Large Cuts with Local Algorithms on Triangle-Free Graphs

Juho Hirvonen
Helsinki Institute for Information Technology HIIT,
Department of Information and Computer Science, Aalto University, Finland
juho.hirvonen@aalto.fi

Joel Rybicki
Helsinki Institute for Information Technology HIIT,
Department of Information and Computer Science, Aalto University, Finland
joel.rybicki@aalto.fi

Stefan Schmid
TU Berlin & T-Labs, Germany
stefan@net.t-labs.tu-berlin.de

Jukka Suomela
Helsinki Institute for Information Technology HIIT,
Department of Information and Computer Science, Aalto University, Finland
jukka.suomela@aalto.fi

Abstract. We study the problem of finding *large cuts in d-regular triangle-free graphs*. In prior work, Shearer (1992) gives a randomised algorithm that finds a cut of expected size $(1/2 + 0.177/\sqrt{d})m$, where m is the number of edges. We give a simpler algorithm that does much better: it finds a cut of expected size $(1/2 + 0.28125/\sqrt{d})m$. As a corollary, this shows that in any d-regular triangle-free graph there exists a cut of at least this size.

Our algorithm can be interpreted as a very efficient *randomised distributed algorithm*: each node needs to produce only one random bit, and the algorithm runs in one synchronous communication round. This work is also a case study of applying *computational techniques* in the design of distributed algorithms: our algorithm was designed by a computer program that searched for optimal algorithms for small values of d.

1 Introduction

We study the problem of finding *large cuts* in *triangle-free graphs*. In particular, we are interested in the design of *fast and simple randomised distributed algorithms*.

1.1 Random Cuts

Let $G = (V, E)$ be a simple undirected graph. A *cut* is a function $c\colon V \to \{\mathsf{a}, \mathsf{b}\}$ that labels the nodes with symbols a and b. An edge $\{u, v\} \in E$ is a *cut edge* if $c(u) \neq c(v)$. We use the convention that the *weight* $w(c)$ of a cut c is the fraction of edges that are cut edges; that is, the weight of the cut is normalised so that it is in the range $[0, 1]$. See Figure 1 for an illustration.

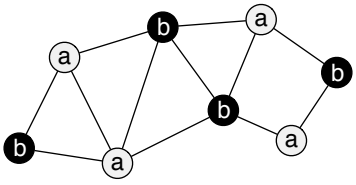


Figure 1: A cut $c\colon V \to \{\mathsf{a}, \mathsf{b}\}$ of weight $w(c) = \frac{10}{12}$.

While the problem of finding a maximum cut (or a good approximation of one) is NP-hard [4, 5, 7, 12, 16], there is a very simple randomised algorithm that finds a relatively large cut: for each node v, pick $c(v) \in \{\mathsf{a}, \mathsf{b}\}$ independently and uniformly at random. We say that c is a *uniform random cut*.

In a uniform random cut, each edge is a cut edge with probability $1/2$. It follows that the expected weight of a uniform random cut is also $1/2$.

1.2 Regular Triangle-Free Graphs

In general graphs, we cannot expect to find cuts that are much better than uniform random cuts. For example, in a complete graph on n nodes, the weight of any cut is at most $1/2 + O(1/n)$.

However, there is a family of graphs that makes for a much more interesting case from the perspective of the max-cut problem: regular triangle-free graphs. Erdős [2] raised the problem of estimating the minimum possible size of a maximum cut in a high-girth graph, and especially the case of triangle-free graphs attracted much interest from the research community [1, 13, 15].

Accordingly, from now on, we assume that G is a d-regular graph for some constant $d \geq 2$, and that there are no triangles (cycles of length three) in G. While focusing on regular triangle-free graphs may seem overly restrictive, our algorithm can be applied in a much more general setting; we will briefly discuss extensions in Section 3.

1.3 Shearer's Algorithm

In triangle-free graphs, it is easy to find cuts that are (in expectation) larger than uniform random cuts. Nevertheless, a uniform random cut is a good starting point.

Shearer's [15] algorithm proceeds as follows. Pick three uniform random cuts c_1, c_2, and c_3. For each node v, let

$$\ell(v) = \big|\{v,u\} \in E : c_1(v) = c_1(u)\}\big|$$

be the number of like-minded neighbours in c_1. Then the output of a node v is

$$c(v) = \begin{cases} c_1(v), & \text{if } \ell(v) < d/2, \\ c_1(v), & \text{if } \ell(v) = d/2 \text{ and } c_3(v) = 0, \\ c_2(v), & \text{if } \ell(v) = d/2 \text{ and } c_3(v) = 1, \\ c_2(v), & \text{if } \ell(v) > d/2. \end{cases} \tag{1}$$

Put otherwise, a node follows c_1 if it seems that there are many cut edges w.r.t. c_1 in its immediate neighbourhood, and it falls back to another cut c_2 otherwise. The value $c_3(v)$ is just used as a random tie-breaker.

Shearer [15] shows that the expected weight of cut (1) is at least

$$\frac{1}{2} + \frac{\sqrt{2}}{8\sqrt{d}} \approx \frac{1}{2} + \frac{0.177}{\sqrt{d}} \tag{2}$$

in d-regular triangle-free graphs.

1.4 Our Algorithm

Shearer's algorithm can be characterised as follows: take a uniform random cut c_1 and then improve it with the help of a *randomised* rule described in (1). In this work, we show that we can do much better with the help of a simple *deterministic* rule.

In our algorithm we pick one uniform random cut c_1. Again, each node v counts the number of like-minded neighbours

$$\ell(v) = \big|\{v,u\} \in E : c_1(v) \neq c_1(u)\}\big|.$$

We define the threshold

$$\tau = \left\lceil \frac{d + \sqrt{d}}{2} \right\rceil. \tag{3}$$

Now the output of a node v is simply

$$c(v) = \begin{cases} c_1(v), & \text{if } \ell(v) < \tau, \\ -c_1(v), & \text{if } \ell(v) \geq \tau. \end{cases} \tag{4}$$

Here $-c_1(v)$ is the complement of $c_1(v)$, that is, $-\mathsf{a} = \mathsf{b}$ and $-\mathsf{b} = \mathsf{a}$. In the algorithm each node simply changes its mind if it seems that there are too many like-minded neighbours.

It is not obvious that such a rule makes sense, or that this particular choice of τ is good. Nevertheless, we show in this work that the expected weight of cut (4) is at least

$$\frac{1}{2} + \frac{9}{32\sqrt{d}} = \frac{1}{2} + \frac{0.28125}{\sqrt{d}}, \tag{5}$$

which is much larger than Shearer's bound (2), at least in low-degree graphs. As a corollary, any d-regular triangle-free graph admits a cut of at least this size.

Our algorithm can be implemented very efficiently in a *distributed* setting: each node only needs to produce one random bit, and the algorithm only requires one communication round. In Shearer's algorithm each node has to produce up to three random bits.

Perhaps the most interesting feature of the algorithm is that it was not designed by a human being—it was discovered by a computer program. Indeed, cuts in triangle-free graphs serve as an example of a computational problem in which *computer-aided methods* can be used to partially *automate algorithm design and analysis* (this process is also known as "algorithm synthesis" or "protocol synthesis"). There is a wide range of other graph problems in which a similar approach has a lot of potential as a shortcut to the discovery of new distributed algorithms.

In Section 2, we outline the procedure that we used to design the algorithm, and then present an analysis of its performance. In Section 3 we discuss how to apply the algorithm in a more general setting beyond regular triangle-free graphs.

2 Algorithm Design and Analysis

We begin this section with an informal overview of so-called neighbourhood graphs. The formal definitions that we use in this work are given after that.

2.1 Neighbourhood Graphs in Prior Work

In the context of distributed systems, the *radius-t neighbourhood* $N(t, v)$ of a node v refers to all information that node v may gather in t communication rounds. Depending on the model of computation that we use, this may include all nodes that are within distance t from v, the edges incident to these nodes, their local inputs, and the random bits that these nodes have generated. The idea is that whatever decision node v takes, it can only depend on its radius-t neighbourhood—any distributed algorithm $\mathcal{A}$ that runs in t communication rounds can be interpreted as a mapping from local neighbourhoods to local outputs.

A *neighbourhood graph* $\mathcal{N}_t$ is a graph representation of all possible radius-t neighbourhoods that a distributed algorithm may encounter. Each node $N \in V(\mathcal{N}_t)$ of the neighbourhood graph corresponds to a possible local neighbourhood: there is at least one communication network in which some node has a local neighbourhood isomorphic to N. We have an edge $\{N_1, N_2\} \in E(\mathcal{N}_t)$ in the neighbourhood graph if there is some communication network in which nodes with local neighbourhoods N_1 and N_2 are adjacent; see Figure 2 for an example.

Neighbourhood graphs are a convenient concept in the study of graph colouring algorithms, both from the perspective of traditional algorithm design [3, 6, 9–11] and from the perspective of computational algorithm design [14]. The key observation is that the following two statements are equivalent:

- $\mathcal{A}\colon V(\mathcal{N}_t) \to \{1, 2, \ldots, k\}$ is a proper colouring of the neighbourhood graph $\mathcal{N}_t$,
- $\mathcal{A}$ is a distributed algorithm that finds a proper k-colouring in t rounds.

To see this, consider any graph G. If nodes u and v are adjacent in G, then their local views $N(t, u)$ and $N(t, v)$ are adjacent in $\mathcal{N}_t$, and by assumption $\mathcal{A}$ assigns a different

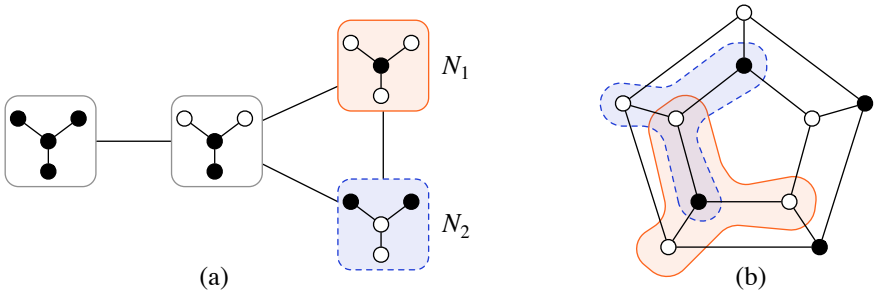


Figure 2: In this example, we study the family $\mathcal{F}$ of 3-regular triangle-free graphs that are labelled with two colours, black and white. (a) A small part of neighbourhood graph $\mathcal{N}_t$ for $t = 1$. (b) There exists a graph $G \in \mathcal{F}$ in which local neighbourhoods N_1 and N_2 are adjacent; hence nodes N_1 and N_2 are adjacent in the neighbourhood graph.

colour to $N(t, u)$ and $N(t, v)$. Hence distributed algorithm $\mathcal{A}$ finds a proper k-colouring of G. Conversely, if algorithm $\mathcal{A}$ finds a proper colouring in any communication network, it defines a proper k-colouring of $\mathcal{N}_t$.

In summary, colourings of the neighbourhood graph correspond to distributed algorithms for graph colouring, and vice versa. In general, a similar property does *not* hold for arbitrary graph problems. For example, there is no one-to-one correspondence between maximal independent sets of $\mathcal{N}_t$ and distributed algorithms that find maximal independent sets [14, Section 8.5].

However, as we will see in this work, we can use neighbourhood graphs also in the context of the maximum cut problem. It turns out that we can define a *weighted* version of neighbourhood graphs, so that there is a one-to-one correspondence between *heavy* cuts in the weighted neighbourhood graph, and randomised distributed algorithms that find *large* cuts in expectation.

2.2 Model of Distributed Computing

Next, we formalise the model of distributed computing that is sufficient for the purposes of our algorithm. Fix the parameter d; recall that we are interested in d-regular triangle-free graphs. Let $G = (V, E)$ be such a graph, and let c be a uniform random cut in G. The *local neighbourhood* of a node v is $N_c(v) = (c(v), \ell_c(v))$, where

$$\ell_c(v) = \big|\{v, u\} \in E : c(v) = c(u)\}\big|$$

is the number of neighbours with the same random bit. Note that there are only $2d + 2$ possible local neighbourhoods.

A distributed algorithm is a function $\mathcal{A}$ that associates an output $\mathcal{A}(N) \in \{\mathsf{a}, \mathsf{b}\}$ with each local neighbourhood N. For any d-regular triangle-free graph $G = (V, E)$, function $\mathcal{A}$ defines a randomised process that produces a random cut c' as follows:

1. Pick a uniform random cut c.
2. For each node v, let $c'(v) = \mathcal{A}(N_c(v))$.

We use the notation $\mathcal{A}(G)$ for the random cut c' produced by algorithm $\mathcal{A}$ in graph G. In particular, we are interested in the quantity $\mathrm{E}[w(\mathcal{A}(G))]$, the expected weight of cut c'.

A priori, we might expect that $\mathrm{E}[w(\mathcal{A}(G))]$ would depend on G. However, as we will soon see, this is not the case—it only depends on parameter d and algorithm $\mathcal{A}$.

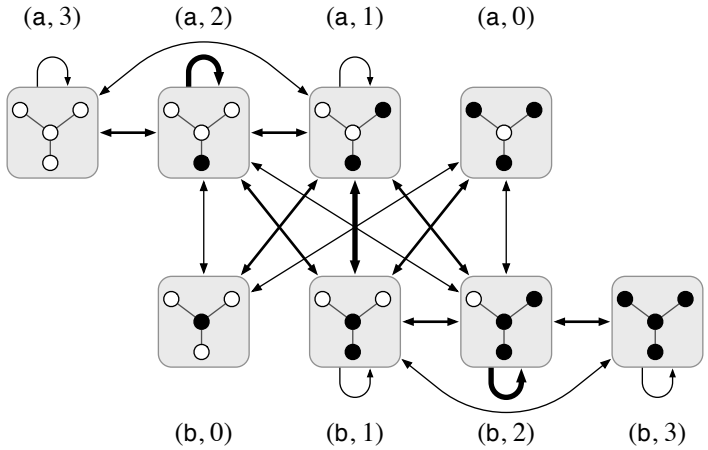


Figure 3: Weighted neighbourhood graph $\mathcal{N}$ for $d = 3$. Edge weights are denoted by line widths; missing edges have weight 0. Note that the digraph is symmetric; however, we prefer the directed representation so that we do not need special treatment for self-loops.

2.3 Weighted Neighbourhood Graph

A *weighted digraph* is a pair $D = (V, w)$ with $w\colon V \times V \to [0, \infty)$. Here V is the set of nodes, and w associates a non-negative *weight* $w(x, y) \geq 0$ with each directed edge $(x, y) \in V \times V$. Let $c\colon V \to \{\mathsf{a}, \mathsf{b}\}$ be a cut in weighted digraph D. The weight of cut c is

$$w(c) = \sum_{\substack{(u,v)\in V\times V,\\ c(u)\neq c(v)}} w(u, v),$$

the total weight of all cut edges.

The *weighted neighbourhood graph* $\mathcal{N} = (V_{\mathcal{N}}, w_{\mathcal{N}})$ is a weighted digraph defined as follows (see Figure 3 for an illustration). The set of nodes

$$V_{\mathcal{N}} = \big\{(k, i) : k \in \{\mathsf{a}, \mathsf{b}\},\, i \in \{0, 1, \ldots, d\}\big\}$$

consists of all possible neighbourhoods that we may encounter in d-regular triangle-free graphs. We define the edge weights as follows:

$$w_{\mathcal{N}}\big((k_1, i_1), (k_2, i_2)\big) = \begin{cases} \dfrac{1}{4^d}\dbinom{d-1}{i_1}\dbinom{d-1}{i_2} & \text{if } k_1 \neq k_2, \\[2ex] \dfrac{1}{4^d}\dbinom{d-1}{i_1-1}\dbinom{d-1}{i_2-1} & \text{if } k_1 = k_2. \end{cases}$$

We follow the convention that $\binom{n}{k} = 0$ for $k < 0$ and $k > n$.

Note that the weights are symmetric, and the total weight of all edges is 1. The following lemma shows that the weight of the edge (N_1, N_2) in the neighbourhood graph equals the probability of "observing" adjacent neighbourhoods of types N_1 and N_2; see Figure 4. Note that the probability does not depend on the choice of graph G or edge $\{u, v\}$.

Lemma 1. *Let G be a d-regular triangle-free graph, and let $\{u, v\}$ be an edge of G. Consider a uniform random cut c of G. Then for any given neighbourhoods $N_1, N_2 \in V_{\mathcal{N}}$ we have*

$$\Pr\big[N_c(u) = N_1 \text{ and } N_c(v) = N_2\big] \;=\; w_{\mathcal{N}}(N_1, N_2).$$

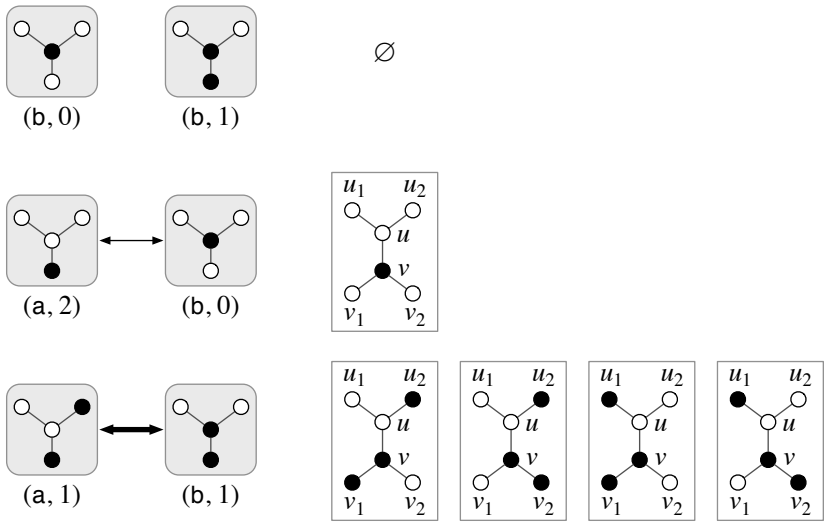


Figure 4: Selected examples of edge weights in the weighted neighbourhood graph $\mathcal{N}$ (see Figure 3). We have $w_{\mathcal{N}}((\mathsf{b},0),(\mathsf{b},1)) = 0$, $w_{\mathcal{N}}((\mathsf{a},2),(\mathsf{b},0)) = 1/64$, and $w_{\mathcal{N}}((\mathsf{a},1),(\mathsf{b},1)) = 1/16$. It is not possible to have a graph in which we have adjacent neighbourhoods of types $(\mathsf{b},0)$ and $(\mathsf{b},1)$. Adjacent neighbourhoods of types $(\mathsf{a},2)$ and $(\mathsf{b},0)$ are fairly rare, while adjacent neighbourhoods of types $(\mathsf{a},1)$ and $(\mathsf{b},1)$ are much more common.

Proof. In what follows, we will denote the neighbours of u by $u_1, u_2, \ldots, u_d$ where $u_d = v$. Similarly, the neighbours of v are $v_1, v_2, \ldots, v_d$ where $v_d = u$. As G is triangle-free, sets $S_u = \{u_1, u_2, \ldots, u_{d-1}\}$ and $S_v = \{v_1, v_2, \ldots, v_{d-1}\}$ are disjoint. In particular, the random variables $c(x)$ for $x \in S_u \cup S_v$ are independent.

Let $N_1 = (k_1, i_1)$ and $N_2 = (k_2, i_2)$. There are two cases. First assume that $k_1 = k_2$. Then

$$\begin{aligned}
\Pr\big[N_c(u) = N_1 \text{ and } N_c(v) = N_2\big] &= \Pr\big[c(u) = k_1 \text{ and } c(v) = k_2\big] \cdot \\
&\quad \Pr\big[|\{y \in S_u : c(y) = k_1\}| = i_1 - 1\big] \cdot \\
&\quad \Pr\big[|\{y \in S_v : c(y) = k_2\}| = i_2 - 1\big] \\
&= \frac{1}{4} \cdot \frac{1}{2^{d-1}} \binom{d-1}{i_1 - 1} \cdot \frac{1}{2^{d-1}} \binom{d-1}{i_2 - 1} \\
&= w_{\mathcal{N}}(N_1, N_2).
\end{aligned}$$

Second, assume that $k_1 \neq k_2$. Then

$$\begin{aligned}
\Pr\big[N_c(u) = N_1 \text{ and } N_c(v) = N_2\big] &= \Pr\big[c(u) = k_1 \text{ and } c(v) = k_2\big] \cdot \\
&\quad \Pr\big[|\{y \in S_u : c(y) = k_1\}| = i_1\big] \cdot \\
&\quad \Pr\big[|\{y \in S_v : c(y) = k_2\}| = i_2\big] \\
&= \frac{1}{4} \cdot \frac{1}{2^{d-1}} \binom{d-1}{i_1} \cdot \frac{1}{2^{d-1}} \binom{d-1}{i_2} \\
&= w_{\mathcal{N}}(N_1, N_2).
\end{aligned}$$

□

2.4 Cuts in Neighbourhood Graphs

Any function $\mathcal{A}\colon V_{\mathcal{N}} \to \{\mathsf{a}, \mathsf{b}\}$ can be interpreted in two ways:

1. A cut of weight $w_{\mathcal{N}}(\mathcal{A})$ in the weighted neighbourhood graph $\mathcal{N}$.
2. A distributed algorithm that finds a cut in any d-regular triangle-free graph: the algorithm picks a uniform random cut c, and then node v outputs $\mathcal{A}(N_c(v))$.

The following lemma shows that the two interpretations are closely related: if $\mathcal{A}$ is a cut of weight w in neighbourhood graph $\mathcal{N}$, then it immediately gives us a distributed algorithm that finds a cut of *expected* weight w in any d-regular triangle-free graph.

Lemma 2. *If $\mathcal{A}\colon V_{\mathcal{N}} \to \{\mathsf{a}, \mathsf{b}\}$ is a cut in neighbourhood graph $\mathcal{N}$, and G is a d-regular triangle-free graph, then* $\mathrm{E}[w(\mathcal{A}(G))] = w_{\mathcal{N}}(\mathcal{A})$.

Proof. Fix a graph G and an edge $\{u, v\}$ of G. By Lemma 1 we have

$$\begin{aligned} w_{\mathcal{N}}(\mathcal{A}) &= \sum_{\mathcal{A}(N_1) \neq \mathcal{A}(N_2)} w_{\mathcal{N}}(N_1, N_2) \\ &= \sum_{\mathcal{A}(N_1) \neq \mathcal{A}(N_2)} \Pr\bigl[N_c(u) = N_1 \text{ and } N_c(v) = N_2\bigr] \\ &= \Pr\bigl[\mathcal{A}(N_c(u)) \neq \mathcal{A}(N_c(v))\bigr]. \end{aligned}$$

The claim follows by summing over all edges $\{u, v\}$ of G. □

2.5 Computational Algorithm Design

Now we have all the tools that we need. Lemma 2 gives a one-to-one correspondence between large cuts of the neighbourhood graph and distributed algorithms that find large cuts. For any fixed value of d, the task of designing a distributed algorithm is now straightforward:

1. Construct the weighted neighbourhood graph $\mathcal{N}$.
2. Find a heavy cut in $\mathcal{N}$.

See Figure 5 for an example. For $d = 3$, the heaviest cut $\mathcal{A}_{\mathrm{opt}}$ of $\mathcal{N}$ is

$$\mathcal{A}_{\mathrm{opt}}((k, i)) = \begin{cases} k & \text{if } i < 3, \\ -k & \text{if } i \geq 3. \end{cases} \tag{6}$$

This is also the best possible algorithm for this value of d, for the model of computing that we defined in Section 2.2.

Remark 1. The reader may want to compare (6) with Section 1.4. For $d = 3$, the algorithms are identical, albeit with a slightly different notation. Note that $\tau_3 = 3$.

Of course finding a maximum-weight cut is hard in the general case. However, in this particular case neighbourhood graphs are relatively small (only $2d + 2$ nodes).

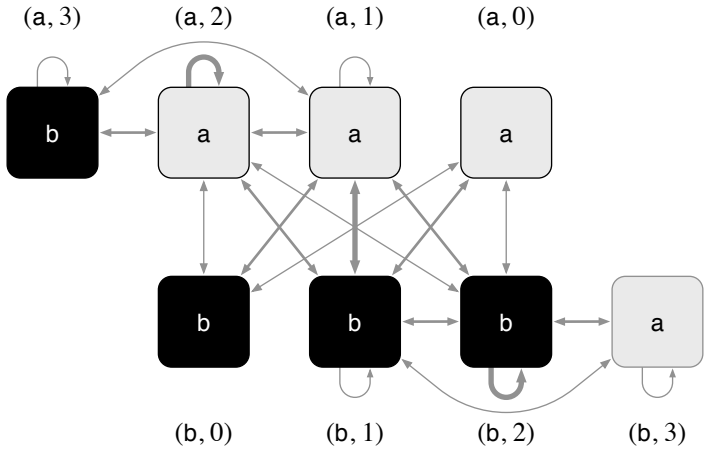


Figure 5: Maximum-weight cut in the weighted neighbourhood graph for $d = 3$.

d:	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
τ_{opt}:	2	3	3	4	5	5	6	6	7	7	8	9	9	10	10	11	11	12	12	13	14	14	15	15	16	16	17	17	18	18	19

Table 1: Optimal threshold τ_{opt} for small values of d.

While the smallest cases could be easily solved with brute force, slightly more refined approaches are helpful for moderate values of d. We took the following approach. First, we reduced the max-weight-cut instance $\mathcal{N}$ to a max-weight-SAT instance ϕ in a straightforward manner:

- For each node $u \in V_{\mathcal{N}}$ we have a Boolean variable x_u in formula ϕ.
- For each edge (u, v) of weight $w_{\mathcal{N}}(u, v)$ we have two clauses in formula ϕ, both of weight $w_{\mathcal{N}}(u, v)$:

  $$x_u \vee x_v \quad \text{and} \quad \neg x_u \vee \neg x_v$$

  Note that at least one of these clauses is always satisfied, while both of them are satisfied if and only if x_u and x_v have different values.

Now it is easy to see that a variable assignment x of ϕ that maximises the total weight of satisfied clauses also gives a maximum-weight cut $\mathcal{A}$ in $\mathcal{N}$: let $\mathcal{A}(u) = \mathsf{a}$ iff x_u is true. More precisely, the total weight of the clauses satisfied by x is $W + w_{\mathcal{N}}(\mathcal{A})$, where W is the total weight of all edges.

With this reduction, we can then resort to off-the-self max-weight-SAT solvers. In our experiments we used *akmaxsat* solver [8]; with it we can solve the cases $d = 2, 3, \ldots, 32$ very quickly (e.g., the case $d = 32$ on a low-end laptop in less than 5 seconds).

Surprisingly, in all cases the max-weight cut has the following simple structure:

$$\mathcal{A}_\tau((k, i)) = \begin{cases} k & \text{if } i < \tau, \\ -k & \text{if } i \geq \tau. \end{cases} \tag{7}$$

The exact values of τ for the heaviest cuts are given in Table 1; note that all values are slightly larger than $d/2$.

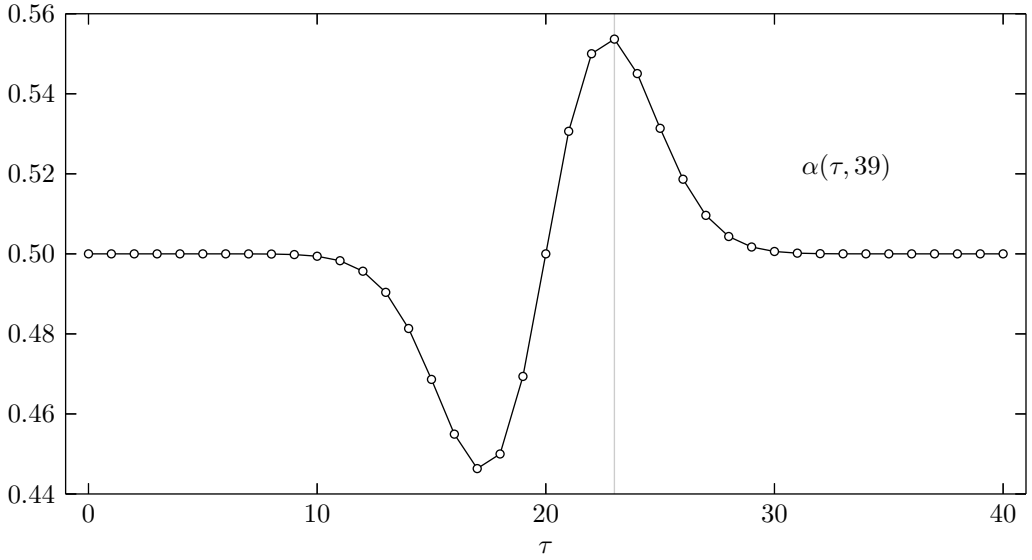


Figure 6: $\alpha(\tau, 39)$ for $\tau = 0, 1, \ldots, 40$.

2.6 Generalisation

Now it is easy to generalise the findings: we can make the educated guess that algorithms of form (7) are good also in the case of a general d. All we need to do is to find a general expression for the threshold τ, and prove that algorithm $\mathcal{A}_\tau$ indeed works well in the general case.

To facilitate algorithm analysis, let us define the shorthand notation

$$\alpha(\tau, d) = w_{\mathcal{N}}(\mathcal{A}_\tau) = \mathrm{E}[w(\mathcal{A}_\tau(G))]$$

for the performance of algorithm $\mathcal{A}_\tau$. It is easy to see that $\alpha(0, d) = \alpha(d + 1, d) = 1/2$, as the threshold value of $\tau = d + 1$ simply means that algorithm $\mathcal{A}_\tau$ outputs a uniform random cut, while $\tau = 0$ means that $\mathcal{A}_\tau$ outputs the complement of the uniform random cut. The general shape of $\alpha(\tau, d)$ is illustrated in Figure 6.

We are interested in the region $\tau > d/2$, where $\alpha(\tau, d) \geq 1/2$. In the following, we derive a relatively simple expression for $\alpha(\tau, d)$ in this region—the proof strategy is inspired by Shearer [15].

Lemma 3. *For all d and $\tau > d/2$ we have*

$$\alpha(\tau, d) = \frac{1}{2} + \frac{1}{4^{d-1}} \binom{d-1}{\tau-1} \sum_{i=d-\tau+1}^{\tau-1} \binom{d-1}{i}.$$

Proof. Fix a triangle-free d-regular graph $G = (V, E)$. Recall that c is a uniform random cut, $N_c(v) = (c(v), \ell_c(v))$ is the local neighbourhood of node $v \in V$, and $\mathcal{A}(N_c(v))$ is the output of algorithm $\mathcal{A}$ at node $v \in V$.

Consider an edge $\{u, v\}$ of G. We will calculate the probability that e is a cut edge. To this end, define

$$\begin{aligned}
p &= \Pr\big[c(u) \neq c(v) \text{ and } \ell_c(u), \ell_c(v) \geq \tau\big], \\
q &= \Pr\big[c(u) \neq c(v) \text{ and } \ell_c(u), \ell_c(v) < \tau\big], \\
r &= \Pr\big[c(u) = c(v) \text{ and either } \ell_c(u) < \tau \leq \ell_c(v) \text{ or } \ell_c(v) < \tau \leq \ell_c(u)\big].
\end{aligned}$$

These are precisely the cases in which $\mathcal{A}(N_c(u)) \neq \mathcal{A}(N_c(v))$; hence $\{u, v\}$ is a cut edge with probability $p + q + r$. For each $x \in \{u, v\}$, let

$$\begin{aligned} p_x &= \Pr\bigl[\ell_c(x) \geq \tau \mid c(u) \neq c(v)\bigr], \\ q_x &= \Pr\bigl[\ell_c(x) < \tau \mid c(u) \neq c(v)\bigr], \\ r_x &= \Pr\bigl[\ell_c(x) \geq \tau \mid c(u) = c(v)\bigr]. \end{aligned}$$

Now we have the following identities:

$$p = \frac{1}{2} p_u p_v, \qquad q = \frac{1}{2} q_u q_v, \qquad r = \frac{1}{2}(r_v(1 - r_u) + r_u(1 - r_v)).$$

By definition, $q_x = 1 - p_x$, and by symmetry, $p_u = p_v$, $q_u = q_v$, and $r_u = r_v$. Hence the probability that $\{u, v\}$ is a cut edge is

$$\begin{aligned} p + q + r &= \frac{1}{2} p_u^2 + \frac{1}{2} q_u^2 + r_u(1 - r_u) = \frac{1}{2} + p_u(p_u - 1) + r_u(1 - r_u) \\ &= \frac{1}{2} - p_u q_u + r_u(p_u + q_u - r_u) = \frac{1}{2} + (r_u - p_u)(q_u - r_u). \end{aligned} \tag{8}$$

An argument similar to what we used in Lemma 1 gives

$$p_u = \frac{1}{2^{d-1}} \sum_{i=\tau}^{d-1} \binom{d-1}{i}, \qquad q_u = \frac{1}{2^{d-1}} \sum_{i=0}^{\tau-1} \binom{d-1}{i}, \qquad r_u = \frac{1}{2^{d-1}} \sum_{i=\tau-1}^{d-1} \binom{d-1}{i}.$$

Recall that we assumed that $\tau > d/2$; hence $\tau - 1 \geq d - \tau$ and

$$\begin{aligned} 2^{d-1}(r_u - p_u) &= \binom{d-1}{\tau-1}, \\ 2^{d-1}(q_u - r_u) &= \sum_{i=0}^{\tau-1} \binom{d-1}{i} - \sum_{i=0}^{d-\tau} \binom{d-1}{i} = \sum_{i=d-\tau+1}^{\tau-1} \binom{d-1}{i}. \end{aligned}$$

From (8) we therefore obtain

$$p + q + r = \frac{1}{2} + \frac{1}{4^{d-1}} \binom{d-1}{\tau-1} \sum_{i=d-\tau+1}^{\tau-1} \binom{d-1}{i}. \qquad \square$$

Now we can easily find an optimal threshold τ for any given d: simply try all $d/2$ possible values and apply Lemma 3. Figure 7 is a plot of optimal τ for $d = 2, 3, \ldots, 1000$. At least for small values of d, it appears that

$$\tau \approx \frac{d+1}{2} + 0.439\sqrt{d}$$

is close to the optimum. For notational convenience, we pick a slightly larger value

$$\tau = \left\lceil \frac{d + \sqrt{d}}{2} \right\rceil.$$

Now we have arrived at the algorithm that we already described in Section 1.4.

What remains is a proof of the performance guarantee (5). Figure 8 gives some intuition on how good the bounds are.

Theorem 4. *Let* $d \geq 2$ *and*

$$\tau = \left\lceil \frac{d + \sqrt{d}}{2} \right\rceil.$$

Then

$$\alpha(\tau, d) \geq \frac{1}{2} + \frac{9}{32\sqrt{d}}.$$

Proof. See Appendix A. □

3 Conclusions

In this work, we have presented a new randomised distributed algorithm for finding large cuts. The key observation was that the task of designing randomised distributed algorithms for finding large cuts can be reduced to the problem of finding a max-weight cut in a weighted neighbourhood graph. This way we were able to use computers to find optimal algorithms for small values of d. The general form of the optimal algorithms was apparent, and hence the results were easy to generalise.

Our algorithm was designed for d-regular triangle-free graphs. However, it can be easily applied in a much more general setting as well. To see this, recall that $\alpha(\tau, d)$ is not only the expected weight of the cut, but it is also the probability that any individual edge $e = \{u, v\}$ is a cut edge. The analysis only assumes that u and v are of degree d and they do not have a common neighbour. Hence we have the following immediate generalisations.

1. Our algorithm can be applied in triangle-free graphs of *maximum* degree d as follows: a node of degree $d' < d$ simulates the behaviour of $d - d'$ missing neighbours. We still have the same guarantee that each original edge is a cut edge with probability $\alpha(\tau, d)$. The running time of the algorithm is still one communication round; however, some nodes need to produce more random bits.

2. Our algorithm can also be applied in *any* graph, even in those that contain triangles. Now our analysis shows that each edge that is not part of a triangle will be a cut edge with probability $\alpha(\tau, d)$. This observation already gives a simple bound: if at most a fraction ϵ of all edges are part of a triangle, we will find a cut of expected size at least $(1 - \epsilon) \cdot \alpha(\tau, d)$.

Acknowledgements

Computer resources were provided by the Aalto University School of Science "Science-IT" project (Triton cluster), and by the Department of Computer Science at the University of Helsinki (Ukko cluster).

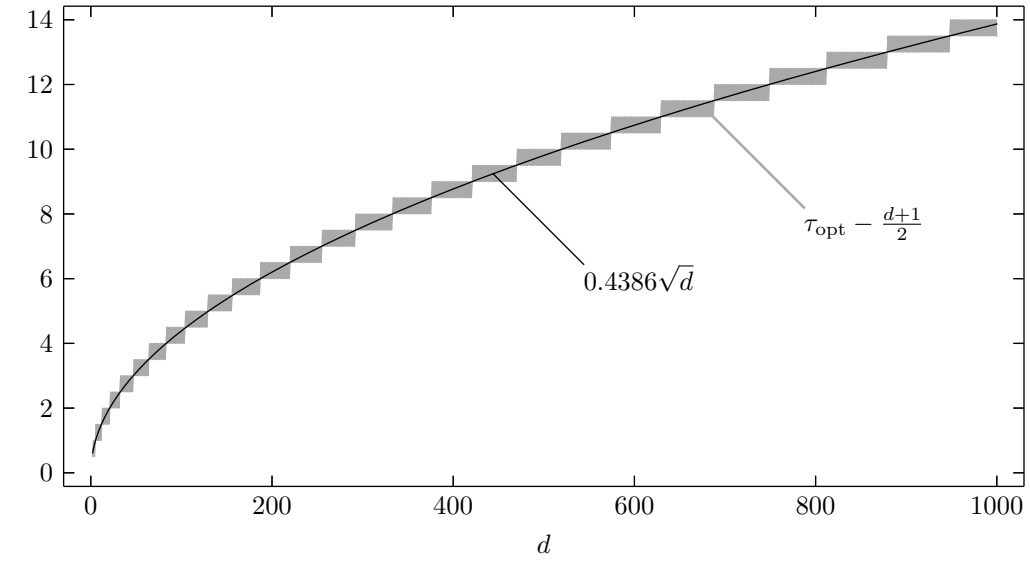
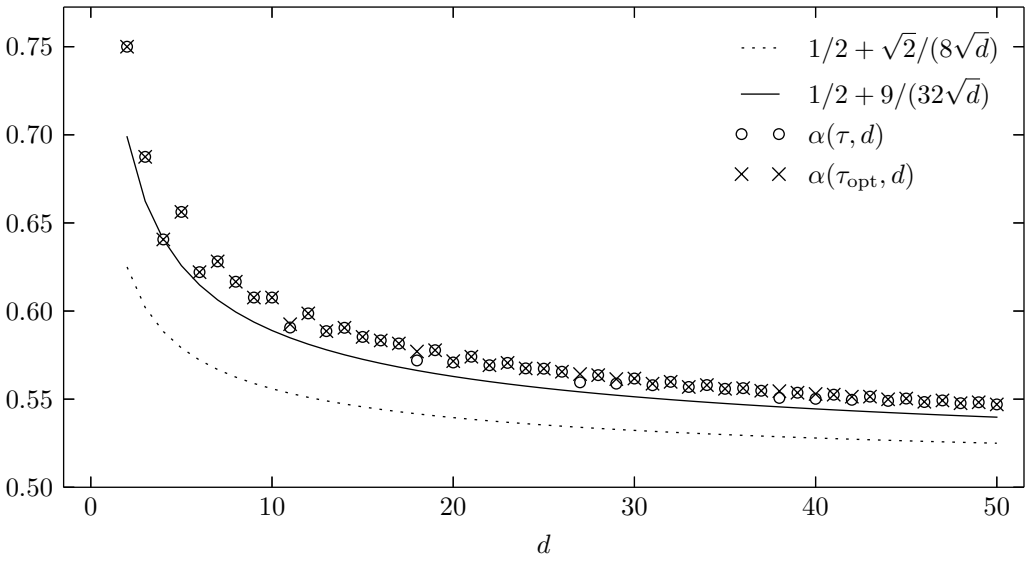


Figure 7: Optimal threshold τ_{opt} for $d = 2, 3, \ldots, 1000$.



$\alpha(\tau_{\text{opt}}, d)$: expected weight of a cut found by an optimal threshold algorithm

$\alpha(\tau, d)$: expected weight of a cut found by our algorithm

$\frac{1}{2} + \frac{9}{32\sqrt{d}}$: a lower bound on $\alpha(\tau, d)$

$\frac{1}{2} + \frac{\sqrt{2}}{8\sqrt{d}}$: a lower bound for Shearer's [15] algorithm

Figure 8: A comparison of performance guarantees for $d = 2, 3, \ldots, 50$. Note that our lower bound and the perfomance of the best threshold algorithm meet at $d = 4$. If we wanted to prove a general bound of the form $1/2 + C/\sqrt{d}$ for a larger $C > 9/32$, we would have to resort to more complicated algorithms (e.g., more than 1 bit of randomness per node or more than 1 communication round).

References

[1] Noga Alon. Bipartite subgraphs. *Combinatorica*, 16(3):301–311, 1996.

[2] Paul Erdős. Problems and results in graph theory and combinatorial analysis. In John Adrian Bondy and U. S. R. Murty, editors, *Proc. Graph Theory and Related Topics (University of Waterloo, July 1977)*, pages 153–163. Academic Press, 1979.

[3] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. In *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2007. doi:10.1007/978-3-540-73420-8_22.

[4] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[5] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48 (4):798–859, 2001. doi:10.1145/502090.502098.

[6] Pierre Kelsen. Neighborhood graphs and distributed $\Delta + 1$-coloring. In *Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT 1996)*, volume 1097 of *Lecture Notes in Computer Science*, pages 223–233. Springer, 1996. doi:10.1007/3-540-61422-2_134.

[7] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

[8] Adrian Kügel. Improved exact solver for the weighted Max-SAT problem. In Daniel Le Berre, editor, *Proc. Pragmatics of SAT Workshop (POS 2010)*, volume 8 of *EasyChair Proceedings in Computing*, pages 15–27, 2012. http://www.easychair.org/publications/?page=2003892821.

[9] Fabian Kuhn and Roger Wattenhofer. On the complexity of distributed graph coloring. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC 2006)*, pages 7–15. ACM Press, 2006. doi:10.1145/1146381.1146387.

[10] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

[11] Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991. doi:10.1137/0404036.

[12] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

[13] Svatopluk Poljak and Zsolt Tuza. Maximum cuts and largest bipartite subgraphs. In William Cook, László Lovász, and Paul Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 181–244. AMS, 1995.

[14] Joel Rybicki. Exact bounds for distributed graph colouring. Master's thesis, Department of Computer Science, University of Helsinki, May 2011. http://urn.fi/URN:NBN:fi-fe201106091715.

[15] James B. Shearer. A note on bipartite subgraphs of triangle-free graphs. *Random Structures & Algorithms*, 3(2):223–226, 1992. doi:10.1002/rsa.3240030211.

[16] Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000. doi:10.1137/S0097539797328847.

A Proof of Theorem 4

We need to prove a lower bound on

$$\alpha(\tau, d) = \frac{1}{2} + \frac{1}{4^{d-1}} \binom{d-1}{\tau-1} \sum_{i=d-\tau+1}^{\tau-1} \binom{d-1}{i}$$

in the region $\tau \approx d/2 + \sqrt{d}/2$. Our general strategy is as follows:

1. Verify cases $d = 2, 3, \ldots, 3000$ with a computer.
2. Prove a closed-form lower bound for $d > 3000$.

The first part is easily solved with a simple Python script or with a short calculation in Mathematica (see Figure 8 for examples of the results for $d = 2, 3, \ldots, 50$). We will now focus on the second part; for that we will need various estimates of binomial coefficients.

The proof given here is certainly not the most elegant way to derive the bound, but it is self-contained and gets the job done. Proving the claim for a "sufficiently large" d would be straightforward. However, we need to show that already a concrete relatively small d such as $d > 3000$ is enough.

We will first approximate binomial coefficients with the normal distribution. Let $J = \{1, 2, 3, 4\}$, and define

$$\delta_j(n) = \left\lfloor j\sqrt{n/32} \right\rfloor, \quad g_j = e^{-j^2/32}$$

for each $j \in \{0\} \cup J$.

Fact 5. *For any $n \geq 1500$ we have*

$$\frac{0.999}{\sqrt{\pi n}} < \frac{1}{4^n}\binom{2n}{n} < \frac{1}{\sqrt{\pi n}}.$$

Lemma 6. *For any $j \in J$, $\delta = \delta_j(n)$, and $n \geq 1500$ we have*

$$\binom{2n}{n+\delta} > 0.995 \cdot g_j \cdot \binom{2n}{n}$$

Proof. We can estimate

$$\begin{aligned}\binom{2n}{n+\delta} \Big/ \binom{2n}{n} &= \frac{n!}{(n+\delta)!} \cdot \frac{n!}{(n-\delta)!} \\ &= \frac{n-\delta+1}{n+1} \cdot \frac{n-\delta+2}{n+2} \cdots \frac{n}{n+\delta} > \left(1 - \frac{\delta}{n}\right)^{\delta} \geq h_j(\delta),\end{aligned}$$

where

$$h_j(\delta) = \left(1 - \frac{j^2}{32\delta}\right)^{\delta}.$$

Now $h_j(\delta) \to g_j$ as $\delta \to \infty$. For each $j \in J$ we can verify that $h_j(\delta) > 0.995 \cdot g_j$ when $\delta \geq \delta_j(1500)$. □

Lemma 7. *For $\delta = \delta_4(n)$ and $n \geq 1500$ we have*

$$\frac{1}{4^n} \sum_{i=-\delta+1}^{\delta} \binom{2n}{n+i} > 0.6088, \qquad \frac{1}{4^n} \sum_{i=-\delta+1}^{\delta-1} \binom{2n}{n+i} > 0.5975.$$

Proof. Here we could apply the Berry–Esseen theorem, but the following simple piecewise estimate is sufficient for our purposes. As

$$\delta_j(n) > j\sqrt{n/32} - 1,$$

we have

$$\sum_{j=1}^{4} \big(\delta_j(n) - \delta_{j-1}(n)\big) \cdot g_j > \left(\sum_{j=1}^{4} g_j \sqrt{n/32}\right) - g_1 > 0.5680\sqrt{n} - 0.9693.$$

Hence using Fact 5 and Lemma 6 we have

$$\begin{aligned}
\frac{1}{4^n} \sum_{i=1}^{\delta} \binom{2n}{n+i} &\geq \frac{1}{4^n} \sum_{j=1}^{4} \big(\delta_j(n) - \delta_{j-1}(n)\big) \binom{2n}{n+\delta_j(n)} \\
&\geq 0.995 \cdot \frac{1}{4^n} \binom{2n}{n} \sum_{j=1}^{4} \big(\delta_j(n) - \delta_{j-1}(n)\big) g_j \\
&> 0.995 \cdot \frac{0.999}{\sqrt{\pi n}} \cdot \big(0.5680\sqrt{n} - 0.9693\big) \\
&> 0.3185 - 0.5436/\sqrt{n} > 0.3044.
\end{aligned}$$

The claim follows from the observations

$$\begin{aligned}
&\frac{1}{4^n} \sum_{i=-\delta+1}^{\delta} \binom{2n}{n+i} > \frac{2}{4^n} \sum_{i=1}^{\delta} \binom{2n}{n+i} > 2 \cdot 0.3044 = 0.6088, \\
&\frac{1}{4^n} \sum_{i=-\delta+1}^{\delta-1} \binom{2n}{n+i} > \Big(2 - \frac{1}{\delta}\Big) \frac{1}{4^n} \sum_{i=1}^{\delta} \binom{2n}{n+i} > 1.9629 \cdot 0.3044 > 0.5975. \qquad \square
\end{aligned}$$

Now we have the estimates that we will use in the proof of Theorem 4. We will consider the odd and even values of d separately.

Odd *d*. Assume that $d = 2n + 1$, $n \geq 1500$. Let

$$\delta = \tau - n = \left\lceil \sqrt{n/2 + 1/4} + 1/2 \right\rceil, \quad \delta' = \delta_4(n),$$

and observe that

$$\sqrt{n/2} < \sqrt{n/2 + 1/4} + 1/2 < \sqrt{n/2} + 1.$$

It follows that

$$\delta' + 1 \leq \delta \leq \delta' + 2.$$

Therefore

$$\begin{aligned}
\alpha(\tau,d) &= \frac{1}{2}+\frac{1}{4^{d-1}}\binom{d-1}{\tau-1}\sum_{i=d-\tau+1}^{\tau-1}\binom{d-1}{i}\\
&= \frac{1}{2}+\frac{1}{4^{2n}}\binom{2n}{n+\delta-1}\sum_{i=-\delta+2}^{\delta-1}\binom{2n}{n+i}\\
&\geq \frac{1}{2}+\frac{1}{4^{2n}}\binom{2n}{n+\delta'+1}\sum_{i=-\delta'+1}^{\delta'}\binom{2n}{n+i}\\
&= \frac{1}{2}+\frac{n-\delta'}{n+\delta'+1}\cdot\frac{1}{4^n}\binom{2n}{n+\delta'}\cdot\frac{1}{4^n}\sum_{i=-\delta'+1}^{\delta'}\binom{2n}{n+i}\\
&> \frac{1}{2}+0.964\cdot 0.995\cdot g_4\cdot\frac{0.999}{\sqrt{\pi n}}\cdot 0.6088 > \frac{1}{2}+\frac{0.2823}{\sqrt{d-1}} > \frac{1}{2}+\frac{9}{32\sqrt{d}}.
\end{aligned}$$

Even *d*. Assume that $d=2n$, $n>1500$. Let

$$\delta=\tau-n=\left\lceil\sqrt{n/2}\right\rceil,\quad \delta'=\delta_4(n).$$

Now we have

$$\delta'\leq\delta\leq\delta'+1.$$

For any $k<n$ we have the identity

$$\begin{aligned}
\sum_{i=-k}^{k}\binom{2n}{n+i} &= \sum_{i=-k}^{k}\left(\binom{2n-1}{n+i-1}+\binom{2n-1}{n+i}\right)\\
&= \sum_{i=-k}^{k}\left(\binom{2n-1}{n-i}+\binom{2n-1}{n+i}\right) = 2\sum_{i=-k}^{k}\binom{2n-1}{n+i}.
\end{aligned}$$

We can use it to derive

$$\begin{aligned}
\alpha(\tau,d) &= \frac{1}{2}+\frac{1}{4^{d-1}}\binom{d-1}{\tau-1}\sum_{i=d-\tau+1}^{\tau-1}\binom{d-1}{i}\\
&= \frac{1}{2}+\frac{1}{4^{2n-1}}\binom{2n-1}{n+\delta-1}\sum_{i=-\delta+1}^{\delta-1}\binom{2n-1}{n+i}\\
&\geq \frac{1}{2}+\frac{1}{4^{2n-1}}\binom{2n-1}{n+\delta'}\sum_{i=-\delta'+1}^{\delta'-1}\binom{2n-1}{n+i}\\
&= \frac{1}{2}+\frac{1}{4^{2n-1}}\cdot\frac{n-\delta'}{2n}\binom{2n}{n+\delta'}\cdot\frac{1}{2}\sum_{i=-\delta'+1}^{\delta'-1}\binom{2n}{n+i}\\
&= \frac{1}{2}+\frac{n-\delta'}{n}\cdot\frac{1}{4^n}\binom{2n}{n+\delta'}\cdot\frac{1}{4^n}\sum_{i=-\delta'+1}^{\delta'-1}\binom{2n}{n+i}\\
&> \frac{1}{2}+0.982\cdot 0.995\cdot g_4\cdot\frac{0.999}{\sqrt{\pi n}}\cdot 0.5975 > \frac{1}{2}+\frac{0.2822}{\sqrt{d}} > \frac{1}{2}+\frac{9}{32\sqrt{d}}.
\end{aligned}$$

This completes the proof of Theorem 4.