Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 27, no. 1, pp. 35–44 (2023)
DOI: 10.7155/jgaa.00612

# Efficient Point-to-Point Resistance Distance Queries in Large Graphs

*Craig Gotsman*[1] *Kai Hormann*[2]

[1]New Jersey Institute of Technology
[2]Università della Svizzera italiana

Submitted: October 2022 Reviewed: December 2022 Revised: January 2023
Accepted: February 2023 Final: February 2023 Published: February 2023
Article type: Regular paper Communicated by: Giuseppe Liotta

**Abstract.** We describe a method to efficiently compute point-to-point resistance distances in a graph, which are notoriously difficult to compute from the raw graph data. Our method is based on a relatively compact hierarchical data structure which "compresses" the resistance distance data present in a graph, constructed by a nested bisection of the graph using compact edge-cuts. Built and stored in a preprocessing step (which is amenable to massive parallel processing), efficient traversal of a small portion of this data structure supports efficient and exact answers to resistance distance queries. The size of the resulting data structure for a graph of $n$ vertices is $O(nk \log n)$, where $k$ is the size of a balanced edge-cut of the graph. Exact queries then require $O(k \log n)$ worst-case time and $O(k)$ average-case time. Approximate values may be obtained significantly faster by applying standard dimension reduction techniques to the "coordinates" stored in the structure.

## 1 Introduction

Resistance distance between two vertices $i$ and $j$ in a graph $G(V, E)$ containing $n$ vertices is a global measure of distance between vertices in a graph, taking into account the lengths of all possible paths between them. Its name stems from the fact that the graph may be thought of as a resistor network with the edges representing unit-value resistors, and the resistance distance $r(i, j)$ is then the effective resistance between the two junctions $i$ and $j$ in this network. Resistance distance is a useful measure with many applications in chemistry and graph analytics, dating back to the seminal paper by Klein and Randić [8]. For more details, the interested reader is referred to the recent survey by Evans and Francis [5].

Resistance distance may be expressed in terms of the classical "flow" in networks [1, Sec. 10.2]. This is an assignment of non-negative real numbers $f_e$ (the flow) to all edges $e$ of a network, along

*E-mail addresses:* gotsman@njit.edu (Craig Gotsman) kai.hormann@usi.ch (Kai Hormann)

with a direction per edge, such that the total incoming flow is equal to the total outgoing flow at all nodes except the source and sink. At the source, the total outgoing flow is 1, as is the total incoming flow at the sink. In this case $r(i,j)$ is just the square of the norm of the minimal flow,

$$r(i,j) = \min \left\{ \sum_{e \in E(G)} f_e^2 : f \text{ is a unit flow in } G \text{ from source } i \text{ to sink } j \right\}.$$

Another way to express the resistance distance as an optimal value uses vectors of real values for the $n$ graph vertices [2, Corollary 6]:

$$\frac{1}{r(i,j)} = \min \left\{ \sum_{(u,v) \in E(G)} (x_u - x_v)^2 : x \in \mathbb{R}^n \text{ and } x_i - x_j = 1 \right\}.$$

The resistance distance is known to be a metric and is usually computed in one of two ways: the "direct" method,

$$r(i,j) = \Gamma_{i,i} + \Gamma_{j,j} - 2\Gamma_{i,j}, \tag{1}$$

where $\Gamma = L^+$ is the pseudoinverse of the positive semi-definite symmetric Laplacian matrix $L$ of $G$, or the "spectral" method,

$$r(i,j) = \sum_{k=2}^{n} \frac{1}{\lambda_k} \left(\phi_i^k - \phi_j^k\right)^2, \tag{2}$$

where $\lambda_k$ and $\phi^k$ are the $k$-th eigenvalue and normalized eigenvector of $L$ (such that $\lambda_1 = 0$). While these are straightforward formulae, they compute simultaneously the resistance distance between all $O(n^2)$ pairs of vertices in the graph, or, at the very least, the resistance distance between a vertex and *all* $O(n)$ other vertices. They require solving global systems before anything can be done for any pair of vertices, with run-time complexity of $O(n^3)$. As such, the methods are not practical to use in an application where queries on the resistance distance between arbitrary pairs of vertices in very large graphs are to be computed rapidly on demand. This arises, for example, in graph sparsification [12], where an edge is deemed important if and only if its resistance distance is large. Thus an edge may be discarded if its resistance distance is small, since there exist many other paths in the graph connecting its two endpoints. Similarly, in the opposite problem of link prediction [10], a link is predicted to appear in a network in the future if the resistance distance between its two endpoints is small, implying that there already is a strong indirect connection between the two edge endpoints.

It may be possible to approximate $r(i,j)$ using (2) by computing only the $m \ll n$ eigenvalue/eigenvector pairs for $L$ having the smallest eigenvalues, as these dominate the spectral sum, thus avoiding the computation of all $n$ pairs. Indeed, if preprocessing and storing the results is allowed, to be used later in a fast online query, then $m$ eigenvalues and eigenvectors of $L$ may be precomputed and stored, at a storage cost of $O(m)$ "coordinates" per graph vertex, and a resistance distance query may be answered in $O(m)$ time. Alas, for many graphs, the value of $m$ necessary to obtain a sufficiently accurate approximation may be quite large, sometimes $m = O(n)$. Spielman and Srivastava [12] expand on this idea to form a (high probability) $\epsilon$-approximation to the resistance distance based on (2), while avoiding explicit computation of eigenvectors. They preprocess the graph using *random projections* and an efficient linear solver, resulting in a matrix with $m = O(\frac{\log n}{\epsilon^2})$ entries ("coordinates") per vertex. This matrix may then be used to answer

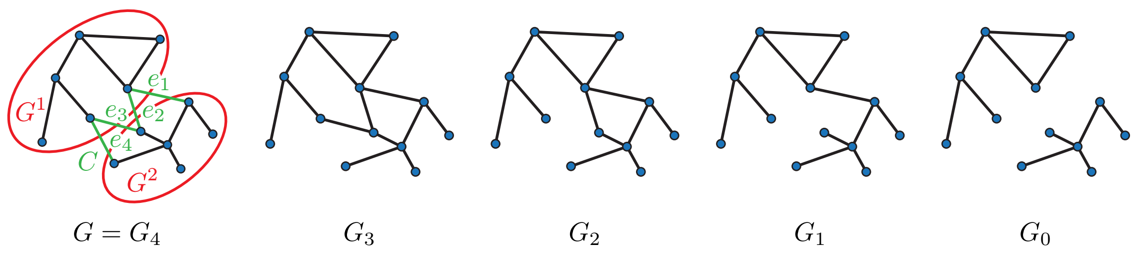

Figure 1: *Left*: graph $G$ with edge-cut $C$ (green) consisting of $k = 4$ edges. *Left to right*: partial graphs $G_i$ resulting from successively removing one edge of the cut at a time.

online queries in $O(m)$ time. Although asymptotically efficient, the values of $\epsilon$ and the implicit constants needed to produce high-quality approximations can make this expensive in practice.

In this paper we describe an alternative method to efficiently answer point-to-point resistance distance queries. This is achieved by preprocessing a graph by recursive edge-cuts, resulting in storage of $m = O(k \log n)$ "coordinates" per vertex and leads to a method to compute $r(i,j)$ *exactly* in $O(m)$ time, where $k$ is the size of a balanced edge-cut of $G$, for example, $k = O(\sqrt{n})$ for planar graphs. Using the same stored data, a very good approximation of $r(i,j)$ may be obtained in much less time if we apply dimension reduction to the coordinates.

## 2 Resistance distance through an edge-cut

If $G(V,E)$ is a connected graph with vertex set $V$ and edge set $E$, an edge-cut of $G$ is a subset $C \subset E$ such that $G(V, E \setminus C)$ consists of two disconnected components $G^1 = G(V_1, E_1)$ and $G^2 = G(V_2, E_2)$, where $V = V_1 \cup V_2$, $V_1 \cap V_2 = \varnothing$ and $E = E_1 \cup C \cup E_2$, $E_1 \cap E_2 = \varnothing$. Denote by $B_1$ the *boundary* of $V_1$ relative to $V_2$, that is, $B_1 = V_1 \cap V(C)$, and similarly $B_2$ the boundary of $V_2$. If $v_1 \in V_1$ and $v_2 \in V_2$, then we say that $C$ *separates* $v_1$ and $v_2$, and show how to express $r_G(v_1, v_2)$ as a function of $r_{G_i}(v_1, u_i)$ and $r_{G_i}(v_2, w_i)$ *only*, where $u_i$ and $w_i$ are the vertices in $B_1$ and $B_2$, respectively and $G_i$ are graphs simpler than $G$, as will be described below. Note that we have added a subscript to the resistance distance function $r$ to indicate the graph to which it is applied.

The starting point for our analysis is the "perturbation formula" of Yang and Klein [14, Theorem 2.1], also mentioned by Ranjan et al. [11], which describes how the resistance distance between two vertices $v_1$, $v_2$ in a graph changes when a new edge is added to the graph.

**Theorem 1** [14] *Let $G(V,E)$ be an undirected graph and $G' = (V, E \cup e)$ be the graph after a new edge $e = (u,w)$ is added. Denote by $r_G(v_1, v_2)$ the resistance distance between vertices $v_1$ and $v_2$ in the graph $G$. Then,*

$$r_{G'}(v_1, v_2) = r_G(v_1, v_2) - \delta(G, v_1, v_2, u, w),$$

*where*

$$\delta(G, v_1, v_2, u, w) = \frac{\big((r_G(v_1,u) - r_G(v_1,w)) - (r_G(v_2,u) - r_G(v_2,w))\big)^2}{4(1 + r_G(u,w))}.$$

Note that the addition of new edge always *decreases* the resistance distance, as expected. We apply Theorem 1 to analyse the effect of the edges in an edge-cut on the resistance distance.

**Theorem 2** *Let $G(V,E)$ be an undirected graph containing an edge-cut $C$ consisting of the $k$ edges $\{e_i = (u_i, w_i) : i = 1, \dots, k\}$ partitioning $V$ into $V_1$ and $V_2$. Denote by $G_i$ the graph containing all the edges of $G$* except *the edges $\{e_j : j = i+1, \dots, k\}$ (see Figure 1). If $v_1, u_i \in V_1$ and $v_2, w_i \in V_2$ are on opposite sides of the edge-cut, then*

$$r_G(v_1, v_2) = r_{G_0}(v_1, u_i) + r_{G_0}(v_2, w_i) + 1 - \Delta(G, v_1, v_2, C), \tag{3}$$

*where*

$$\Delta(G, v_1, v_2, C) = \sum_{i=2}^{k} \delta(G_i, v_1, v_2, u_i, w_i). \tag{4}$$

**Proof:** Theorem 1 may be applied $k-1$ times by starting with the graph $G_1$ containing just *one* edge of $C$ and repeatedly adding one more edge of $C$ at a time, so that

$$r_G(v_1, v_2) = r_{G_1}(v_1, v_2) - \Delta(G, v_1, v_2, C). \tag{5}$$

Observing that the removal of the first edge $e_1 = (u_1, w_1)$ disconnects $G_1$ into $G_0$ consisting of two disconnected vertex sets $V_1$ and $V_2$ with $u_1 \in V_1$ and $w_1 \in V_2$, we trivially have

$$r_{G_1}(v_1, v_2) = r_{G_0}(v_1, u_1) + r_{G_0}(v_2, w_1) + 1,$$

from which (3) results. □

## 3 Nested bisection

Assuming the cut $C$ is balanced, namely, partitions $V$ into two sets $V_1$ and $V_2$ of approximately equal size, then Theorem 2 is applicable in approximately 50% of the cases, when vertex pairs $(v_1, v_2)$ are separated by $C$. In this case, it provides an efficient way to compute $r_G(v_1, v_2)$ if only $r_{G_i}(v_1, u_i)$ and $r_{G_i}(v_2, w_i)$ are known. However, if $v_1$ and $v_2$ are not separated by $C$, for example, without loss of generality, $v_1, v_2 \in V_1$, then obviously

$$r_{G_1}(v_1, v_2) = r_{G_0}(v_1, v_2)$$

and the computation of (5) may proceed recursively on $G^1 = G_0$ to evaluate the first term. This means that an edge-cut $C^1$ must be found for $G^1$, and then (5) applied again on $G^1$. This process will continue recursively and terminate when $v_1$ and $v_2$ are separated by the edge-cut, at which point (3) is applied. Note that the superscript $i$ in $G^i$ denotes the subgraph of $G$ treated at recursion level $i$.

Thus, in order that this recursive method apply to *any* vertex pair, a *nested bisection tree* [6] must be constructed, where edge-cuts are computed recursively until small enough vertex sets are obtained. Each node of the binary tree represents a subgraph of $G$ and an edge-cut of the subgraph. Each descendant of the node represents one of the two connected components obtained from the edge-cut. Leaves of the tree represent small subgraphs which are not partitioned further. If the edge-cuts are balanced, the height of the binary tree will be $O(\log n)$. For any vertex $v$, we say that $v$ is *associated with* all nodes of the tree that contain $v$, and also associated with the subgraphs and edge-cuts in those nodes. In the opposite direction, we say that these subgraphs and edge-cuts are associated with $v$.

Once the binary tree is constructed, the resistance distance between $v_1$ and $v_2$ may be computed by traversing the tree and considering all nodes associated with $v_1$ *and* associated with $v_2$, namely

$$r_G(v_1, v_2) = r_{G^d}(v_1, u) + r_{G^d}(v_2, w) + 1 - \sum_{i=0}^{d} \Delta(G^i, v_1, v_2, C^i),$$

where the $G^i$ are the subgraphs of $G$ contained in the nodes along the path of the tree from its root at depth 0 to the node at depth $d$ containing the edge-cut $C^i$ separating $v_1$ from $v_2$. The vertices $u$ and $w$ are the endpoints of the final edge of the edge-cut $C^d$ in $G^d$.

## 4 Implementation details

### 4.1 Preprocessing

In order to use the method described in the previous section to efficiently answer resistance distance queries between any two vertices of a given graph, it is necessary to precompute the bisection tree and resistance distances between each vertex and the endpoints of its associated cut edges in the relevant subgraphs. This is performed just once on the graph, namely, we compute and store "coordinates" of each vertex to all associated cut edges. This will result in a very compact representation of the resistance distance information and will facilitate efficient computation of the resistance distance between any two vertices at query time. In practice the graph is recursively partitioned and a binary tree built. The resistance distances from each vertex to all associated edge-cuts are then computed *postorder* (bottom-up). For a leaf of the tree, the resistance distance matrix is computed simply by applying (1) to the subgraph stored in that leaf. Once we have the resistance distance matrix of two components of a graph, $r_{G_1}$ and $r_{G_2}$, the resistance distance matrix $r_G$ of the complete graph is computed by inserting one edge of the cut at a time and updating the matrix accordingly. Adding the first edge $(u, w)$ (where $u \in V(G_1)$ and $w \in V(G_2)$) triggers the computation

$$r_G(v_1, v_2) := \begin{cases} r_{G_1}(v_1, v_2), & v_1, v_2 \in G_1, \\ r_{G_2}(v_1, v_2), & v_1, v_2 \in G_2, \\ r_{G_1}(v_1, u) + r_{G_2}(w, v_2) + 1, & v_1 \in G_1, v_2 \in G_2. \end{cases}$$

Adding any of the other edges $e_i = (u_i, w_i)$ triggers the update

$$r_G(v_1, v_2) := r_G(v_1, v_2) - \delta(G, v_1, v_2, u_i, w_i).$$

As this proceeds, the following "coordinates" (one per each edge $e_i = (u_i, w_i)$ in the cut) are stored for all $v \in G$,

$$c(v, e_i) := \begin{cases} r_G(v, u_i), & i = 1, v \in G_1 \\ r_G(v, w_i), & i = 1, v \in G_2 \\ \dfrac{r_G(v, u_i) - r_G(v, w_i)}{2\sqrt{1 + r_G(u_i, w_i)}}, & i > 1. \end{cases} \tag{6}$$

The end result is a tree data structure containing $O\left(\sum_{i=1}^{d} |C^i|\right)$ values for vertex $v$, where $C^i$ are the edge-cuts associated with $v$. There are $d = O(\log n)$ such edge-cuts.

Balanced edge-cuts of a graph may be obtained using a variety of methods (see the survey by [3]), some implemented quite efficiently in the METIS software package [7]. A simple method is the spectral method [13] which uses the so-called Fiedler eigenvector of the graph Laplacian matrix. Whether these edge-cuts are compact or not depends on the type of graph. For example, it is well known that both planar graphs and unit disk graphs with $n$ vertices admit balanced edge-cuts of size $O(\sqrt{n})$ [4, 9]. Unit disk graphs model well radio, wifi and IoT networks. For these cases, the storage requirements of our method are $O(\sqrt{n}\log n)$ values per vertex, as opposed to the naive $O(n)$ values per vertex if all pairwise resistance distances are precomputed and naively stored for lookup at query time.

## 4.2 Answering point-to-point queries

At query time, given a pair of vertices $v_1, v_2 \in G$, the coordinates computed in (6) and stored in preprocessing are used to compute $r_G(v_1, v_2)$ exactly and efficiently by traversing a path of the binary tree, starting at the root and ending at the node whose associated cut separates $v_1$ and $v_2$. Starting with $r_G(v_1, v_2) = 0$, at each node with associated cut $C$ consisting of $k$ edges $e_i = (u_i, w_i)$, $i = 1, \ldots, k$, this is updated as

$$r_G(v_1, v_2) := r_G(v_1, v_2) + \sum_{i=2}^{k} \big(c(v_1, u_i) - c(v_2, w_i)\big)^2. \tag{7}$$

At the terminal tree node that separates $v_1$ and $v_2$, we also perform the following update involving the first edge in the associated cut

$$r_G(v_1, v_2) := c(v_1, u_1) + c(v_2, w_1) + 1 - r_G(v_1, v_2). \tag{8}$$

The time complexity of this computation is $O(k \log n)$, where $k$ is the size of an edge-cut. However, observing that for a balanced edge-cut, 50% of the possible vertex pairs will be separated already at depth $d = 1$, 25% at depth $d = 2$, 12.5% at depth $d = 3$ and so on, we conclude (by summing a geometric series) that the average time complexity for a query is $O(k)$.

We note that as the resistance distance is accumulated bottom-up, its value can only decrease (because of the negative sign in front of $r_G(v_1, v_2)$ in (8)). This is to be expected, as climbing the tree towards the root exposes more and more of the graph, thus more possible paths between the two vertices, which can only reduce the resistance distance between them.

## 5 Approximating the resistance distance

The contribution to the resistance distance between two vertices in a graph separated by an edge-cut of size $k$, as described in (7), is the sum of squares of the differences of $k - 1$ values associated with each of the edges, namely the square of an Euclidean distance between the embeddings of the vertices in a space of dimension $k - 1$. As such, it is amenable to dimension reduction by principal component analysis (PCA) to a space of much smaller dimension which captures most of this distance, in which the coordinates are sorted in decreasing order of “importance”. This has the potential to reduce the storage requirements of this method dramatically, at the cost of a minor loss of accuracy of the computed resistance distance. This optimization also reduces the computation runtime, since (4) may be replaced by a sum of much less than $k$ terms. Note that since the number of vectors $m$ is typically much larger than the dimension $k$, we use the

PCA method which performs a cheap eigendecomposition of the covariance matrix of size $k \times k$ (rather than the standard multidimensional scaling method which would require a very expensive eigendecomposition of a matrix of size $m \times m$), followed by a projection of the input vectors on to the reduced space.

Another possible optimization leading to an even shorter query time is possible when vertices are close to each other in the graph. This means that separation occurs deep in the nested bisection tree and most of the resistance distance is accumulated at the lower levels. In this case, the resistance distance may be approximated well by using most of the coordinates at the lower levels, and much fewer at the higher levels. At each level, coordinates are used, starting from the most important, as long as the contribution of that coordinate is above a threshold. Once the threshold is crossed, all other coordinates at that level are ignored completely.

## 6 Examples

We have implemented our methods in MATLAB and run them on some sample graphs. Our code may be found here. Edge-cuts are computed using the spectral method [13] and resistance distance of the leaf clusters are computed using the Laplacian pseudo-inverse (1). Although our entire implementation is serial, we note that the method is "embarrassingly parallel" in the sense that it can easily be parallelized to significantly reduce both the preprocessing time and the query time.

Some results on a planar graph and a unit disk graph are shown in Figures 2 and 3. Figure 2 illustrates the computation of the *exact* resistance distance between two vertices in both types of graphs, using four levels of edge-cuts within a nested bisection in both examples until the vertices are separated. In the planar graph of 401 vertices, 62 "coordinates" are required, and in the unit disk graph of 379 vertices, 113 "coordinates" are required. Figure 3 illustrates how the distance may be *approximated* well using far less "coordinates" than needed for the exact distance computation. A unit disk graph containing 4,653 vertices is preprocessed with a nested bisection of depth 9 and the dimension of the coordinate space is reduced at each level using PCA. As a result the resistance distance may be approximated well: 0.8047 instead of 0.8057 using 40 instead of 254 coordinates, and 0.6054 instead of 0.6063 using 38 instead of 291 coordinates. In contrast, were we to use that number of coordinates in the truncated spectral approximation (2), the approximation would have been completely off by two orders of magnitude.

## 7 Discussion and conclusion

We have described a method that preprocesses a graph $G$, building a data structure that may be used to rapidly answer online queries approximating the point-to-point resistance distance between two vertices of the graph. The fundamental idea is that the resistance distance between two vertices on opposite sides of an edge-cut $C$ may be expressed using *only* values relating the two vertices and the edges of the cut in each of the two components $G_1$ and $G_2$. This is analogous to the fact that the shortest-path distance $d(v_1, v_2)$ between these two vertices may also be expressed in a similar manner as

$$d_G(v_1, v_2) = \min\left\{d_{G_1}(v_1, u_i) + d_{G_2}(w_i, v_2) + 1 : (u_i, w_i) \in C\right\}$$

Equation (4) expresses the resistance distance in $G$ in terms of the resistance distances to the separating edges in $G_i$, which contain partial edge-cuts. It would be more satisfying if it could be

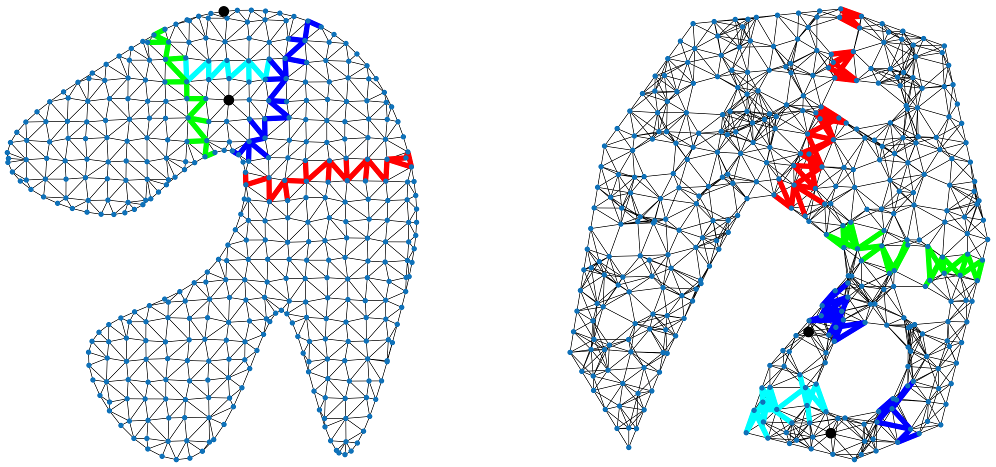

Figure 2: Example of the portion of the nested bisection needed to compute the resistance distance between the two black vertices in two (unweighted) graphs. *Left*: planar graph containing 401 vertices. Four recursive edges-cuts are shown, each partitioning the relevant subgraph into two balanced components until the two vertices are separated. The edge-cuts are coloured (red, green, blue, cyan) with increasing depth $(0, 1, 2, 3)$. In order to compute the required resistance distance, each of the two vertices must store "coordinates" related *only* to the 62 edges in their associated cuts. As the cuts are traversed top-down and the resistance distance then refined bottom-up, the values start at $r = 1.0151$ and decrease to 0.9572, 0.9173, and the final value 0.9164. *Right*: Unit disk graph within polygonal domain containing 379 vertices. Similar to (left), four cuts are required to separate the two black vertices, in total 113 edges. The resistance distance is refined bottom-up as 0.5004, 0.3772, 0.3741, 0.3741.

expressed in terms of resistance distances to the fully separated graph $G_0$. Theorems 4.9 and 4.12 in [14] have such expressions for the cases $k = 2, 3$, but it seems like the expression for a larger $k$ would be quite cumbersome.

Our method relies on the fact that the graph admits compact edge-cuts. By "compact" we mean $O(n^p)$, preferably for some $p \leq \frac{2}{3}$. This is true for planar graphs, minor-free graphs, unit disk graphs, hyperbolic random graphs and geometric inhomogeneous random graphs [4]. For more general graphs, such as the scale-free ("power law") graphs that model the internet and social networks, the edge-cuts will be less compact and we have to rely on the compression of "coordinates" in the approximation stage to reduce the resulting number to manageable proportions. For example, in a social network graph consisting of 7,623 vertices and 27,805 edges, two vertices are separated at the second level (level 1 = 1,393 edges, level 2 = 577 edges), thus in principle requiring 1,970 coordinates to compute the resistance distance between them exactly. After dimension reduction, $415 + 49 = 464$ coordinates suffice, incurring an error of 1.3%.

Finally, we mention that while our description has dealt with the simple case of unit resistances on the edges (i.e., unweighted graphs), the entire analysis applies also to arbitrary resistance values (i.e., weighted graphs). In this case, the off-diagonal entries of the Laplacian matrix mentioned in (1) and (2) are the negative inverses of the edge weights, and the value "1" appearing in the update formulae should be replaced by the appropriate edge weight.

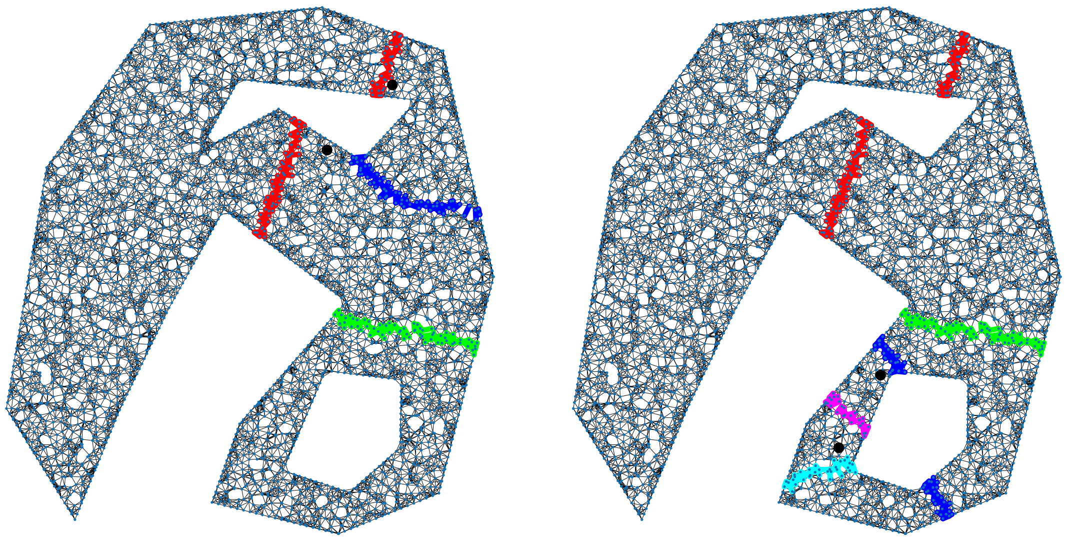

Figure 3: Unit disk graph on 4,653 vertices with 8.6 neighbors on average, decomposed using nested bisection of depth 9. *Left*: exact resistance distance of 0.8047 between the two black vertices is computed using three levels of the tree (red, green, blue) involving $105+78+71=254$ coordinates. The approximate distance of 0.8057 may be computed on these levels using 24 of the 55 reduced coordinates on the top level, 5 of 49 at the second level, and 11 of 50 at the third, in total 40 reduced coordinates. *Right*: Exact resistance distance of 0.5956 is computed on five levels (red, green, blue, cyan, magenta) using $105+78+69+39+31=322$ coordinates. Approximate resistance distance of 0.5958 is computed using $4/55+11/49+5/39+7/28+11/20$ reduced coordinates, in total 38 reduced coordinates.

## References

[1] R. B. Bapat. *Graphs and Matrices*. Springer, London, 2nd edition, 2014. doi:10.1007/978-1-4471-6569-9.

[2] B. Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer, New York, 1998. doi:10.1007/978-1-4612-0619-4.

[3] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. Springer, Cham, 2016. doi:10.1007/978-3-319-49487-6_4.

[4] P. Carmi, M. K. Chiu, M. J. Katz, M. Korman, Y. Okamoto, A. van Renssen, M. Roeloffzen, T. Shiitada, and S. Smorodinsky. Balanced line separators of unit disk graphs. *Comput. Geom.*, 86:Article 101575, 14 pages, Jan. 2020. doi:10.1016/j.comgeo.2019.101575.

[5] E. J. Evans and A. E. Francis. Algorithmic techniques for finding resistance distances on structured graphs. *Discrete Appl. Math.*, 320:387–407, Oct. 2022. doi:10.1016/j.dam.2022.04.012.

[6] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10(2):345–363, 1973. doi:10.1137/0710032.

[7] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998. The METIS source code is available at http://glaros.dtc.umn.edu/gkhome/views/metis. doi:10.1137/S1064827595287997.

[8] D. J. Klein and M. Randić. Resistance distance. *J. Math. Chem.*, 12:81–95, 1993. doi:10.1007/BF01164627.

[9] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979. doi:10.1137/0136016.

[10] B. Pachev and B. Webb. Fast link prediction for large networks using spectral embedding. *J. Complex Netw.*, 6(1):79–94, Feb. 2018. doi:10.1093/comnet/cnx021.

[11] G. Ranjan, Z.-L. Zhang, and D. Boley. Incremental computation of pseudo-inverse of Laplacian. In Z. Zhang, L. Wu, W. Xu, and D.-Z. Du, editors, *Combinatorial Optimization and Applications*, volume 8881 of *Lecture Notes in Computer Science*, pages 729–749. Springer, Cham, 2014. doi:10.1007/978-3-319-12691-3_54.

[12] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

[13] D. A. Spielman and S.-H. Teng. Spectral partitioning works: Planar graphs and finite element meshes. *Linear Alg. Appl.*, 421(2–3):284–305, Mar. 2007. doi:10.1016/j.laa.2006.07.020.

[14] Y. Yang and D. J. Klein. A recursion formula for resistance distances and its applications. *Discrete Appl. Math.*, 161(16–17):2702–2715, Nov. 2013. doi:10.1016/j.dam.2012.07.015.