Study of Planar Separators
—— CS 598 Project Report

Qixin (Stark) Zhu

Table of Contents

1. Introduction

"Divide and conquer" is one of the oldest and most widely used techniques for designing efficient algorithms [1]. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem [2]. This strategy can be successfully and efficiently applied to graph problems, provided we can quickly separate the graph into roughly equal subgraphs [1].

The separator theorem states that for any n-vertex planar graph G=(V, E) (n = |V|) and for any weight function w: V → $\mathbb{R}^+$, V can be partitioned into 3 sets A, B, S ⊆ V such that (1) A and B are α-balanced: w(A), w(B) ≤ $\alpha \cdot w(V)$ for some $\alpha \in (0,1)$; (2) A and B are separated: no edge joins a vertex in A with a vertex in B; (3) separator S is small: $|S| \leq f(n)$; (4) the partition can be found efficiently in linear time.

Finding planar separators correctly and efficiently is important because it is a fundamental tool for many more complicated tools or algorithms. Separators can be used recursively to form a separator hierarchy, namely r-division. An r-division of G is a decomposition into O(N/r) edge-disjoint pieces (called regions), each of which has vertices less than or equal to r and has $O(\sqrt{r})$ boundary vertices. The root of the separator hierarchy tree is the entire graph itself, and the two children are the roots of separator trees constructed recursively for the subgraphs A and B induced by the root-level separator S. A naïve method of constructing this r-division, by Frederickson (1986) [7], is to apply the linear-time separator when traversing the separator tree, which will take a total of O(N logN) time. A second approach is to preprocess the graph with a ρ-clustering of size $\sqrt{r}$, and contract each piece into a single node, thus shrinking the graph into n'=O($N/\sqrt{r}$) vertices. Doing naïve recursion on this shrunk graph takes O($N/\sqrt{r}$ logn) time, which is governed by the time of expanding pieces back and doing another O(logr) levels of recursion. So the total time cost of this approach is O(N logr). The time bound of separator hierarchy construction can be further improved to linear by using appropriate data structures to perform the partitions, given by Goodrich (1995) [7].

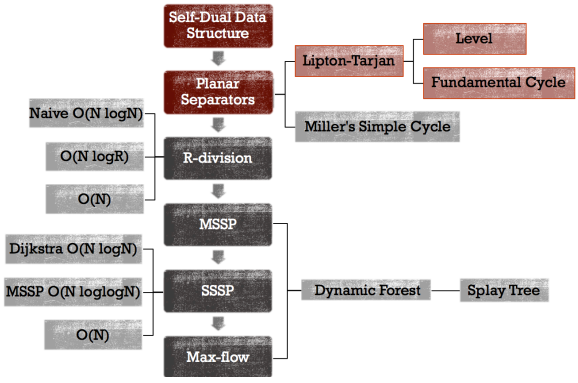


Figure 1. Road map of long-term research plan

The long-term plan of this research topic (Figure 1) includes further implementation and evaluations of all 3 above mentioned r-division methods, using them as tools to solve Multi-Source Shortest Path (MSSP) problems. Further Single-Source Shortest Path (SSSP) problem solution

will have 2 approaches for study. One is using MSSP as a subroutine to find all-to-all shortest path length with in a region, then treat the region as a big merged vertex and solve the SSSP in the merged graph, combines results with path lengths inside each region to reconstruct the answers for the original graph. This approach has O(N loglogN) time complexity, which the same and worth comparison to another approach described in Klein (1997) [10]. The SSSP algorithm can be further used in the practical study of the O(N logN) Max-flow problem in the preprocessing phase [11], which is expected to be the main time consuming Step.

2. Existing Research

Holzer (2009) implemented and evaluated the 3 linear-time planar separator algorithms mentioned above. They presented a comprehensive experimental study of the algorithms applied to a variety of graphs scaled from 10^1 to 10^4 vertices [12]. They concluded that the Fundamental Cycle Separator (FCS) almost always outperforms other algorithms (Level Separator and Lipton-Tarjan Separator), even for graphs with large diameter. In their implementation, no dual graph is explicitly stored. Therefore, to find a non-tree edge forming the cycle separator, they iterate through all non-tree edges and keep track of the vertices inside the cycle in order to return the "best" (in terms of balance and separator size) cycle found. This approach helps the algorithm to find a good cycle within linear time and thus performs well in most cases, but does extra work and consumes more time than the original algorithm, which returns the first non-tree edge that forms a valid separator.

Fox-Epstein (2016) implemented the Miller's Simple Cycle Separator (SCS) and compared to Holzer's work. Similar to Holzer (2009), there is no explicit dual graph stored. Fox-Epstein confirmed Holzer's finding that FCS work well in most inputs, but also notice FCS may output large cycle depending on the root selection of the primal spanning tree. They propose to use SCS which not only have worst-case guarantee but also output a cycle as the separator with competitive performance. They also push the scale of testing graph to 40 million vertices with 12 core CPU and 48GB RAM.

3. Self-Dual Data Structure

A self-dual data structure is an overlay of the sorted incidence lists of primal graph G and its dual G* [1]. It consists of two maps, one for vertices of G and the other for faces of G. Each dart object stores pointers of its head vertex, tail vertex, left and right face, reverse dart, successor and predecessor dart, next and previous dart. Each record in the 2 maps points to an arbitrary incidental dart of the vertex or face, as shown in Figure 2. As a result, a self-dual data structure of G is also a representation of G* and all pointers are explicitly stored. There is a tradeoff between storing all pointers explicitly versus computing them on the fly. It is easier to track and debug with explicit pointers, but it also takes more space to store the information. This could be a problem for RAM when graph gets too big and is a potential place for future improvements.

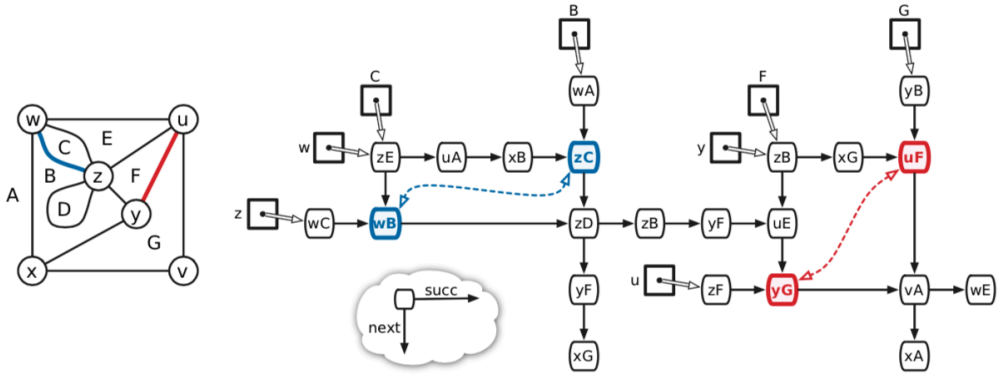


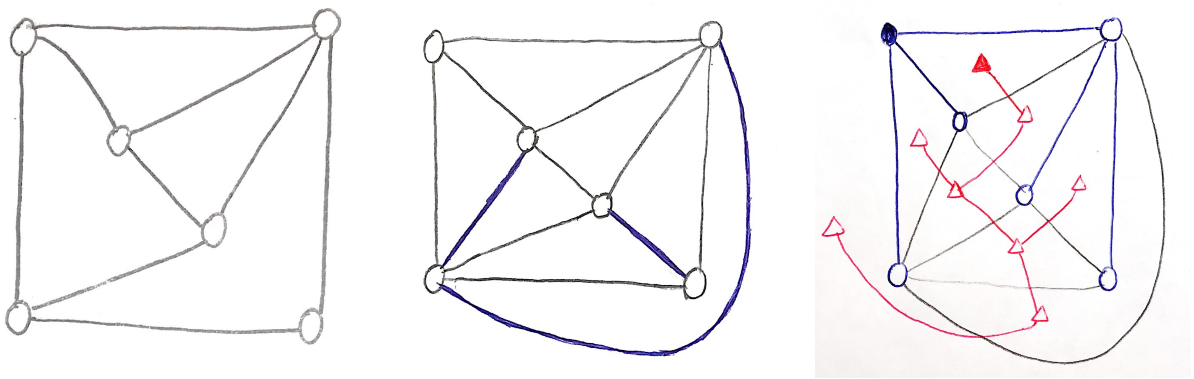
Figure 2. Partial representation of a self-dual data structure of a sample graph [1]

The Self-Dual Data Structure is implemented in this project because there is no existing data structure that performs the operations needed for updating and modifying primal-dual planar graphs. The closest existing data structure is called the half-edge data structure, in which each undirected edge is stored as a pair of half-edges for each direction. This data structure is most often used for rendering in computer graphic or meshing in finite element analysis. In both cases, the mesh is done by a separate tool which can generates a valid planar graph to be stored in the data structure. Once the mesh is generated, the vertices, edges and faces will not change. Changes can be done by re-generating a new mesh and stored in the data structure. Thus, the data structure itself does not support operations such as deleting an edge and merging its two incidental faces, adding a vertex on a face and split that face or deleting parallel edges and self-loops. Therefore, the entire Self-Dual Data Structure is implemented.

3.1. Graph Operations

Flatten

Flatten takes a planar graph, not necessary simple, checks for all parallel edges and self-loop then removes them to make the graph simple. The result on the sample graph shown in Figure 2 is shown in Figure 3(a).



(a) Flattened sample graph (b) Triangulated graph after flattened (c) Tree-coTree

Figure 3. Operations supported by self-dual data structure

Triangulate
Triangulate takes a flattened simple planar graph, examine all its faces. If a face has degree more than 3, pick an incidental vertex V and add edges between V and all its non-neighboring vertices incidental to the same face, thus splitting the face into multiple triangles. This step is needed for building binary Dual Tree (coTree) for FCS and Lipton-Tarjan Separator. One of the many triangulation results of the previously flattened graph is shown in Figure 3(b).

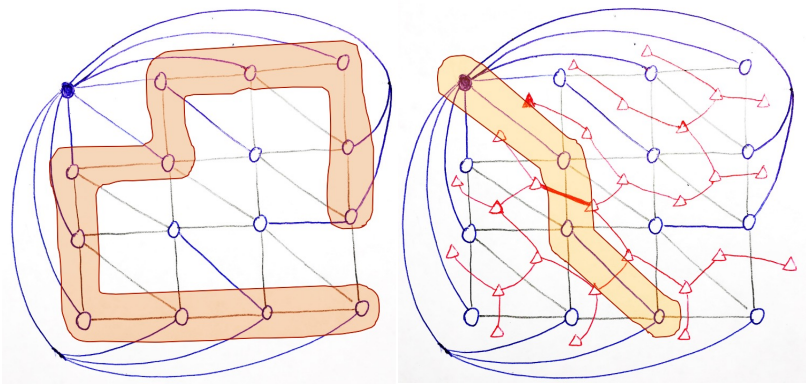
Build Tree (primal) and coTree (dual)
After triangulation, the spanning tree of the primal graph (Tree) can be found using BFS, which then gives the unique corresponding dual tree (coTree). The Tree and coTree is shown in Figure 3(c). The circles are primal vertices, the triangles are dual vertices (primal faces), and the root vertex is solid. By selecting the root of dual tree as the right face of an edge in the primal tree, it is ensured that the dual tree root has a degree no more than 2. This root choice further ensures the dual tree is a binary tree and guarantees the existence of an edge separating the tree by 1/3-2/3 in the worst case.

4. Separator Algorithms

4.1. Level Separator

To find the level separator, one can pick a random vertex as root, use BFS to build the spanning tree of the primal graph and define levels of all vertices to be the distance from the root vertex. Return the median level as the separator. "median" level Lm is defined to be the level such that the number of vertices with level less than Lm is less than half, but the number of vertices with level less than or equal to Lm is greater than or equal to half. The set of vertices return by level separator is balanced but may be big. The entire algorithm takes linear time to build and traverse the tree and count vertices number at each level.

Level separator works well for grid graphs, as shown in Figure 4(a), in the sense that when a balanced level separator is found, it will have size of $O(\sqrt{N})$. If the grid is not triangulated, pick a corner vertex as root, the algorithm will find the diagonal vertices as the median level, which contains exactly $\sqrt{N}$ vertices. If the grid is triangulated (in the not so elegant way as shown), the algorithm may find a rectangular shape cycle of size at most $4\sqrt{N}$ which separates the grid as "inside" and "outside".



(a) Level Separator (b) Fundamental Cycle Separator

Figure 4. Separators for sample grid graph

4.2. Fundamental Cycle Separator

Similar as in level separator, the algorithm picks a random vertex as root, uses BFS to find the primal spanning tree. The corresponding coTree is unique and can be built easily in linear time. Traverse the coTree and return the edge that separates it into 2 balanced sub-trees of size between 1/3 and 2/3. The cycle formed by adding that edge into the primal spanning tree is returned as the separator. The entire algorithm takes linear time to build Tree-coTree and linear time to find the edge from coTree to form the cycle.

The fundamental cycle separator may give separators of large size due to the large depth of the primal tree. This means it is dependent on the structure of the input graph as well as the selection of the root vertex and the building process of the spanning tree. The result is shown for the sample grid graph as in Figure 4(b).

4.3. Lipton-Tarjan Separator

Lipton and Tarjan (1979) [3] combined the ideas from the naïve separators and constructed a balanced separator whose sized can be limited to $O(\sqrt{N})$. First, construct the BFS spanning tree and find the median level, same as the level separator method. Then find a level above and below the median level Lm that contains $\sqrt{N}$ vertices, named as L_a and L_z, as shown in Figure 5. Since all levels in between have size more than $\sqrt{N}$, there are at most $\sqrt{N}$ such levels. The algorithm then constructs a new graph by replacing all vertices above level L_z with one super vertex as root. It also deletes all vertices below level L_z. Triangulate the newly constructed graph and applies FCS to it. The cycle C found in this modified graph is guaranteed to have size at most $2\sqrt{N}$ because the tree depth is at most $\sqrt{N}$. The union of L_a, L_z and C can be returned as the Lipton-Tarjan Separator. The original graph is separated into 4 parts, which can be combined to form 2 subgraphs satisfying the balance requirement. In this approach, the size of the final separator is guaranteed to be no more than $4\sqrt{N}$. The entire algorithm takes linear time to perform operations similar to level separator and FCS, and the extra time to build a new graph is also linear.

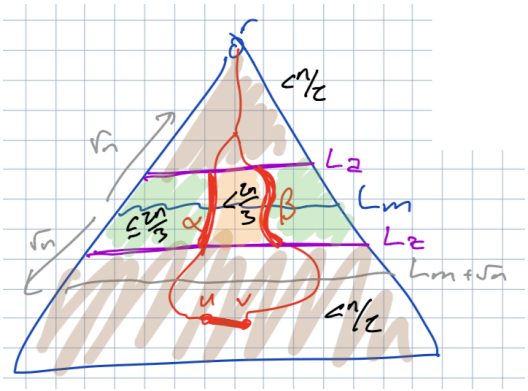


Figure 5. Sketch for Lipton-Tarjan algorithm from reference [1]

The Lipton-Tarjan algorithm is slightly modified in this project. Instead of building a new graph and re-triangulate it, the original coTree is re-assigned with new weight such that dual vertices (primal faces) outside the central zone (between L_a and L_z) is assigned zero weight. In this way, the FCS is actually finding a cycle balanced in the number of faces/vertices inside the central zone,

which is equivalent to applying FCS on the new modified graph. This change improves the constant factor of the algorithm's performance by avoiding build new graph and do triangulation.

5. Performance Evaluation

5.1. Metrics

Separator Size is defined to be the number of vertices in the separator returned by the above algorithms. The smaller separator size is preferred.
Balance Ratio is defined to be the ratio between the size of the two subgraphs separated by the separator. Note that the size of one subgraph is the number of vertices in the subgraph, which also includes the vertices in the separator. The "perfect" balance is equivalent to balance ratio 1.0, and the "worst" is 2.0 since both Tree and coTree are binary, in which case a 1/3-2/3 separation is guaranteed.
Runtime is defined to be the clock time of invoking find-separator method. None of the preprocesses such as read graph from disk, flattening and triangulation is included in runtime.

5.2. Datasets

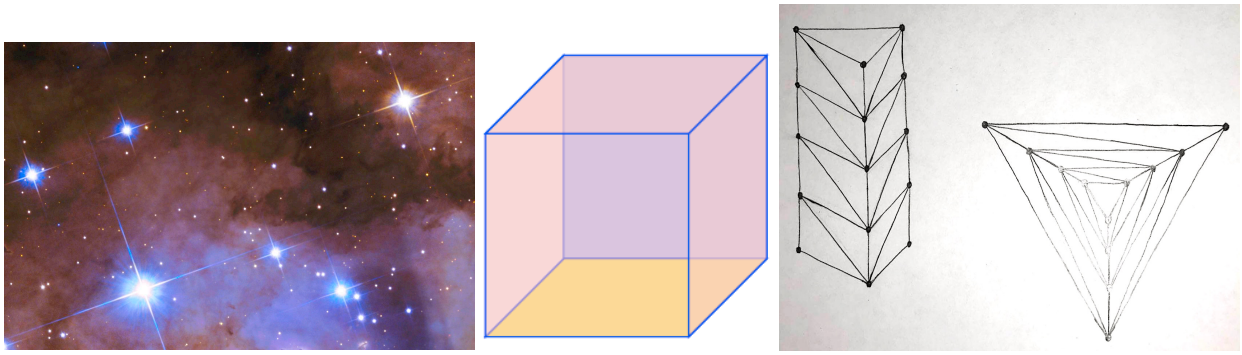
The number of vertices in testing graphs is scaled from 10^1 to 10^5. For each testing graph, 32 trials are run with a random selected vertex as root, then average and standard deviation is shown in the figures. The algorithm supports vertices and faces with different weights, but in this stage of the project, all elements are weighted equally.

Grids
A serials of grid planar graphs are generated from a fraction of a photo from NASA [14], as shown in Figure 6(a). Each vertex represents a pixel in the rectangular photo and is connected with 4 neighboring vertices (up, down, left, right) through a pair of darts.

Spheres
Spheres are generated from a cube by iteratively adding a vertex to each face, connecting to all its incidental vertices and splitting that face, as shown in Figure 6(b). The most important characteristic of sphere type graphs is that the size of their BFS spanning tree levels grows linearly with the number of vertices

Cylinders
Cylinders are generated from a triangle by iteratively adding a parameter triangle outside existing ones then connecting and triangulating incidental triangles, as shown in Figure 6(c). The important characteristic of cylinder type graphs is that the depth of their BFS spanning trees grows linearly with the number of vertices, whereas the optimum size of separators is clearly constant 3 (one triangle can separate its inside from its outside).

Random graphs
Random graphs are generated from a triangle by iteratively adding a vertex on a randomly chosen face and triangulating that face.



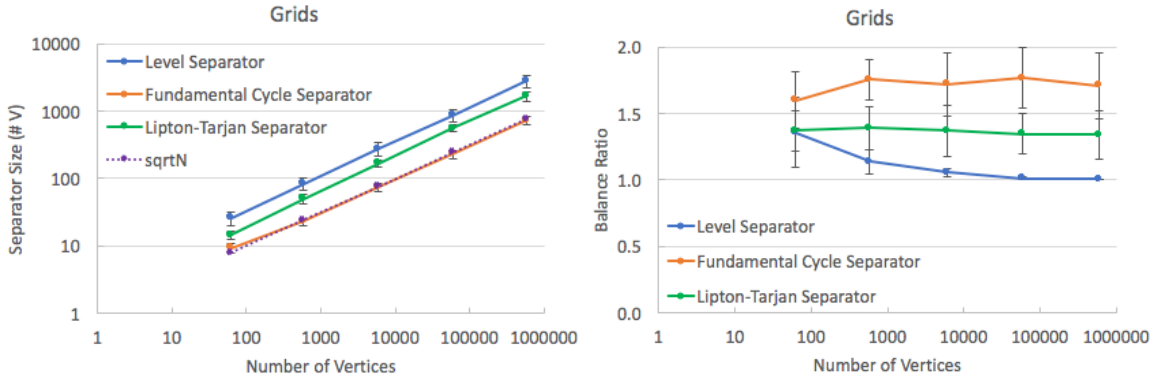
(a) NASA photo for grids (b) Base cube to generate spheres (c) Cylinder and planar embedding
Figure 6. Datasets of planar graphs

6. Results and Discussions

6.1. Grids

Separators have $O(\sqrt{N})$ in size as shown in Figure 7(a). 3 types separator algorithms differ in terms of constant factor, among which FCS has constant factor almost 1 and level separator has constant close to 4. It is noticed that although the worst-case theoretical size bound is $4\sqrt{N}$ for Lipton-Tarjan, in grids the constant factor is about 2, less than the worst-case.

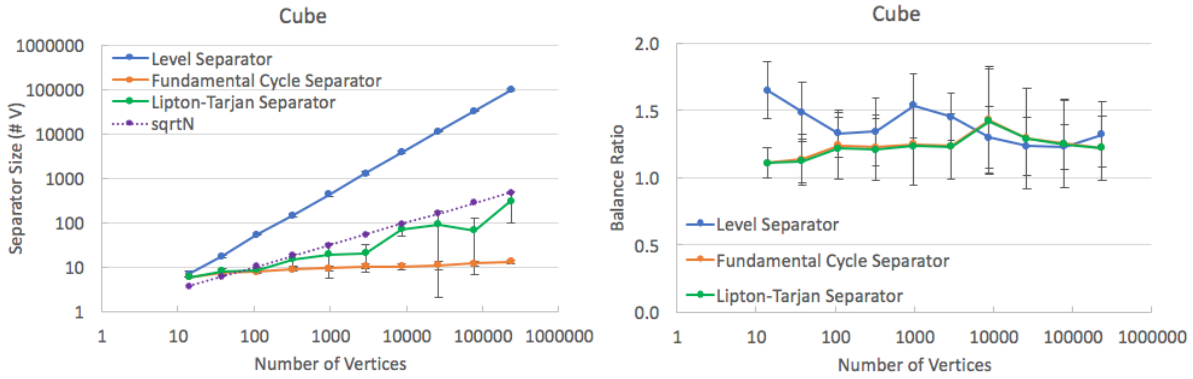
Balance ratios are between 1.0 and 2.0 as expected. It is noted that FCS gives the worst balance ratio on average, because the FCS algorithm returns the first edge it finds that can split coTree into 1/3-2/3, when there are edges that provide much better-balanced splits



(a) Separator size (b) Balance ratio
Figure 7. Results for grids

6.2. Spheres

Due to the large middle levels of spheres, the size of level separators grows linearly with the number of vertices in the sphere graphs. Lipton-Tarjan still follows the $O(\sqrt{N})$ trend, with some variance, as guaranteed by the theoretical proof [3]. FCS performs best in sphere graphs with an almost O(logN) growth, since it only needs to find a parameter of the sphere to cut it into 2 halves.



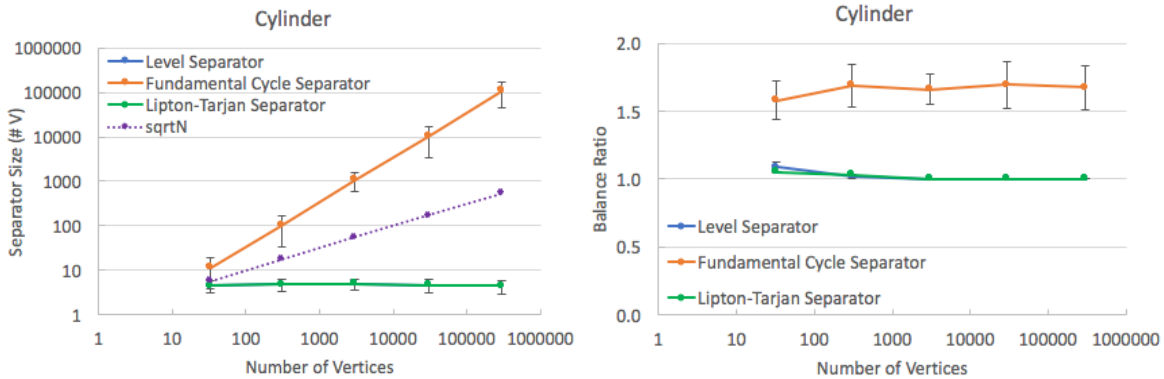
(a) Separator size (b) Balance ratio

Figure 8. Results for spheres

6.3. Cylinders

The characteristic of large spanning tree depth of cylinders leads to very different results from previous. Since the size of FCS is highly dependent on the tree depth, it grows linearly in the case of cylinders. The balance ratio is also close to 2.0 as explained before. It is noticed that 1 out of 32 trials of FCS did find an optimum separator of size 3. The FCS needs to be lucky "twice" to find an optimum separator: once in selecting the random root luckily need the middle of the cylinder, once in finding the optimum non-tree edge at the first time traversing coTree. Both Lipton-Tarjan and Level separator can find the optimum separators of constant size for cylinders, mush less than the $O(\sqrt{N})$ theoretical bound. They also have almost 1.0 balance ratio, which means they both find the separator near the middle of the cylinder regardless where the random root is, which is an impressive property.

The observation regarding separator size is different from Holzer (2009) [13] due the different implementation choices. In their paper, since no explicit dual graph stored and no dual tree built, they examine all non-tree edges of the primal spanning tree and keep track of the inside of the cycle formed by that non-tree edge. They return the edge that gives a good balance ratio and generates the smallest cycle, which is the best option after the spanning tree is determined. This approach yields a better result while maintain the linear running time. In this project, FCS returns the first edge forming a balanced cycle, but may not be the best choice. Therefore, Holzer (2009) concluded FCS performs well in most cases, but the results shown here indicates that FCS performs not so well on large diameter graphs.

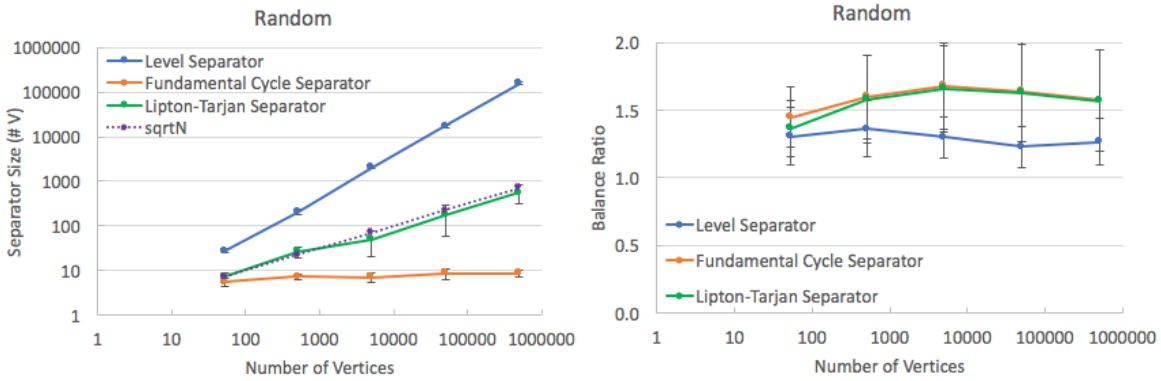


(a) Separator size (b) Balance ratio

Figure 9. Results for cylinders

6.4. Random graphs

The trend of separator sizes is similar to the sphere graphs, discussion is skipped here.

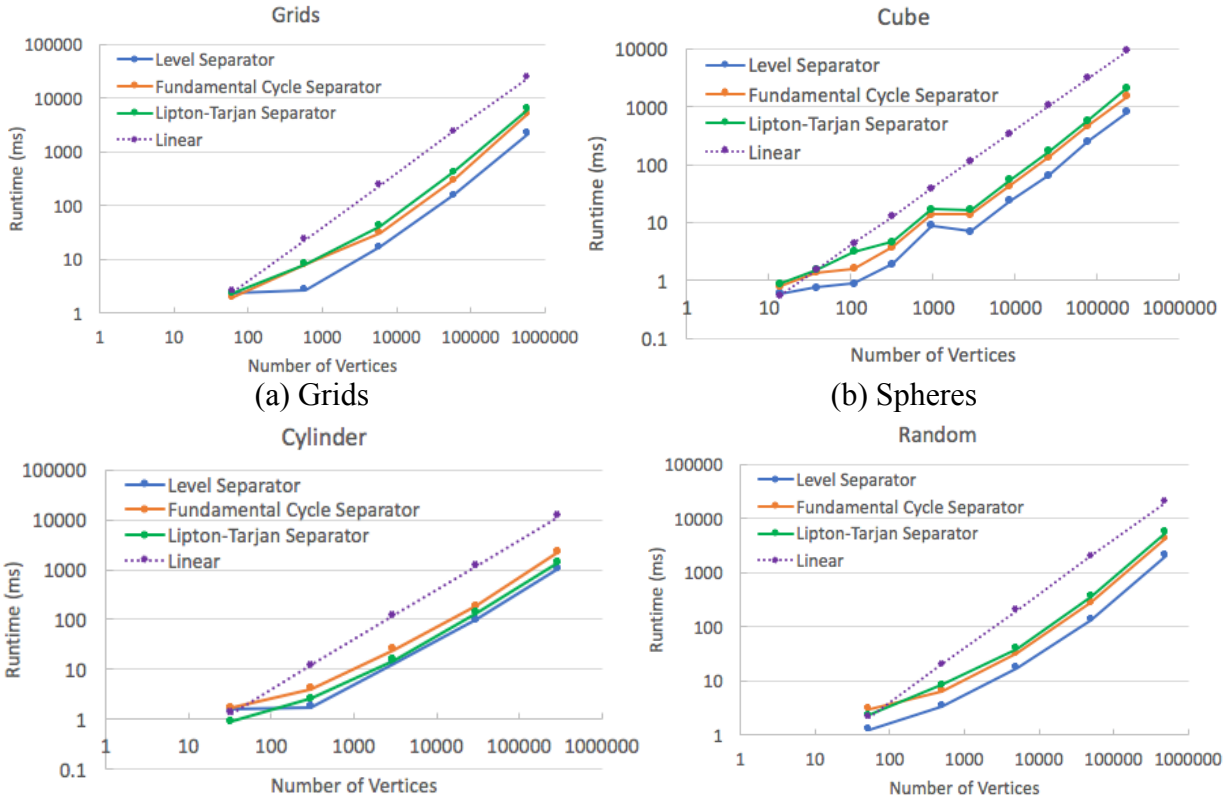


(a) Separator size (b) Balance ratio

Figure 10. Results for random graphs

6.5. Runtime

Figure 10 (a)-(d) clearly shows that all 3 separator algorithms run in linear time. There may be some non-trivial overhead to do sanity check for small graphs, which is negligible when graph is big enough.



(a) Grids (b) Spheres

(c) Cylinders (d) Random graphs

Figure 10. Results for random graphs

7. Future work

1. More types of testing graphs can be used in performance evaluation. One type is to duplicate an existing planar graph, and connect both parts with a few added vertices. This type of Twin graphs has an explicit know minimum size separator. The other type is to use real world road maps as planar graphs, which may need some preprocessing to ensure the connectivity and planarity.
2. FCS can be optimized using ideas similar to Holzer (2009). A set of non-tree edges that can form balanced cycles can be maintained, then all such candidates can be examined in order to find the smallest cycle with respect to the given primal spanning tree. The overall time complexity will remain O(N), but the size of separator may be reasonably reduced.
3. Lipton0Tarjan implementation can be optimized as well. The current algorithm returns the union of entire level L_a, L_z and the cycle. It is possible to only select the vertices that are effective separators. For example, if the central orange region in Figure 5 is one subgraph and the rest forms the other subgraph, the separator only needs to include the central part of level L_a, L_z and the cycle. This potential improvement will not change the theoretical size bound of Lipton-Tarjan separator, but may be helpful in practice, without increase the time complexity.
4. Miller's Simple Cycle separator is worth implementing and comparing with all the above 3 algorithms. As suggested by Fox-Epstein (2016) [13], the Simple Cycle separator has some nice properties for further usage in r-division, and has competitive performance on almost all kinds of planar graphs.
5. Further study may use separators as a tool to develop more complicated tools like r-division, and/or use them to implement more algorithms including but not limited to MSSP, SSSP and max-flow.

8. Conclusions

Runtime of all 3 planar separator algorithms (Level separator, FCS, Lipton-Tarjan separator) are linear with respect to the number of vertices in the planar graphs. The size of separators found by FCS grows linearly when the graph has large diameter. The size of separators found by Level separator grows linearly in sphere type graphs. Lipton-Tarjan yield separators of much more stable sizes, which is consistent with the theoretical guaranteed worst-case $O(\sqrt{N})$ bound. All source codes in this project is saved in public repository on GitHub [15].

The implementation of FCS and Lipton-Tarjan can be improved in practice and their performances and comparison with Miller's Simple Cycle separators worth further study.

9. References

[1] Jeff's notes
[2] https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
[3] Lipton, Tarjan, A separator theorem for planar graphs, SIAM Journal on Applied Mathematics, 36 (2): 177–189, 1997
[4] Djidjev, On the problem of partitioning planar graphs, SIAM Journal on Algebraic and Discrete Methods, 3 (2): 229–240, 1982.
[5] Miller, Finding small simple cycle separators for 2-connected planar graphs, Journal of Computer and System Sciences, 32 (3): 265–279, 1986
[6] Djidjev, Venkatesan, Reduced constants for simple cycle graph separation, Acta Informatica, 34 (3): 231–243, 1997.
[7] Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, SIAM J. Computing, 1004-1022, 1987.
[8] Goodrich, Planar separators and parallel polygon triangulation, J. Comput. System Sci. 51, 374–389, 1995.
[9] Klein, Multiple-source shortest paths in planar graphs, Proceedings of 16th ACM-SIAM Symposium on Discrete Algorithms, 146-155, 2005
[10] Klein, Rao, Rauch, Subramanian, Faster shortest-path algorithms for planar graphs, Journal of Computer and System Sciences, 55 (1): 3–23, 1997
[11] Erickson. Maximum Flows and Parametric Shortest Paths in Planar Graphs. Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, 794-804, 2010.
[12] Martin Holzer, Frank Schulz, Dorothea Wagner, Grigorios Prasinos, and Christos D. Zaroliagis. 2009. Engineering planar separator algorithms. ACM Journal of Experimental Algorithmics 14 (2009), 5:1.5–5:1.31
[13] Eli Fox-Epstein, Shay Mozes, Phitchaya Mangpo Phothilimthana, and Christian Sommer. 2016. Short and simple cycle separators in planar graphs. J. Exp. Algorithmics 21, 2, Article 2.2
[14] https://www.spacetelescope.org/images/heic1509a/
[15] https://github.com/StarkZhu/PlanarGraphAlgo