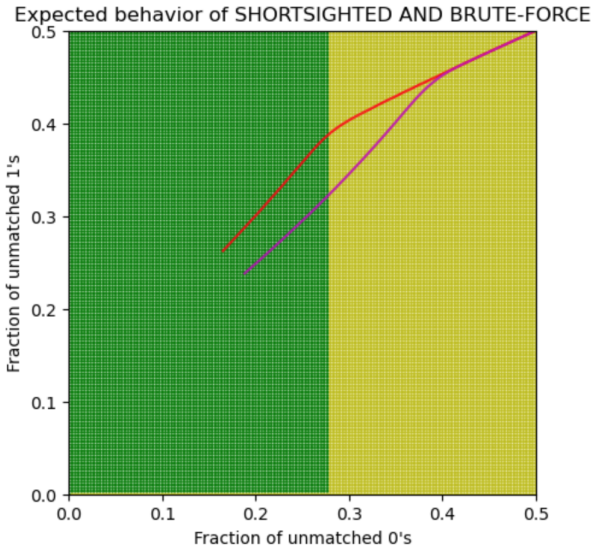


Figure 6. Behavior of SHORTSIGHTED and OPT for graph in Fig 5, $n = 300$. Yellow indicates SHORTSIGHTED's preference for class 0, green for class 1. The red curve is the average of the SHORTSIGHTED's path on 10,000 instances of the model. Magenta shows BRUTE-FORCE's path.

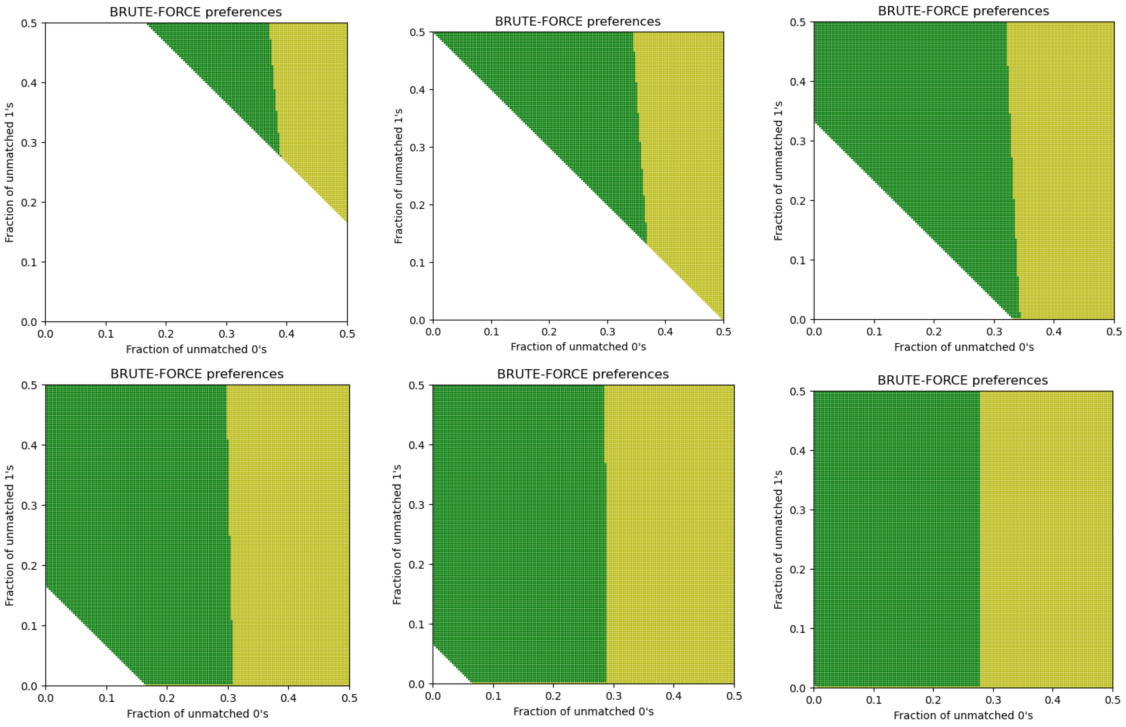


Figure 7. Preferences of BRUTE-FORCE for the graph in Fig 5 varying over time. Yellow represents a preference for class 0, green for class 1. The number of vertices remaining for each graph are 200,150,100,50,20, and 2 from left to right. Portions of the graph that are uncolored represent infeasible states at the given time.

To show this is suboptimal, consider the following algorithm: we prefer class R_0 until we have received $.88n$ vertices, then thereafter prefer R_1. With essentially the same differential equation analysis as above except with $T = .88n$, we have the expected size of the matching is $\approx 0.575597n$. This algorithm out-performs SHORTSIGHTED by a linear factor in n (albeit small, $\approx .0006n$), implying at least as big of a difference between SHORTSIGHTED and BRUTE-FORCE.

Although SHORTSIGHTED is provably worse asymptotically than BRUTE-FORCE in some cases, experimental evidence suggests that they tend to perform very similarly. We have been unable to find a model on which the their expected matching sizes differ by more than 0.1%. This suggests that it may be possible to show that, even though SHORTSIGHTED is not optimal, it has a very similar competitive ratio to the optimal online algorithm.

4 Conclusion

In Erdős–Rényi graphs, the structure looks uniform everywhere, and the performance of simple linear-time matching heuristics, both in the offline and online settings, is well understood. In this paper, we've explored what happens when we reduce those uniformity assumptions, requiring edges to be indistinguishable on a local level but allowing large-scale global density differences (a model motivated by more realistic examples of graphs arising in nature). We showed that, so long as no class of vertices is expected to have noticeably different degrees from the others, these global density differences have no asymptotic effect on the performances of either those simple online or offline algorithms. When some vertices are allowed to be biased towards higher degree than others, however, we find that the situation becomes much more complex. There is ample ground for further work on this subject. In the offline setting, we have left open a complete characterization of the critical threshold for the Karp–Sipser core, and would be interested to see classes of graphs on which the Label-Aware Karp–Sipser algorithm finds near optimal matchings even though oblivious Karp–Sipser does not. In the online setting, it would be valuable

either to find a more efficient means of calculating the optimal that BRUTE-FORCE finds, or else to prove that SHORTSIGHTED (or some other reasonable alternative) always achieves a very similar competitive ratio.

5 Acknowledgements

This project was conducted as a part of the 2023 Summer Program for Undergraduate Research at MIT. We're grateful to all the organizers of the program for facilitating it, to Elchanan Mossel for suggesting the topic, and to Professor David Jerison in particular for his helpful feedback. We would especially like to thank our wonderful mentors, Anna Brandenberger and Byron Chin.

References

[1] Jonathan Aronson, Alan M. Frieze, and Boris G. Pittel. Maximum matchings in sparse random graphs: Karp-sipser revisited. *Random Struct. Algorithms*, 12:111–177, 1998.

[2] Tom Bohman and Alan M. Frieze. Karp–sipser on random graphs with a fixed degree sequence. *Combinatorics, Probability and Computing*, 20:721 – 741, 2011.

[3] Thomas Budzinski, Alice Contat, and Nicolas Curien. The critical Karp–Sipser core of random graphs, 2022.

[4] Byron Chin and Allan Sly. Optimal reconstruction of general sparse stochastic block models, 2021.

[5] R.W.R. Darling and J.R. Norris. Differential equation approximations for Markov chains. *Probability Surveys*, 5(none):37 – 79, 2008.

[6] Jon Feldman, Aranyak Mehta, Vahab S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. *CoRR*, abs/0905.4100, 2009.

[7] Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

[8] Alexander E. Holroyd and James B. Martin. Galton–watson games. *Random Structures & Algorithms*, 59(4):495–521, 2021.

[9] Zhiyi Huang, Xinkai Shu, and Shuyi Yan. The power of multiple choices in online stochastic matching, 2022.

[10] R. M. Karp and M. Sipser. Maximum matching in sparse random graphs. In *22nd Annual Symposium on Foundations of Computer Science (SFCS 1981)*, pages 364–375, 1981.

[11] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Symposium on the Theory of Computing*, 1990.

[12] Kamer Kaya, Johannes Langguth, Ioannis Panagiotas, and Bora Uçar. *Karp-Sipser based kernels for bipartite graph matching*, pages 134–145. 12 2020.

[13] Thomas G. Kurtz. Solutions of ordinary differential equations as limits of pure jump markov processes. *Journal of Applied Probability*, 7(1):49–58, 1970.

[14] Johannes Langguth, Ioannis Panagiotas, and Bora Uçar. Shared-memory implementation of the karp-sipser kernelization process. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 71–80, 2021.

[15] Vahideh H. Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics, 2011.

[16] Andrew Mastin and Patrick Jaillet. Greedy online bipartite matching on random graphs. *CoRR*, abs/1307.2536, 2013.

[17] Elchanan Mossel, Joe Neeman, and Allan Sly. A proof of the block model threshold conjecture. *Combinatorica*, 38(3):665–708, 2018.

[18] Flore Sentenac, Nathan Noiry, Matthieu Lerasle, Laurent Ménard, and Vianney Perchet. Online matching in geometric random graphs, 2023.

[19] Nahuel Soprano-Loto, Matthieu Jonckheere, and Pascal Moyal. Online matching for the multiclass stochastic block model, 2023.

[20] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285 – 309, 1955.

[21] Nicholas C. Wormald. Differential Equations for Random Processes and Random Graphs. *The Annals of Applied Probability*, 5(4):1217 – 1235, 1995.

A Proof of Markov Property

Consider the distribution of multigraphs resulting from drawing from the blocked configuration model with fixed block sizes and edge counts, and then running the Karp–Sipser algorithm for i steps. Conditional on the value of Y for the resulting graph, we show that this distribution is the same as a blocked configuration model with those edge counts, conditioned on the numbers of thin and fat vertices in each class.

Proof. We proceed by induction. First, note that we start with a blocked configuration model with the given edge counts – once we discard multigraphs where the number of thin and fat vertices don't fit T_i, F_i, we will still be uniform over the remaining multigraphs. Now, suppose that, conditional on the value of its tuple Y_i, the distribution of multigraphs after i steps of the algorithm follows the blocked configuration model conditioned on number of thin and fat vertices. We would like to show that after $i+1$ steps, conditional on any fixed Y_{i+1}, the distribution still follows blocked configuration model. To do so, we let Y_{i+1} be arbitrary, and let G_1,G_2 be outputs of the blocked configuration model (that is, ordered lists of edges for each pair of labels) that correspond to multigraphs with tuple Y_{i+1} – we will argue that the probability of reaching G_1 after $i+1$ steps is the same as the probability of reaching G_2, meaning that all such multigraphs are equally likely (this is what it means to follow blocked configuration model distribution). The way to show this is a straightforward counting argument: enumerate all the ways to reach G_1 or G_2 from a multigraph in the previous step, then appeal to the inductive hypothesis to argue that the equality of these counts implies equal probabilities. To lay out this argument explicitly, start with either G_1 or G_2 and imagine performing the following "undoing" process:

i) First, choose a value for the tuple Y_i of the multigraph preceding G. Of course, for some choices of Y_i there are no ways to yield G on the next step (for instance, if Y_i has fewer total edges than Y_{i+1}, there's no way to choose a graph with tuple Y_i and get a multigraph with tuple Y_{i+1} after a step of Karp–Sipser) – the goal is just to show that the number of ways to yield G_1 is always the same as the number of ways to yield G_2.

ii) Next, choose which two label classes a and b the $i+1$st step of the algorithm chose an edge between. (Again, it is possible that some of these choices won't correspond to any valid predecessor states.)

iii) When the algorithm chooses an edge for the matching, it removes all edges adjacent to the endpoints from the multigraph. So, the edge chosen on step $i+1$ must have been between two vertices that are now degree 0 in G. The number of degree-0 vertices in a given class j is equal to the total number of vertices in that class minus F_j+T_j, so the number of ways to choose one degree-0 a-vertex and one degree-0 b-vertex is the same for either G_1 or G_2. We choose one such pair (x,y) to be the edge chosen in step $i+1$ of the algorithm.

iv) Once we fix which (x,y) was chosen, determining the multigraph state prior to step $i+1$ just entails choosing what neighbour multisets x and y had. We have the following constraints on what vertices we can choose for these edges sets:

- Y_i encodes the number of degree-1 vertices in the preceding graph – if this number is greater than 0, then the edge chosen by the Karp–Sipser algorithm must involve such a thin vertex, and so either x or y must have no additional edges, so we now choose which one.

- Because we know Y_i and Y_{i+1}, we know for each pair of classes exactly how many edges the $i+1$st step removed between that pair. So, we have to ensure that the number of edges we add of each type exactly equals these values.
- Similarly, the difference between Y_i and Y_{i+1} tells us for each class j the change in the number of thin and fat vertices in that class. A valid choice of edge set must, for each class j, add an edge to exactly $-\Delta F_j - \Delta T_j$ degree-0 vertices of class j; the number of ways to choose those vertices is $\binom{\text{initial size of class } j - (F_j)_i - (T_j)_i}{-\Delta F_j - \Delta T_j}$. Then, we have to add edges to exactly $-\Delta F_j$ degree-1 vertices of class j; there are $\binom{(T_j)_i - \Delta F_j - \Delta T_j}{-\Delta F_j}$ ways to choose those. Any remaining edges must be distributed over vertices of degree at least 2.

v) Finally, once we fix which edges to add, we can choose any indices within the ordered edge-lists to add them.

The crucial point of this is that the number of options available as we progress from one step to the next in the procedure is always the same regardless of whether we're considering G_1 or G_2 – the restrictions on edge sets depend only on the number of edges between each pair of classes, and the number of thin/fat vertices in each class, all of which is the same for both G_1 and G_2 because we're conditioning on them having the same tuple. So, we find that for any given predecessor tuple Y_i, the number of ways to choose a multigraph of tuple Y_i, remove a valid edge, and reach G_1 is exactly the same as the number of ways to choose a multigraph of tuple Y_i, remove a valid edge, and reach G_2. By the inductive hypothesis, all ordered edge-lists with the same Y-tuple are equally likely on step i. The number of valid edges to remove from an ordered edge-list just depends on how many edges there are and whether there are thin vertices, both of which are captured in the tuple. So, all ways to choose a multigraph of tuple Y_i and remove an edge are equally likely to have occurred on step $i+1$ of the algorithm, meaning that the resulting multigraph is equally likely to be G_1 or G_2. ■

B Conditions of Wormald's Theorem

We on several occasions in this paper claim that particular Markov processes remain close to a limiting system of differential equations. In this section, we step through for each of those instances the justification of those claims. The key tool is a theorem of Wormald, restated in its general form below. Here, n indexes a family of discrete time random processes, each of which has "history" sequence $H_n \in S_n^+$. The notation Y_t is shorthand for $y(H_t)$.

Theorem 5 (Wormald [21])**.** Let a be fixed. For $1 \le l \le a$, let $y^{(l)} : \bigcup_n S_n^+ \to \mathbb{R}$ and $f_l : \mathbb{R}^{a+1} \to \mathbb{R}$, such that for some constant C and all l, $|y^{(l)}(h_t)| < Cn$ for all $h_t \in S_n^+$ for all n. Suppose also that for some function $m = m(n)$:

i) for some functions $w = w(n)$ and $\lambda = \lambda(n)$ with $\lambda^4 \log n < w < n^{2/3}/\lambda$ and $\lambda \to \infty$ as $n \to \infty$, for all l and uniformly for all $t < m$,
$$\mathbb{P}\left[|Y_{t+1}^{(l)} - Y_t^{(l)}| > \frac{\sqrt{w}}{\lambda^2 \sqrt{\log n}} \middle| H_t\right] = o(n^{-3});$$

ii) for all l and uniformly over all $t < m$, we always have
$$\mathbb{E}(Y_{t+1}^{(l)} - Y_t^{(l)} | H_t) = f_l(t/n, Y_t^{(1)}/n, \ldots, Y_t^{(a)}/n) + o(1);$$

iii) for each l, the function f_l is continuous and satisfies a Lipschitz condition on D, where D is some bounded connected open set containing the intersection of $\{(t, z^{(1)}, \ldots, z^{(a)}) : t \ge 0\}$ with some neighbourhood of $\{(0, z^{(1)}, \ldots, z^{(a)}) : \mathbb{P}(Y_0^{(l)} = z^{(l)} n, 1 \le l \le a) \ne 0 \text{ for some } n\}$.

Then,

i) For $(0, \hat{z}^{(1)}, \ldots, \hat{z}^{(a)}) \in D$, the system of differential equations
$$\frac{dz_l}{ds} = f_l(s, z_1, \ldots, z_a), \qquad l = 1, \ldots, a,$$
has a unique solution in D for $z_l : \mathbb{R} \to \mathbb{R}$ passing through
$$z_l(0) = \hat{z}^{(l)}, \qquad 1 \le l \le a$$
and which extends to points arbitrarily close to the boundary of D.

ii) Almost surely,

$$Y_t^l = nz_l(t/n) + o(n)$$

uniformly for $0 \le t \le \min\{\sigma n, m\}$ and for each l, where $z_l(t)$ is the solution in (i) with $\hat{z}^{(l)} = Y_0^{(l)}/n$, and $\sigma = \sigma(n)$ is the supremum of those s to which a solution can be extended.

B.1 Phase 1 of the Karp–Sipser algorithm

The first time we make use of this differential equations method is in the analysis of the first phase of the Karp–Sipser algorithm. We will outline how to apply Wormald's theorem in this case. Here, we take Y as we defined it, f as the derivative we wrote down for the corresponding ODE, and note that with high probability, $y^{(l)}(h_t) < 100(\max_{ij} c_{ij})n$ for all sufficiently large n (the F_i and T_i components of $y^{(l)}(h_t)$ are clearly bounded by n; whp the edge counts are all initially within a factor of 100 of their expectations, and once they start that way they never increase). We take the stopping time m to be the first time all of the T_i entries of Y drop below $n^{0.01}$. Now,

i) Take for instance $w = n^{0.5}$ and $\lambda = \log n$. The probability that any vertex in the initial graph has degree polynomial in n decays exponentially in n, and it can be observed that the magnitude of a transition is bounded by twice the maximum degree, so we certainly have the desired condition.

ii) This convergence of expected transition size of the Markov process to f is precisely what is guaranteed by the convergence of our degree estimates. Note that this convergence holds as the total number of thin vertices, fat vertices, and edges is going to infinity, so taking our stopping time to be m prevents border cases once these values drop down to constant sizes.

iii) To show that f is Lipschitz in the neighbourhood of solutions, it suffices to show that solutions to $z'(t) = f(t,z)$ always maintain constant average degree in each label class. To do so, note that with high probability the initial average degrees are at most $100(\max_{ij} c_{ij})$ in each class, and then observe from the equations that whenever the average degree in a given class i is more than 100 at some time t, $-\sum_j \bar{\mathcal{E}}'_{ij}(t) > \bar{\mathcal{F}}'_i(t) + \bar{\mathcal{T}}'_i(t)$.

Note that we have only shown that the conditions of the theorem hold with high probability over initial graph configurations; clearly, this is sufficient for our desired result.

B.2 Phase 2 of the Karp–Sipser algorithm (equitable case)

We also use Wormald's theorem to justify our analysis of the second phase of the algorithm in the equitable case. Here, Y_i is the state of the tuple after the ith run of the algorithm – i.e., the ith time where there are no thin vertices. We define f_l to be the expected change in the tuple that one of these runs would incur if the transition probability estimates from Section 2.5 held exactly and did not change throughout the run.

In order for this process to have the desired Lipschitz properties, we will need to define a more restricted domain than the entire possible space of tuples. In particular, we will fix some constants γ and ϵ, and consider the domain of Y to consist only of tuples where the average degree into each class differs by at most γ, and is at least $2+\epsilon$. The value of ϵ is chosen in the analysis; to determine γ, we observe the following:

- λ_i is defined such that $\frac{\lambda_i(e^{\lambda_i}-1)}{e^{\lambda}_i - 1 - \lambda_i}$ is equal to the average degree of class i. This is monotonically increasing in the average degree of class i; so there is some constant $\lambda > 0$ such that if the average degree in class i is at least $2+\epsilon$, then $\lambda_i > \lambda$.
- The function $\delta_i \theta_i = \frac{\lambda_i}{1-e^{-\lambda_i}} \cdot \frac{\lambda_i}{e^{\lambda_i}-1}$ is bounded above by 1 and monotonically decreasing for $\lambda_i > 0$. So, there exists some constant $\eta > 0$ such that $\frac{\lambda}{1-e^{-\lambda}} \cdot \frac{\lambda}{e^{\lambda}-1} < 1 - 2\eta$.
- $\lambda_i, \lambda_j \mapsto \frac{\lambda_i}{1-e^{-\lambda_i}} \cdot \frac{\lambda_j}{e^{\lambda_j}-1}$ is uniformly continuous, so there exists some constant κ such that $|\lambda_i - \lambda_j| < \kappa$ implies that $|\left(\frac{\lambda_i}{1-e^{-\lambda_i}} \cdot \frac{\lambda_j}{e^{\lambda_j}-1}\right) - \left(\frac{\lambda_i}{1-e^{-\lambda_i}} \cdot \frac{\lambda_i}{e^{\lambda_i}-1}\right)| < \eta$ – which, if $\lambda_i > 2+\epsilon$, implies $\left(\frac{\lambda_i}{1-e^{-\lambda_i}} \cdot \frac{\lambda_j}{e^{\lambda_j}-1}\right) < 1 - \eta$.
- When defined on the domain $[2+\epsilon, \infty)$, λ_i is uniformly continuous as a function of the average degree of i. So, we can define γ such that |average degree in class i − average degree in class j| $< \gamma$ implies $|\lambda_i - \lambda_j| < \kappa$ whenever average degrees are greater than $2+\epsilon$.

We will define the stopping time m of the process to be the first time it leaves this domain. Note that, as in Phase 1, the value of $|Y|$ is bounded by $100c_{\max}n$ with probability $1-o(1)$, since it is initially and can only decrease. We're now ready to verify the criteria of Wormald's theorem.

i) Take again $w=n^{0.5}$ and $\lambda=\log n$. We can describe a single run of the algorithm as a branching process: each degree-1 vertex created by the algorithm corresponds to a node that has children according to the number of degree-2 vertices adjacent to its neighbour; this corresponds to expected offspring number of $\delta_i\theta_j<1-\eta$. After each removal we are uniform over graphs with the remaining statistics; so, as long as the branching process size is $o(n)$, we can treat the treat these offspring distributions roughly independently for each node (in particular, as long as the branching process is $o(n)$ with high probability, all offspring distributions have expectation at most $1-\frac{\eta}{2}$ regardless of the values for the other nodes). To prove that $\mathbb{P}\left[|Y_{t+1}^{(l)}-Y_t^{(l)}|>\frac{\sqrt{w}}{\lambda^2\sqrt{\log n}}\middle|H_t\right]=o(n^{-3})$, we therefore just need to show that the probability that a Galton–Watson tree with $\mu<1-\frac{\eta}{2}$ reaches size $\frac{n^{0.25}}{(\log n)^{5/2}}$ is $o\left(\frac{1}{n^3}\right)$; this follows from a standard Chernoff bound.

ii) The fact that $\mathbb{E}(Y_{t+1}^{(l)}-Y_t^{(l)}|H_t)=f_l(t/n,Y_t^{(1)}/n,...,Y_t^{(a)}/n)+o(1)$ holds is because we expect a constant run duration, and we know our initial degree estimates will hold with small error terms when we remove a constant portion of the graph.

iii) Continuity of f_l is clear from continuity of our degree estimates and the fact that small changes in the edge densities of the graph can't bias the branching process too heavily. Similar justification that it's Lipschitz on the given domain can be found by examining the degree estimate functions.

Since Y_0/n is with high probability $o(1)$ from having equal average degrees, and this ODE keeps equal average degrees equal, with high probability the degrees stay within $o(1)$ of equal until the stopping time.

B.3 Analysis of SHORTSIGHTED

Define the 2-D Markov Chain Z_t, where the first coordinate represented the number of unmatched R_0 vertices while the second coordinate represents the number of unmatched R_1 vertices at time t during a run of SHORTSIGHTED. We have that $Z_0=(|R_0|,|R_1|)$. If we are in the regime where we prefer class 0, we have transition probabilities as follows:

$$\mathbb{P}(Z_{t+1}=(x-1,y)|Z_t=(x,y))=\frac{1}{2}(1-(1-p_{0,0})^x)+\frac{1}{2}(1-(1-p_{1,0})^x)\to\frac{1}{2}\left(1-e^{-c_{0,0}x/n}\right)+\frac{1}{2}\left(1-e^{-c_{1,0}x/n}\right)$$

$$\begin{aligned}\mathbb{P}(Z_{t+1}=(x,y-1)|Z_t=(x,y))&=\frac{1}{2}(1-p_{0,0})^x(1-(1-p_{0,1})^y)+\frac{1}{2}(1-p_{1,0})^x(1-(1-p_{1,1})^y)\\&\to\frac{1}{2}e^{-c_{0,0}x/n}\left(1-e^{-c_{0,1}y/n}\right)+\frac{1}{2}e^{-c_{1,0}x/n}\left(1-e^{-c_{1,1}y/n}\right)\end{aligned}$$

$$\mathbb{P}(Z_{t+1}=(x,y)|Z_t=(x,y))=\frac{1}{2}(1-p_{0,0})^x(1-p_{0,1})^y+\frac{1}{2}(1-p_{1,0})^x(1-p_{1,1})^y\to\frac{1}{2}e^{-c_{0,0}x/n}e^{-c_{0,1}y/n}+\frac{1}{2}e^{-c_{1,0}x/n}e^{-c_{1,1}y/n}$$

We now verify the three conditions of Wormald's:

i) This simply comes from the fact that at each time step, each coordinate of Z_t can change by at most 1.

ii) We use the following fact from [16]: for $n>0$, $c\le n/2$, and $x\in[0,1]$, we have

$$\left|e^{-cx}-\left(1-\frac{c}{n}\right)^{nx}\right|\le\frac{c}{ne}$$

We can apply this term-wise to each of our probabilities, giving us the desired condition.

iii) e^x is Lipschitz continuous on [0,1], therefore we have the third condition as well.

An essentially identical proof follows for the Markov chain where we prefer class 1. Therefore we may apply Wormald's theorem to obtain the differential equations stated in the lemma. The second system of differential equations follows as well, because the initial conditions hold almost surely (within $o(n)$).

We have also applied Wormald's in the analysis of DUMB-GREEDY in the equitable case. Due to the similar structure of transition probabilities between DUMB-GREEDY and SHORTSIGHTED, the verification of the conditions of Wormald's is also very similar.

C Games on Multitype Branching Processes

This appendix concerns games played on multitype branching processes. The setting and results are the same as those considered by Holroyd and Martin with regards to the normal game on GW trees, but extended slightly to cover the multitype case [8]. At the end of this section, we will explain the connection to our claim about criticality of the Karp–Sipser core.

C.1 Definitions

Definition 6. A game on a (possibly infinite) game tree is played as follows: Both players know the full structure of the tree. On a player's turn, they choose one child of the current node to walk down to, and the next player starts at that node. A player loses when there are no valid moves (i.e. when they start their turn on a leaf node). We denote by $\mathcal{N}_d$ the set of states from which an optimal player can force a win within d turns of the game, $\mathcal{P}_d$ the set of states from which a player is guaranteed to lose within d steps if their opponent plays optimally, and $\mathcal{D}_d$ to be the set of states in neither $\mathcal{N}_d$ nor $\mathcal{P}_d$ (i.e. the set of states in which either player can force the game to continue for d steps if the other is unwilling to lose).

We will consider games played on a random game tree drawn from a multitype branching process. Although for the purposes of the results in the paper we only need to understand the case where the offspring distributions are independent Poissons in each type, we will do our proofs here for arbitrary offspring distributions.

Definition 7. Consider a multitype branching process with types $1...q$, where the probability of a node of type i having offspring set containing exactly o_j offspring of each type j is $p_i(o_1,...,o_q)$. The **offspring generating function** for this process is the polynomial function

$$G:[0,1]^q \to [0,1]^q$$

$$\begin{bmatrix} x_1 \\ ... \\ x_q \end{bmatrix} \mapsto \sum_{o_1=0}^{\infty} ... \sum_{o_q=0}^{\infty} \left(\prod_j x_j^{o_j} \cdot \begin{bmatrix} p_1(o_1,...,o_q) \\ ... \\ p_q(o_1,...,o_q) \end{bmatrix} \right)$$

For fixed offspring distributions, we are interested in the values of $N_d, P_d, D_d \in [0,1]^q$, where the ith entry of N_d is the probability that the root of this branching process is an $\mathcal{N}_d$ node, given that it is of class i (likewise P_d and D_d are the probabilities the root is in $\mathcal{P}_d$ or $\mathcal{D}_d$, respectively).

C.2 Convergence

We can calculate the values for N_d, P_d, and D_d recursively. Note the following:

- A vertex is in $\mathcal{N}_d$ if and only if it has at least one child in $\mathcal{P}_{d-1}$
- A vertex is in $\mathcal{P}_d$ if and only if all of its children are in $\mathcal{N}_{d-1}$
- A vertex is in $\mathcal{D}_d$ when neither of the above conditions are true (and, in particular, when $d=0$)

We can consider each of the root's children to be independent multitype branching processes, with roots of their given types. This lets us write recursions for these values in terms of the offspring generating function:

- $N_d = \mathbf{1} - G(\mathbf{1} - P_{d-1})$
- $P_d = G(N_{d-1})$
- $D_d = \mathbf{1} - N_d - P_d$

Letting $F(x) = \mathbf{1} - G(\mathbf{1} - G(x))$, these recurrences become

- $N_{d+2} = F(N_d)$
- $\mathbf{1} - P_{d+2} = F(\mathbf{1} - P_d)$

Note that for any k, $N_{2k}=N_{2k+1}$, $P_{2k}=P_{2k+1}$, and $D_{2k}=D_{2k+1}$; this is because the player starting from the root will make all the odd moves of the game, so they can force the other player to get stuck in the first $2k+1$ steps if and only if they can force a them to get stuck in the first $2k$ steps. So, to understand the behaviour of these values for large d, it suffices to show that the even values of both N_d and P_d converge to fixed points. That is, we want to describe the eventual behavior of $F\circ F\circ\cdots\circ F(\mathbf{0})$ and $F\circ F\circ\cdots\circ F(\mathbf{1})$ – if we can show that these converge to fixed values, N_d and $1-P_d$ also converge to those values, respectively. This fact is implied by the following lemma:

Lemma 13. If a function $F:[0,1]^q\to[0,1]^q$ has every output coordinate continuous and non-decreasing in every input coordinate, then there exists a minimal fixed point x of F; that is, a fixed point of F such that for all i, the ith coordinate of x is minimal among all the ith coordinates of fixed points of F; iterating F starting from $\mathbf{0}$ will reach this point. Similarly, there exists a maximal fixed point of F; iterating F starting from $\mathbf{1}$ will reach that point.

Proof. This statement is directly implied by Tarski's fixed point theorem, applied on the complete lattice obtained by coordinate-wise ordering on $[0,1]^q$ (that is, $x\le y$ if and only if $x_i\le y_i$ for all i) [20]. ■

Thus, we have that

- $\lim_{d\to\infty}N_d$ is equal to the minimum fixed point of F
- $\lim_{d\to\infty}P_d$ is equal to $\mathbf{1}$ minus the maximum fixed point of F
- $\lim_{d\to\infty}D_d$ is equal to the maximum fixed point of F minus the minimum fixed point of F

In particular, the probability of neither player having a winning strategy within d steps tends to 0 in the limit of d for all root types if and only if F has exactly one fixed point on $[0,1]^q$.

C.3 Implications for Karp–Sipser core

The connection between this analysis of games and our analysis of the Karp–Sipser algorithm comes from the following fact:

Lemma 14. If the root of a tree is in $\mathcal{N}_d$ or $\mathcal{P}_d$, there exists a sequence of removals that the Karp–Sipser algorithm can make within depth d that result in the removal of the root.

Proof. This fact is shown in Karp and Sipser's original paper [10]; the proof is a straightforward induction ($\mathcal{P}$ nodes correspond to vertices that can remove their parent, $\mathcal{N}$ nodes correspond to vertices that can be removed by one of their children). ■

So, we find that as long as $D_d\to 0$, the root of the tree is removed with probability approaching 1; by the above analysis, this occurs whenever F has only one fixed point. Plugging in the case of the process where nodes of type i have independently $\text{Pois}(c_{ij}\bar{S}_j)$ children of type j yields the result we use as Lemma 7 in the paper.

D Optimality of BRUTE-FORCE among online algorithms

Our notation here is for the case where $q=2$; the proof method easily generalizes to higher q.

Proof. Let $M_A(\cdot)$ denote the expected matching size at a given state for an algorithm A, and define $p_{i,A}(\cdot)$ as the probability we match to class i at a given state under algorithm A and $p_{\emptyset,A}(\cdot)$ as the probability there are no available vertices at the given state (which does not depend on A). We claim that $M_{BF}(i,j,r)\ge M_A(i,j,r)$ for all i,j,r and algorithms A. We have for any algorithm A,

$$M_A(i,j,r)=p_{0,A}(i,j,r)M_A(i-1,j,r-1)+p_{1,A}(i,j,r)M_A(i,j-1,r-1)+p_{\emptyset,A}(i,j,r)M_A(i,j,r-1)$$

where if i is 0, we have $M_A(i-1,j,r)=0$ (and similarly for j). We have assumed that any algorithm will match a vertex iff there is an available edge, as not matching when there is an available edge is sub-optimal. We prove our claim by induction on the number of remaining vertices of a state.

Base case: $M_A(i,j,0)=n-i-j$ for all A, implying $M_{BF}(i,j,0)\ge M_A(i,j,0)$ for all A. This is trivial as if there are no vertices left to arrive, the size of our matching is the number of R_0 vertices that have been matches plus the number of R_1 vertices that have been matched, which is equal to $|R_0|-i+|R_1|-j=n-i-j$.

Now we show that assuming $M_{BF}(i,j,r) \geq M_A(i,j,r)$ for all i,j and A, then $M_{BF}(i,j,r+1) \geq M_A(i,j,r+1)$ for all i,j and A for $r \in \{0,1,...,n-1\}$. Here we have two cases: $M_{BF}(i-1,j,r) \geq M_{BF}(i,j-1,r)$ or $M_{BF}(i-1,j,r) < M_{BF}(i,j-1,r)$. The proof is essentially identical in both cases, so we just examine the first case. In this case, we have

$$p_{0,BF}(i,j,r) = \mathbb{P}(\text{there is an available edge to } R_0 \text{ at state } (i,j,r)) \geq p_{0,A}(i,j,r)$$

Also, we have for any A that $p_{0,A}(i,j,r) + p_{1,A}(i,j,r) = 1 - \mathbb{P}(\text{there are no edges available})$, so the LHS is constant over all algorithms. Therefore, since $M_{BF}(i-1,j,r) \geq M_{BF}(i,j-1,r)$ we have

$$\begin{aligned}
M_{BF}(i,j,r+1) &= p_{0,BF}(i,j,r+1)M_{BF}(i-1,j,r) + p_{1,BF}(i,j,r+1)M_{BF}(i,j-1,r) + p_{\emptyset,BF}(i,j,r+1)M_{BF}(i,j,r) \\
&\geq p_{0,A}(i,j,r+1)M_{BF}(i-1,j,r) + p_{1,A}(i,j,r+1)M_{BF}(i,j-1,r) + p_{\emptyset,BF}(i,j,r+1)M_{BF}(i,j,r) \\
&\geq p_{0,A}(i,j,r+1)M_A(i-1,j,r) + p_{1,A}(i,j,r+1)M_A(i,j-1,r) + p_{\emptyset,A}(i,j,r+1)M_A(i,j,r) \\
&= M_A(i,j,r+1)
\end{aligned}$$

Our claim is proved, and hence $M_{BF}(|R_0|,|R_1|,n) \geq M_A(|R_0|,|R_1|,n)$ implying that no algorithm beats BRUTE-FORCE in expectation. ■ ■