On Vertex Bisection Width of Random $d$-Regular Graphs

Josep Díaz$^{1,4}$, Öznur Yaşar Diner$^{1,3}$, Maria Serna$^{1,4}$, and Oriol Serra$^{1,4}$

$^1$ALBCOM Research Group, Computer Science Department. Universitat Politècnica de Catalunya, Barcelona, Spain
$^2$Computer Engineering Department, Kadir Has University, Istanbul, Turkey
$^3$Mathematics Department. Universitat Politècnica de Catalunya, Barcelona, Spain
$^4$Institut de Matemàtiques de la UPC-BarcelonaTech (IMTech), Universitat Politècnica de Catalunya, Barcelona

November 8, 2022

Abstract

Vertex bisection is a graph partitioning problem in which the aim is to find a partition into two equal parts that minimizes the number of vertices in one partition set that have a neighbor in the other set. We are interested in giving upper bounds on the vertex bisection width of random $d$-regular graphs for constant values of $d$. Our approach is based on analyzing a greedy algorithm by using the Differential Equations Method. In this way, we obtain the first known upper bounds for the vertex bisection width in random regular graphs. The results are compared with experimental ones and with lower bounds obtained by Kolesnik and Wormald, (Lower Bounds for the Isoperimetric Numbers

$^{*}$Research supported by the Spanish Agencia Estatal de Investigación under project PID2020-112581GB-C21

$^{†}$Ö. Y. Diner is partially supported by the Scientific and Technological Research Council Tübitak under project BIDEB 2219-1059B191802095 and by Kadir Has University under project 2018-BAP-08

$^{‡}$Research supported by the Spanish Agencia Estatal de Investigación under project PID2020-113082GB-I00.
of Random Regular Graphs, SIAM J. on Disc. Math. 28(1), 553-575, 2014).

Keywords: Vertex bisection width, random regular graphs, differential equations method.

1 Introduction

Partitioning the vertices of a given graph $G$ into subsets, subject to certain constraints, generates an important collection of algorithmic problems, which are used to model a large range of practical applications, see e.g. [2].

The quest for designing fast algorithms for partition models has generated an important body of theoretical study of graph partitioning problems.

One of the partition problems that has received considerable attention, from both the theoretical and the experimental point of view, is the Minimum Edge Bisection as, among other applications, it measures the bottleneck on the amount of information that can flow through a network.

Let $G = (V,E)$ be a graph with $n$ vertices. For any $S \subseteq V$, with $|S| = \lfloor n/2 \rfloor$, the pair $(S, V \setminus S)$ is a partition of the vertex set $V$ into equal sized sets when $n$ is even and differing by one when $n$ is odd. Such a partition is called a bisection of the graph. The edge width $bw(S, V \setminus S)$ of a bisection $(S, V \setminus S)$ is the number of edges connecting vertices in its different parts. The minimum edge bisection or bisection width of $G$ is the minimum width over all bisections:

$$bw(G) = \min \{bw((S, V \setminus S)) : (S, V \setminus S) \text{ is a bisection}\}.$$

The corresponding decision problem, Minimum Edge Bisection, has been a very productive research area, using theoretical and empirical methods.

Concerning its algorithmic complexity, Minimum Edge Bisection is NP-complete for general graphs [19] and it remains in NP-complete when restricted to many graph classes; in particular $d$-regular graphs even for $d = 3$ [7], grids with an arbitrary number of holes and their equivalent class of unit disk graphs [15]. The problem is efficiently solvable for trees [35], hypercubes [33], grid graphs with a constant number of holes [17] and on graphs with bounded treewidth [24]. Regarding poly-time approximation algorithms, it is known that there exists no polynomial time approximation scheme to approximate bisection width within a constant factor [28]. The best approximation algorithm for the problem, achieves a ratio of $O(\log n)$
From the practical point of view, there has been a slow improvement of
efficient heuristics, from the simulated annealing [27] to one that in practice
is very efficient [10]. There are also various software packages dedicated to
solving partitioning problems, including the MINIMUM EDGE BISECTION, see
for example METIS 0.7.3 [26].

A fruitful line of research has been the study of asymptotic tight bounds
for the bisection width of random \(d\)-regular graphs, in particular for cubic
graphs \((d = 3)\). Buser [8] showed the existence of a cubic graph \(G\) with
\(bw(G) > \frac{n}{256}\). Bollobás [4] extended Buser’s result to show that for
almost all cubic graphs \(G\) one has \(bw(G) > \frac{n}{17}\), and obtained a lower bound
for \(d\)-regular graphs for all \(d \geq 3\). Clark and Entringer [9] showed that
for almost all cubic graphs, \(bw(G) > \frac{(n+138)}{3}\). They also proved that almost
every \(d\)-regular \(G\) with \(cdn \leq bw(G)\) where \(cd \to d/4\) as \(d \to \infty\), which
is asymptotically correct. In the last 30 years, several improvements have
tightened the bounds for bisection width. Alon [1] give upper bounds for
the edge bisection width for general values of the degree \(d\). The method
in [1] can be turned into a randomized algorithm to obtain bisections with
small edge width. These bounds were later improved by Lyons [32]. Lichev
and Mitsche [31] obtained very tight upper and lower bounds for the cubic
case. Díaz, Serna and Wormald [14] gave a randomized greedy algorithm for
constructing bisections with small edge width which provide, for \(4 \leq d \leq 12\),
the best known asymptotically almost surely (a.a.s.) upper bounds for the
edge-bisection of random \(d\)-regular graphs. Recall that an event is said to
happen a.a.s. if it happens with probability tending to 1 when \(n \to \infty\). In
[14], the greedy algorithm is analyzed by means of the Differential Equation
Method (DEM). The DEM, introduced by Kurtz [30], has been used in var-
ious combinatorial optimization problems [25, 11]. The general framework
to use the DEM to bound the solution to a given problem is to design a
greedy algorithm that at each step increments the size of a solution; if \(X_t\) is
a random variable indicating the size of the solution at step \(t\), we consider
the expected change of \(|X_{t+1} - X_t|\) in one step of the algorithm, and define
the differential equation indicated by this expected change. The solution
to that differential equation is the expected size of the solution. More-
over, as the stochastic sequence of those random variables should form a
semi-martingale, the expected value is concentrated, giving an approximate
solution to the problem, see e.g. Wormald [40, Theorem 5.1].

In this paper we adapt this strategy to obtain upper bounds on the vertex
bisection width of random \(d\)-regular graphs for \(d\) constant.

Given a graph \(G = (V,E)\) and a partition \((S,V \setminus S)\), let \(S^* = \{u \in S :\)
∃v ∈ V \ S, uv ∈ E} and let let \( S^* = \{v ∈ V \setminus S : ∃w ∈ S, with uv ∈ E\} \). The vertex width of the bisection \((S, V \setminus S)\) is defined as

\[
\text{vbw}(S, V \setminus S) = \min\{|S^*|, |\overline{S^*}|\},
\]

and the vertex bisection width of \( G \) is defined as

\[
\text{vbw}(G) = \min_{\{(S, V \setminus S)\}} \{\text{vbw}(S, V \setminus S)\}.
\]

In the Min vertex bisection problem the aim is to find the bisection with minimal bisection width.

We note that a slightly different definition of vertex bisection width, which considers instead the minimum number of vertices to remove in order to obtain a disconnected partition of the graph in two equal sized subgraphs, can be found in the literature, see for example [38]. In this paper we will stick to the former, somewhat more natural, definition of vertex bisection width.

It is straightforward to obtain the following trivial bounds connecting edge and vertex bisection widths for any \( d \)-regular graph \( G \):

\[
\frac{\text{bw}(G)}{d} ≤ \text{vbw}(G) ≤ \text{bw}(G).
\]

The problem of estimating the vertex bisection width of \( d \)-regular graphs asks for the appropriate value of \( c \in [1/d, 1] \) such that \( \text{vbw}(G) = c \cdot \text{bw}(G) \).

Both the edge and vertex bisection problems are particular instances of the edge and vertex graph isoperimetric problems, where the minimization is taken over all sets of a given cardinality (not necessarily \( n/2 \)). The example of the \( n \)-cube, where an exact solution for the two problems is known (see Harper [20, 21]) illustrates the different structural behaviour of the two problems. Roughly speaking, the solutions for the edge isoperimetric problem are close to subspaces of the space of binary sequences, which turn out to maximize vertex boundary, whereas the ones for the vertex isoperimetric problem are close to balls, which maximize edge boundary.

In spite that the Min Vertex Bisection arises in important applications, as the study of message passing and fault tolerance in interconnection networks, see e.g. [12, 37], the problem has received little attention, in particular from the theoretical point of view.

The problem is known to be NP-complete for general graphs and it is solvable in polynomial time for hypercubes [6]. There have been heuristics based in integer linear programming and branch and bound [16, 22], where
due to the exhaustive search nature of the techniques, their applicability is reduced only to graphs of small size.

Recently, Kolesnik and Wormald [29] obtained a.a.s. lower bounds for the vertex bisection width of random $d$-regular graphs for every $d \geq 3$.

One of the main motivations of this paper is to provide corresponding upper bounds by producing bisections with small width. This is achieved by proposing a randomized greedy algorithm which is suited for the vertex bisection problem, and analyze it by means of the Differential Equation Method.

The paper is organized as follows. In Section 2 we describe a greedy algorithm to obtain a partition with small vertex bisection of a $d$-regular graph, which provides upper bounds for the parameter. In section 3 we present a modification of the previous algorithm, suitable for its analysis by the Differential Equations Method, when applied to random $d$-regular graphs. The results and discussion, both experimental and by the differential equation method, are presented in Section 4. The paper concludes in section 5 with some final remarks and lines of future research. In the Appendix we provide additional information on the experimental part of our contribution.

2 A greedy algorithm

Upper bounds for the vertex bisection width $\text{vbw}(G)$ of a graph $G = (V, E)$ are obtained by exhibiting vertex bisections, or equivalently, by providing balanced Red/Green vertex colorings $(R, G)$, with $|R| - |G| \leq 1$, with a set $R_0 \subset R$ of interior vertices of color red as large as possible, as the vertex bisection width of such a partition is $|R| - |R_0|$.

For a vertex $x_0$ and an integer $r$, the ball $B(x_0, r)$ with center $x_0$ and radius $r$ is set of vertices of $G$ at distance at most $r$ from $x_0$. A natural guess for bisections with small vertex width may consider that the set in the partition will contain a large ball. A second consideration is that, in such a partition, one expects to have vertices with a large number of neighbors on the other side of the partition. These two considerations suggest the greedy Algorithm 1 to produce suitable vertex bisections with small vertex bisection.

We next explain Algorithm 1. Let $r_c(x_0)$ be the smallest $r$, $1 \leq r \leq n$, such that $|B(x_0, r)| > n/2$. Our algorithm for computing a vertex bisection starts by coloring RED a ball around a randomly chosen vertex $x_0$ of radius $r_0(x_0) = r_c(x_0) - 2$. For simplicity we write $r_0$ and $r_c$ when $x_0$ is clear form the context. We next proceed to enlarge the set $R$ of red vertices up to $n/2$
Algorithm 1 Greedy algorithm to create a vertex bisection of a given $d$-regular graph $G = (V, E)$

Input: A $d$-regular graph $G$ with $|V| = n$
Choose a random vertex $x_0$
$R = \emptyset$, $B = \{x_0\}$, $D = N(x_0)$
$\{R \cup B = B(x_0, 0), R \cup B \cup D = B(x_0, 1)\}$

{Phase 1}
while $|R \cup B \cup D| \leq n/2$ do
$R = R \cup B$, $B = D$
$D = \{N(u) \cap (V \setminus (R \cup B)) \mid u \in R \cup B\}$
$\{R = B(x_0, r_0), R \cup B = B(x_0, r_0 + 1), R \cup B \cup D = B(x_0, r_c)\}$

{Phase 2}
while $|R| < n/2$ do
For $0 < i < d$, let $R_i = \{u \in R \mid |N(u) \setminus R| = i\}$
Let $j = \min\{i \mid |R_i| > 0\}$
Choose a random vertex $v$ from $R_j$
Choose a random vertex $w$ from $N(v) \setminus R$
$R = R \cup \{w\}$
Return $(R, V \setminus R)$.

in a greedy fashion. This is done by first selecting a colored vertex with the smallest possible number of uncolored neighbors and then by coloring red one of its uncolored neighbors. The set $R$ of red vertices returned by the algorithm contains a ball centered at $x_0$ of radius $r_0 - 1$, ensuring a large subset of interior vertices, so that the expected bisection width will be small. In the second phase of Algorithm 1 we color RED new vertices until we get a bisection. In this phase, we greedily select the vertices with the fewer uncolored neighbors.

We ran versions of the algorithm with distinct choices of $r_0$ to start the second phase of the algorithm. With the choice $r_0 = r_c - 1$ an undesired irregular behaviour is observed for distinct values of the number $n$ of vertices in random $d$-regular graphs, depending on how close is the ball $B(x_0, r_c)$ to a set of cardinality $n/2$. The closer this size to $n/2$, the shorter is the range of execution of the second phase of the algorithm, resulting in larger bisection widths. On the other hand, smaller choices of $r_0$ below $r_c - 2$ slow down the execution time with not producing better results. In Table 1 we show the proportion of vertices in the bisection width obtained in a run of Algorithm 1 with two selections for $r_0$ on graphs of different sizes.
Empirically, it can be seen that the choice $r_0 = r_c - 2$ yields better bisection widths. In Appendix A, we provide several plots comparing the sizes of the balls before and after reaching size $n/2$ as a function of $n$. From the plots, it can be seen that, as expected, the size of the balls follow closely a step function.

Given the graph $G$ as an adjacency list, Algorithm 1 visits the list of neighbours of only red vertices at most $d$ times each, so the overall running time of the algorithm is $O(dn)$.

In our experiments, we use Algorithm 1 (coded in Python-3) to obtain upper bounds for the bisection width of random $d$-regular graphs. Experiments are run on several instances of random regular graphs with number of vertices ranging from $n = 10^5$ to $n = 6 \cdot 10^5$. We selected this range because it covers, for most of the values of $d$, the highest variability on ball’s sizes (see Appendix A). We have used the generator of random $d$–regular graphs included in the Python package NetworkX-2.5. The results, reporting the average of 5 runs of the algorithm on each graph together with the maximum and minimum value are given in Appendix B. Table 2 in Section 4 reports the maximum obtained vertex bisection value for $4 \leq d \leq 10$.

### Table 1: Experimental results of Algorithm 1, for $d = 4$ and different values of $n$, with different selections of $r_0$.

| $n$          | $10^5$ | $2 \cdot 10^5$ | $3 \cdot 10^5$ | $4 \cdot 10^5$ | $5 \cdot 10^5$ | $6 \cdot 10^5$ |
|-------------|--------|----------------|----------------|----------------|----------------|----------------|
| $r_0 = r_c - 2$ | 0.46180 | 0.46630 | 0.46135 | 0.45910 | 0.45905 | 0.46594 |
| $r_0 = r_c - 1$ | 0.47974 | 0.514479 | 0.47929 | 0.473765 | 0.46867 | 0.51469 |

3 Analyzing a greedy algorithm with the differential equation method

Algorithm 1 is not suited for its analysis with the differential equation method as in the first phase, at each step all the vertices in the boundary of a ball are colored RED instead of being colored one by one. This makes the algorithm more efficient in terms of running time, however the difference of increment of vertices between two consecutive steps is too large to transform it into a continuous smooth function, as needed in the DEM.

A more suitable algorithm for the application of the differential equation method to the first phase of Algorithm 1 to be run on random $d$-regular graphs is the one presented in Algorithm 2.
We use the pairing model to generate random $d$-regular graphs with $n$ vertices, where $nd$ is assumed to be even. The model was introduced by Bollobás [3] and by Wormald [40], see Gao and Wormald [18] for a very efficient way to generate a random $d$-regular graph. The basic working of the pairing model chooses a random matching on a set of $nd$ points that are grouped into bags of size $d$, each bag corresponding to a vertex. The resulting object is a $d$-regular multigraph (it can have loops and multiple edges), but a basic result on the pairing model is that, with positive probability bounded away from zero, one gets a $d$-regular simple graph with the uniform distribution. Therefore it follows that every property holding a.a.s. in the pairing model also holds a.a.s. for random $d$-regular graphs.

The pairing model can be equivalently described as a graph process $(G_t, t = 0, 1, \ldots, nd)$, by arbitrarily choosing at each time $t$ an unpaired point and by pairing it with a randomly chosen unpaired point, exposing in this way an edge of the graph, at time $t$ (see e.g. [40].)

### 3.1 Algorithm 2

In this subsection, we first introduce Algorithm 2 which is an adaptation of the first phase of Algorithm 1 to deal with the pairing model. The algorithm implements the first phase of Algorithm 1 coupled with the graph process generating a $d$-regular graph in the pairing model. Algorithm 2 obtains the next ball, by exposing (one by one) the edges out of the current ball. In this way, one step of the algorithm makes only small changes on the status of the RED vertices. In order to deal directly with edges we need to keep additional information about the partially exposed graph.

Algorithm 2 keeps a partition of $V$ with respect to the current set of RED vertices:

- $R_i$ is the set of red vertices with $i$ unpaired points, $0 \leq i < d$.
- $Z_i$ is the set of non red vertices with $i$ unpaired points, $0 \leq i \leq d$.
- $R = \bigcup_0^{d-1} R_i$, is the set of red vertices.
Algorithm 2 Create a description of $B(x_0, r_0 + 1)$ on a random $d$-regular graph $G$ on $n$ vertices in the pairing model

Input: $d$ and $V$ with $|V| = n$
{Notice the random $d$-regular graph is generated as we expose edges}
\[\text{for } i = 0, \ldots, d - 1 \text{ do}\]
\[R_i = \emptyset, Z_i = \emptyset\]
Choose a random vertex $x_0$
\[R_0 = \{x_0\}, Z_{d-1} = \{N(x_0)\}, Z_d = V \setminus (R_0 \cup Z_{d-1}), R = R_0\]
\[\text{while } |R| < n/2 \text{ do}\]
\[\text{for } i = 1 \text{ to } d - 1 \text{ do}\]
\[R_i = Z_i, \text{ Raux}_i = R_i\]
\[Z_i = \emptyset\]
\[R = \cup_{i=0}^{d-1} R_i\]
\[\text{Zaux}_d = Z_d, \text{ Raux} = R\]
{Phase}
\[\text{while } |R| - |R_0| > 0 \text{ do}\]
\[\text{Choose a random unpaired point } x \text{ in vertices from } R \setminus R_0\]
\[\text{Compute the } i \text{ such that } x \in R_i\]
\[\text{Remove } x \text{ from } R_i \text{ and add it to } R_{i-1}\]
\[\text{Choose a random unpaired point } y \text{ from } V \setminus (R_0 \cup Z_0)\]
\[\text{if } y \in R \text{ then}\]
\[\text{Compute the value } j \text{ such that } y \in R_j\]
\[\text{Remove } y \text{ from } R_j \text{ and add it to } R_{j-1}\]
\[\text{else}\]
\[\text{Compute the value } j \text{ such that } y \in Z_j\]
\[\text{Remove } y \text{ from } Z_j \text{ and add it to } Z_{j-1}\]
{The edge $xy$ has been exposed, and the corresponding sets updated}
\[\text{Return } (G, \text{Raux}, \text{Raux}_0, \ldots, \text{Raux}_{d-1}, \text{Zaux}_d).\]
Algorithm 2 enlarges a ball of red vertices around $x_0$ by, at each iteration of the inner while, choosing randomly one unpaired point from a red vertex, then by choosing randomly an unpaired point, from a red or a non red vertex, and by pairing the two selected points. This operation exposes a new edge. After this, it updates the sets $R, R_i$ and $Z_i$. This basic operation is repeated as long as (i) there are unpaired points in RED vertices, and (ii) there are less than $n/2$ RED points. When Algorithm 2 runs out of unpaired points but there are less than $n/2$ red points, we color RED all uncolored points in $\bigcup_{i=1}^{d-1} Z_i$, updating the variables $R, R_i$ and $Z_i$ accordingly. Observe that this process is equivalent to grow $B(x_0, r)$ from $B(x_0, r - 1)$. A phase of Algorithm 2 corresponds to the growth of one additional layer in the ball.

In order to apply the Differential Equation Method to estimate the sizes of the different sets in the output of Algorithm 2, we define several random variables.

- $r_i = |R_i|$, $0 \leq i < d$, counts the number of red vertices with $i$ unpaired points.
- $nR = \sum_{i=0}^{d-1} r_i$, counts the number of red vertices.
- $pR = \sum_{i=0}^{d-1} ir_i$, counts the number of unmatched points in vertices colored red.
- $z_i$, $1 \leq i \leq d$, counts the number of non red vertices with $i$ unmatched points.
- $nZ = \sum_{i=0}^{d} z_i$ counts the number of non red vertices.
- $pZ = \sum_{i=0}^{d} iz_i$ counts the number of unmatched points in vertices not colored red.
- $pW = pR + pW$, counts the total number of unmatched points.

All the variables depend on the step $t$ of the algorithm, but this dependence is not made explicit in the notation.

We run Algorithm 2 in phases, each phase being finished when $|R| - |R_0| = 0$, which correspond to the inner while loop. The input of the next phase, as long as $|R| < n/2$, is initialized by updating the variables $R_i$ to $Z_i$ and $Z_i$ to zero, $0 \leq i \leq d - 1$.

At each step $t$ of the inner loop of the Algorithm 2 we discover one edge of the graph. We compute the expected changes in the values of the variables (up to $O(1/nW)$ terms). For the variables $r_i$, $0 \leq i < d$, these changes are displayed in the following Table,
monitoring the two random choices for the points to be paired, the first one among points counted by \( pR \) and the second one among all \( nW \) available points. The variable \( r_i \) is modified if the points are chosen in \( R_i \) (decreasing its value) or \( R_{i+1} \) (increasing its value). According to the displayed values, the expected change of the variables \( r_i \) is

\[
\Delta(r_i) = -2 \frac{ir_i}{pR} \frac{ir_i}{nW} - \frac{ir_i}{pR} \frac{nW - ir_i - (i + 1)r_{i+1}}{nW} - \frac{pR - ir_i - (i + 1)r_{i+1}}{pR} \frac{ir_i}{nW}
+ \frac{(i + 1)r_{i+1}}{pR} \frac{nW - ir_i - (i + 1)r_{i+1}}{nW} + \frac{pR - ir_i - (i + 1)r_{i+1}}{pR} \frac{(i + 1)r_{i+1}}{nW}
+ 2 \frac{(i + 1)r_{i+1}}{pR} \frac{(i + 1)r_{i+1}}{nW},
\]

A similar analysis applies to the expected change of the variables \( Z_i \) for \( 0 \leq i < d \),

\[
\Delta(Z_i) = -\frac{iz_i}{nW} + \frac{(i + 1)z_{i+1}}{nW}.
\]

and

\[
\Delta(Z_d) = -\frac{dz_d}{nW}.
\]

By applying the differential equation method, these expected changes are translated to the system of differential equations:

\[
r_i' = \frac{1}{nW \cdot pR} (2(ir_i)((i + 1)r_{i+1}) + (nW + pR)((i + 1)r_{i+1} - ir_i)), \quad 0 \leq i < d,
\]
\[
z_i' = \frac{1}{nW} ((i + 1)z_{i+1} - iz_i), \quad 0 \leq i < d,
\]
\[
z_d' = -\frac{1}{nW} dz_d.
\]

The solution to this system of differential equations at the stopping time provides an asymptotically good approximation of the values of the variables.
on the output of Algorithm 2 as ensured by the main result in Wormald [39, Theorem 5.1]. The applicability of that theorem requires only that the random variables in the process have bounded variation (in our case these variations are bounded by two in absolute value) and that the corresponding continuous versions are Lipschitz (in our case they are quadratic functions of the variables) together with the natural integrability conditions of the system in the domain of interest.

Several phases of the algorithm governed by the same system of differential equations are run where each phase stops when there are no unmatched red points left. These results are feeding the initial values of the variables in the next phase, until the final phase, which stops when \( nR = n/2 \).

To analyze the successive systems of differential equations we need to determine suitable initial conditions. For doing so, we take into account that a random \( d \)-regular graph with \( n \) vertices contains a.a.s. a rooted complete \( d \)-ary tree of depth \( k = \log \sqrt{n} \), i.e. all vertices except the root and the leaves have \( (d - 1) \) children (see Bollobás and de la Vega [5]). More explicitly, Makover and McGowan [34] showed that a.a.s. a random \( d \)-regular graph contains a complete \( d \)-ary tree with at least \( n^{1/2-\epsilon} \) vertices for every \( \epsilon > 0 \). The interior vertices of the tree have all its \( d \) points already paired. Each leaf has \( d - 1 \) unpaired points.

For the implementation, we integrate the system of differential equations with the Runge-Kutta solver DOP853 in Python-3 with \( 10^6 \) steps. The system is initialized with \( \epsilon + \epsilon/(d - 1) \) RED vertices, corresponding to a complete \( d \)-ary tree, with \( \epsilon = 10^{-5} \) leaves. Thus the initial values of the variables are \( r_0 = \epsilon/(d - 1) \), \( r_{d-1} = \epsilon \), \( z_d = 1 - \epsilon - \epsilon/(d - 1) \) and the remaining variables are set to zero.

When approximately solving the system of differential equations for a phase in Algorithm 2, we do not expect to get exact zero values on all variables. We stop a phase just before one of the variables in the solution gets below zero. Then, the initial values for the following phase are obtained by, for \( 0 \leq i \leq d - 1 \), accumulating in \( r_i \) the value \( z_i \) and then setting to zero \( z_i \). We stop the process just before over-passing the 0.5 bound. We illustrate, for \( d = 4 \), the evolution of the \( r_i \) and the \( z_i \) variables along phases in Figure 1.

### 3.2 Algorithm 3

The output of Algorithm 2 provides the values of the variables in the one before the last round, when we reach \( |R| = n/2 \). To complete the algorithm we want next to mimic the second phase of Algorithm 1 and analyze it with
Figure 1: The evolution of $r_i$ and $z_i$ variables in the solutions to the system of differential equations. The first line contains the first two phases of the solution of the DEM for Algorithm $\mathcal{A}$, the second line the two last ones. The last two the phase in which the red points form a bisection.
Algorithm 3 Second phase to get a vertex bisection on a random $d$-regular graph $G$ on $n$ vertices

Input: $(G, R, R_0, \ldots, R_{d-1}, Z_d)$, a partially exposed graph and the output of Algorithm 2

\begin{align*}
R_d &= Z_d \\
R &= \bigcup_{i=0}^{d-1} R_i \\
L &= \bigcup_{i=1}^{\lfloor d/2 \rfloor} R_i
\end{align*}

\begin{algorithm}
\textbf{while} $|R| - |R_1| < n/2$ \textbf{do}
\begin{itemize}
  \item Choose a random unpaired point $x$ in vertices from $L$
  \item Choose a random unpaired point $y$ in vertices from $V \setminus R_0$
  \item \{The Edge $xy$ has been exposed and the corresponding sets updated\}
\end{itemize}
Add to $R$ enough neighbors of vertices in $R_1$ to get $|R| = n/2$
Return $(R \setminus R_1, V \setminus (R \setminus R_1)).$
\end{algorithm}

We have been unable to analyze the second phase of Algorithm 1 via
the DEM. Instead, we analyze a variant of the second phase described as
Algorithm 3. In this variant instead of keeping a strict priority by number
of uncovered points, we select the first uncovered point from vertices with
$d/2$ or less uncovered points and the second one from all available points.
As a minor modification, the stopping time is set to $|R| - |R_1| = n/2$ instead
of $|R| = n/2$. This is so because the vertices in $R_1$, when placed in the
complement, contribute in at most $|R_1|$ to the vertex bisection width, which
provides a slight improvement in the sought upper bound.

We keep the same notation as in Algorithm 2 for $R_i$ and $r_i$, $0 \leq i < d$
but now we denote by
\begin{align*}
pL &= \sum_{i=1}^{\lfloor d/2 \rfloor} ir_i, \\
pR &= \sum_{i=1}^{d} R_i,
\end{align*}
the number of unmatched points in vertices colored red that are used as first
point, by $R_d$ the set of vertices with $d$ unmatched points and by

In the same way as for Algorithm 2 we can derive expressions for the
expected changes of the variables $r_i$. For $0 \leq i \leq \lceil d/2 \rceil$, we have

$$\Delta(r_i) = -2 \frac{ir_i^2}{pL pR} \left( \frac{pL - ir_i - (i + 1)r_{i+1}}{pL} \right) - \frac{ir_i}{pL} \left( \frac{pR - ir_i - (i + 1)r_{i+1}}{pR} \right) + \frac{ir_i}{pL} \left( \frac{pL - ir_i - (i + 1)r_{i+1}}{pR} \right) + \frac{(i + 1)r_{i+1}^2}{pR},$$

and for $\lceil d/2 \rceil < i \leq d$, we have

$$\Delta(r_i) = - \frac{ir_i}{pR} + \frac{(i + 1)r_{i+1}}{pR},$$

and

$$\Delta(r_d) = - \frac{dr_d}{pR}.$$ 

By applying the differential equation method, these expected changes are translated to the system of differential equations:

$$r_i' = \frac{1}{pR \cdot pL} \left( 2(ir_i)((i + 1)r_{i+1}) + (pR + pL)((i + 1)r_{i+1} - ir_i) \right), \quad 0 \leq i < \lceil d/2 \rceil$$

$$r_i' = \frac{1}{pR} ((i + 1)r_{i+1} - ir_i), \quad \lceil d/2 \rceil < i < d$$

$$r_d' = - \frac{1}{pR} dr_d.$$ 

The approximate solution to this system of differential equations at the stopping time provides an asymptotically good approximation of the values of the variables on the output of Algorithm 3. For the implementation, as before, we integrate the system of differential equations with the Runge-Kutta solver DOP853 in Python-3 with $10^6$ steps. The system is initialized with the values obtained with the DEM for Algorithm 2.

4 Results

Table 2 summarizes the approximate upper and lower bounds for the vertex bisection width. In all of the cases, following [29], the displayed values give
Figure 2: Values of $\alpha$ as upper bound via the DEM (except for $d = 3$), as upper bound via empirical calculation and as lower bound, with degree $3 \leq d \leq 10$.

| Degree $d$ | DEM       | Exper  | LB      |
|-----------|-----------|--------|---------|
| 3         | 0.27964   | 0.30924| 0.14420 |
| 4         | 0.58103   | 0.46552| 0.28966 |
| 5         | 0.61018   | 0.55903| 0.40859 |
| 6         | 0.65693   | 0.62588| 0.50190 |
| 7         | 0.65640   | 0.67865| 0.57466 |
| 8         | 0.72031   | 0.72051| 0.63178 |
| 9         | 0.88097   | 0.75354| 0.67716 |
| 10        | 0.83769   | 0.77800| 0.71371 |

The proportion $\alpha$ of vertices in the red part which have neighbors in the other part, so that the value of the bisection is $\alpha(n/2)$. The first column corresponds to the DEM upper bound, obtained by approximating the solution of the continuous differential equations generating by the changes of solution provided by the greedy algorithm 2 (except for $d = 3$). As mentioned before, it uses the Runge-Kuta solver DOP853 in Python, with a number $10^7$ of steps initialized with $\epsilon = 10^{-5}$, except for $d = 9$ in which to avoid numerical errors we set $\epsilon = 10^{-4}$. In Appendix B, we provide the empirical results of the execution of Algorithm 1. From these results, in the second column, we present the experimental upper bound for the vertex bisection width on random $d$-regular graphs, the highest minimum value attained in the executions of Algorithm 1 on such graphs. The third column contains the lower bounds values obtained by Kolesnik and Wormald.

Some comments are in order for an interpretation of the results. For $d = 3$, the best upper and lower bounds obtained by Lichev and Mitsche [31] for the edge bisection width of random cubic graphs are

$$0.103295n \leq \text{bw}(G_{n,3}) \leq 0.139822n.$$  

The upper bound for the edge bisection width is also upper bound for vertex bisection width. Algorithm 1 on random cubic graphs gives an upper bound of 0.15782n, while Algorithm 2 with DEM gives 0.24093n, none of which beat the upper bound that can be obtained from the edge bisection width. This may be an indication that for random cubic graphs the structure of the bisection is closer to a matching than to the boundary of a ball.
For \( d > 4 \), the upper bounds on edge bisection for random \( d \)-regular graphs reported in [14], do not provide too much information, as all of them are bigger than \( 0.5n \). For \( d = 4 \), an upper bound for vertex bisection of \( 0.333n \) is given in [13]. This gives an \( \alpha \) value of 0.6666 which is improved by our DEM upper bound.

A second remark concerns the choice of \( n \) in the range \( 10^5 \) to \( 6 \cdot 10^5 \) for the size of graphs in our experiments. As mentioned before, the bisection width computed by Algorithm 1 is not monotone in \( n \). For instance, for \( d = 4 \), the results of Algorithm 1 are displayed in Table 1. The selected range covers graph sizes that are large enough to be significant and that allow to carry out the computation. Besides, the range expands over sufficiently many values of \( n \) to capture the effects of the closeness of the size of the initial ball to \( n/2 \) for most of the considered values of \( d \). Additional results for smaller values of \( n \) are summarized in Appendix B.

The third remark concerns the values of the degrees up to \( d = 10 \). The trivial upper bound for the minimum vertex bisection with is \( n/2 \), meaning that all vertices in one side of the bisection have a neighbour in the other side. Indeed, for larger values of \( d \) the lower bounds in [29] come close to \( n/2 \). For example, for \( d = 100 \) the lower bound in [29] is \( 0.9785(n/2) \). This is also close to the expected value of a random bisection, showing very small variability among distinct bisections. We note that the expected vertex bisection of a random partition into two equal parts is \( n/2(1 - 1/2^d) \) which tends to \( n/2 \) with \( d \to \infty \). Therefore, the problem is meaningful only for a range of small values of the degree. Both algorithms presented here can be run for larger values of \( d \). We have chosen to report only the range of degrees displayed in Table 2 which seem to us the relevant ones, for which the upper bound obtained from the edge bisection width is meaningless and the trivial upper bound of \( n/2 \) can be significantly improved.

A fourth remark obviously concerns the gap between the lower and upper bounds for the vertex bisection width. The lower bound is obtained by a first moment method. However, the analysis is considerably more involved than the analogous application of the first moment method for the lower bound of the edge bisection width used by Bollobás [4]. The union bound involved in the computation is likely to weaken the strength of the lower bounds, which we believe have room for improvement. On the other hand, the choice of a ball as a candidate for minimizing the vertex bisection width looks natural and we suspect that it may be closer to the optimal value.

As a final remark, we have concentrated on the vertex bisection problem instead of the general vertex isoperimetric problem. The latter asks for the minimum boundary of sets with given cardinality, other than \( n/2 \). Both
algorithms presented here are suited to analyze this more general problem, by setting the stopping time at $cn$ for different values of $c$ other than $1/2$. A Python implementation of the algorithms is available on demand.

5 Conclusions

The edge bisection problem, or more general the edge isoperimetric problem, has been intensively studied in the literature and has a wide range of applications. One of the reasons is that there are available tools, from randomized algorithms to spectral techniques, which make the problem more approachable than its related vertex bisection problem, or the more general vertex isoperimetric problem. For constant degree, most of the applications to computer science and other areas, bounds on edge isoperimetry are usually enough even if vertex isoperimetry is concerned. However, from the theoretical point of view, it is a natural and important question to understand the behaviour of the two parameters.

The vertex isoperimetric problem is solved for a few classes of graphs. A particularly significant one is the solutions by Harper [20, 21] for the vertex and edge isoperimetric problems on the hypercube. Interestingly, the minimizers for both problems in the hypercube are structurally very different. This prompted us to investigate the vertex bisection problem for random regular graphs.

For fixed small degree, the best bounds for the edge vertex bisection of random regular graphs have been obtained by Diaz, Serna and Wormald [14] by analyzing a greedy algorithm with the differential equations method. The greedy algorithm used in [14] does not provide good vertex bisectors, indicating that the minimizing bisectors for the case of edge cuts and vertex cuts have distinct structural properties. In this paper we consider a different randomized greedy algorithm to obtain upper bounds on the vertex bisection width of random regular graphs, which is also analyzed with the differential equation method. The results are compared with the lower bounds for this parameter obtained by Kolesnik and Wormald [29] and also with experimental results with a simplified version of the randomized greedy algorithm. To the best of our knowledge our algorithm provides the first upper bounds for the vertex bisection problem of random regular graphs. The gaps between lower and upper bounds certainly leave room for improvement, which is the more relevant natural question left open.
References

[1] N. Alon. On the edge-expansion of graphs. *Combinatorics, Probability and Computing*, 6:145–152, 1997.

[2] C. E. Bichot, P. Siarry (Eds.). *Graph Partitioning*, Wiley, 2011.

[3] B. Bollobáš, A probabilistic proof of an asymptotic formula for the number of labelled regular graphs, *Europ. J. Combinatorics*, 1 (1980) 311–316.

[4] B. Bollobáš, The isoperimetric number of random regular graphs, *European J. Combinatorics*. 9, 241–244, 1988.

[5] B. Bollobáš, W. Fernández de la Vega, The diameter of random regular graphs, *Combinatorica*, 2 125–134, 1982.

[6] Brandes, U., and, Fleischer, D. Vertex bisection is hard, too. *Journal of Graph Algorithms and Applications*, 13(2), 119-131, 2009.

[7] T. Bui, S. Chaudhuri, T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7:171–191, 1987.

[8] P. Buser, On the bipartition of graphs. *Discrete Applied Mathematics*, 9 105-109, 1984.

[9] L. H. Clarke and R. C. Entringer. The bisection width of cubic graphs. *Bull. of the Australian Math. Society*, 39(3), 389 - 396, 1989.

[10] D. Delling, A.V. Goldberg, I. Razenshteyn and R.F. Werneck. *Exact combinatorial branch-and-bound for graph bisection*. In: Proc. of the 14th ALENEX, pp. 30–44, 2012.

[11] J. Díaz and D. Mitsche, The cook-book approach to the differential equation method, *Computer Science Review*. 4 , 129–151, 2010.

[12] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34:313–356, 2002.

[13] J. Díaz, N. Do, M.J. Serna, N.C. Wormald. Bounds on the max and min bisection of random cubic and random 4-regular graphs, Theoretical Computer Science, Volume 307, Issue 3, 2003, Pages 531-547.

[14] J. Díaz, M. Serna, N.C. Wormald. Bounds on the bisection width for random d-regular graphs, *Theoretical Computer Science*, 382 (2), 120-130, 2007.

[15] J. Díaz and G. B. Mertzios. Minimum Bisection is NP-hard on Unit Disk Graphs. *Information and Computation*, 256: 83-92, 2017.

[16] H. Fraire, D. Terán-Villanueva, N.C. García N.C., J.J.G. Barbosa, E.R. Angel and Y.G. Rojas. Exact Methods for the Vertex Bisection Problem. In: O. Castillo, P. Melin, W. Pedrycz and J. Kacprzyk (eds.) *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*. Studies in Computational Intelligence, vol. 547. Springer, 2014.

19
[17] A.E. Feldmann, P. Widmayer. An $O(n^4)$ time algorithm to compute the bisection width of solid grid graphs. In: Proc. of 19th ESA, 143–154, 2011.

[18] P. Gao, and N. Wormald. Uniform generation of random regular graphs. SIAM Journal Computing, 46, 1395–1427, 2017.

[19] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. Theoretical Computer Science, 1, 237–267, 1976.

[20] L. Harper. Optimal assignments of number to vertices. Journal of SIAM, 12, 131–135, 1964.

[21] L. Harper. Optimal numbering and isoperimetric problems. Journal of Combinatorial Theory, 1, 385–393, 1966

[22] P. Jain, G. Saran, K. Srivastava. Branch and bound algorithms for vertex bisection minimization problems. In: Advanced Computing and Communication Tech., 452, Springer, 2016.

[23] K. Jansen, M. Karpinski, A. Lingas, and E. Seidel. Polynomial time approximation schemes for max-bisection on planar and geometric graphs. SIAM J. on Computing. 35 (1) 110–119, 2005.

[24] S. Janson, T. Łuczak and A. Ruciuński, Random Graphs, John Wiley, 2000.

[25] R.M. Karp and M. Sipser, Maximum matchings in sparse random graphs, Proc. 22 IEEE FOCS, 364–375, 1981.

[26] G. Karypis, V. Kumar, METIS Manual, Version 4.0, University of Minnesota, Department of Computer Science, 1998.

[27] B. Kernighan, S. Lin. An efficient heuristic procedure for partitioning graphs. Bell Systems and Technology Journal, 49 (2), 291–307, 1970.

[28] S. Khot. Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In Proc of 43 IEEE FOCS-04, 136–145, 2004.

[29] B. Kolesnik and N. Wormald. Lower Bounds for the Isoperimetric Numbers of Random Regular Graphs. SIAM Journal on Discrete Math., 28(1), 553–575, 2014.

[30] T.G. Kurtz. Solutions of ordinary differential equations as limits of pure Markov jump processes. Journal of Applied Probability, 7, 49–58, 1970.

[31] L. Lichev and D. Mitsche. On the minimum bisection of Random 3-regular graphs. arXiv:2009.00598, 2020.

[32] R. Lyons. Factors of IID on trees. Combinatorics, Probability and Computing, 26(2):285—300, 2017.

[33] R. M. MacGregor. On Partitioning a Graph: A Theoretical and Empirical Study, PhD Thesis, University of California, Berkeley, 1978.
[34] Makover, E. and McGowan, Regular trees in random regular graphs, arXiv:Math:0610858v2, 2010.

[35] K. Nakano. Linear layouts of generalized hypercubes. Intl. J. Foundations of Computer Science, 14, 137–156, 2003.

[36] H. Räcke. Optimal Hierarchical Decompositions for Congestion Minimization in Networks. In: Proc. of the 40th ACM STOC, 255–264, 2008.

[37] T. Sauerwald, and A. Stauffer. Rumor spreading and vertex expansion on regular graphs. In ACM-SIAM SODA, 462–475. SIAM, 2011.

[38] R. van Bevern, A.E. Feldmann, M. Sorge, and O. Suchý. On the Parameterized Complexity of Computing Balanced Partitions in Graphs. Theory of Computing Systems, 57:1–35, 2015.

[39] N.C. Wormald. The Differential Equation Method for random graph processes and greedy algorithms. In: M. Karonski and H. Prömel (Eds.), Lectures on Approximation and Randomized Algorithms, PWN, Warsaw, 73–155, 1999.

[40] N.C. Wormald. Models of random regular graphs. In: Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 267, CUP; 239–298, 1999.
A The evolution of the Balls

The following plots depict the sizes of the balls ($y$-coordinate) as function of the number of nodes ($x$-coordinate), for the different values of $d$. In the plots, $B_0$ corresponds to the size of the ball $B(x_0, r_0)$. Recall that $r_0$ is selected in such a way that $|B(x_0, r_0 + 2)| \geq n$. $B_1$ is the size of $B(x_0, r_0 + 1)$ and $B_2$ that of $B(x_0, r_0 + 2)$. 

![Plots showing the evolution of balls for different values of $d$.](image)
B Experimental results

In the following tables, we present the computed values of the vertex bisections obtained after running Algorithm 1 rounded to the fifth digit.

For each value of $d$ in the interval $[4, 10]$, we generated 5 random graphs of each size and executed the algorithm 5 times on each graph. In the tables, the runs are identified as $r_0$ to $r_5$. We also show the average, maximum and minimum vertex bisection of each graph. We highlight in red the highest minimum vertex bisection and in blue, the same value for each graph size.

For small values of $n$, we omit the results of each run and provide the average, max and min values for the graph having the worst seen vertex bisection.

| $d$ | $n$ | Avg | Max | Min | $n$ | Avg | Max | Min |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 4   | 1000| 0.47480 | 0.48800 | 0.47000 | 4000| 0.46150 | 0.46300 | 0.45950 |
| 4   | 2000| 0.46540 | 0.47700 | 0.45700 | 5000| 0.46008 | 0.46360 | 0.45680 |
| 4   | 3000| 0.46360 | 0.46600 | 0.46133 | 6000| 0.46080 | 0.46233 | 0.45933 |
| 5   | 1000| 0.56680 | 0.57000 | 0.56200 | 4000| 0.56170 | 0.56500 | 0.55900 |
| 5   | 2000| 0.55900 | 0.56300 | 0.55500 | 5000| 0.55920 | 0.56440 | 0.55600 |
| 5   | 3000| 0.56453 | 0.56733 | 0.55933 | 6000| 0.55833 | 0.56233 | 0.55500 |
| 6   | 1000| 0.63400 | 0.64000 | 0.62800 | 4000| 0.62600 | 0.63050 | 0.62150 |
| 6   | 2000| 0.62920 | 0.63200 | 0.62500 | 5000| 0.62848 | 0.63120 | 0.62520 |
| 6   | 3000| 0.62787 | 0.63000 | 0.62600 | 6000| 0.62433 | 0.62667 | 0.62300 |
| 7   | 1000| 0.68240 | 0.68600 | 0.68000 | 4000| 0.67990 | 0.68400 | 0.67650 |
| 7   | 2000| 0.68300 | 0.68500 | 0.68100 | 5000| 0.67952 | 0.68080 | 0.67680 |
| 7   | 3000| 0.68173 | 0.68733 | 0.67733 | 6000| 0.68093 | 0.68567 | 0.67867 |
| 8   | 1000| 0.72440 | 0.72800 | 0.72000 | 4000| 0.72220 | 0.72650 | 0.71900 |
| 8   | 2000| 0.72020 | 0.72200 | 0.71800 | 5000| 0.72440 | 0.72760 | 0.72120 |
| 8   | 3000| 0.72400 | 0.72800 | 0.72133 | 6000| 0.72153 | 0.72333 | 0.72033 |
| 9   | 1000| 0.75640 | 0.76600 | 0.75400 | 4000| 0.75490 | 0.75600 | 0.75300 |
| 9   | 2000| 0.75460 | 0.75700 | 0.75300 | 5000| 0.75504 | 0.75640 | 0.75400 |
| 9   | 3000| 0.75520 | 0.75667 | 0.75400 | 6000| 0.75453 | 0.75733 | 0.75233 |
| 10  | 1000| 0.78360 | 0.78600 | 0.77800 | 4000| 0.78000 | 0.78250 | 0.77750 |
| 10  | 2000| 0.78100 | 0.78300 | 0.78000 | 5000| 0.77944 | 0.78080 | 0.77760 |
| 10  | 3000| 0.77960 | 0.78000 | 0.77867 | 6000| 0.77867 | 0.77967 | 0.77800 |

Table 2: Experimental results of Algorithm 1 for small values of $n$. 
Table 3: Experimental results of Algorithm 1 for $d = 3$.
Table 4: Experimental results of Algorithm 1, for $d = 4$. 

| $n$     | $r_0$      | $r_1$      | $r_2$      | $r_3$      | $r_4$      | Avg        | Max        | Min        |
|---------|------------|------------|------------|------------|------------|------------|------------|------------|
| 100000  | 0.46270    | 0.46048    | 0.46104    | 0.45950    | 0.46116    | 0.46270    | 0.45950    | 0.45920    |
| 100000  | 0.46190    | 0.46246    | 0.45972    | 0.46142    | 0.46132    | 0.46246    | 0.45972    | 0.46072    |
| 100000  | 0.46204    | 0.46166    | 0.46096    | 0.46046    | 0.46166    | 0.46116    | 0.46270    | 0.46270    |
| 100000  | 0.45968    | 0.4612    | 0.45928    | 0.46180    | 0.46147    | 0.46426    | 0.46535    | 0.46416    |

| $n$     | $r_0$      | $r_1$      | $r_2$      | $r_3$      | $r_4$      | Avg        | Max        | Min        |
|---------|------------|------------|------------|------------|------------|------------|------------|------------|
| 200000  | 0.46535    | 0.46584    | 0.46698    | 0.46416    | 0.46561    | 0.46698    | 0.46416    | 0.46552    |
| 200000  | 0.46647    | 0.46552    | 0.46702    | 0.46620    | 0.46630    | 0.46702    | 0.46620    | 0.46552    |
| 200000  | 0.46454    | 0.46642    | 0.46588    | 0.46612    | 0.46727    | 0.46454    | 0.46454    | 0.46454    |
| 200000  | 0.46687    | 0.46653    | 0.46419    | 0.46446    | 0.46551    | 0.46687    | 0.46687    | 0.46419    |

| $n$     | $r_0$      | $r_1$      | $r_2$      | $r_3$      | $r_4$      | Avg        | Max        | Min        |
|---------|------------|------------|------------|------------|------------|------------|------------|------------|
| 300000  | 0.46076    | 0.46293    | 0.46062    | 0.46041    | 0.46111    | 0.46293    | 0.46041    | 0.46041    |
| 300000  | 0.46143    | 0.46169    | 0.46154    | 0.46090    | 0.46105    | 0.46169    | 0.46169    | 0.46169    |
| 300000  | 0.46085    | 0.46136    | 0.46140    | 0.46135    | 0.46106    | 0.46140    | 0.46140    | 0.46140    |
| 300000  | 0.46111    | 0.46082    | 0.46127    | 0.46112    | 0.46099    | 0.46127    | 0.46127    | 0.46127    |

| $n$     | $r_0$      | $r_1$      | $r_2$      | $r_3$      | $r_4$      | Avg        | Max        | Min        |
|---------|------------|------------|------------|------------|------------|------------|------------|------------|
| 400000  | 0.45826    | 0.45841    | 0.45890    | 0.45849    | 0.45856    | 0.45890    | 0.45890    | 0.45826    |
| 400000  | 0.45788    | 0.45823    | 0.45821    | 0.45946    | 0.45856    | 0.45946    | 0.45856    | 0.45788    |
| 400000  | 0.45814    | 0.45783    | 0.45900    | 0.45881    | 0.45846    | 0.45900    | 0.45900    | 0.45783    |
| 400000  | 0.45884    | 0.45929    | 0.45901    | 0.45906    | 0.45910    | 0.45906    | 0.45906    | 0.45884    |

| $n$     | $r_0$      | $r_1$      | $r_2$      | $r_3$      | $r_4$      | Avg        | Max        | Min        |
|---------|------------|------------|------------|------------|------------|------------|------------|------------|
| 500000  | 0.45882    | 0.45930    | 0.45794    | 0.45813    | 0.45880    | 0.45860    | 0.45930    | 0.45794    |
| 500000  | 0.45952    | 0.45899    | 0.45800    | 0.45856    | 0.45821    | 0.45860    | 0.45952    | 0.45800    |
| 500000  | 0.45833    | 0.45933    | 0.45906    | 0.45869    | 0.45905    | 0.45983    | 0.45983    | 0.45833    |
| 500000  | 0.45890    | 0.45908    | 0.45836    | 0.45914    | 0.45886    | 0.45914    | 0.45886    | 0.45836    |

| $n$     | $r_0$      | $r_1$      | $r_2$      | $r_3$      | $r_4$      | Avg        | Max        | Min        |
|---------|------------|------------|------------|------------|------------|------------|------------|------------|
| 600000  | 0.46539    | 0.46567    | 0.46519    | 0.46570    | 0.46540    | 0.46547    | 0.46614    | 0.46540    |
| 600000  | 0.46555    | 0.46517    | 0.46573    | 0.46590    | 0.46558    | 0.46590    | 0.46590    | 0.46590    |
| 600000  | 0.46611    | 0.46532    | 0.46589    | 0.46624    | 0.46591    | 0.46624    | 0.46624    | 0.46624    |
| 600000  | 0.46540    | 0.46590    | 0.46635    | 0.46605    | 0.46582    | 0.46635    | 0.46635    | 0.46540    |
| 600000  | 0.46523    | 0.46638    | 0.46563    | 0.46632    | 0.46614    | 0.46594    | 0.46638    | 0.46523    |
Table 5: Experimental results of Algorithm 1, for $d = 5$. 

| $n$   | $r_0$  | $r_1$  | $r_2$  | $r_3$  | $r_4$  | Avg  | Max  | Min  |
|-------|--------|--------|--------|--------|--------|------|------|------|
| 10000 | 0.55856| 0.55482| 0.55674| 0.55654| 0.55606| 0.55654| 0.55856| 0.55482|
| 10000 | 0.55614| 0.55488| 0.55604| 0.55652| 0.55607| 0.55676| 0.55676| 0.55488|
| 10000 | 0.55642| 0.55522| 0.55622| 0.55866| 0.55708| 0.55672| 0.55866| 0.55522|
| 10000 | 0.55600| 0.55546| 0.55802| 0.55710| 0.55653| 0.55802| 0.55546| 0.55546|
| 10000 | 0.55648| 0.55626| 0.55692| 0.55616| 0.55744| 0.55654| 0.55616| 0.55616|
| 20000 | 0.56019| 0.55838| 0.55864| 0.55914| 0.55929| 0.55913| 0.56019| 0.55838|
| 20000 | 0.55961| 0.55907| 0.55936| 0.55903| 0.55998| 0.55941| 0.55998| 0.55903|
| 20000 | 0.55915| 0.55881| 0.55874| 0.55920| 0.55941| 0.55906| 0.55941| 0.55874|
| 20000 | 0.55897| 0.55857| 0.55939| 0.55874| 0.55941| 0.55907| 0.55941| 0.55857|
| 20000 | 0.55941| 0.56002| 0.55881| 0.55868| 0.55951| 0.55929| 0.56002| 0.55868|
| 30000 | 0.55817| 0.55749| 0.55803| 0.55745| 0.55693| 0.55761| 0.55817| 0.55761|
| 30000 | 0.55700| 0.55735| 0.55733| 0.55743| 0.55757| 0.55734| 0.55757| 0.55700|
| 30000 | 0.55740| 0.55729| 0.55787| 0.55799| 0.55736| 0.55758| 0.55799| 0.55729|
| 30000 | 0.55704| 0.55790| 0.55741| 0.55703| 0.55767| 0.55741| 0.55790| 0.55703|
| 30000 | 0.55697| 0.55602| 0.55881| 0.55868| 0.55951| 0.55929| 0.56002| 0.55868|
| 40000 | 0.55574| 0.55636| 0.55708| 0.55633| 0.55663| 0.55643| 0.55708| 0.55574|
| 40000 | 0.55735| 0.55679| 0.55613| 0.55627| 0.55625| 0.55656| 0.55735| 0.55613|
| 40000 | 0.55716| 0.55660| 0.55716| 0.55709| 0.55600| 0.55680| 0.55716| 0.55600|
| 40000 | 0.55644| 0.55556| 0.55676| 0.55668| 0.55669| 0.55643| 0.55676| 0.55556|
| 40000 | 0.55717| 0.55619| 0.55646| 0.55686| 0.55598| 0.55653| 0.55682| 0.55598|
| 50000 | 0.55618| 0.55634| 0.55999| 0.55627| 0.55637| 0.55623| 0.55637| 0.55599|
| 50000 | 0.55600| 0.55633| 0.55690| 0.55615| 0.55633| 0.55620| 0.55659| 0.55606|
| 50000 | 0.55667| 0.55641| 0.55539| 0.55555| 0.55594| 0.55599| 0.55667| 0.55539|
| 50000 | 0.55608| 0.55592| 0.55536| 0.55627| 0.55604| 0.55593| 0.55627| 0.55536|
| 50000 | 0.55622| 0.55589| 0.55563| 0.55658| 0.55549| 0.55596| 0.55658| 0.55549|
| 60000 | 0.55573| 0.55561| 0.55525| 0.55591| 0.55518| 0.55553| 0.55591| 0.55518|
| 60000 | 0.55586| 0.55524| 0.55593| 0.55525| 0.55573| 0.55560| 0.55593| 0.55524|
| 60000 | 0.55520| 0.55529| 0.55530| 0.55589| 0.55591| 0.55552| 0.55591| 0.55520|
| 60000 | 0.55632| 0.55534| 0.55597| 0.55538| 0.55564| 0.55573| 0.55632| 0.55534|
| 60000 | 0.55510| 0.55536| 0.55612| 0.55547| 0.55561| 0.55553| 0.55612| 0.55510|
|     | $r_0$   | $r_1$   | $r_2$   | $r_3$   | $r_4$   | Avg    | Max    | Min    |
|-----|---------|---------|---------|---------|---------|--------|--------|--------|
| 100000 | 0.62408 | 0.62328 | 0.62370 | 0.62486 | 0.62393 | 0.62486 | 0.62328 |
| 100000 | 0.62522 | 0.62402 | 0.62434 | 0.62492 | 0.62462 | 0.62522 | 0.62384 |
| 100000 | 0.62356 | 0.62564 | 0.62524 | 0.62436 | 0.62486 | 0.62564 | 0.62356 |
| 100000 | 0.62436 | 0.62432 | 0.62442 | 0.62416 | 0.62402 | 0.62442 | 0.62282 |
| 200000 | 0.62647 | 0.62691 | 0.62602 | 0.62527 | 0.62626 | 0.62691 | 0.62527 |
| 200000 | 0.62605 | 0.62558 | 0.62717 | 0.62633 | 0.62627 | 0.62717 | 0.62558 |
| 200000 | 0.62581 | 0.62690 | 0.62649 | 0.62646 | 0.62655 | 0.62710 | 0.62581 |
| 200000 | 0.62587 | 0.62624 | 0.62647 | 0.62603 | 0.62622 | 0.62648 | 0.62587 |
| 200000 | 0.62631 | 0.62688 | 0.62658 | 0.62639 | 0.62639 | 0.62688 | 0.62588 |
| 300000 | 0.62447 | 0.62466 | 0.62501 | 0.62541 | 0.62493 | 0.62541 | 0.62447 |
| 300000 | 0.62491 | 0.62491 | 0.62537 | 0.62609 | 0.62510 | 0.62537 | 0.62491 |
| 300000 | 0.62483 | 0.62497 | 0.62440 | 0.62489 | 0.62475 | 0.62497 | 0.62440 |
| 300000 | 0.62559 | 0.62497 | 0.62514 | 0.62595 | 0.62537 | 0.62595 | 0.62497 |
| 300000 | 0.62487 | 0.62537 | 0.62523 | 0.62610 | 0.62539 | 0.62610 | 0.62487 |
| 400000 | 0.62483 | 0.62454 | 0.62389 | 0.62435 | 0.62475 | 0.62483 | 0.62389 |
| 400000 | 0.62469 | 0.62457 | 0.62441 | 0.62443 | 0.62481 | 0.62481 | 0.62441 |
| 400000 | 0.62384 | 0.62503 | 0.62509 | 0.62498 | 0.62415 | 0.62462 | 0.62509 |
| 400000 | 0.62443 | 0.62392 | 0.62475 | 0.62435 | 0.62445 | 0.62478 | 0.62392 |
| 400000 | 0.62471 | 0.62467 | 0.62488 | 0.62508 | 0.62485 | 0.62508 | 0.62467 |
| 500000 | 0.62425 | 0.62447 | 0.62390 | 0.62460 | 0.62428 | 0.62430 | 0.62460 |
| 500000 | 0.62448 | 0.62438 | 0.62461 | 0.62500 | 0.62448 | 0.62459 | 0.62500 |
| 500000 | 0.62442 | 0.62438 | 0.62473 | 0.62432 | 0.62404 | 0.62438 | 0.62404 |
| 500000 | 0.62472 | 0.62447 | 0.62448 | 0.62433 | 0.62454 | 0.62451 | 0.62433 |
| 500000 | 0.62417 | 0.62432 | 0.62418 | 0.62441 | 0.62491 | 0.62440 | 0.62417 |
| 600000 | 0.62422 | 0.62433 | 0.62406 | 0.62411 | 0.62347 | 0.62426 | 0.62483 |
| 600000 | 0.62380 | 0.62376 | 0.62404 | 0.62353 | 0.62422 | 0.62387 | 0.62242 |
| 600000 | 0.62449 | 0.62469 | 0.62438 | 0.62376 | 0.62367 | 0.62420 | 0.62367 |
| 600000 | 0.62402 | 0.62424 | 0.62404 | 0.62402 | 0.62483 | 0.62423 | 0.62483 |
| 600000 | 0.62432 | 0.62416 | 0.62416 | 0.62379 | 0.62414 | 0.62412 | 0.62379 |

Table 6: Experimental results of Algorithm 1 for $d = 6$. 
| $n$    | $r_0$   | $r_1$   | $r_2$   | $r_3$   | $r_4$   | Avg   | Max    | Min   |
|--------|---------|---------|---------|---------|---------|-------|--------|-------|
| 100000 | 0.67908 | 0.67916 | 0.6784 | 0.67800 | 0.67784 | 0.67860 | 0.67916 | 0.67780 |
| 100000 | 0.67816 | 0.67956 | 0.67910| 0.67942| 0.67904 | 0.67854 | 0.67922 | 0.68000 |
| 100000 | 0.67908 | 0.67814 | 0.67852| 0.67820| 0.67855 | 0.67880 | 0.67921 | 0.67908 |
| 100000 | 0.67878 | 0.67988 | 0.67808| 0.67976| 0.67900 | 0.67910 | 0.67998 | 0.67880 |
| 200000 | 0.67725 | 0.67692 | 0.67663| 0.67638| 0.67678 | 0.67678 | 0.67725 | 0.67638 |
| 200000 | 0.67649 | 0.67660 | 0.67693| 0.67644| 0.67675 | 0.67675 | 0.67728 | 0.67644 |
| 200000 | 0.67775 | 0.67656 | 0.67672| 0.67753| 0.67717 | 0.67755 | 0.67912 | 0.67656 |
| 200000 | 0.67695 | 0.67659 | 0.67672| 0.67656| 0.67708 | 0.67816 | 0.67956 | 0.67816 |
| 200000 | 0.67714 | 0.67753 | 0.67621| 0.67690 | 0.67689 | 0.67652 | 0.67753 | 0.67621 |
| 300000 | 0.67643 | 0.67735 | 0.67777 | 0.67731 | 0.67688 | 0.67735 | 0.67643 | 0.67643 |
| 300000 | 0.67645 | 0.67699 | 0.67821 | 0.67635 | 0.67678 | 0.67678 | 0.67635 | 0.67635 |
| 300000 | 0.67703 | 0.67665 | 0.67657 | 0.67693 | 0.67671 | 0.67703 | 0.67638 | 0.67638 |
| 300000 | 0.67693 | 0.67693 | 0.67633 | 0.67763 | 0.67685 | 0.67663 | 0.67638 | 0.67638 |
| 300000 | 0.67631 | 0.67619 | 0.67709 | 0.67662 | 0.67637 | 0.67652 | 0.67699 | 0.67619 |
| 400000 | 0.67627 | 0.67678 | 0.67672 | 0.67677 | 0.67671 | 0.67671 | 0.67721 | 0.67627 |
| 400000 | 0.67662 | 0.67663 | 0.67671 | 0.67638 | 0.67645 | 0.67671 | 0.67671 | 0.67671 |
| 400000 | 0.67591 | 0.67690 | 0.67637 | 0.67645 | 0.67636 | 0.67690 | 0.67691 | 0.67691 |
| 400000 | 0.67681 | 0.67672 | 0.67633 | 0.67589 | 0.67649 | 0.67681 | 0.67659 | 0.67659 |
| 400000 | 0.67661 | 0.67642 | 0.67609 | 0.67650 | 0.67649 | 0.67682 | 0.67699 | 0.67661 |
| 500000 | 0.67662 | 0.67625 | 0.67636 | 0.67673 | 0.67653 | 0.67673 | 0.67625 | 0.67625 |
| 500000 | 0.67666 | 0.67654 | 0.67654 | 0.67588 | 0.67639 | 0.67666 | 0.67588 | 0.67588 |
| 500000 | 0.67640 | 0.67652 | 0.67625 | 0.67632 | 0.67652 | 0.67652 | 0.67652 | 0.67652 |
| 500000 | 0.67679 | 0.67624 | 0.67624 | 0.67644 | 0.67679 | 0.67624 | 0.67624 | 0.67624 |
| 500000 | 0.67640 | 0.67658 | 0.67674 | 0.67614 | 0.67642 | 0.67674 | 0.67614 | 0.67614 |
| 600000 | 0.67840 | 0.67686 | 0.67867 | 0.67873 | 0.67861 | 0.67873 | 0.67840 | 0.67840 |
| 600000 | 0.67807 | 0.67841 | 0.67885 | 0.67859 | 0.67849 | 0.67884 | 0.67807 | 0.67807 |
| 600000 | 0.67826 | 0.67880 | 0.67892 | 0.67896 | 0.67872 | 0.67896 | 0.67826 | 0.67826 |
| 600000 | 0.67855 | 0.67890 | 0.67842 | 0.67821 | 0.67879 | 0.67857 | 0.67890 | 0.67821 |
| 600000 | 0.67869 | 0.67872 | 0.67867 | 0.67865 | 0.67869 | 0.67872 | 0.67865 | 0.67865 |

Table 7: Experimental results of Algorithm 1 for $d = 7$. 

28
Table 8: Experimental results of Algorithm 1 for \( d = 8 \).

| \( n \) | \( r_0 \) | \( r_1 \) | \( r_2 \) | \( r_3 \) | \( r_4 \) | Avg | Max | Min |
|---|---|---|---|---|---|---|---|---|
| 100000 | 0.72096 | 0.71836 | 0.72024 | 0.71960 | 0.71926 | 0.71968 | 0.72096 | 0.71836 |
| 100000 | 0.72024 | 0.71910 | 0.71934 | 0.71950 | 0.71965 | 0.71968 | 0.72024 | 0.71910 |
| 100000 | 0.71956 | 0.71920 | 0.71908 | 0.72098 | 0.71981 | 0.72098 | 0.71981 | 0.71908 |
| 100000 | 0.71972 | 0.71926 | 0.71938 | 0.71966 | 0.71954 | 0.71951 | 0.71972 | 0.71926 |
| 100000 | 0.71958 | 0.71940 | 0.71932 | 0.71956 | 0.72048 | 0.71967 | 0.72048 | 0.71932 |
| 200000 | 0.71921 | 0.71836 | 0.71965 | 0.71978 | 0.71920 | 0.71924 | 0.71978 | 0.71836 |
| 200000 | 0.71943 | 0.71886 | 0.71948 | 0.71907 | 0.71937 | 0.71924 | 0.71948 | 0.71886 |
| 200000 | 0.71921 | 0.71965 | 0.71992 | 0.71954 | 0.71969 | 0.71960 | 0.71992 | 0.71921 |
| 200000 | 0.71894 | 0.71955 | 0.71973 | 0.71952 | 0.71918 | 0.71938 | 0.71973 | 0.71894 |
| 200000 | 0.71939 | 0.71848 | 0.71955 | 0.71973 | 0.71952 | 0.71918 | 0.71938 | 0.71848 |
| 300000 | 0.72090 | 0.72050 | 0.72036 | 0.72111 | 0.72041 | 0.72066 | 0.72111 | 0.72036 |
| 300000 | 0.72079 | 0.72099 | 0.72051 | 0.72079 | 0.72077 | 0.72099 | 0.72077 | 0.72051 |
| 300000 | 0.72051 | 0.72076 | 0.72032 | 0.72076 | 0.72000 | 0.72047 | 0.72076 | 0.72000 |
| 300000 | 0.72043 | 0.72038 | 0.72052 | 0.72023 | 0.72013 | 0.72034 | 0.72052 | 0.72013 |
| 300000 | 0.72028 | 0.72043 | 0.72079 | 0.72067 | 0.72065 | 0.72057 | 0.72079 | 0.72028 |
| 400000 | 0.71964 | 0.71969 | 0.72048 | 0.71968 | 0.72017 | 0.71993 | 0.72048 | 0.71964 |
| 400000 | 0.71974 | 0.71987 | 0.72020 | 0.72062 | 0.72002 | 0.72009 | 0.72062 | 0.71974 |
| 400000 | 0.72015 | 0.71991 | 0.72001 | 0.72059 | 0.71945 | 0.72002 | 0.72059 | 0.71945 |
| 400000 | 0.72006 | 0.72022 | 0.72024 | 0.71998 | 0.71964 | 0.72003 | 0.72024 | 0.71964 |
| 400000 | 0.72047 | 0.72002 | 0.71995 | 0.72017 | 0.71987 | 0.72009 | 0.72047 | 0.71987 |
| 500000 | 0.71994 | 0.71979 | 0.71970 | 0.71983 | 0.71984 | 0.71982 | 0.71994 | 0.71970 |
| 500000 | 0.71974 | 0.71987 | 0.72026 | 0.71978 | 0.71965 | 0.71974 | 0.71990 | 0.72026 |
| 500000 | 0.71966 | 0.72000 | 0.71991 | 0.71982 | 0.72008 | 0.71990 | 0.72008 | 0.71966 |
| 500000 | 0.71966 | 0.71950 | 0.71956 | 0.71953 | 0.71982 | 0.71961 | 0.71982 | 0.71950 |
| 500000 | 0.71953 | 0.71945 | 0.71990 | 0.72046 | 0.71972 | 0.71981 | 0.72046 | 0.71945 |
| 600000 | 0.71971 | 0.72080 | 0.72004 | 0.71969 | 0.71984 | 0.71982 | 0.72080 | 0.71968 |
| 600000 | 0.71965 | 0.71996 | 0.71950 | 0.71966 | 0.71967 | 0.71969 | 0.71996 | 0.71950 |
| 600000 | 0.71943 | 0.71968 | 0.71952 | 0.71987 | 0.71950 | 0.71960 | 0.71987 | 0.71943 |
| 600000 | 0.72003 | 0.71962 | 0.71965 | 0.71960 | 0.71962 | 0.71970 | 0.72001 | 0.71960 |
| 600000 | 0.71951 | 0.71962 | 0.71977 | 0.71972 | 0.71998 | 0.71997 | 0.71998 | 0.71951 |
| $n$  | $r_0$  | $r_1$  | $r_2$  | $r_3$  | $r_4$  | Avg   | Max   | Min   |
|------|--------|--------|--------|--------|--------|-------|-------|-------|
| 100000 | 0.75212 | 0.75228 | 0.75282 | 0.75272 | 0.75264 | 0.75252 | 0.75282 | 0.75212 |
| 100000 | 0.75272 | 0.75186 | 0.75222 | 0.75142 | 0.75350 | 0.75234 | 0.75350 | 0.75142 |
| 100000 | 0.75190 | 0.75218 | 0.75322 | 0.75270 | 0.75236 | 0.75247 | 0.75322 | 0.75218 |
| 100000 | 0.75204 | 0.75320 | 0.75336 | 0.75290 | 0.75324 | 0.75295 | 0.75336 | 0.75204 |
| 100000 | 0.75336 | 0.75342 | 0.75308 | 0.75194 | 0.75286 | 0.75293 | 0.75342 | 0.75194 |
| 200000 | 0.75242 | 0.75252 | 0.75261 | 0.75196 | 0.75213 | 0.75233 | 0.75261 | 0.75196 |
| 200000 | 0.75257 | 0.75247 | 0.75256 | 0.75259 | 0.75250 | 0.75254 | 0.75259 | 0.75247 |
| 200000 | 0.75213 | 0.75250 | 0.75211 | 0.75206 | 0.75287 | 0.75233 | 0.75287 | 0.75206 |
| 200000 | 0.75270 | 0.75227 | 0.75240 | 0.75274 | 0.75215 | 0.75245 | 0.75274 | 0.75215 |
| 200000 | 0.75260 | 0.75189 | 0.75213 | 0.75220 | 0.75266 | 0.75230 | 0.75266 | 0.75189 |
| 300000 | 0.75157 | 0.75225 | 0.75203 | 0.75183 | 0.75190 | 0.75191 | 0.75225 | 0.75157 |
| 300000 | 0.75188 | 0.75232 | 0.75239 | 0.75215 | 0.75201 | 0.75213 | 0.75239 | 0.75188 |
| 300000 | 0.75201 | 0.75264 | 0.75266 | 0.75209 | 0.75229 | 0.75234 | 0.75266 | 0.75201 |
| 300000 | 0.75177 | 0.75227 | 0.75273 | 0.75194 | 0.75237 | 0.75221 | 0.75273 | 0.75177 |
| 300000 | 0.75256 | 0.75189 | 0.75215 | 0.75221 | 0.75165 | 0.75219 | 0.75256 | 0.75165 |
| 400000 | 0.75225 | 0.75225 | 0.75220 | 0.75194 | 0.75207 | 0.75214 | 0.75225 | 0.75194 |
| 400000 | 0.75207 | 0.75214 | 0.75206 | 0.75209 | 0.75199 | 0.75207 | 0.75214 | 0.75199 |
| 400000 | 0.75183 | 0.75218 | 0.75175 | 0.75226 | 0.75196 | 0.75200 | 0.75226 | 0.75175 |
| 400000 | 0.75209 | 0.75211 | 0.75209 | 0.75205 | 0.75226 | 0.75212 | 0.75226 | 0.75205 |
| 400000 | 0.75218 | 0.75213 | 0.75206 | 0.75197 | 0.75256 | 0.75218 | 0.75256 | 0.75197 |
| 500000 | 0.75369 | 0.75356 | 0.75349 | 0.75358 | 0.75385 | 0.75363 | 0.75385 | 0.75349 |
| 500000 | 0.75354 | 0.75379 | 0.75365 | 0.75368 | 0.75339 | 0.75365 | 0.75339 | 0.75354 |
| 500000 | 0.75373 | 0.75384 | 0.75354 | 0.75342 | 0.75361 | 0.75363 | 0.75384 | 0.75342 |
| 500000 | 0.75368 | 0.75354 | 0.75314 | 0.75377 | 0.75360 | 0.75355 | 0.75377 | 0.75314 |
| 500000 | 0.75366 | 0.75354 | 0.75347 | 0.75366 | 0.75372 | 0.75361 | 0.75372 | 0.75347 |
| 600000 | 0.75333 | 0.75300 | 0.75322 | 0.75339 | 0.75305 | 0.75320 | 0.75317 | 0.75321 |
| 600000 | 0.75304 | 0.75324 | 0.75331 | 0.753107 | 0.75276 | 0.75309 | 0.75310 | 0.75308 |
| 600000 | 0.75359 | 0.75331 | 0.75309 | 0.75310 | 0.75318 | 0.75325 | 0.75319 | 0.75316 |
| 600000 | 0.75335 | 0.75358 | 0.75326 | 0.75308 | 0.75332 | 0.75332 | 0.75331 | 0.75326 |
| 600000 | 0.75334 | 0.75338 | 0.75309 | 0.75320 | 0.75325 | 0.75325 | 0.75323 | 0.75320 |

Table 9: Experimental results of Algorithm [1] for $d = 9$. 
| $n$   | $r_0$ | $r_1$ | $r_2$ | $r_3$ | $r_4$ | Avg   | Max   | Min   |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 100000| 0.77768| 0.77860| 0.77746| 0.77798| 0.77790| 0.77800| 0.77860| 0.77746|
| 100000| 0.77726| 0.77844| 0.77850| 0.77734| 0.77790| 0.77850| 0.77746| 0.77860|
| 100000| 0.77782| 0.77770| 0.77760| 0.77770| 0.77740| 0.77764| 0.77782| 0.77740|
| 100000| 0.77760| 0.77734| 0.77786| 0.77732| 0.77726| 0.77748| 0.77760| 0.77726|
| 100000| 0.77734| 0.77754| 0.77826| 0.77796| 0.77724| 0.77767| 0.77826| 0.77724|
| 200000| 0.77788| 0.77814| 0.77824| 0.77830| 0.77819| 0.77838| 0.77888| 0.77788|
| 200000| 0.77804| 0.77806| 0.77841| 0.77821| 0.77814| 0.77841| 0.77799| 0.77799|
| 200000| 0.77810| 0.77819| 0.77841| 0.77798| 0.77812| 0.77841| 0.77794| 0.77794|
| 200000| 0.77838| 0.77820| 0.77800| 0.77869| 0.77831| 0.77869| 0.77726| 0.77726|
| 200000| 0.77745| 0.77820| 0.77793| 0.77810| 0.77789| 0.77814| 0.77850| 0.77850|
| 300000| 0.77805| 0.77791| 0.77801| 0.77805| 0.77807| 0.77802| 0.77807| 0.77791|
| 300000| 0.77801| 0.77828| 0.77797| 0.77773| 0.77789| 0.77798| 0.77828| 0.77773|
| 300000| 0.77735| 0.77793| 0.77810| 0.77805| 0.77775| 0.77784| 0.77810| 0.77735|
| 300000| 0.77817| 0.77819| 0.77877| 0.77809| 0.77817| 0.77808| 0.77819| 0.77777|
| 300000| 0.77806| 0.77789| 0.77729| 0.77827| 0.77856| 0.77809| 0.77856| 0.77745|
| 400000| 0.77782| 0.77759| 0.77809| 0.77795| 0.77780| 0.77785| 0.77809| 0.77759|
| 400000| 0.77791| 0.77826| 0.77783| 0.77759| 0.77798| 0.77791| 0.77826| 0.77759|
| 400000| 0.77789| 0.77780| 0.77757| 0.77790| 0.77793| 0.77781| 0.77793| 0.77757|
| 400000| 0.77789| 0.77777| 0.77789| 0.77801| 0.77792| 0.77789| 0.77801| 0.77777|
| 400000| 0.77789| 0.77746| 0.77794| 0.77772| 0.77766| 0.77773| 0.77794| 0.77746|
| 500000| 0.77770| 0.77782| 0.77748| 0.77784| 0.77754| 0.77767| 0.77784| 0.77748|
| 500000| 0.77748| 0.77744| 0.77805| 0.77779| 0.77777| 0.77771| 0.77805| 0.77744|
| 500000| 0.77812| 0.77790| 0.77765| 0.77769| 0.77754| 0.77778| 0.77812| 0.77754|
| 500000| 0.77775| 0.77765| 0.77772| 0.77772| 0.77778| 0.77772| 0.77778| 0.77765|
| 500000| 0.77816| 0.77780| 0.77766| 0.77766| 0.77790| 0.77784| 0.77816| 0.77766|
| 600000| 0.77793| 0.77785| 0.77769| 0.77762| 0.77780| 0.77778| 0.77775| 0.77775|
| 600000| 0.77772| 0.77762| 0.77764| 0.77768| 0.77786| 0.77770| 0.77770| 0.77772|
| 600000| 0.77789| 0.77788| 0.77759| 0.77758| 0.77750| 0.77769| 0.77765| 0.77760|
| 600000| 0.77748| 0.77772| 0.77745| 0.77748| 0.77754| 0.77753| 0.77755| 0.77751|

Table 10: Experimental results of Algorithm 1 for $d = 10$. 